По договору между издательством «Символ-Плюс» и Интернет-магазином «Books.Ru – Книги России» единственный легальный способ получения данного файла с книгой ISBN 5-93286-069-3, название «Запускаем Linux, 4-е издание» – покупка в Интернет-магазине «Books.Ru – Книги России». Если Вы получили данный файл каким-либо другим образом, Вы нарушили международное законодательство и законодательство Российской Федерации об охране авторского права. Вам необходимо удалить данный файл, а также сообщить издательству «Символ-Плюс» (piracy@symbol.ru), где именно Вы получили данный файл.

RUNNING LINUX

Fourth Edition

Matt Welsh, Matthias Kalle Dalheimer,
Terry Dawson and Lar Kaufman

O'REILLY®

ЗАПУСКАЕМ LINUX

Четвертое издание

Мэтт Уэлш, Маттиас Калле Далхаймер,
Терри Доусон и Лар Кауфман

СИМВОЛ®

Санкт-Петербург — Москва
2004

Мэтт Уэлш, Маттиас Калле Далхаймер,
Терри Доусон и Лар Кауфман

Запускаем Linux, 4-е издание

Перевод С. Маккавеева

Главный редактор	*А. Галунов*
Зав. редакцией	*Н. Макарова*
Научный редактор	*М. Деркачев*
Редактор	*А. Петухов*
Корректоры	*С. Беляева, С. Журавина*
Верстка	*Н. Гриценко*

Уэлш М., Далхаймер К., Доусон Т. и Кауфман Л.
Запускаем Linux, 4-е издание. – Пер. с англ. – СПб: Символ-Плюс, 2004. – 736 с., ил.
ISBN 5-93286-069-3

Бестселлер «Запускаем Linux», выходящий уже четвертым изданием, признан в сообществе Linux лучшим руководством для начинающих. В книге рассказано обо всем, что необходимо для понимания, установки и начала работы с ОС Linux. Авторы, опытные специалисты по Linux, не ограничиваются рассмотрением ядра, или оболочки, или графического интерфейса, или даже основных приложений. Они предвидят возможные трудности и предлагают надежные решения, которые обеспечат успешную работу в Linux. Глубоко освещаются вопросы конфигурирования, на которых часто спотыкаются пользователи, но о которых умалчивают другие издания. Материал излагается просто и ясно, но представлен достаточно полно, чтобы служить руководством для начинающих и в то же время обеспечить новой информацией опытных пользователей.

В четвертом издании глубже рассмотрены установка системы и ее администрирование, конфигурирование настольных рабочих сред и сетевое взаимодействие. В число новых тем входят: популярная конфигурация LAMP, объединяющая Linux, Apache, MySQL и PHP, настольная рабочая среда GNOME, основы безопасности и брандмауэры, управление пакетами в Debian, почтовый агент Postfix и настройка звуковой подсистемы.

ISBN 5-93286-069-3
ISBN 0-596-00272-6 (англ)

© Издательство Символ-Плюс, 2004
Authorized translation of the English edition © 2002 O'Reilly Media, Inc. This translation is published and sold by permission of O'Reilly Media, Inc., the owner of all rights to publish and sell the same.

Все права на данное издание защищены Законодательством РФ, включая право на полное или частичное воспроизведение в любой форме. Все товарные знаки или зарегистрированные товарные знаки, упоминаемые в настоящем издании, являются собственностью соответствующих фирм.

Издательство «Символ-Плюс». 199034, Санкт-Петербург, 16 линия, 7,
тел. (812) 324-5353, edit@symbol.ru. Лицензия ЛП N 000054 от 25.12.98.
Налоговая льгота – общероссийский классификатор продукции
ОК 005-93, том 2; 953000 – книги и брошюры.
Подписано в печать 27.02.2004. Формат 70x100 1/16 . Печать офсетная.
Объем 46 печ. л. Тираж 3000 экз. Заказ N
Отпечатано с диапозитивов в Академической типографии «Наука» РАН
199034, Санкт-Петербург, 9 линия, 12.

Оглавление

Предисловие 9

1. Введение в Linux 19

Об этой книге 20
Краткая история Linux 22
Кто использует Linux? 25
Характеристики системы 27
Характеристики программного обеспечения 32
Copyright для Linux 46
Программное обеспечение Open Source и философия Linux 47
Источники информации по Linux 53
Получение помощи 55

2. Подготовка к установке Linux 59

Дистрибутивы Linux 59
Подготовка к установке Linux 61

3. Установка и начальное конфигурирование 69

Установка программного обеспечения Linux 69
Послеустановочные процедуры 86
Устранение неполадок 91

4. Основные команды и понятия Unix 105

Регистрация пользователя в системе 106
Установка пароля 107
Виртуальные консоли 108
Часто используемые команды 108
Оболочки 112
Полезные клавиши и как ими пользоваться 114
Сокращенный ввод с клавиатуры 115
Расширение имен файлов 117
Сохранение выводимых данных 118

Что такое команда? . . . 121
Запуск страницы руководства . . . 123
Страницы руководства . . . 123
Владение файлом и права доступа . . . 126
Изменение владельца, группы и прав доступа . . . 129
Стартовые файлы . . . 132
Важные каталоги . . . 134
Обслуживающие программы . . . 135
Процессы . . . 136

5. Основы управления системой . . . 141

Сопровождение системы . . . 142
Начальная загрузка системы . . . 146
Запуск системы и инициализация . . . 156
Однопользовательский режим . . . 163
Останов системы . . . 164
Файловая система /proc . . . 165
Управление учетными записями пользователей . . . 167

6. Управление файловыми системами, свопингом и устройствами . . . 178

Управление файловыми системами . . . 178
Управление пространством свопинга . . . 196
Файлы устройств . . . 200

7. Обновление программного обеспечения и ядра . . . 204

Утилиты архивирования и сжатия . . . 205
Обновление программного обеспечения . . . 215
Использование RPM . . . 224
Использование dpkg и apt . . . 227
Сборка нового ядра . . . 237
Загружаемые драйверы устройств . . . 250
Автоматическая загрузка модулей . . . 254

8. Другие задачи администрирования . . . 256

Создание резервных копий . . . 256
Запуск задач по расписанию с помощью cron . . . 264
Управление системными журналами . . . 269
Управление службами печати . . . 272
Установка атрибутов терминала . . . 306
Действия в аварийных ситуациях . . . 307

9. Редакторы, средства обработки текста, графика и печать 313

Редактирование файлов с помощью vi 313
Редактор Emacs 323
Обработка текста и обработка документов 338
Графика 359
Настройка и работа со звуком в Linux 364
Печать 378

10. Установка X Window System 387

Основы X 388
Требования к оборудованию 389
Установка XFree86 393
Настройка XFree86 395
Запуск XFree86 403
Возможные проблемы 404

11. Настройка среды X 405

Основы настройки X Window 406
K Desktop Environment 409
Приложения KDE 419
Настольная среда GNOME 427
Приложения GNOME 433
Другие приложения X 440

12. Совместимость с Windows и Samba 447

Совместный доступ к дискам с помощью MTools 449
Совместное использование дисковых разделов 453
Выполнение программ MS-DOS и Windows под Linux 469

13. Языки программирования 471

Программирование для gcc 472
Файлы проектов 483
Программирование на языке оболочки 494
Использование Perl 496
Java 505
Другие языки 511

14. Инструментарий программиста 516

Отладка с помощью gdb 516
Средства программирования 534
Интегрированные среды разработки 558

15. Протоколы TCP/IP и PPP 560
Сетевое взаимодействие по протоколу TCP/IP 560
Коммутируемые линии и протокол PPP 581
PPP поверх ISDN 589
ADSL 598
Настройка NFS и NIS 599

16. World Wide Web и электронная почта 605
World Wide Web 605
Электронная почта 616

17. Основы безопасности 636
Общий взгляд на систему безопасности 636
Первые шаги в организации защищенной системы 639
Настройка TCP wrappers 643
Брандмауэры: фильтрация IP-пакетов 646

18. LAMP 659
MySQL 661
PHP 669
Сервер LAMP в действии 674

A. Источники информации по Linux 677

B. Установка Linux на системы Digital/Compaq Alpha 685

Библиография 702

Алфавитный указатель 707

Предисловие

Недостаточно владеть техникой. Нужно превзойти технические приемы, и тогда искусство станет безыскусственным, проистекая из подсознания.

Дайсецу Судзуки (Daisetsu Suzuki) (1870–1966)

Эта книга о Linux, бесплатной операционной системе с открытым исходным кодом (open source), благодаря которой происходят изменения в мире компьютеров. В этой книге показывается, что, изучив эту мощную и бесплатную операционную систему, можно начать работать на компьютере совершенно по-иному. Разрабатываемая свободно образованной группой, состоящей из тысяч добровольцев, связанных через Интернет, Linux противоречит традиционному компьютерному «мэйнстриму». Linux зародилась как настоящее подпольное движение, можно сказать, хакеров-партизан и в значительной мере вернула атмосферу радости, открытий и самостоятельности в культуру вычислительной техники, в которой в настоящее время доминируют корпорации. Мы приглашаем вас погрузиться в этот мир, получить удовольствие и присоединиться к той массе людей, которые понимают, что значит управлять синхросигналом дисплея и устанавливать корневой раздел в образе ядра с помощью *rdev*.

Цитата из Дзэн, приведенная в начале, подытоживает философию, которой мы придерживаемся в этой книге, рассчитанной на пытливого и творческого читателя, готового с головой окунуться в мир Linux и желающего разобраться с самыми основаниями этой системы. Linux – это восстание против коммерческих операционных систем, и многие ее пользователи принадлежат к тому типу людей, которые любят находиться на переднем крае технологических новшеств. Разумеется, любой читатель может без особых хлопот установить и запустить систему Linux, но цель этой книги – помочь глубже вникнуть в систему и позволить полностью овладеть характерным для Linux стилем мышления. Вместо того чтобы вдаваться в подробности запутанных моментов, мы стараемся объяснить принципы работы системы, чтобы читатель мог самостоятельно решать возникающие проблемы. Предоставляя в общее распоряжение опыт, накопленный несколькими специалистами по Linux, мы надеемся вселить в вас достаточную уверенность, чтобы назвать себя настоящим гуру Linux. (Вот ваш первый коан: какой смысл заниматься хакерством в одиночку?)

У вас в руках четвертое издание «Запускаем Linux» – классической, по общему мнению, книги по установке, сопровождению и изучению системы Linux. Первое издание (на англ. языке) вышло еще в 1996 году и ведет свое происхождение от свободно распространявшейся книги «Linux Installation and Getting Started», которая все еще циркулирует в Интернете. С тех пор книга подверглась ряду улучшений и изменений с целью привести ее в соответствие с последними разработками в мире Linux. В третьем издании к Мэтту Уэлшу (Matt Welsh) и Лару Кауфману (Lar Kaufman) в качестве соавтора присоединился Калле Далхаймер (Kalle Dalheimer), который выполнил большинство обновлений и в настоящем издании. Терри Доусон (Terry Dawson) является автором ряда материалов, относящихся к системе безопасности.

В данной редакции мы полностью обновили информацию, касающуюся установки, конфигурирования и обучения, чтобы привести ее в соответствие с последними дистрибутивами Linux (включая Red Hat и его производные, SuSE и Debian) и многочисленными прикладными пакетами. Однако в основной материал книги не было внесено больших изменений. Это было сделано намеренно: в первых трех изданиях мы приложили много усилий, чтобы сделать книгу возможно более зрелой несмотря на постоянное развитие самой Linux. Никакая книга не может точно отразить все, что известно о Linux. (Вы не найдете здесь глав, посвященных применению Linux в космическом челноке или описывающих слабости алгоритмов шифрования, хотя соответствующие работы проведены.) Эта философия оправдала себя в полной мере и сохранена в настоящем, исправленном издании. Мы надеемся, что книга послужит вам долгие годы.

Мир Linux сильно изменился со времени предыдущего издания «Запускаем Linux». Не только повысились производительность и устойчивость системы, но и непрерывно расширяется спектр приложений, от офисных приложений до серверов баз данных. Под Linux работают критически важные приложения и популярные интернет-сайты, поисковые механизмы и сети доставки контента. Все чаще Linux выбирают в качестве ОС для настольных систем, а такие графические среды, как KDE и GNOME, еще больше облегчают эффективное использование Linux.

В предисловии к первому изданию говорилось о том, что у Linux есть все возможности полностью изменить лицо мира операционных систем для PC. Теперь мы можем с полной уверенностью сказать, что это предсказание оправдалось. Linux с поразительной силой вторглась в центральное русло развития вычислительной техники: о ней рассказывают все крупные средства массовой информации, она дала толчок так называемой революции Open Source (разработке программного обеспечения с открытым кодом) и многими рассматривается как наиболее реальный конкурент, способный противостоять засилью Microsoft на рынке операционных систем. Большинство оценок сходится на том, что количество пользователей Linux в мире превысило 200 миллионов. Развитие Linux привело к тому, что большинству пользователей не обязательно разбираться в сложностях драйверов устройств, файлах конфигурации XFree86 и начальных загрузчиках. Тем не менее мы считаем целесообразным дать читателю некоторое представление о внутреннем устройст-

ве системы, чтобы он мог понять механизм ее работы, даже несмотря на то, что для обычной работы с Linux в этом нет строгой необходимости.

Почему людям нравится Linux

Есть много причин, по которым люди при выборе операционной системы останавливаются на Linux. Они могут быть связаны со стоимостью, производительностью, гибкостью, размерами или функциями. Либо выбор бывает связан с волнением, которое возникает при работе с собственной системой вместо простой установки пакета программ, извлеченного из коробки. Windows XP и Mac OS X – хорошие операционные системы, но они нацелены на удовлетворение потребностей пользователей домашних компьютеров и потому страдают от некоторых ограничений и не так гибки, как Linux. Вот некоторые из причин, по которым пользователи переходят на Linux:

Linux бесплатна. Это значит, что Linux является бесплатно распространяемым клоном операционной системы Unix. Вы можете свободно взять Linux у того, у кого она уже есть, или загрузить из Интернета, или за разумную цену приобрести на CD-ROM у поставщика, который подготовил ее для распространения (возможно, с добавлением стоимости или предоставлением услуг по поддержке). Linux также распространяется свободно «free as in speech, not just free as in beer» (т. е. как свобода слова, а не бесплатное пиво): каждый может модифицировать систему и распространять модификации и усовершенствования. Подробнее мы займемся этим далее, когда будем говорить об open source (программном обеспечении с открытым исходным кодом) и свободно распространяемом программном обеспечении.

Linux пользуется популярностью. Она поддерживает большое число аппаратных платформ, включая распространенные процессоры Pentium (Pentium II, III и 4), AMD и Cyrix, и даже более старые машины 386/486. Linux работает также на компьютерах более высокого класса с архитектурой Itanium, SPARC или Alpha, а также PowerPC и Macintosh с процессорами 68k. Linux работает даже на мэйнфреймах IBM 390, а урезанные версии – на PDA, таких как Palm Pilot и Compaq iPAQ. Linux поддерживает широкий спектр аппаратных устройств, включая видеокарты, звуковые карты, приводы CD-ROM, жесткие диски, принтеры, сканеры и многие другие.

Linux представлен в Интернете весьма многочисленным сообществом пользователей. Существует множество веб-сайтов, предоставляющих информацию по системе и обеспечивающих дискуссии. Все большее число коммерческих поставщиков программного обеспечения разрабатывает приложения для Linux, в числе которых Corel WordPerfect Office 2000 Suite, пакет StarOffice, распространяемый Sun Microsystems, и ряд пакетов для работы с базами данных таких известных компаний, как Oracle, Informix и IBM.

Linux – мощная система. Linux отличается эффективностью и скоростью, прекрасно используя возможности аппаратуры. Пользователи, переходя-

щие на Linux с Windows, часто бывают удивлены скоростью работы системы и быстротой ее реакции даже при большом числе работающих процессов и открытых окон. Машина под Linux с довольно быстрым процессором и достаточным объемом оперативной памяти работает с такой же скоростью, как рабочие станции с Unix стоимостью в десятки тысяч долларов, а иногда и быстрее. Linux – многопользовательская, многозадачная операционная система, в которой может одновременно выполняться несколько задач (и даже быть зарегистрировано несколько пользователей). Linux поддерживает также многопроцессорные архитектуры и часто применяется в серверах верхнего уровня, где такого рода архитектура – обычное явление. С помощью Linux строят большие «кластеры», состоящие из сотен машин, объединенных в высокоскоростную сеть, и используемые в трудоемких научных расчетах и крупных веб-сайтах.

Linux можно управлять. В противоположность большинству коммерческих систем с графическим интерфейсом, отражающих стремление держать пользователя в неведении относительно системных процессов, Linux совершенно открыта и позволяет разобраться в том, что происходит у нее внутри. В то же время можно при желании частично ослабить контроль, воспользовавшись удобными инструментами типа *yast* из дистрибутива SuSE.

Linux устойчива. Linux разрабатывается открыто тысячами программистов, а также многочисленными фирмами и университетами, и все они добавляют новые функции, повышают производительность системы и устраняют найденные ошибки. Плоды трудов этих многочисленных разработчиков представлены в виде современных компиляторов, редакторов и утилит. В результате Linux оказывается очень надежной. У многих пользователей она работает без перезапуска месяцами (забудьте про голубой экран смерти). Для Linux создано колоссальное количество бесплатных приложений, от настольных издательских систем и пакетов офисных приложений до научных инструментов, мультимедийных приложений и игр.

Linux – полнофункциональная система. Linux поддерживает все функции современных Unix-подобных систем, включая виртуальную память, потоки, многопроцессорность и развитые сетевые функции (в том числе IPv6, DHCP, брандмауэры, трансляцию сетевых адресов и др.). Linux поддерживает широкий спектр пакетов программного обеспечения, языков программирования и аппаратных устройств. Она использует GUI (графический интерфейс пользователя) X Window System и предлагает несколько развитых интерактивных сред, включая KDE и GNOME, о чем будет рассказано в книге.

Linux в значительной степени совместима с Windows. Linux может быть установлена на одной машине с любой версией Windows (в том числе Windows 95/98/NT, Windows 2000 или Windows XP), а также вместе с другими ОС, например OS X и FreeBSD. Linux может непосредственно работать с файлами Windows – через сеть или с Windows-разделами на той же машине. Популярный пакет Samba позволяет использовать Linux в

качестве файлового сервера или сервера печати для Windows. Обратите внимание, что Linux не работает под Windows. Она совершенно независима, но в нее внесены функции, допускающие совместную работу различных систем.

Linux маленькая. Основная часть системы может работать на 8 Мбайт памяти, включая GUI и несколько приложений. Базовая система может уместиться в 20 Мбайт дискового пространства, и многие используют ее в качестве «аварийной системы», запускаемой с гибкого диска объемом 1,44 Мбайт! Linux может даже быть настроена для работы со встроенными системами с малым объемом памяти (такими как сетевые маршрутизаторы или роботы) либо для работы в портативных PDA.

Linux большая. Некоторые большие дистрибутивы занимают несколько гигабайт дискового пространства под приложения, файлы исходного кода и данных. Количество мощных утилит и приложений, перенесенных на Linux, неуклонно растет. В большинстве случаев полная система Linux требует около 300 Мбайт дискового пространства. В такую систему входят базовые файлы и полезные дополнительные средства типа библиотек программирования, компиляторов, текстовых процессоров и др. Но продвинутый пользователь может значительно расширить систему.

Linux хорошо поддерживается. Основная поддержка осуществляется через многочисленные веб-сайты, посвященные Linux, а также многочисленные телеконференции и почтовые списки рассылки. Но можно заключить договор на поддержку с независимой фирмой или купить дистрибутив с поддерживаемой версией Linux.

Linux хорошо документирована. Существует эта книга (отличное начало!), переведенная также на испанский, немецкий, французский, итальянский, португальский, чешский, польский, китайский, корейский и японский языки. Сообщество разработчиков Linux давно учредило Linux Documentation Project (LDP) – проект документирования Linux, в рамках которого поддерживается большой объем электронной документации по системе. Многочисленные книги, списки FAQ (ЧаВо) и документы «howto» этого проекта помогут справиться практически с любой задачей, которую потребуется выполнить под Linux. После преодоления некоторых препятствий при установке Linux ведет себя в основном так же, как любая другая Unix-система, поэтому многие книги по использованию и администрированию Unix окажут вам необходимую помощь. Наконец, существуют сотни популярных изданий по Linux как для начинающих, так и для опытных пользователей, переведенных на основные языки.

Она привлекательна. Скажем прямо, не требуется особой отваги, чтобы запустить последний заархивированный релиз, выпущенный крупнейшей софтверной компанией мира (не станем называть имен). В Linux есть позиция, философия и *joie de vivre*, которых вы не найдете ни в одной другой операционной системе. Linux – это гораздо больше, чем собрание битов на CD-ROM… если вы можете это оценить.

Структура этой книги

Каждая глава этой книги содержит изрядную порцию информации. Полное, подробное изложение материала могло бы занять несколько книг, но мы быстро пройдем темы, которые необходимо знать.

В главе 1 делается попытка свести вместе различные темы. Объясняются причины появления Linux и ее отличия от других Unix-систем, а также других операционных систем для персональных компьютеров.

В главе 2 описываются предварительные задачи, решение которых может потребоваться перед установкой, например создание разделов на жестком диске (в случае необходимости работы с другой операционной системой помимо Linux).

Глава 3 служит полным руководством по установке и конфигурированию Linux.

Глава 4 представляет собой введение в Unix для системного администратора и предназначена тем, кто в этом введении нуждается. Данная информация обеспечит вас достаточными средствами для решения основных задач, которые будут возникать на протяжении всей книги. В этой главе рассказывается об основных командах и некоторых понятиях, которые необходимо знать, а также даются советы администраторам.

Главы 5, 6, 7 и 8 рассказывают о системном администрировании и сопровождении. Возможно, это наиболее важные и полезные главы во всей книге; в них обсуждаются управление бюджетами пользователей, резервное копирование данных, обновление программного обеспечения, сборка нового ядра, настройка аудиосистемы и многое другое.

Глава 9 знакомит с самыми популярными и часто используемыми средствами работы с текстом и редакторами под Linux – *vi* и Emacs – и показывает, как напечатать документ и использовать различные графические программы.

В главе 10 рассказывается, как установить и сконфигурировать X Window System – мощный графический интерфейс для Linux и Unix. Мы покажем, как справиться с проблемами, которые могут возникнуть при установке программного обеспечения с дистрибутива, и как настроить его для наилучшей работы с видеосистемой.

Глава 11 показывает, как установить собственную визуальную среду под X Window System, и рассказывает о многочисленных мощных настройках, которые позволяет осуществить система, об интерактивных средах KDE и GNOME и ряде полезных программ, работающих под X.

Глава 12 представляет различные средства для взаимодействия с системами DOS и Windows, в частности сервер Samba, интегрирующий Linux с другими пользователями PC.

Главы 13 и 14 предназначены программистам. Представлены компиляторы, интерпретаторы, отладчики и многие другие инструментальные средства для программирования под Linux.

В главе 15 рассказывается об установке соединения с внешним миром, имеющего огромное значение. Показано, как сконфигурировать систему для работы в локальной сети и как осуществлять связь с интернет-провайдером, используя протокол «точка-точка» (PPP).

В главе 16 более глубоко рассматриваются вопросы конфигурации сети, показывается, как настроить электронную почту, установить почтовые программы Elm и Netscape и даже запустить собственный веб-сервер.

В главе 17 рассказывается об основных и наиболее важных правилах обеспечения защиты системы, подключенной к Интернету. Нельзя гарантировать полностью, что соблюдение этих правил защитит вашу систему от проникновения извне, но они помогут устранить очевидные и тривиальные слабые места.

Глава 18 знакомит с самым популярным комплектом средств, с помощью которого администраторы веб-сайтов публикуют контент. Этот набор носит название LAMP – по первым буквам слов Linux, Apache, MySQL и PHP.

В приложении А рассказывается о другой полезной документации по Linux и дополнительных источниках получения помощи.

В приложении В обсуждается установка Linux на не-Intel-системе, которая первой стала ее поддерживать, – 64-разрядной машине Digital Alpha.

В библиографии перечислен ряд книг, документов HOWTO и RFC, представляющих интерес для пользователей и администраторов Linux.

Соглашения, используемые в книге

Ниже приводится список типографских соглашений, принятых в этой книге:

Курсив

Используется в именах файлов и каталогов, названиях программ и именах команд, параметрах командной строки, адресах электронной почты и путях, именах сайтов, а также при определении новых терминов.

`Моноширинный шрифт`

Используется в примерах для отображения содержимого файлов, данных, возвращаемых командами, для обозначения переменных окружения, ключевых слов в коде программ, имен машин, хостов, пользователей, идентификаторов, а также команд *vi* и Emacs.

`Моноширинный полужирный шрифт`

Используется в примерах для команд или другого текста, который пользователь должен ввести буквально.

`Моноширинный курсив`

Используется для обозначения переменных параметров, ключевых слов или текста, которые пользователь должен заменить фактическим значением.

Этим значком отмечены важные примечания к расположенному рядом тексту.

Этим значком отмечены предупреждения, относящиеся к расположенному рядом тексту.

Как с нами связаться

Информация, приведенная в данной книге, была со всей тщательностью протестирована и проверена, но вы можете обнаружить, что некоторые функции изменились (или даже найти ошибку!). Пожалуйста, пишите о найденных ошибках, а также присылайте ваши предложения относительно будущих изданий по адресу:

O'Reilly & Associates, Inc.
1005 Gravenstein Highway North
Sebastopol, CA 95472
1-800-998-9938 (в США или Канаде)
1-707-829-0515 (международный/местный)
1-707-829-0104 (факс)

Можно писать по электронной почте. Для включения в список рассылки электронной почты или заказа каталога пишите по адресу:

info@oreilly.com

С техническими вопросами и комментариями по поводу книги можно обратиться по адресу:

bookquestions@oreilly.com

У этой книги есть веб-сайт, на котором представлены примеры, найденные ошибки и планы относительно новых изданий. Эта страница находится по адресу:

http://www.oreilly.com/catalog/runux4/

Дополнительные сведения об этой и других книгах можно найти на веб-сайте O'Reilly & Associates:

http://www.oreilly.com/

Благодарности

Эта книга – результат работы многих людей и, по-видимому, всех их перечислить здесь невозможно. Прежде всего, мы хотим поблагодарить Энди Орама (Andy Oram), который проделал отличную работу по редактированию, написанию и щелканью кнутом, в результате чего книга приняла должный вид. Помимо общего редактирования Энди написал главу с руководством по

Unix, а также разделы по X и Perl. Именно Энди первым предложил нам написать книгу для O'Reilly, и он проявил долготерпение святого в ожидании наших исправлений.

Те из вас, кто уже знаком с Linux, могут обратить внимание на то, что некоторые части этой книги, например введение и глава, посвященная установке, были выпущены как часть книги «Linux Installation and Getting Started» (Установка и начало работы с Linux), свободно доступной в Интернете. Издательство O'Reilly разрешило нам опубликовать в Интернете эти части (первоначально написанные для данной книги) с пользой для сообщества пользователей Linux и с целью получения нами откликов и поправок от тех, кто их прочтет. Спасибо всем принявшим участие в редактировании этих разделов.

Мы хотим также поблагодарить следующих людей за работу над операционной системой Linux (без них не о чем было бы и книгу писать): Линуса Торвальдса (Linus Torvalds), Ричарда Столлмена (Richard Stallman), Дональда Бекера (Donald Becker), Алана Кокса (Alan Cox), Реми Карда (Remy Card), Эрика Реймонда (Eric Raymond), Теда Тсо (Ted T'so), Х. Дж. Лу (H. J. Lu), Мигеля де Иказу (Miguel de Icaza), Росса Биро (Ross Biro), Дрю Экхардта (Drew Eckhardt), Эда Карпа (Ed Carp), Эрика Янгдейла (Eric Youngdale), Фреда ван Кемпена (Fred van Kempen), Стивена Твиди (Steven Tweedie), Патрика Волькердинга (Patrick Volkerding), Дирка Хондела (Dirk Hohndel), Матиаса Эттриченда (Matthias Ettrichand) и всех других хакеров – от борцов с ядром до скромных составителей документации, слишком многочисленных, чтобы всех их здесь перечислить.

Особая благодарность за вклад в проект документирования Linux, техническое рецензирование книги или просто дружескую поддержку Филу Хьюзу (Phil Hughes), Мелинде Мак-Брайд (Melinda McBride), Биллу Хану (Bill Hahn), Дэну Ирвингу (Dan Irving), Майклу Джонстону (Michael Johnston), Джоэлу Гольдбергеру (Joel Goldberger), Майклу К. Джонсону (Michael K. Johnson), Адаму Рихтеру (Adam Richter), Роману Яновски (Roman Yanovsky), Джону Мэджиду (Jon Magid), Эрику Троэну (Erik Troan), Ларсу Вирцениусу (Lars Wirzenius), Олафу Кирху (Olaf Kirch), Грегу Хенкинсу (Greg Hankins), Алану Сондхейму (Alan Sondheim), Джону Дэвиду (Jon David), Анне Кларк (Anna Clark), Адаму Гудмену (Adam Goodman), Ли Гомесу (Lee Gomes), Робу Уокеру (Rob Walker), Робу Мальде (Rob Malda), Джефу Бэйтсу (Jeff Bates) и Волькеру Лендеке (Volker Lendecke). Мы благодарим Шона Уоллеса (Shawn Wallace) и Умберто Кренку (Umberto Crenca) за прекрасный снимок рабочего окна Gimp в главе 9.

За работу над третьим изданием мы благодарны Филу Хьюзу (Phil Hughes), Роберту Дж. Часселу (Robert J. Chassell), Тони Капеллини (Tony Cappellini), Крейгу Смоллу (Craig Small), Нату Макаревичу (Nat Makarevitch), Крису Дэвису (Chris Davis), Чаку Торопеку (Chuck Toporek), Фредерику Хонгфенгу (Frederic HongFeng) и Дэвиду Пранате (David Pranata), добавившим многочисленные комментарии и исправления. Особое впечатление произвел труд целой команды разработчиков и пользователей Debian, которую нам предоставили Оссама Отмен (Ossama Othman) и Джулиан Т. Дж. Мидгли (Julian

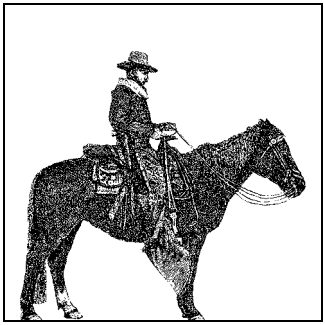
T. J. Midgley). Джулиан организовал хранение комментариев в системе контроля версий (CVS). Книгу вместе изучали он, Крис Лоуренс (Chris Lawrence), Роберт Дж. Чассел (Robert J. Chassell), Кирк Хиллиард (Kirk Hilliard) и Стефен Зандер (Stephen Zander).

За рецензирование четвертого издания мы благодарны Дэвиду Коллье-Брауну (David Collier-Brown), Оливеру Флимму (Oliver Flimm), Филу Хьюзу (Phil Hughes), Крису Лоуренсу (Chris Lawrence), Ричу Пейну (Rich Payne), Крейгу Смоллу (Craig Small), Джефу Трантеру (Jeff Tranter) и Арону Веберу (Aaron Weber). Мэтт Уэлш хочет выразить особую благодарность своей невесте Эйми Бауэр (Amy Bauer) за любовь и поддержку, а также оплату половины стоимости домашней линии DSL.

Кале хотел бы поблагодарить Валерика Ватафу (Valerica Vatafu) из Бузэу, Румыния, за большую помощь в работе над главой, посвященной LAMP. Он также хочет поблагодарить своих коллег из компании Klarälvdalens Datakonsult AB – Мишеля Бойера де ля Жироде (Michel Boyer de la Giroday), Таню Далхаймер (Tanja Dalheimer), Стеффена Хансена (Steffen Hansen), Джеспера Педерсена (Jesper Pedersen), Лутца Роговски (Lutz Rogowski), Бо Торсена (Bo Thorsen) и Карла-Хайнца Циммера (Karl-Heinz Zimmer) – за конструктивные комментарии к черновым наброскам книги и общее «стимулирование Linux-мышления».

Большой вклад в четвертое издание внесли эсперты в различных специальных областях. В частности, мы хотим поблагодарить Джефа Трантера (Jeff Tranter) за сведения о настройке аудиосистемы в главе 9 и обновление информации по сетевым ресурсам Linux в приложении A и библиографии; Арона Вебера (Aaron Weber) из Ximian за материал по среде GNOME в главе 11; Кайла Дента (Kyle Dent) за материал по почтовому агенту Postfix в главе 16; Джея Тс (Jay Ts) за переработку раздела по Samba в главе 12; Криса Лоуренса (Chris Lawrence) за материал по обновлению пакетов в Debian главы 7; Баррета Дж. Лайона (Barrett G. Lyon) и Ричарда Пейна (Richard Payne) за материал по установке Linux на системах Compaq/Digital Alpha в приложении B (который Ричард Пейн обновил для четвертого издания).

Если у вас есть вопросы, комментарии или исправления к этой книге, не стесняйтесь войти в контакт с авторами. Адреса электронной почты в Интернете: Мэтт Уэлш (Matt Welsh) – *mdw@cs.berkeley.edu*, Лар Кауфман (Lar Kaufman) – *lark@conserve.org*, Калле Далхаймер (Kalle Dalheimer) – *kalle@dalheimer.de*.



1

Введение в Linux

Это книга о Linux, бесплатной операционной системе с открытым исходным кодом (open source), полностью поддерживающей многозадачность, систему X Window, сетевой протокол TCP/IP и многое другое. Наберитесь терпения и читайте дальше: на следующих страницах мы опишем систему очень подробно.

В компьютерных кругах Linux вызвала бóльший резонанс, чем любая другая разработка за последние несколько лет. Ее удивительно быстрое распространение и способность вызывать приверженность к себе напоминают о том характерном для времени первых успехов компьютерных технологий волнении, которое вызывала возможность самостоятельного программирования. По иронии судьбы успех Linux связан со второй молодостью одной из старейших и все еще широко используемых операционных систем, Unix. Linux является одновременно и новой и старой технологией.

В узко техническом понимании Linux представляет собой лишь ядро операционной системы, предлагающее базовые сервисы планирования процессов, виртуальной памяти, управления файлами и ввода/вывода. Иными словами, сама по себе Linux является низкоуровневой частью операционной системы.

Однако термин «Linux» используется, как правило, для обозначения системы в целом, т. е. как ядра, так и многочисленных работающих с ним приложений: полной среды разработки и исполнения, включающей в себя компиляторы, редакторы, графические интерфейсы, текстовые процессоры, игры и т. д.

Эта книга будет вашим проводником в многогранном и изменяющемся мире Linux. Linux превратилась в операционную систему для бизнеса, учебных заведений и повышения персональной производительности. Данная книга поможет вам использовать ее с максимальной эффективностью.

Linux может превратить любой персональный компьютер в сервер и рабочую станцию высокого уровня. В деловом мире Linux устанавливается на целые вычислительные сети, обеспечивая обработку данных в финансовом и медицинском секторе, распределенных вычислительных средах, системах телекоммуникаций и т. д. Университеты всего мира используют Linux в курсах

программирования и проектирования операционных систем. И, разумеется, энтузиасты вычислительной техники повсюду используют Linux в домашних компьютерах для программирования, создания документов и всевозможных компьютерных изысканий. Linux применяют в мощных настольных машинах, карманных PDA, портативных лэптопах и даже устанавливают в какую-нибудь рухлядь, которая занимается лишь спулингом печати.

Помимо рабочих станций и личного использования Linux применяется также в больших серверах. Все большее число людей обнаруживают, что Linux обладает мощью, устойчивостью и гибкостью, достаточными для работы с самыми большими дисковыми массивами, многопроцессорными системами и приложениями, охватывающими диапазон от веб-серверов до корпоративных баз данных. Под управлением Linux работают многие критически важные бизнес-приложения, сайты Интернета, поисковые механизмы и сети доставки контента. Ученые соединяют группы машин, работающих под Linux, в огромные кластеры для решения физических и инженерных задач, требующих огромных вычислительных ресурсов. Последняя версия пакета Samba даже позволяет Linux выполнять функции файлового сервера и сервера печати для Windows – и с большей производительностью, чем Windows!

Отличительная особенность Linux состоит в том, что она является *свободно* распространяемой реализацией Unix. Ее разработка была начата – и продолжается до сих пор – группой добровольцев, общающихся в основном через Интернет, которые обмениваются исходным кодом, сообщают о найденных ошибках и устраняют проблемы в открытой среде. К участию в проекте разработки Linux приглашаются все желающие: все, что требуется, это интерес в работе над бесплатным клоном Unix и определенные познания в программировании.

Для понимания этой книги достаточно уметь уверенно обращаться с персональным компьютером, работающим под любой операционной системой, такой как Windows 98 или какая-либо разновидность Unix. Предполагается также, что читатель готов немного поэкспериментировать, чтобы все заработало правильно, – в конце концов, в этом половина удовольствия, получаемого от освоения Linux. Linux превратилась в систему, которую удивительно просто устанавливать и конфигурировать, но ее мощь сопряжена с тем, что некоторые детали оказываются более сложными, чем при работе с Windows. Мы надеемся, что благодаря этой книге вы обнаружите, что установка собственной системы Linux и работа в ней достаточно просты и доставляют большое удовольствие.

Об этой книге

Эта книга является обзором и руководством начального уровня по системе Linux. Мы постарались представить достаточный объем общей и интересной информации по ряду тем, чтобы удовлетворить как новичков, так и искушенных пользователей Unix. Книга должна обеспечить почти всякого пользователя сведениями, достаточными для установки и использования Linux,

а также получения от нее максимальной отдачи. Вместо того чтобы говорить о многочисленных непостоянных технических деталях, имеющих тенденцию к изменению ввиду быстрого развития системы, мы даем подготовку, которой достаточно для самостоятельных исследований.

Эта книга нацелена на тех, кто действительно стремится к использованию всей мощи, предоставляемой Linux. Вместо того чтобы рассказывать о всяких хитрых трюках, мы подготовим вас к тому, что вы действительно будете понимать, как работают различные части системы, и поэтому сможете самостоятельно настраивать, конфигурировать систему и устранять возникающие проблемы. Установить и эксплуатировать Linux не сложно, однако, как и во всякой реализации Unix, для того чтобы все работало должным образом, часто требуется привлечение своего рода черной магии.

В этой книге освещаются следующие темы:

- Что такое Linux? Архитектура и идеология этой уникальной операционной системы, и чем она может быть для вас полезна.
- Все подробности, которые нужно знать для запуска Linux, включая советы по аппаратной конфигурации, рекомендуемой для полноценной системы.
- Где взять и как установить Linux. О дистрибутивах Red Hat, SuSE и Debian мы расскажем более подробно, чем о других, но представленных сведений должно быть достаточно для установки любой версии системы.
- Введение в Unix-системы для новичков, включая обзор наиболее важных команд и понятий.
- Как заботиться и ухаживать за Linux-системой, включая системное администрирование и поддержку системы в рабочем состоянии, обновление системы и устранение неполадок.
- Получение наибольшей отдачи от системы Linux с использованием таких мощных инструментов, как TEX, Emacs, KDE, GNOME и многих других.
- Linux как среда разработки. Профессиональные средства программирования и разработки приложений для Linux. Мы познакомим читателей с компиляцией и отладкой программ на C и C++, Java, Perl и сценариями оболочки.
- Использование Linux для телекоммуникаций и сетевого взаимодействия, включая основы конфигурирования TCP/IP, PPP для модемных соединений с Интернетом, конфигурирование ISDN, организацию электронной почты, доступа к телеконференциям и к сети. Мы даже покажем, как сконфигурировать Linux-систему для работы в качестве веб-сервера.

Нам бы хотелось рассказать вам о тысячах вещей, которые можно делать с Linux, но, к сожалению, для этого понадобилась бы книга размером с полный *Оксфордский словарь английского языка*, с которой не справился бы никто, включая несчастных авторов. Вместо этого мы попытались рассказать о наиболее ярких и интересных сторонах системы и сообщить, как можно узнать о ней больше.

Хотя значительная часть изложенного в книге материала не является сугубо технической, полезно иметь опыт работы с какой-нибудь Unix-системой. Для тех, у кого нет опыта работы с Unix, мы включили в главу 4 краткое учебное руководство. Глава 5 полностью посвящена администрированию системы и может оказаться полезной даже опытным пользователям Unix при работе с Linux.

Если вы новичок в Unix, вам понадобится более детальное руководство по ее основам. Мы не будем долго задерживаться на основах, предпочитая скорее перейти к более интересным частям системы. Во всяком случае, хотя этой книги должно быть достаточно, чтобы приступить к работе, большинству читателей может потребоваться дополнительная информации о Unix и ее многочисленных инструментах. Список источников информации можно получить в приложении А.

Краткая история Linux

Unix является одной из популярнейших в мире операционных систем благодаря широкой базе поддержки и распространения. Она была разработана в середине 70-х годов как многозадачная система для мини-компьютеров и мэйнфреймов. С тех пор она стала одной из наиболее распространенных операционных систем, несмотря на свой не всегда удобный интерфейс и отсутствие единой стандартизации. Единой реализации Unix не существует. Изначально она была разработана в Bell Labs, впоследствии разделившись на несколько версий, в число которых входит популярная разработка Калифорнийского университета в Беркли под названием BSD. С течением времени ряд фирм разработал собственные реализации Unix, начав с самого начала или развив уже имеющуюся версию. Linux создавалась с чистого листа, хотя в ранних версиях присутствовала часть кода из BSD.

Рынок систем Unix сократился в начале 90-х годов в связи с появлением системы Windows NT, но она вернулась окрепшей, став главной опорой больших компьютеров.

В сообществе разработчиков операционных систем существует определенный культ Unix. Многие хакеры считают, что Unix – это «правильная штука», Единственная Настоящая Операционная Система. В этом и причина разработки Linux все более расширяющейся группой энтузиастов Unix, которые хотят поковыряться с собственной системой. Кроме того, Linux не является продуктом, привязывающим вас к определенному поставщику или разработчику ПО. Поскольку Linux распространяется свободно вместе со всем исходным кодом (о чем подробнее будет сказано ниже), каждый желающий может модифицировать эту систему, чтобы приспособить ее к своим нуждам. Не нужно ждать, пока какая-нибудь крупная компания реализует новые функции и выпустит пакеты обновления; сообщество пользователей Linux располагает возможностью самостоятельно совершенствовать, адаптировать и исправлять систему. Такая возможность и обусловила мощь, которой достигла Linux.

Linux – свободно распространяемая версия Unix, первоначально разработанная Линусом Торвальдсом (Linus Torvalds), начавшим работу над ней будучи студентом Хельсинкского университета в Финляндии. Сейчас Линус работает в фирме Transmeta Corporation, находящейся в городе Санта-Клара в Калифорнии, и продолжает поддерживать *ядро* Linux, т. е. основной, низкоуровневый компонент операционной системы.

Линус поместил первоначальную версию Linux в Интернет для бесплатного использования, нечаянно породив при этом один из крупнейших феноменов всех времен в разработке программного обеспечения. Сегодня Linux создается и поддерживается тысячами разработчиков, свободно сотрудничающих между собой при помощи Интернета. Некоторые компании стали обеспечивать поддержку Linux, создавать легкие для установки дистрибутивы системы и продавать рабочие станции с предустановленной на них Linux. В марте 1999 года в Сан-Хосе в Калифорнии состоялась первая всемирная выставка Linux World Expo, которую, как сообщается, посетили 12 000 человек. Согласно большинству оценок, количество пользователей Linux в мире измеряется миллионами.

Первоначально Linux создавалась Линусом Торвальдсом как курсовая работа, вдохновением для которой послужила созданная Эндрю Таненбаумом (Andrew Tanenbaum) операционная система Minix (одна из ранних версий Unix, предназначенная для обучения проектированию операционных систем). Линус хотел создать простую Unix-систему, которая могла бы работать на 386-м PC. Первое обсуждение Linux состоялось в рамках телеконференции Usenet *comp.os.minix*. Это обсуждение касалось в основном создания маленькой Unix-системы для учебных и научных заведений и тех пользователей, которым хотелось бóльших возможностей, чем предоставляла Minix.

В самом начале разработка Linux была связана в основном с возможностями переключения задач в защищенном режиме 80386 и велась на ассемблере. Линус пишет об этом:

> «После этого началось спокойное плавание: по-прежнему трудное кодирование, но у меня были различные вспомогательные средства, и отладка стала легче. На этом этапе я стал использовать C, и это существенно ускорило дело. В это же время я стал серьезно носиться с маниакальной идеей, как сделать «Minix, лучший, чем Minix». Я надеялся, что в один прекрасный день перекомпилирую *gcc* под Linux.
>
> Два месяца ушло на написание основных программ, но затем прошло еще немного времени, и у меня появился драйвер диска (с большим количеством ошибок, но все-таки работавший на моей машине) и простая файловая система. Тогда я выпустил версию 0.01 (примерно в конце августа 1991 года). Она была не слишком хороша, в ней не было драйвера гибких дисков и она мало что умела делать. Не думаю, что кто-нибудь когда-либо ее компилировал. Но я уже был одержим и не мог остановиться, не выставив Minix за дверь».

Версия Linux 0.01 никогда не анонсировалась. Из исходных текстов 0.01 нельзя было даже получить исполняемый код: в них были лишь зачатки ядра и предполагалось наличие доступа к машине, работающей под Minix, чтобы откомпилировать их и повозиться с ними.

Пятого октября 1991 года Линус объявил о первой «официальной» версии Linux, версии 0.02. К этому моменту в Linux уже работали *bash* (GNU Bourne Again Shell) и *gcc* (GNU C compiler), но больше почти ничего. По-прежнему эта система предназначалась для хакеров. Внимание было сосредоточено на создании ядра; никакие вопросы поддержки пользователей, документации, распространения и т. п. даже не обсуждались. Сегодня ситуация в корне изменилась: мир Linux в основном озабочен графическими средами пользователей, созданием простых в установке дистрибутивов и приложениями высокого уровня, такими как графические утилиты и офисные пакеты.

Линус писал в *comp.os.minix*:

> «Не тоскуете ли вы по добрым временам Minix-1.1, когда мужчины были мужественны и писали собственные драйверы устройств? У вас нет интересного проекта и вам смертельно хочется вонзить свои зубы в какую-нибудь ОС, которую можно переделать по своему вкусу? Вас удручает, что под Minix все работает? Вы скучаете без необходимости работы на машине по ночам, чтобы заставить хитрую программу работать? Тогда это сообщение может быть как раз для вас.
>
> Как я уже писал месяц назад, я работаю над некоммерческой Minix-подобной ОС для AT-386. Она доведена, наконец, до такого состояния, когда ею можно даже пользоваться (хотя и не во всех случаях), и я хочу выложить исходные тексты для широкого распространения. Это всего лишь версия 0.02, но я успешно запускаю в ней *bash*, *gcc*, GNU *make*, GNU *sed*, *compress* и т. д.»

После версии 0.03 Линус сразу перескочил в нумерации к версии 0.10. В это время над проектом стало работать много людей. После нескольких последующих исправлений Линус присвоил очередной версии номер 0.95, чтобы тем самым показать, что до «официальной» версии уже совсем недалеко. (Обычно программа получает номер версии 1.0, когда теоретически завершена и отлажена.) Это было в марте 1992 года. Но прошло еще два года, прежде чем, наконец, в марте 1994 года появилась версия 1.0. Во время написания книги (сентябрь 2002 года) текущая версия стабильного ядра имеет номер 2.4.19, и одновременно разрабатываются версии 2.5 (ниже мы подробнее обсудим принципы нумерации версий Linux).

Linux не смогла бы появиться без инструментария GNU, созданного Free Software Foundation. FSF – это группа, созданная Ричардом Столманом (Richard Stallman) в 1984 году, чтобы способствовать созданию программного обеспечения, которое каждый может разрабатывать, передавать другим и модифицировать. «Free» (свободный) означает здесь свободу, а не только отсутствие платы. В основе философии Free Software Foundation лежит глубокая нравственная уверенность в том, что всякое программное обеспечение должно свободно распространяться (опять-таки, «свободно» в широком смысле). Эту философию разделяют многие в сообществе пользователей Linux. Этот идеал отражен в GNU General Public License (Универсальной общественной лицензии GNU), лицензии на авторские права, под которой выпускается Linux. Подробнее об этом мы расскажем далее в этой главе.

Проект GNU – основной результат трудов FSF – привел к созданию многих ценных инструментов и приложений, от которых зависит Linux, в том числе текстовый редактор Emacs, пакет компиляции *gcc* и ряд других. Инструментарий GNU с самого начала тесно переплетен с разработкой Linux. Поскольку инструментарий GNU внес значительный вклад в Linux, Free Software Foundation даже требует, чтобы дистрибутивы Linux с сопровождающими утилитами назывались GNU/Linux.

Разработанная в университете Беркли операционная система Berkeley Unix (BSD) также сыграла важную роль в отношении Linux – не столько в ее создании, сколько в предоставлении инструментов, которые делают ее популярной. Эта система «Berkeley Software Distribution» была разработана в калифорнийском университете Беркли в конце 1970-х годов группой разработчиков, положивших в основу системы исходные коды Unix разработки AT&T. Группа разработчиков BSD внесла ряд усовершенствований в исходную архитектуру ядра Unix, и через некоторое время BSD начала самостоятельное существование. В настоящее время есть много версий BSD для различных аппаратных платформ, и сообщества пользователей BSD и Linux соперничают между собой в популярности. Даже операционная система Mac OS X основана на одной из версий BSD! Некоторые сетевые утилиты и демоны Linux ведут свое происхождение от BSD.

Сегодня Linux является полнофункциональной законченной реализацией Unix, поддерживающей большое количество приложений, языков программирования, инструментов и аппаратных устройств. Linux поддерживает графический интерфейс X Window, сетевые протоколы TCP/IP, многопроцессорную архитектуру, сложную аппаратуру и программное обеспечение для научных расчетов и параллельных вычислений и многое другое. Почти все основные свободно распространяемые программные пакеты перенесены на Linux, для нее есть масса коммерческого ПО. Фактически многие разработчики сначала пишут свои приложения для Linux, а затем переносят их на другие Unix-системы. Поддерживается большее разнообразие аппаратных средств, чем в первоначальных версиях ядра. Было выполнено много тестов производительности Linux-систем, показавших, что они работают быстрее дорогих рабочих станций Sun и Compaq. В большом числе тестов Linux-системы работают лучше или, по крайней мере, не хуже, чем Windows NT/2000/XP. Кто мог предположить, что этот «маленький» клон Unix вырастет настолько, что бросит вызов целому миру персональных компьютеров и серверов?

Кто использует Linux?

Разработчики приложений, системные администраторы, интернет-провайдеры, специалисты по ядру операционных систем, создатели мультимедиа – вот лишь некоторые группы людей, для которых Linux обладает особой привлекательностью.

Unix-программисты все чаще используют Linux из-за соображений стоимости – можно создать завершенную программную среду, потратив несколько долларов, и пользоваться ею на дешевом персональном компьютере благодаря широкой переносимости программ для Linux. Это современная операционная система, совместимая со стандартом POSIX и в значительной мере похожая на System V, поэтому программы, работающие под Linux, должны работать с остальными современными Unix-системами.

Сетевые возможности являются сильной стороной Linux. Ее с удовольствием приняли те, кто поддерживает большие сети, оценив простоту администрирования, высокую производительность и низкую стоимость. Многие интернет-сайты используют Linux для поддержки больших веб-серверов приложений е-коммерции, поисковых машин и т. д. Linux поддерживает распространенные сетевые стандарты, такие как NFS (Network File System, сетевая файловая система) и NIS (Network Information Service, сетевой информационный сервис), что позволяет легко подсоединить машину с Linux к корпоративной или университетской сети с Unix-машинами. Она позволяет легко организовать совместный доступ к файлам, поддерживать регистрацию на удаленных машинах и запускать на них приложения. Linux поддерживает также программный пакет Samba, позволяющий Linux-машине действовать в качестве файлового сервера и сервера печати для Windows. Многие убедились, что сочетание Linux и Samba оказывается быстрее (и дешевле), чем использование Windows 2000.

Очень часто Linux используется для поддержки крупных промышленных приложений, включая веб-серверы, базы данных, приложения B2B (business-to-business) и сайты для торговли через Интернет. Большое количество предприятий убеждается в том, что Linux может служить недорогой, эффективной и устойчивой системой для поддержки самых ответственных приложений. То обстоятельство, что Linux легко перестраивается для решения конкретных задач, вплоть до модификации ядра, делает эту систему очень привлекательной для компаний, которым требуется контроль за работой внутренних механизмов системы. Linux поддерживает RAID – механизм, позволяющий работать с группой жестких дисков как с одним логическим устройством и значительно повысить надежность хранения данных. Сочетание Linux, веб-сервера Apache, базы данных MySQL и языка сценариев PHP настолько распространено, что у него есть собственный акроним – LAMP. Об этом комплекте мы подробно расскажем в главе 18.

Специалисты по ядру первыми признали Linux – фактически именно они помогли Линусу Торвальдсу создать Linux и до сих пор образуют крепкое сообщество. Списки почтовой рассылки по ядру Linux весьма активны, и если вы не хотите отстать от достижений в проектировании операционных систем, необходимо подписаться на них. Если вам нужно отрегулировать алгоритм замены страниц, поиграть с сетевыми протоколами или оптимизировать работу с кэшем, то Linux предоставляет для этого очень хорошие возможности. Linux также оказывает помощь при изучении проектирования

операционных систем, и многие университеты используют системы Linux в продвинутых курсах по операционным системам.

Наконец, Linux становится ареной использования мультимедиа, поскольку она совместима с огромным числом аппаратных средств, включая большинство современных звуковых и видеокарт. Несколько программных сред, в том числе MESA 3D (бесплатная реализация OpenGL), было перенесено на Linux. GIMP (бесплатный пакет, сходный с Adobe Photoshop), первоначально разработанный под Linux, для многих становится предпочтительным средством работы с графикой. Многие кинематографические компании систематически пользуются Linux для получения сложных спецэффектов. Например, при съемках фильмов «Титаник» и «Матрица» для создания сложных спецэффектов использовались «рендер-фермы» Linux-машин.

Linux используется и в практических приложениях. Linux-системы добрались до высоких широт Тихого океана, осуществляя телекоммуникации и анализ данных на исследовательском океанографическом судне. Linux-системы используются на исследовательских станциях в Антарктике, а большие кластеры Linux-систем применяются во многих исследовательских учреждениях для моделирования сложных физических явлений типа образования звезд или землетрясений. В ряде больниц Linux используется для ведения историй болезней. Один из рецензентов этой книги использует Linux в морской пехоте США. Linux оказывается такой же надежной и удобной системой, как и другие реализации Unix.

Таким образом, Linux расширяет свое присутствие во многих направлениях. Даже неискушенные пользователи могут успешно с ней работать при наличии поддержки, обычно оказываемой университетами и корпорациями своим пользователям компьютеров. Конфигурирование и обслуживание требуют некоторой специальной подготовки, однако использование Linux окупает затраты, предоставляет мощь и новые возможности для тех, кто хочет получить дополнительные средства управления своей программной средой.

Характеристики системы

Linux поддерживает большинство функций, присутствующих в других реализациях Unix, а также много функций, которых больше нигде нет. В этом разделе дан беглый обзор характеристик ядра Linux.

Замечание о нумерации версий Linux

Тех, кто начинает изучать Linux, может запутать способ присвоения номеров версий различным программам. Приступая к работе с Linux, вы, скорее всего, будете держать в руках какой-нибудь дистрибутив на компакт-диске, например «Red Hat Version 7.1» или «SuSE Linux Version 6.0». Важно понимать, что эти номера относятся только к конкретному дистрибутиву, представляющему собой упаковку с некоторой версией Linux и огромным количеством прикладных программ, обычно распространяемому на компакт-диске. По-

этому версия, присвоенная Red Hat, SuSE или Debian, может не иметь никакого отношения к версиям программ, находящихся на этом дистрибутиве.

Ядро Linux, как и любое приложение, компонент, библиотека или программный пакет, входящий в дистрибутив Linux, обычно имеет *собственный* номер версии. Например, могут быть установлены *gcc* версии 2.96 и графический интерфейс пользователя XFree86 версии 4.0.3. Как можно догадаться, чем больше номер версии, тем новее программное обеспечение. При установке дистрибутива, такого как Red Hat или SuSE, все проще, поскольку в дистрибутив обычно включаются самые последние версии каждого пакета, а продавец следит за тем, чтобы программы, входящие в конкретный дистрибутив, могли работать совместно.

Ядро Linux имеет особую схему нумерации версий, о которой следует знать. Как уже говорилось, ядро – это центральная часть самой операционной системы, ответственная за управление всеми аппаратными ресурсами машины – дисками, сетевыми интерфейсами, памятью и т. д. В отличие от Windows, ядро Linux не включает в себя библиотеки уровня приложений или графические интерфейсы. В известном смысле, пользователь взаимодействует с ядром не непосредственно, а лишь через такие интерфейсы, как оболочка (shell) или графический интерфейс пользователя (GUI), о чем будет сказано ниже.

Однако многие продолжают считать номер версии ядра Linux номером версии системы «в целом», что может ввести в заблуждение. Если кто-то заявляет, что у него работает ядро версии 2.5.12, то это мало что значит, если остальная часть его системы на несколько лет старше.

Система версий ядра Linux действует так. В любой момент времени имеются (в смысле возможности загрузки из Интернета) *две* «последние» версии ядра – *стабильная* и *разрабатываемая*. Стабильная версия предназначена для большинства пользователей Linux, не намеренных погружаться вглубь новейших экспериментальных функций и которым нужна устойчиво работающая и не меняющаяся каждодневно система. Напротив, разрабатываемая версия очень быстро меняется по мере того, как разработчики добавляют и тестируют новые функции через Интернет. Изменения стабильной версии обычно состоят в исправлениях ошибок и пробелов в защите, в то время как изменения в разрабатываемой версии могут касаться чего угодно – от основных подсистем ядра до мелких усовершенствований в драйвере устройства. Разработчики Linux не гарантируют, что разрабатываемая версия ядра будет работать у каждого, но стабильную версию они поддерживают с целью обеспечения устойчивой работы во всех конфигурациях.

У стабильной версии ядра дополнительный номер версии четный (например, 2.4), а у разрабатываемой – нечетный (например, 2.5). Дополнительный номер у текущей разрабатываемой версии всегда на единицу больше, чем у текущей стабильной. Поэтому, если текущая стабильная версия имеет номер 2.6, разрабатываемая версия имеет номер 2.7 (если, конечно, Линус не решит перескочить сразу к версии 3.0, и тогда, естественно, разрабатываемая версия будет иметь номер 3.1).

У номеров каждой из этих версий есть третий уровень, «уровень патчей» (заплаток), например 2.4.19 или 2.5.12. Уровень патчей указывает на конкретный номер ревизии данной версии ядра, и больший номер соответствует более поздней ревизии. Когда писались эти строки в ноябре 2002 года, последняя стабильная версия ядра имела номер 2.4.19, а текущая разрабатываемая – 2.5.45.

Характеристики системы

Linux – это полностью многозадачная и многопользовательская операционная система (как и все остальные версии Unix). Это означает, что одновременно на одной машине могут работать несколько пользователей и выполняться несколько программ. Linux поддерживает также многопроцессорные системы (такие как материнские платы с двумя процессорами Pentium), содержащие до 32 процессоров, что очень ценно для высокопроизводительных серверов и научных приложений.

Linux хорошо совместима с рядом стандартов Unix (насколько можно говорить о стандартизации Unix) на уровне исходных текстов, включая функции IEEE POSIX.1, System V и BSD. Она создавалась с учетом требования переносимости исходного кода, поэтому вы найдете в Linux функции, присутствующие во многих Unix-системах. Большинство бесплатных программ для Unix, распространяемых через Интернет или иным образом, может быть скомпилировано для Linux без дополнительных изменений.

При наличии опыта работы в Unix вас могут заинтересовать другие внутренние возможности Linux, включая управление заданиями в стандарте POSIX (используемое в таких оболочках, как C shell, *csh* и *bash*), псевдотерминалы (*pty*) и поддержка национальных или специальных клавиатур с помощью динамически загружаемых драйверов. Linux поддерживает также *виртуальные консоли* (*virtual consoles*), которые позволяют в текстовом режиме системной консоли переключаться между разными сеансами регистрации. Тем, кто пользовался программой *screen*, виртуальная консоль Linux покажется знакомой (хотя почти все предпочитают графическую среду).

Linux благополучно соседствует с другими операционными системами, такими как Windows 95/98, Windows NT/2000/XP, Mac OS или другими версиями Unix, если они установлены на той же машине. Начальные загрузчики Linux (LILO) и Grand Unified Bootloader (GRUB) позволяют выбрать операционную систему, запускаемую при начальной загрузке, при этом Linux совместима и с другими загрузчиками (например, с загрузчиком Windows 2000).

Linux может работать с широким рядом архитектур центрального процессора, в том числе Intel x86 (вся линейка Pentium, а также 386/486), Itanium, SPARC/UltraSPARC, ARM, PA-RISC, Alpha, PowerPC, MIPS, m68k и мэйнфрэймы IBM 370/390. Linux перенесена на ряд встроенных процессоров, а для различных PDA созданы сокращенные версии, в том числе для PalmPilot и Compaq iPaq. С другой стороны, Linux рассматривается в качестве ОС для машин верхнего ряда. Hewlett-Packard объявила о намерении выпустить суперкомпьютер с установкой на нем в качестве операционной системы Linux.

Большое количество масштабируемых кластеров – суперкомпьютеров, образованных из множества PC, – также работает под Linux.

Linux поддерживает различные типы файловых систем для хранения данных. Некоторые файловые системы, такие как «вторая расширенная файловая система» *ext2fs*, были специально созданы для Linux. Поддерживаются и другие типы файловых систем Unix, такие как Minix-1 и Xenix. Реализованы также файловые системы NTFS (Windows 2000 и NT), VFAT (Windows 95/98) и FAT (MS-DOS), что позволяет непосредственно обращаться к файлам Windows. Включена также поддержка файловых систем Macintosh, OS/2 и Amiga. Также поддерживается файловая система ISO 9660 CD-ROM, позволяющая читать компакт-диски всех стандартных форматов. Подробнее о файловых системах говорится в главах 3 и 5.

Поддержка сетевого взаимодействия является одной из наиболее сильных сторон Linux, как в отношении поддерживаемых функций, так и в отношении производительности. Linux обеспечивает полную реализацию протоколов TCP/IP. В нее входят драйверы для многих распространенных сетевых карт Ethernet, протоколы PPP и SLIP (обеспечивающие доступ к сети TCP/IP через последовательные порты и модем), PLIP (Parallel Line Internet Protocol – интернет-протокол через параллельный порт) и сетевая файловая система NFS. Поддерживается также современный набор протоколов IPv6 и многие другие протоколы, включая DHCP, Appletalk, IRDA, DECnet и даже AX.25 для пакетных радиосетей. Поддерживается полный набор клиентов и служб TCP/IP, таких как FTP, Telnet, NNTP и Simple Mail Transfer Protocol (SMTP). Ядро Linux содержит полную поддержку сетевых брандмауэров (которые фильтруют сетевые пакеты, препятствуя, например, несанкционированному доступу к интрасетям). Широко распространено мнение, что Linux превосходит другие операционные системы в производительности работы сети. Подробнее о сетевом взаимодействии мы поговорим в главе 15.

Ядро

Ядро является самой сущностью операционной системы. Это программный код, управляющий интерфейсами между пользовательскими программами и аппаратными устройствами, распределением времени между процессами, обеспечивающим многозадачность, и многими другими сторонами системы. Ядро – это не отдельный процесс, работающий в системе. Его можно представить себе как набор постоянно находящихся в памяти программ, к которым есть доступ у всех процессов. К программам ядра можно обращаться разными способами. Один из прямых методов использования ядра – это выполнение процессом *системного вызова*, являющегося функцией, заставляющей ядро выполнить некоторый программный код в интересах процесса. Например, системный вызов *read* осуществляет чтение данных из файла. С точки зрения программиста *read* выглядит как обычная функция C, но в действительности программный код *read* находится внутри ядра.

Код ядра выполняется и в других ситуациях. Например, обработчики прерываний, генерируемых аппаратными устройствами, находятся в ядре. Ког-

да процесс запускает некоторую операцию и должен дождаться ее результатов, в игру вступает ядро, которое переводит процесс в режим ожидания и передает его время другому процессу. Аналогично, ядро переключается между процессами через короткие промежутки времени, используя для переключения прерывание по таймеру (и другие средства). Это лежит в основе осуществления многозадачной работы.

Тип архитектуры ядра Linux известен как *монолитный*. Это означает, что все драйверы устройств являются частью ядра. В некоторых операционных системах реализована архитектура *микроядра*, когда драйверы устройств и другие компоненты (такие как файловые системы и код, управляющий памятью) *не* входят в ядро, а выступают как независимые службы или обычные пользовательские приложения. Обе архитектуры имеют свои достоинства и недостатки. Монолитная архитектура более распространена в реализациях Unix и применяется в таких классических системах, как System V и BSD. В Linux все же есть поддержка загружаемых драйверов устройств (т. е. таких, которые могут загружаться и выгружаться из памяти по команде пользователя). Эта тема рассматривается в разделе «Загружаемые драйверы устройств» главы 7.

Ядро Linux для платформ Intel создано с учетом использования специальных возможностей защищенного режима процессоров Intel x86 (начиная с 80386 и далее вплоть до Pentium 4). В частности, Linux использует парадигму управления памятью в защищенном режиме с помощью дескрипторов и другие передовые возможности этих процессоров. Каждый, кто знаком с программированием в защищенном режиме процессора 80386, знает, что этот чип проектировался для многозадачных систем типа Unix (x86 возник под влиянием Multics). Linux использует эти функции.

Как и большинство современных операционных систем, Linux является многопроцессорной ОС: она поддерживает системы с несколькими процессорами на материнской плате. Это позволяет различным программам работать на разных процессорах одновременно (или «параллельно»). Linux также поддерживает *потоки*, распространенную технологию программирования, позволяющую одной программе создать несколько «потоков управления», совместно использующих данные в памяти. Linux поддерживает несколько пакетов уровня ядра и уровня пользователя, а потоки ядра Linux выполняются на нескольких процессорах, используя преимущества настоящей параллельной аппаратной обработки. Пакет потоков ядра Linux совместим со стандартом POSIX 1003.1c.

Ядро Linux поддерживает загрузку страниц исполняемых файлов по запросу. Это означает, что в память считываются с диска только те сегменты программы, которые фактически используются. Кроме того, при одновременном выполнении нескольких экземпляров программы в памяти находится лишь один ее экземпляр. Исполняемые модули используют динамически связываемые библиотеки совместного доступа. Это означает, что исполняемые модули совместно используют код библиотеки, находящейся в единственном библиотечном файле, расположенном на диске. В результате испол-

няемые файлы занимают значительно меньшее дисковое пространство. Кроме того, одновременно в памяти находится лишь один экземпляр кода библиотеки, что сокращает общий объем используемой памяти. Можно использовать и статически связываемые библиотеки, если есть стремление к созданию «законченных» исполняемых модулей, которым не нужно наличие совместно используемых библиотек. Динамическое связывание совместно используемых библиотек на этапе выполнения позволяет программистам заменять библиотечные модули своими собственными.

Для более эффективного использования памяти машины в Linux реализована так называемая *виртуальная память* (*virtual memory*) со страничной подкачкой файлов. Это означает, что на диске может быть выделено пространство для свопинга.[1] Когда приложению требуется больше физической памяти, чем фактически есть в системе, неактивные страницы сбрасываются на диск. (*Страница* – это просто единица измерения памяти, выделяемой операционной системой. В большинстве архитектур она составляет 4 Кбайт.) При очередном обращении к таким страницам они считываются с диска обратно в оперативную память. Такой механизм позволяет выполнять приложения большего размера и поддерживать одновременно большее число пользователей. Однако свопинг не может служить равноценной заменой физической оперативной памяти: необходимость чтения страниц с диска замедляет работу.

Ядро Linux хранит в памяти фрагменты файлов, к которым недавно осуществлялся доступ, чтобы избежать лишних обращений к диску (относительно медленных). Вся свободная память системы используется для кэширования обращений к диску, поэтому при невысокой загрузке системы доступ к большому количеству файлов может быстро осуществляться через оперативную память. Когда пользовательским приложениям требуется больший объем физической памяти, размер дискового кэша уменьшается. Таким образом, физическая память всегда используется полностью.

Для облегчения отладки ядро Linux осуществляет *дампы памяти* программ, попытавшихся выполнить недопустимую операцию, такую как обращение к недействительному адресу памяти. Дамп памяти, записываемый в файл с именем *core* в рабочем каталоге программы, помогает программисту определить причину возникновения аварийной ситуации. Об использовании дампа памяти при отладке мы поговорим в разделе «Исследование файла дампа памяти» главы 14.

---

[1] Строго говоря, это не следует называть пространством для свопинга: выгружаются не процессы целиком, а отдельные страницы памяти. В отдельных случаях могут оказаться выгруженными и целые процессы, но это не обязательно. Термин «swap space» происходит от ранних разработок Linux, и формально следовало бы говорить о «paging space» – страничном файле.

Характеристики программного обеспечения

В этом разделе мы познакомимся со многими имеющимися программными приложениями для Linux и поговорим о ряде распространенных вычислительных задач. В конце концов, самое главное в системе – спектр существующих для нее приложений. Что еще производит впечатление в Linux, так это бесплатность большей части имеющихся приложений.

Основные команды и утилиты

Практически все утилиты, которые можно встретить в обычных реализациях Unix, перенесены в Linux. В их числе основные команды, такие как *ls*, *awk*, *tr*, *sed*, *bc*, *more* и др. На Linux перенесены многие распространенные программные пакеты, включая Perl, Python, Java Development Kit и др. Что бы вы ни назвали, в Linux это есть. Поэтому можно рассчитывать встретить на Linux рабочую среду, знакомую по другим Unix-системам. В Linux есть все стандартные команды и утилиты.

В Linux работают многие текстовые редакторы, в том числе *vi* (а также «современные» версии, такие как *vim*), *ex*, *pico* и *jove*, а также GNU Emacs и его разновидности, такие как XEmacs (с расширениями для X Window System) и *joe*. К какому бы текстовому редактору вы ни привыкли, скорее всего, он перенесен на Linux.

Выбор текстового редактора – интересный вопрос. Многие пользователи Unix по-прежнему работают в таких «простых» редакторах, как *vi* (и действительно, текст для первого издания этой книги был написан в *vi* под Linux). Однако *vi*, в силу его возраста, присущи многочисленные ограничения, поэтому популярность приобретают более новые (и сложные) редакторы, такие как Emacs. Emacs поддерживает полный, основанный на LISP макроязык и интерпретатор, мощный командный синтаксис и другие замечательные расширения. Существуют пакеты макросов для Emacs, которые позволяют читать электронную почту и телеконференции, редактировать содержимое справочников и даже организуют сеансы психотерапии с использованием искусственного интеллекта (незаменимые для доведенных до стресса фанатиков Linux). В главу 9 мы включили полное руководство по *vi* и подробно описали Emacs.

Интересно отметить, что Linux-утилиты по большей части являются программным приложением GNU. Эти утилиты GNU поддерживают развитые функции, которых нет в BSD и AT&T. Например, GNU-версия редактора *vi*, называемая *elvis*, включает структурированный макроязык, отличающийся от оригинальной версии AT&T. Однако в утилитах GNU прослеживается стремление сохранить совместимость со своими аналогами в BSD и System V. Многие считают, что версии GNU превосходят оригиналы. Примером служат GNU-утилиты сжатия файлов *gzip* и *bzip2*, более эффективные, чем оригинальная Unix-утилита сжатия. (Если вы хотите продемонстрировать «старую закалку», то можете продолжать пользоваться такими программами,

как *ex* и *compress*. Это может произвести впечатление на ваших друзей, привыкших делать все в GUI щелчком мыши.)

Для многих пользователей самой важной утилитой является *оболочка* (shell), или командный процессор. Оболочка – это программа, которая принимает команды пользователя и выполняет их. Кроме того, многие оболочки имеют такие возможности, как *управление заданиями* (что позволяет пользователю одновременно управлять несколькими процессами – не обращайте внимания, что это звучит, как у Оруэлла), перенаправление ввода/вывода и командный язык для написания *командных сценариев* (*shell scripts*). Командный сценарий – это файл, содержащий программу на языке оболочки, аналогичный «командному файлу» в MS-DOS.

Для Linux есть много оболочек. Важнейшим различием между оболочками является командный язык. Например, оболочка C shell (*csh*) использует язык, напоминающий язык программирования C. В классической оболочке Борна используется иной командный язык. Выбор оболочки часто основывается на том, какой командный язык она предоставляет. Используемая оболочка определяет в известной мере вашу рабочую среду в Linux.

К какой бы оболочке Unix вы ни привыкли, ее версия, скорее всего, перенесена на Linux. Самой популярной оболочкой является GNU Bourne Again Shell (*bash*) – «еще одна оболочка Борна». В *bash* включены многие развитые возможности, такие как управление заданиями, история команд, автозавершение имен команд и файлов, Emacs-образный (или при желании *vi*-образный) интерфейс редактирования командной строки и мощные расширения языка стандартной оболочки Борна. Другой популярной оболочкой является *tcsh* – версия C-оболочки с расширенными функциями, аналогичными имеющимся в *bash*. В число других оболочек входят оболочка Корна (*ksh*), *ash* от BSD, *zsh* – маленькая Борн-образная оболочка, и *rc* – оболочка Plan 9.

Что важно в этих базовых утилитах? Linux предоставляет уникальную возможность перекроить систему соответственно своим потребностям. Например, если вы – единственный пользователь системы и предпочитаете редактор *vi* и оболочку *bash*, нет причин устанавливать другие редакторы или оболочки. Среди фанатиков и пользователей Linux преобладает позиция «сделай сам».

Обработка текста и обработка слов

Почти каждому пользователю компьютера требуется та или иная система подготовки текстов. (Действительно, один из авторов книги почти разучился писать ручкой на бумаге.) В мире PC *обработка слов* (*word processing*) является нормой: она подразумевает редактирование и обработку текста, часто в среде WYSIWYG («What-You-See-Is-What-You-Get» – «что видишь, то и имеешь»), а также изготовление печатных экземпляров текста с рисунками, таблицами и прочими украшениями.

В мире Unix чаще встречается *обработка текста* (*text processing*) что совсем иное, нежели классическая обработка слов. В системе обработки текста

автор вводит текст с помощью «языка набора», описывающего то, как текст должен быть форматирован. Автор может модифицировать текст не в специальной среде обработки слов, а в любом текстовом редакторе, таком как *vi* или Emacs. Когда ввод исходного текста (на языке набора) закончен, пользователь форматирует текст отдельной программой, преобразующей исходный текст в формат, пригодный для печати. Это в известной мере аналогично программированию на некотором языке, таком как C, и «компиляции» текста в пригодную для печати форму.

Есть много систем обработки текста, работающих под Linux. Одной из них является *groff*, GNU-версия классического форматтера текста *troff*, разработанного первоначально в Bell Labs и все еще использующегося во многих Unix-системах во всем мире. Другой системой обработки текста является TEX, созданный достославным Дональдом Кнутом. Существуют диалекты TEX, такие как LATEX, а также многочисленные расширения и пакеты. Примером может служить PDF&LATEX – пакет, генерирующий файлы PDF прямо из документов LATEX.

Текстовые процессоры, например TEX и *groff*, различаются в основном синтаксисом языка форматирования. Выбор той или иной системы форматирования основывается на том, существуют ли для нее утилиты, удовлетворяющие ваши потребности; кроме того, это дело вкуса.

Например, язык форматирования, используемый *groff*, некоторым кажется не вполне понятным, поэтому они предпочитают TEX, язык которого более удобочитаем. Однако *groff* позволяет выводить текст в чистом ASCII и просматривать его на терминале, в то время как TEX предназначен в основном для вывода на печать. Вместе с тем, существуют программы, позволяющие получать чистый ASCII-текст из документов, форматированных в TEX, или, скажем, преобразовывать документы TEX в *groff*.

Еще одной системой обработки текста является Texinfo – расширение TEX, используемое для документирования программного обеспечения в FSF. Texinfo позволяет из одного исходного файла получить печатный документ или гипертекст типа «Info» для просмотра броузером. Файлы «info» являются основным форматом документации, используемым программным обеспечением GNU, в частности Emacs.

Текстовые процессоры широко используются в компьютерном мире для подготовки заметок, диссертаций, журнальных статей и книг. (Например, эта книга была сначала написана в формате LATEX, отфильтрована в самодельную SGML-систему и отпечатана в издательстве из *groff*.) Доступность языка исходного текста как простого текстового файла открывает возможность для создания расширений самого текстового процессора. Благодаря тому что исходные документы не хранятся в неизвестном формате, который может читать только специальный процессор слов, программисты могут создавать синтаксические анализаторы и трансляторы языка форматирования, расширяя таким образом систему. Такой подход вполне согласуется с философией Unix создания приложений в виде набора небольших утилит, взаи-

модействующих друг с другом, вместо больших монолитных систем, представляющих собой «черные ящики».

Как выглядит такой язык форматирования? Обычно исходный текст на языке форматирования состоит в основном из собственно текста, а также «управляющих кодов» для создания определенных эффектов, например смены шрифта, установки полей, создания списков и т. п.

Самым известным языком форматирования является HTML – язык разметки, на котором написана практически каждая страница в World Wide Web. Другим популярным языком обработки текстов является Docbook, своего рода отраслевой стандарт набора тегов для разметки технической документации, используемый и в Проекте документации Linux (обсуждаемом далее в этой главе). Вот как выглядит один из предыдущих разделов в Docbook:

```
<sect2><title> Основные команды и утилиты </title>

<para>

Практически все утилиты, которые можно встретить в обычных реализациях Unix,
перенесены в Linux. В их числе основные команды, такие как <command>ls</command>,
<command>awk</command>, <command>tr</command>, <command>sed</command>,
<command>bc</command>, <command>more</command> и другие. На Linux перенесены многие
распространенные программные пакеты, включая Perl, Python, Java Development Kit
и другие. Что бы вы ни назвали, в Linux это есть. Поэтому вы можете рассчитывать
встретить на Linux рабочую среду, знакомую по другим Unix-системам. В Linux есть все
стандартные команды и утилиты.
</para>
```

С первого взгляда язык набора может показаться неясным, но на практике его очень легко освоить. В результате использования системы обработки текста выдерживаются типографские стандарты. Например, все нумерованные списки в документе будут выглядеть одинаково, если только автор не изменит определение «окружения» нумерованного списка.

Основной целью наборного языка является предоставление автору возможности сосредоточиться на содержании, не беспокоясь о типографском оформлении. В результате при печати показанного фрагмента команды внутри тегов `<command>` будут напечатаны шрифтом, цветом и с прочими атрибутами, которые выберет издатель, а кроме того, легко составить указатель команд. Более того, вместо странно выглядящего тега `<xref>` будут вставлены соответствующий номер и название главы, которые будут правильными, даже если авторы переставят главы после написания раздела.

Хотя для HTML существуют редакторы типа WYSIWYG, немного попрактиковавшись, можно привыкнуть вручную вводить теги, подобные имеющимся в примере. В развитых текстовых редакторах, таких как Emacs и *vim*, есть специальные макросы и окружения для редактирования HTML, LaTeX и других типов документов, а также удобные возможности в виде особых шрифтов и цвета для выделения тегов различных видов. С помощью специальных средств можно создать выходной текст в стандартном формате, например PostScript или PDF, и вывести его на экран или принтер.

Процессоры типа WYSIWYG привлекательны по ряду причин. Они предоставляют мощный и иногда сложный визуальный интерфейс для редактирования документов. Однако этот интерфейс неизбежно ограничен доступными пользователю сторонами форматирования текста. Например, во многих процессорах слов есть специальный «язык форматирования» для создания сложных выражений, таких как математические формулы. Этот язык форматирования сходен с обработкой текста, хотя имеет значительно более ограниченные возможности.

Преимущество текстовых процессоров состоит в том, что они позволяют точно указать, что вам требуется. Они позволяют также редактировать исходный текст любым текстовым редактором и затем преобразовать его в другой формат. Платой за такую гибкость и мощь является отсутствие WYSIWYG-интерфейса. Многие пользователи процессоров слов привыкли видеть форматированный текст сразу в процессе редактирования. С другой стороны, при работе с текстовым процессором обычно не заботятся о том, как будет выглядеть текст после форматирования. Пишущий привыкает мысленно представлять, как будет выглядеть текст в результате выполнения вставленных в него команд форматирования.

Есть программы, позволяющие перед печатью просмотреть отформатированный документ на графическом дисплее. Например, программа *xdvi* отображает «независимый от устройства» файл, генерируемый под X Window System. Другие приложения, например *xfig*, предоставляют графический интерфейс WYSIWYG для создания рисунков и диаграмм, которые впоследствии конвертируются в команды текстового процессора, включаемые в документ.

Существует еще много других утилит текстовых процессоров. С перенесенным под Linux TEX поставляется мощная система METAFONT, используемая для создания шрифтов для TEX. В число других программ входят *ispell* – интерактивная программа проверки орфографии, *makeindex* – используется для создания предметных указателей в документах LATEX, а также многочисленные пакеты макросов *groff* и TEX для форматирования документов различных типов и математических текстов. Существуют также программы, транслирующие тексты TEX в *groff* и обратно, а также во многие другие форматы.

Коммерческие приложения

Коммерческие разработчики программного обеспечения проявили взлет интереса к созданию программ для Linux. В число этих программ входят системы автоматизации делопроизводства, процессоры слов, научные приложения, средства сетевого администрирования и крупномасштабные базы данных. Linux стала играть важную роль на рынке коммерческого ПО, и количество популярных коммерческих приложений для Linux может поразить. Нет никакой возможности рассказать здесь обо всех них, поэтому мы коснемся самых популярных приложений и кратко отметим ряд других.

StarOffice – это полный пакет автоматизации офиса для Linux, выпускаемый Sun Microsystems (первоначально его разработала небольшая компания

Star Division, которую купила Sun). Этот пакет, версии которого есть для Windows и Solaris, является в некоторой степени клоном Microsoft Office и содержит процессор слов, электронную таблицу, редактор HTML, администратора презентаций и другие средства. Он может читать файлы во многих форматах аналогичных приложений (в том числе Microsoft Office) и бесплатен для некоммерческого использования.

Corel выпустила WordPerfect Office 2000 для Linux – еще один пакет автоматизации офиса в составе процессора слов, электронной таблицы, программы презентаций, менеджера личных данных и других приложений. Он бесплатен для личного пользования, но есть и коммерческая лицензия. Corel выпустила также профессиональный графический пакет для Linux CorelDRAW.

Oracle, IBM, Informix, Sybase и Interbase выпустили коммерческие СУБД для Linux. Многие базы данных для Linux продемонстрировали более высокую производительность, чем соответствующие версии для Windows 2000.

Очень популярна такая база данных для Linux, как MySQL, бесплатная и простая в эксплуатации СУБД, которую можно получить на *http://www.mysql.com*. Поскольку MySQL легко установить, настроить и использовать, она быстро стала предпочтительным выбором для тех приложений, которые не нуждаются в сложностях различных коммерческих СУБД. Кроме того, несмотря на свою бесплатность, MySQL поддерживается компанией, которая ее разработала – MySQL AB. Основы работы с MySQL мы осветим в главе 18.

Все же в MySQL отсутствуют некоторые более развитые функции коммерческих баз данных. Некоторые пользователи предпочитают базу данных с открытым исходным кодом PostgresSQL, и Red Hat включает ее в некоторые свои продукты.

Помимо баз данных для Linux существует целый ряд промышленных приложений. Linux – одна из самых популярных платформ для поддержки предоставления услуг в Интернете, поэтому естественно, что для Linux выпущены мощные платформы для масштабируемых веб-сайтов, включая BEA WebLogic и IBM WebSphere. Есть также коммерческие высокопроизводительные виртуальные машины Java, поставляемые Sun, IBM и др. IBM выпустила популярный сервер коммуникаций и веб-приложений Lotus Domino, а также платформу коммуникаций WebSphere MQ (прежнее название – MQSeries).

Ученые, инженеры и математики обнаружат, что в Linux можно работать с рядом популярных коммерческих продуктов, таких как Maple, Mathematica, MATLAB и Simulink. В число других коммерческих приложений для Linux входят мощные системы автоматизированного проектирования, средства управления сетями, брандмауэры и среды разработки программного обеспечения.

Языки программирования и утилиты

Linux предоставляет полную среду программирования под Unix, включая все стандартные библиотеки, средства программирования, компиляторы и

отладчики, которые есть в других Unix-системах. Чаще всего в Linux применяют GNU Compiler Collection (набор компиляторов GNU), или *gcc*. *gcc* может компилировать C, C++, Objective-C (еще один объектно-ориентированный диалект C), Chill (язык программирования, используемый в основном для телекоммуникаций), FORTRAN и Java. В мире Unix разработка приложений и системное программирование обычно осуществляются на C или C++, и *gcc* представляет собой один из лучших имеющихся компиляторов C/C++, который поддерживает массу развитых функций и оптимизаций.

Java, относительно новое явление среди языков программирования, полностью поддерживается в Linux. Несколько компаний и независимых проектов выпустили переносы Java Development Kit для Linux, включая Sun, IBM и Blackdown Project (выполнивший один из первых переносов Java на Linux). Java – это объектно-ориентированный язык программирования и среда исполнения, которая поддерживает широкий спектр приложений, включая апплеты веб-страниц, базирующиеся на Интернете распределенные системы, соединение с базами данных и многое другое. Написанные для Java приложения могут выполняться на любой системе (независимо от архитектуры ЦПУ и операционной системы), поддерживающей виртуальную машину Java. Существует ряд JIT-компиляторов Java («just-in-time»), и в поставку Java Development Kit для Linux (JDK) входят высокопроизводительные JIT-компиляторы, работающие не хуже, чем компиляторы для Windows или других Unix-систем. IBM выпустила полную интегрированную среду разработки на Java, VisualAge for Java. *gcc* может компилировать Java-программы непосредственно в выполняемые модули и частично поддерживает стандартные библиотеки JDK.

Помимо C, C++ и Java в Linux были перенесены многие другие компилируемые и интерпретируемые языки, такие как Smalltalk, FORTRAN, Pascal, LISP, Scheme и Ada. Кроме того, есть ряд ассемблеров для написания программ в машинных кодах. Группа разработчиков работает над проектом open source под названием Mono, цель которого – скопировать основные элементы проекта Microsoft .NET для Unix- и Linux-систем. Возможно, самую важную группу языков программирования для Linux составляют многочисленные языки сценариев, включая Perl (язык сценариев, являющийся вершиной всех языков этого типа), Python (первый язык сценариев, исходно проектировавшийся как объектно-ориентированный) и Tcl/Tk (подобная оболочке система обработки команд с поддержкой возможности создания простых приложений для X Window System).

В системах Linux применяется развитый отладчик *gdb*, позволяющий пошагово выполнять программы в поисках ошибок или анализировать дампы памяти для выяснения причины краха программы. Утилита профилирования *gprof* позволяет получать статистику работы программы для выяснения того, на что программа тратит больше всего времени. Текстовые редакторы Emacs и *vim* предоставляют интерактивную среду редактирования и компиляции для различных языков программирования. В числе других инструментов, перенесенных на Linux, – утилита GNU *make*, используемая для

управления компиляцией больших программ, а также системы контроля версий исходного кода CVS и Revision Control System.

Linux идеально подходит для создания Unix-приложений. Она обеспечивает современную среду программирования со всеми дополнительными возможностями, и многие профессиональные Unix-программисты заявляют, что Linux – их любимая ОС для разработки и отладки. Студенты, изучающие вычислительную технику, могут использовать Linux для изучения программирования в Unix и других особенностей системы, например архитектуры ядра. Linux предоставляет доступ не только к полному набору библиотек и утилит, но и к исходным текстам ядра и библиотек. Языкам программирования и инструментальным средствам, доступным в Linux, посвящены главы 13 и 14.

Система X Window

X Window System служит стандартным графическим интерфейсом для Unix-машин. Первоначально она была разработана в MIT в 1980-х годах с целью дать возможность приложениям выполняться на ряде рабочих станций Unix разных производителей. X – мощная графическая среда, поддерживающая многие приложения. Написано много специальных приложений для X, включая игры, графические утилиты, средства программирования и работы с документацией и др.

В отличие от Microsoft Windows, X Window System располагает встроенной поддержкой сетевых приложений. Например, можно запустить приложение X на сервере, при этом его окна будут передаваться на рабочую станцию по сети. Кроме того, X в очень большой степени доступна настройке: можно изменить по своему вкусу почти любую характеристику системы. Можно менять шрифты, цвета, оформление окон и значки в соответствии со своими предпочтениями. Можно даже эмулировать в X рабочий стол Windows или Macintosh, если желательно сохранить привычный интерфейс.

X Window System распространяется свободно. Однако ряд разработчиков стали распространять коммерческие расширения первоначального программного обеспечения X. Версия X Window для Linux известна как XFree86 и является переносом X11R6 (X Window System Version 11, Release 6), объявленного свободно распространяемым для Unix-систем, устанавливаемых на PC, таких как Linux. XFree86 поддерживает широкий спектр видеоустройств, включая стандартный VGA и ряд видеоадаптеров с ускорителями. XFree86 является полным комплектом программного обеспечения X, включающим в себя сам X-сервер, ряд прикладных программ и утилит, программные библиотеки и документацию. Он входит практически в каждый дистрибутив Linux.

В число стандартных X-приложений входят: *xterm* (эмулятор терминала, используемый для большинства текстовых приложений в X Window); *xdm* (менеджер сеансов X, обслуживающий регистрацию); *xclock* (простое представление часов); *xman* (программа вывода в X системного руководства) и др. X-приложений для Linux слишком много, чтобы все их здесь перечислить, но в базовый комплект XFree86 входят «стандартные» приложения,

содержавшиеся в исходной версии MIT. Отдельно доступны и многие другие, и, теоретически, все написанные для X Window приложения должны без проблем компилироваться под Linux.

За внешний вид и поведение интерфейса X Window отвечает *менеджер окон* (*window manager*). Эта дружественная программа отвечает за размещение окон, интерфейс пользователя, посредством которого изменяется размер окон, преобразование в иконки, перемещение окон, вид оконных рамок и т. д. Стандартный дистрибутив XFree86 включает в себя несколько менеджеров окон, в том числе популярный *fvwm2*. Последний обеспечивает ряд развитых функций, включая виртуальный рабочий стол: если пользователь передвигает мышь к краю экрана, смещается весь рабочий стол, как если бы дисплей имел значительно большие размеры, чем на самом деле. *fvwm2* в значительной мере настраиваем и разрешает доступ ко всем функциям, как с клавиатуры, так и с помощью мыши.

Дистрибутив XFree86 содержит программные библиотеки и файлы для программистов, желающих создавать X-приложения. Поддерживаются различные наборы графических элементов (widgets), такие как Athena, Open Look и Xaw3D. Включены все стандартные шрифты, растровые изображения и документация. Поддерживается также PEX (программный интерфейс для трехмерной графики) и MESA, свободно распространяемая реализация графических примитивов OpenGL 3D.

В главах 10 и 11 мы расскажем о том, как устанавливать и использовать X Window System на Linux-машине.

KDE и GNOME

Хотя X Window System предоставляет гибкую оконную систему, многим пользователям хотелось бы иметь законченную среду рабочего стола с возможностью настройки внешнего вида всех окон и графических элементов (например, кнопок и полос прокрутки), простым интерфейсом пользователя и развитыми возможностями, такими как перетаскивание данных между приложениями. KDE и GNOME – два отдельных проекта, имеющие целью предоставить такую развитую среду для Linux. Путем создания мощного комплекта средств разработки, библиотек и приложений, интегрированных в среду рабочего стола, KDE и GNOME нацелены на то, чтобы открыть новую эру настольных систем Linux. Обе системы предоставляют богатый графический интерфейс пользователя, менеджер окон, утилиты и приложения, по возможностям соперничающие с такими системами, как рабочий стол Windows 2000, или превосходящие их.

Благодаря KDE и GNOME даже неопытные и начинающие пользователи будут чувствовать себя в Linux как дома. Большинство дистрибутивов автоматически конфигурируют одну из этих рабочих сред во время установки, что исключает необходимость сталкиваться с текстовым интерфейсом консоли.

Хотя как KDE, так и GNOME ставят задачей сделать Unix-окружение более дружественным для пользователя, упор в них сделан на разных аспектах.

Главными целями KDE являются простота использования, устойчивость и совместимость на уровне интерфейса пользователя с другими вычислительными средами (например, Windows 2000). GNOME же нацелен на хорошее представление и максимум возможностей настройки.

Работа в сети

Linux считается одной из наиболее мощных и надежных сетевых систем в мире, и все большее число людей приходит к выводу, что Linux – это прекрасный выбор для сетевого сервера. Linux поддерживает семейство сетевых протоколов TCP/IP, на которых основан весь Интернет, включая IPv6 (новая версия протокола IP для Интернета следующего поколения), и UUCP (используемый для связи Unix-машин через последовательные линии). Linux позволяет связаться с любым компьютером в Интернете через Ethernet (в том числе Fast и Gigabit Ethernet), Token Ring, коммутируемые соединения, беспроводные сети, пакетные радиосети, последовательные порты, ISDN, ATM, IRDA, Appletalk, IPX (Novell NetWare) и многие другие сетевые технологии. Доступен весь спектр приложений для Интернета, включая веб-броузеры, веб-серверы, FTP, электронную почту, чат, телеконференции, *ssh*, *telnet* и др.

Большинство пользователей Linux подключается к Интернету из дома с помощью коммутируемого соединения с интернет-провайдером. Linux поддерживает популярные протоколы PPP и SLIP, используемые большинством провайдеров Интернета для доступа по коммутируемой линии. Если у вас есть широкополосный доступ через канал T1, кабельный модем, DSL или другое устройство, то Linux позволяет пользоваться и этими технологиями. Можно даже настроить Linux-машину для работы в качестве маршрутизатора и брандмауэра для целой сети компьютеров, которые будут подключены к Интернету через одно коммутируемое или широкополосное соединение.

В Linux поддерживаются многие веб-броузеры, в том числе Netscape Navigator, Mozilla (отделившийся от Netscape броузер open source), Konqueror (еще один броузер open source, поставляемый с KDE) и текстовый броузер Lynx. Даже текстовый редактор Emacs содержит небольшой текстовый веб-броузер.

Под Linux может также работать ряд веб-серверов, в том числе популярный бесплатный сервер Apache. Установлено, что Apache под Linux обслуживает больше веб-сайтов, чем какая бы то ни была другая платформа в мире. Как будет показано в главе 16, установить Apache и пользоваться им просто.

В Linux есть широкий выбор программ для электронной почты и телеконференций, таких как MH, Elm, Pine, *trn*, а также средства чтения почты/телеконференций, входящие в веб-броузеры Netscape и Mozilla. Многие из них совместимы со стандартными протоколами, например IMAP и POP. Каковы бы ни были ваши требования, Linux можно настроить для отправки и получения любых сообщений электронной почты и телеконференций.

Есть целый ряд других сетевых служб, которые могут работать под Linux. Samba является пакетом, позволяющим Linux-машинам действовать в ка-

честве файл-сервера и сервера печати Windows. NFS позволяет беспрепятственно совместно использовать файлы вместе с другими машинами сети. Благодаря NFS файлы на удаленных машинах выглядят так же, как если бы они располагались на ваших собственных дисках. FTP позволяет обмениваться файлами между машинами в сети. В число других сетевых возможностей входят основанные на протоколе NNTP системы электронных новостей, например C-News и INN; почтовые агенты *sendmail*, *exim* и *smail*; *ssh*, *telnet* и *rsh*, позволяющие регистрироваться и выполнять команды на других машинах в сети; *finger*, позволяющий получать информацию о других пользователях Интернета. В общем, существует масса различных приложений и протоколов, базирующихся на TCP/IP.

Если у вас есть опыт работы с приложениями TCP/IP на других Unix-системах, то Linux не будет для вас новинкой. Эта система предоставляет стандартный интерфейс программирования сокетов, поэтому практически любая программа, использующая TCP/IP, может быть перенесена на Linux. X-сервер Linux также поддерживает TCP/IP, что позволяет отображать на вашем дисплее приложения, выполняемые на других машинах. Сетевое администрирование также окажется знакомым для тех, кто пришел с других Unix-систем, поскольку средства конфигурации и мониторинга сходны со своими аналогами в BSD.

В главе 15 мы расскажем о настройке и установке под Linux TCP/IP, в том числе PPP. Мы разберем также вопросы настройки веб-броузеров, веб-серверов и почтовых программ.

Поддержка портативных компьютеров

В Linux есть ряд функций, специфических для лэптопов, таких как поддержка PCMCIA (или «PC Card») и APM. Пакет PCMCIA Tools для Linux содержит драйверы многих устройств PCMCIA, включая модемы, карты Ethernet и адаптеры SCSI. APM позволяет ядру следить за заряженностью аккумулятора и выполнять определенные действия (например, автоматически завершать работу) при снижении ее ниже допустимого уровня. Кроме того, эта функция переводит процессор в режим малого энергопотребления, когда он не используется. Такую функцию легко настроить как параметр ядра. Есть ряд утилит для взаимодействия с APM, в том числе *apm*, которая отображает состояние аккумулятора, и *apmd*, которая регистрирует состояние аккумулятора и может генерировать события, связанные с состоянием питания. Эти утилиты присутствуют в большинстве дистрибутивов Linux.

Взаимодействие с Windows и MS-DOS

Существуют различные утилиты для связи с Windows и MS-DOS. Наиболее известным приложением является проект Wine – эмулятор Microsoft Windows для X Window System под Linux. Цель этого еще не законченного проекта – позволить программам Windows выполняться непосредственно в Linux и других операционных системах на платформе Intel. Есть аналогичный коммерческий эмулятор Windows WABI, разработанный в Sun Microsys-

tems. Wine находится в процессе непрерывного развития и сейчас позволяет запускать многочисленные программы для Windows, включая настольные приложения и некоторые игры. Подробности о состоянии проекта можно получить на *http://www.winehq.com*.

Linux обеспечивает прозрачный интерфейс для перемещения файлов между Linux и Windows. Под Linux можно смонтировать раздел Windows или гибкий диск и непосредственно обращаться к файлам Windows, как и любым другим. Кроме того, существуют пакеты *mtools* для непосредственного доступа к гибким дискам, отформатированным в MS-DOS, и *htools*, делающий то же самое в отношении гибких дисков Macintosh.

Еще одним приложением из этой области является Linux MS-DOS Emulator, или DOSEMU, позволяющее запускать многие MS-DOS-программы прямо из Linux. Несмотря на то что программы MS-DOS быстро уходят в небытие, все же существует ряд интересных утилит и игр под MS-DOS, которые хотелось бы запускать в Linux. В DOSEMU можно даже запускать старую Microsoft Windows 3.1.

Linux не предоставляет полной поддержки эмуляции среды Windows и MS-DOS, но эти ОС можно запускать на той же машине, где стоит Linux, и выбирать операционную систему для запуска во время начальной загрузки. Мы покажем, как установить начальный загрузчик LILO, позволяющий выбирать между Linux, Windows и другими ОС при начальной загрузке.

Еще одним популярным решением является запуск на системном уровне «виртуальной машины», которая буквально позволяет выполнять Linux и Windows *одновременно*. *Виртуальная машина* – это программное приложение, которое эмулирует большую часть аппаратных средств машины, заставляя операционную систему поверить в то, что она работает на реальном физическом компьютере. С помощью виртуальной машины можно загрузить Linux, а затем запустить Windows – и на машине будут одновременно выполняться приложения Linux и Windows. Можно сделать наоборот – загрузить Windows, а затем запустить на виртуальной машине Linux. Хотя при работе с виртуальной машиной несколько снижается производительность, часто очень удобно применять такую технологию, чтобы, например, запустить под Linux текстовый процессор, написанный под Windows. Самые распространенные виртуальные машины – это VMWare (*http://www.vmware.com/*), являющаяся коммерческим продуктом, и Plex86 (*http://savannah.nongnu.org/projects/plex86*), представляющая собой проект open source.

Другие приложения

Существует тьма приложений для Linux, чего и следовало ожидать от операционной системы с таким разнообразным составом пользователей. В настоящее время Linux сосредоточена, главным образом, на персональных вычислениях в Unix, но ситуация быстро меняется. Расширяется спектр приложений для деловых и научных целей, появляется все больше коммерческих приложений.

В научном мире Linux бесспорно считается лучшей платформой для экономичного решения вычислительных задач. Для Linux разработано большое число научных программ, включая популярные технические инструменты MATLAB и Mathematica. Есть также большое число бесплатных пакетов, в том числе FELT (Finite Element Analysis Tool – анализ методом конечных элементов), Spice (средство проектирования и анализа микросхем) и Khoros (система обработки изображений/цифровых сигналов и визуализации). На Linux перенесены многие популярные библиотеки вычислительных алгоритмов, например LAPACK – библиотека методов линейной алгебры. Есть также оптимизированная для Linux версия кода BLAS, на котором основывается LAPACK.

Linux – одна из самых популярных платформ для организации параллельных вычислений с помощью кластеров, которые представляют собой группу недорогих машин, обычно соединенных между собой высокоскоростной (порядка гигабит в секунду или выше) сетью. Проект NASA Beowulf первым популяризировал идею связать большое число PC, работающих под Linux, в один большой суперкомпьютер, чтобы осуществлять научные или числовые расчеты. В настоящее время кластеры на базе Linux стали скорее правилом, чем исключением, во многих научных приложениях. Кластеры Linux находят себе применение в самых разнообразных приложениях. Например, поисковый механизм Google использует кластер из 4000 Linux-машин!

Как и во всякой операционной системе, в Linux есть свои игры. На Linux переведен ряд популярных коммерческих игр, в том числе Quake, Quake II, Quake III Arena, Doom, SimCity 3000, Descent и др. Большинство популярных игр позволяют играть через Интернет или локальную сеть. Возникают Linux-клоны других коммерческих игр. Есть также классические текстовые подземельные игры, такие как Nethack и Moria; игры типа MUD (многопользовательские «темницы», позволяющие многим пользователям участвовать в текстовых приключениях), например DikuMUD и TinyMUD; множество графических игр, таких как *xtetris*, *netrek* и *Xboard* (X11-интерфейс для *gnuchess*).

Меломаны найдут в Linux широкую поддержку различной звуковой аппаратуры и сопутствующих программ, таких как CDplayer (программа, управляющая приводом CD-ROM, как обычным CD-плеером, что довольно удивительно), MIDI-секвенсоры и редакторы (позволяющие сочинять музыку для воспроизведения через синтезатор или другой инструмент с поддержкой MIDI) и звуковые редакторы для оцифрованного звука.

Не можете найти нужную программу? Есть ряд веб-сайтов, на которых находятся обширные каталоги приложений для Linux. Наиболее популярны каталоги Linux-программ Freshmeat (*http://www.freshmeat.net*), Icewalkers (*http://www.icewalkers.com*) и Linux на Dave Central (*http://linux.davecentral.com*). Все эти каталоги далеко не полны, но содержат очень много программ, разбитых по категориям и с оценками пользователей. Посмотрите эти сайты, чтобы, по крайней мере, убедиться, как много программ написано для Linux.

Если требуемое никак не найти, всегда можно попытаться перенести на Linux приложение с другой платформы. В крайнем случае можно написать приложение самому. Это в духе Free Software – если хочешь, чтобы что-то было сделано хорошо, сделай это сам! Начинать одному крупный программный проект бывает страшно, но те, кто открывает публике раннюю версию своей программы, обнаруживают, что возникает множество помощников, готовых присоединиться к проекту.

Copyright для Linux

Linux подпадает под так называемую GNU *General Public License* (Универсальная общественная лицензия GNU), или *GPL*. GPL, которую иногда называют лицензией «copyleft» («права оставлены», как противоположность copyright – «права удержаны»), разработана Free Software Foundation (Фондом свободнораспространяемого программного обеспечения) для проекта GNU. Она налагает ряд условий на распространение и модификацию «свободного программного обеспечения». В этом смысле «свободное» относится к свободе, а не только к бесплатности. GPL всегда была подвержена неправильному толкованию, и мы надеемся, что данный обзор поможет понять границы и цели GPL, а также ее отношение к Linux. Полный текст GPL можно получить на *http://www.gnu.org/copyleft/gpl.html*.

Первоначально Линус Торвальдс выпустил Linux под более строгой, чем GPL, лицензией, которая допускала свободное распространение и модификацию, но запрещала денежные расчеты за распространение и использование. GPL разрешает продавать и получать прибыль от свободного программного обеспечения, но не позволяет каким-либо образом ограничивать права на дальнейшее распространение программного обеспечения.

Прежде всего, мы хотим разъяснить, что «свободное программное обеспечение», охватываемое GPL, *не* находится в общественном владении (public domain). Программное обеспечение в общественном владении не имеет копирайта и находится в общественном владении в буквальном смысле. Напротив, программное обеспечение, охватываемое GPL, имеет копирайт, принадлежащий одному или нескольким авторам. Это значит, что программное обеспечение защищено обычными международными законами об авторских правах и автор его юридически определен. Тот факт, что программное обеспечение может свободно распространяться, не означает, что оно находится в общественном владении.

Программное обеспечение с лицензией GPL не является также «shareware». Обычно программное обеспечение с лицензией «shareware» и авторские права на него принадлежат автору, который требует, чтобы пользователи прислали деньги за пользование полученным ПО. Напротив, программное обеспечение с лицензией GPL может бесплатно распространяться и использоваться.

GPL разрешает также модификацию свободного программного обеспечения и распространение модифицированных версий. Однако результаты модификации также должны охватываться GPL. Иными словами, фирма не может

взять Linux, модифицировать ее и продавать под ограничительной лицензией. Если какое-либо программное обеспечение является производным от Linux, оно также должно быть охвачено лицензией GPL.

Лица и организации могут распространять программное обеспечение GPL за плату и даже извлекать прибыль из продажи и распространения, однако продавец программного обеспечения GPL не может лишить таких же прав покупателя. Это значит, что если вы где-либо приобрели программное обеспечение GPL, то сами можете распространять его бесплатно или продавать.

Поначалу в этом можно усмотреть противоречие. Как можно с прибылью продавать программное обеспечение, когда GPL позволяет любому приобрести его бесплатно? Когда фирма собирает вместе много бесплатных программ, помещает их на компакт-диск и распространяет, ей должны быть оплачены расходы на производство и распространение компакт-диска, а кроме того, она может пожелать получить прибыль от продажи программного обеспечения. GPL это разрешает.

Организации, продающие свободное программное обеспечение, должны следовать некоторым ограничениям, установленным GPL. Во-первых, они не могут ограничить права пользователей, приобретающих программное обеспечение. Это значит, что, купив компакт-диск с программным обеспечением GPL, вы можете копировать и распространять его бесплатно или перепродавать самостоятельно. Во-вторых, распространитель должен сделать очевидным для пользователей, что программное обеспечение действительно охватывается GPL. И в-третьих, распространитель должен бесплатно предоставить полный исходный код распространяемого программного обеспечения или указать по требованию покупателя, откуда он может быть загружен. Это позволяет каждому покупателю программного обеспечения GPL модифицировать это программное обеспечение.

Разрешение распространять и продавать свободное программное обеспечение – очень полезная вещь. Не у всех есть доступ к Интернету для бесплатной загрузки такого программного обеспечения, как Linux. GPL позволяет фирмам продавать и поставлять программное обеспечение тем, у кого нет возможности загрузить его бесплатно или по разумной цене. Например, многие организации продают Linux на гибких дисках, лентах или компакт-дисках по почте и получают от этого прибыль. Разработчикам Linux из этой прибыли, вероятно, ничего не достанется. Таковы отношения между разработчиком и распространителем, устанавливаемые GPL. Иными словами, Линус знал, что фирмы могут начать продавать Linux, и он не получит из их прибыли ни гроша. (Если Линус и не богат, то по крайней мере он знаменит!)

В мире свободного программного обеспечения важны не деньги. Цель свободного программного обеспечения – разработка и распространение фантастических программ и возможность каждого получить их и пользоваться ими. В следующем разделе мы обсудим, как это отражается на разработке Linux.

Программное обеспечение Open Source и философия Linux

У новичков в Linux часто встречается ряд заблуждений и неверных ожиданий в отношении системы. Linux является уникальной операционной системой, для эффективного использования которой важно понимать ее философию и архитектуру. В центре философии Linux лежит идея, которую мы называем сейчас программным обеспечением open source (с открытым исходным кодом).

Open source означает программное обеспечение, исходный код которого (т. е. внутреннее устройство программы) каждый желающий может свободно загрузить, изменить и повторно распространить. Программное обеспечение, охватываемое GPL, описанной в предыдущем разделе, укладывается в категорию open source. Не должно удивлять, что в ту же категорию попадает много программного обеспечения с лицензиями, сходными с GPL, но не совпадающими с ней. Например, программное обеспечение, которое можно свободно модифицировать, но не имеющее таких же строгих требований к последующему распространению, как GPL, тоже считается Open Source. В эту категорию попадает ряд лицензий, в том числе BSD и Apache.

Модели разработки «open source» и «free software» – это явление, начавшееся с Фонда свободного программного обеспечения и ставшее популярным вместе с Linux. Это совершенно новый способ создания программного обеспечения, при котором все стороны разработки, отладки, тестирования и исследования открыты всем, кто проявит к этому интерес. Не полагаясь на какую-то одну фирму в разработке и поддержке программы, open source обеспечивает открытое развитие кода в сообществе разработчиков и пользователей, движимых желанием *создавать хорошее программное обеспечение*, а не просто стремлением к прибыли.

Издательство O'Reilly & Associates, Inc., опубликовало книгу «Open Sources», служащую хорошим введением в модель разработки open source. Это сборник очерков о процессе open source, написанных ведущими разработчиками (включая Линуса Торвальдса (Linus Torvalds) и Ричарда Столлмана (Richard Stallman)) под редакцией Chris DiBona, Sam Ockman и Mark Stone.

Пресса уделила большое внимание open source, и некоторые называют это явление «новой волной» в разработке программного обеспечения, которая заставит забыть прежние методы, когда все скрывалось от посторонних глаз. Поживем – увидим, но есть обнадеживающие признаки того, что это не столь уж несбыточно. Например, Netscape Corporation опубликовала исходный код своего веб-броузера как проект open source под названием Mozilla, и такие фирмы, как Sun Microsystems, IBM и Apple, объявили о планах выпуска некоторых своих продуктов как open source в надежде, что в условиях совместного общественного проекта разработки программного обеспечения они достигнут успеха.

Linux находится в центре внимания прессы к явлению Open Source. Чтобы понять, откуда происходит идеология разработки Linux, есть смысл взглянуть на то, как традиционно создавалось коммерческое программное обеспечение.

Коммерческие фирмы стремятся основывать разработку на строгой политике контроля качества, использования средств контроля исходного кода и версий, документирования, учета выявленных ошибок и их устранения. Разработчикам не разрешается по своей прихоти добавлять новые характеристики или изменять ключевые участки кода: они должны утверждать изменения как ответные действия на сообщение об ошибке и затем вводить все изменения в систему контроля версий, чтобы при необходимости их можно было «откатить». За каждым разработчиком закреплен один или несколько участков кода, и только этот разработчик может изменять эти участки, когда они «взяты для редактирования».

Внутри фирмы отдел технического контроля на каждом очередном проходе программного обеспечения выполняет наборы строгих тестов (так называемые «регрессивные тесты») и составляет отчеты об обнаруженных ошибках. Устранение этих ошибок является обязанностью разработчика. Применяется сложная система статистического анализа для того, чтобы при выпуске очередной версии был устранен определенный заданный процент ошибок и операционная система в целом удовлетворила некоторым выходным критериям.

В целом, процесс сопровождения и поддержки кода разработчиками коммерческого программного обеспечения очень сложен, и это правильно. Фирма должна иметь количественный критерий того, что очередная версия программного обеспечения готова к поставке. Разработка коммерческой программной системы – очень большая работа, для выполнения которой часто требуются сотни (если не тысячи) программистов, тестеров, составителей документации и административного персонала. Конечно, у каждой фирмы-разработчика есть свои особенности, но в целом картина такова, как описано. Небольшие фирмы-разработчики обычно копируют такую организацию в уменьшенном масштабе.

Полную противоположность этому представляет собой Linux, которая является хакерской операционной системой[1] и, видимо, всегда будет такой. Хотя многие проекты open source приняли элементы коммерческой технологии разработки приложений, такие как системы контроля версий и регистрации ошибок, все же неотъемлемой чертой разработки Linux являются сотрудничество и рассредоточение, что в корне отлично от традиционного подхода.

Linux разрабатывается, главным образом, как совместный проект группы добровольцев из Интернета, живущих в разных концах света. Единой организации, ответственной за разработку системы, не существует. По большей

---

[1] Под словом «хакер» мы подразумеваем одержимого своим делом программиста, человека, который получает удовольствие от работы на компьютере и обычно делает на нем что-либо интересное. Такое значение противостоит распространившемуся представлению о хакере как человеке, делающем с компьютером что-либо нехорошее или противозаконное.

части, Linux-сообщество общается с помощью различных почтовых списков рассылки и веб-сайтов. В ходе работы над проектом возник ряд соглашений; например, программисты, которые хотят, чтобы их код был включен в «официальное» ядро, должны послать его Линусу Торвальдсу. Он протестирует код и включит его в ядро (если код ничего не портит и не противоречит общей архитектуре системы, то, скорее всего, он это сделает). По мере развития Linux эта задача стала слишком трудоемкой, чтобы Линус мог справиться с ней один (к тому же он теперь обзавелся детьми), поэтому есть другие участники, отвечающие за тестирование кода и включение его в определенные части ядра, например в сетевую подсистему.

Архитектура самой системы такова, что делает ее весьма открытой для добавления новых возможностей и функций. Новая версия ядра обычно выходит раз в несколько недель, а иногда и чаще. Конечно, это очень приблизительная цифра; все зависит от того, сколько обнаружено ошибок, подлежащих исправлению, велик ли поток сообщений от пользователей, тестирующих предварительные версии, и много ли удалось Линусу поспать в течение недели.

Следует сказать, что выход новой версии не означает, что устранены все до единой ошибки и решены все проблемы. (Это в такой же мере относится и к коммерческим программам!) Как только система представляется свободной от критических или часто возникающих ошибок, она признается «стабильной» и осуществляется выпуск новой версии. Главная задача разработки Linux не в том, чтобы создать совершенный и лишенный ошибок код, цель состоит в разработке бесплатной реализации Unix. Больше чем кому бы то ни было, Linux предназначена разработчикам.

Каждый, кто хочет добавить к системе новую функцию или программу, обычно делает ее доступной в версии «альфа». Это стадия для тестирования теми отчаянными пользователями, которые хотят устранить проблемы в исходном коде. Поскольку Linux-сообщество базируется в основном в Интернете, альфа-версии обычно выкладываются на один или несколько веб-сайтов Linux (см. приложение А), а в список почтовой рассылки Linux помещается сообщение о том, как получить и протестировать этот код. Загрузившие и протестировавшие программу пользователи могут по почте сообщить автору о результатах, исправленных ошибках и задать ему вопросы.

После устранения первоначальных проблем альфа-код переходит на стадию «бета», на которой он обычно считается стабильным, но не завершенным (т. е. могут быть реализованы не все функции). Он может и непосредственно перейти в «финальную» стадию, в которой программное обеспечение считается законченным и пригодным к использованию. Если это код ядра, то по завершении разработчик может попросить Линуса о включении его в стандартное ядро или в качестве необязательной дополнительной функции ядра.

Учтите, что это только принятые соглашения, а не правила. Некоторые разработчики настолько уверены в своих программах, что не выпускают версий стадии «альфа» или для отладки. Такие решения всегда принимает сам разработчик.

Что же происходит с регрессивным тестированием и строгим процессом контроля качества? Вместо него действует правило «выпускай скорее и выпускай чаще». Лучшими контролерами являются сами пользователи, поскольку они испытывают систему во множестве сред и с разнообразными реальными приложениями, что не так просто осуществить в любой штатной группе контроля качества. Одной из самых примечательных сторон такой модели разработки и выпуска является то, что ошибки (и бреши в системе безопасности) порой могут быть выявлены, объявлены и устранены в течение нескольких *часов*, а не дней и недель.

Поразительно, что такая неорганизованная система добровольцев, программирующих и отлаживающих целую Unix-систему, может вообще что-то произвести. Как показывает практика, это один из наиболее эффективных и целеустремленных программных проектов, когда-либо осуществлявшихся. Ядро Linux было написано полностью *с нуля*, без использования какого-либо фирменного кода. Добровольцами была проделана огромная работа по перенесению на Linux всего существующего на свете бесплатного программного обеспечения. Были написаны и перенесены библиотеки, созданы файловые системы, написаны аппаратные драйверы для многих распространенных устройств.

Программное обеспечение Linux обычно выпускается в виде *дистрибутива*, являющегося собранным в один пакет набором программ, в целом образующих систему. Большинству пользователей было бы слишком сложно вручную собрать систему с самого основания, начав с ядра, затем добавив утилиты и установив все необходимое программное обеспечение. Вместо этого предлагается целый ряд дистрибутивов, в состав которых входит все необходимое для установки и запуска всей системы. Опять-таки, стандартного дистрибутива не существует, их много, и у каждого есть свои преимущества и недостатки. Эта книга описывает установку дистрибутивов Red Hat, SuSE и Debian, но она может оказаться полезной при установке любого выбранного дистрибутива.

Несмотря на то что Linux является завершенной системой, для установки и запуска полной системы требуется некоторый опыт работы в Unix. Нет такого дистрибутива Linux, в котором не нашлось бы каких-то ошибок, поэтому после инсталляции может потребоваться вручную исправить небольшие проблемы. Некоторым читателям это может показаться морокой, но правильнее воспринимать данное обстоятельство как «радость Linux», т. е. удовольствие, получаемое от возни с системой, изучения и исправления ее. Именно такое отношение отличает энтузиастов Linux от простых пользователей. Linux может быть хобби, увлекательным спортом или стилем жизни. (Так же как у любителей сноуборда или горного велосипеда, у «линуксоидов» есть свой жаргон и стиль одежды; если не верите – потолкайтесь на какой-нибудь выставке Linux!) Многие новички в Linux пишут о том, с каким удовольствием они осваивают эту новую систему, и считают, что Linux воскрешает очарование, испытывавшееся при первом знакомстве с компьютером.

Советы новичкам в Unix

Для установки и запуска собственной Linux-системы не требуется большого опыта работы в Unix. На самом деле многие новички в Unix успешно устанавливают Linux на своих машинах. Приобрести опыт установки полезно, но учтите, что некоторым процедура установки может показаться весьма обескураживающей. Если повезет, то вы сможете установить Linux и начать в ней работать, не имея никакого опыта работы в Unix. Однако как только вы будете готовы заняться более сложными задачами – установкой новых программ, перекомпиляцией ядра и т. д., так знание Unix станет необходимостью. (Отметим, однако, что многие дистрибутивы Linux так же просто установить и сконфигурировать, как Windows 98, и уж конечно проще, чем Windows 2000.)

К счастью, запустив собственную Linux-систему, можно изучить те важные функции Unix, которые необходимо знать для выполнения этих задач. В этой книге содержится много информации, которая поможет начать изучение Unix. Глава 4 представляет собой учебник по основам Unix, а в главе 5 содержатся сведения по администрированию Linux. Возможно, лучше прочесть эти главы, прежде чем вообще начинать установку Linux. Содержащиеся в них сведения могут оказаться очень ценными, если придется столкнуться с проблемами.

Не следует рассчитывать на превращение из новичка в системного администратора Unix за одну ночь. Никакая реализация Unix не заработает без проблем и без сопровождения, и нужно быть готовым к сложностям. Рассматривайте это как возможность лучше изучить Linux и Unix и не теряйте мужества, если не все будет так гладко, как хотелось бы.

Советы для гуру Unix

Даже тем, у кого за плечами годы программирования и системного администрирования Unix, может понадобиться содействие, чтобы собрать и установить Linux. Знатокам Unix следует ознакомиться с некоторыми аспектами Linux, прежде чем пускаться в плавание. Во-первых, Linux не является коммерческой Unix-системой. Она не пытается соблюдать те же стандарты, что и другие Unix-системы, с которыми вы могли сталкиваться, но в известном смысле Linux *изменяет* мир Unix, заставляя все прочие системы потрудиться за те деньги, которых они стоят. Точнее, хотя стабильность – важный фактор в разработке Linux, он не является *единственным*.

Большее значение, вероятно, имеет функциональность. Во многих случаях новый код включается в ядро, даже если в нем все еще есть ошибки и не все функции реализованы. При этом предполагается, что важнее выпустить код, который пользователи могут тестировать и которым они будут пользоваться, чем задерживать релиз, пока он не достигнет совершенства. Почти все проекты open source выходят в альфа-версии, прежде чем они будут окончательно протестированы. В результате сообщество open source в целом имеет возможность работать с кодом, тестировать и разрабатывать его даль-

ше, а те, кто находит альфа-версию «достаточно хорошей» для своих нужд, могут ею пользоваться. Коммерческие производители Unix редко занимают такую позицию при выпуске программного обеспечения, если вообще когда-либо делают это.

Даже если вы суперспециалист по Unix и можете дизассемблировать ядро Solaris во сне, а также перекодировать суперблок AIX со связанными руками, некоторое время вам придется привыкать к Linux. Это современная и динамичная система, новые версии ядра которой выходят каждые несколько месяцев, а новые утилиты появляются постоянно. Сегодня ваша система вполне современна, а завтра вы оказываетесь с ней в каменном веке.

Как же при такой динамичности не отстать в изменчивом мире Linux? По большей части обновление лучше производить пошагово, т. е. обновлять те части системы, которые *требуют* обновления, и только тогда, когда вы считаете обновление необходимым. Например, если вы никогда не пользуетесь Emacs, нет смысла устанавливать в системе каждую новую его версию. Более того, даже если вы активно используете Emacs, обычно нет необходимости обновлять его, пока вы не обнаружите отсутствие какой-то функции, имеющейся в новой версии. Нет причин всегда стремиться к обладанию самой последней версией программы.

Помните, что Linux разработана своими пользователями. Из этого следует, что чаще всего Linux поддерживает ту аппаратуру, с которой приходится сталкиваться ее пользователям и разработчикам. Это приводит к тому, что поддерживается большая часть популярной аппаратуры и периферийных устройств для систем, основанных на 80x86 (Linux, возможно, поддерживает больше аппаратных устройств, чем любая коммерческая реализация Unix). Однако отсутствует поддержка некоторых темных и таинственных устройств, как и тех, изготовители которых поставляют фирменные драйверы, не раскрывая спецификаций. Со временем спектр поддерживаемой аппаратуры расширяется, поэтому если вы не найдете своего любимого устройства в списке, не исключено, что его поддержка грядет в скором будущем.

Другая трудность поддержки аппаратных устройств в Linux обусловлена тем, что многие компании решили сделать интерфейсы производимой ими аппаратуры своей собственностью. В итоге добровольные разработчики Linux просто не могут написать драйверы для таких устройств (если бы они это сделали, владельцами драйверов стали бы компании, владеющие интерфейсами, что нарушило бы условия GPL). Компании, сделавшие интерфейсы своей собственностью, пишут собственные драйверы для операционных систем, например Microsoft Windows. Конечному пользователю (т. е. вам) ничего не требуется знать об интерфейсе. К сожалению, это не дает возможности разработчикам Linux писать драйверы для этих устройств.

Едва ли можно что-то изменить в этой ситуации. Иногда программисты пытаются писать хакерские драйверы исходя из догадок о том, каков может быть интерфейс. В других случаях разработчики пытаются работать с соответствующей компанией, пытаясь получить информацию об интерфейсе устройства, с разной степенью успеха.

Источники информации по Linux

Как вы, вероятно, догадываетесь, помимо этой книги есть много источников информации по Linux.

Электронные документы

При наличии доступа в Интернет много документации по Linux можно получать через Web или с анонимных FTP-серверов, разбросанных по всему свету. При отсутствии прямого доступа в Интернет вы все же можете получить эти документы, поскольку многие дистрибутивы Linux на CD-ROM содержат все перечисленные здесь документы, и их часто можно купить в розницу. Кроме того, они распространяются через многие другие сети, такие как Fidonet и CompuServe.

Существует много веб-сайтов и FTP-архивов с программным обеспечением для Linux и сопутствующей документацией. В приложении А перечислены некоторые документы по Linux, имеющиеся в Интернете.

Примерами существующих электронных документов по Linux служат: Linux FAQ – сборник часто встречающихся вопросов по Linux; документы Linux HOWTO, каждый из которых описывает отдельный аспект системы, в том числе Installation HOWTO (установка), Printing HOWTO (печать) и Ethernet HOWTO (локальные сети); Linux META-FAQ – список иных информационных ресурсов Linux в Интернете.

Большинство этих документов регулярно поступает в одну или несколько телеконференций, относящихся к Linux (см. ниже раздел «Телеконференции»).

Исходная страница сервера Linux Documentation находится на *http://www.tldp.org*. Там находится много HOWTO и других документов, а также ссылки на другие сайты, интересные для пользователей Linux, в том числе на руководства из Проекта документирования Linux – Linux Documentation Project (см. следующий раздел).

Книги и другие печатные издания

В конце книги приведена библиография, в которой перечислено великое множество источников, полезных при работе с системой. Опубликован ряд работ, специально посвященных Linux. Кроме того, ряд бесплатных книг распространяется в рамках проекта Linux Documentation Project (LDP). Это осуществляемый в Интернете проект, задача которого – распространение добросовестного комплекта «руководств» по Linux. Эти руководства аналогичны комплектам документации, поставляемым с коммерческими версиями Unix: они охватывают все темы, от установки до работы в системе, программирования, сетевого взаимодействия, разработки ядра и др.

Руководства LDP можно получить через Web, а также заказать по почте из различных источников. В библиографии подробно перечислены имеющиеся руководства и способы получения их. Издательство O'Reilly & Associates

опубликовало *Linux Network Administrator's Guide* (Руководство сетевого администратора Linux) из проекта LDP.

Помимо растущего числа книг по Linux есть большое число книг собственно по Unix, которые, конечно, применимы к Linux: в том, что касается использования и программирования системы, Linux в большинстве отношений не сильно отличается от других реализаций Unix. Данная книга фактически подразумевает дополнение в виде большой библиотеки доступных в настоящее время книг по Unix. Мы излагаем наиболее важные, специфические для Linux детали, и надеемся, что за более глубокими сведениями вы обратитесь к другим источникам.

Вооружившись несколькими хорошими книгами по Unix и книгой, которую вы держите сейчас в руках, вы сможете справиться почти с любыми проблемами. В библиографии перечислены настоятельно рекомендуемые книги по Unix как для новичков, так и для опытных пользователей.

Есть по крайней мере два ежемесячных журнала по Linux: *Linux Journal* и *Linux Magazine*. Они дают прекрасную возможность следить за событиями, происходящими в Linux-сообществе.

Телеконференции Usenet

Usenet является всемирным форумом электронных новостей и дискуссий, с большим количеством так называемых «телеконференций» – дискуссионных клубов, посвященных какой-либо избранной теме. Значительная часть разработки Linux прошла по волнам Интернет и Usenet, и неудивительно, что есть много телеконференций, посвященных Linux.

Телеконференций, посвященных Linux, слишком много, чтобы можно было здесь перечислить их. Те, которые непосредственно относятся к Linux, входят в иерархию *comp.os.linux*, но есть и другие по родственным темам, например, *comp.windows.x*.

Почтовые списки рассылки в Интернете

При наличии доступа к электронной почте можно принять участие в ряде почтовых списков рассылки. Спектр обсуждаемых в них тем простирается от программирования ядра до основных вопросов, возникающих у пользователей. Многие популярные почтовые списки имеют веб-сайты, хранящие архивы, в которых можно вести поиск, что позволяет легко найти ответы на частые вопросы. Часть этих ресурсов перечислена в приложении А.

Получение помощи

Вам наверняка понадобится та или иная помощь во время путешествия по миру Linux. Даже самые искушенные из искушенных в Unix бывают озадачены какими-либо странностями или функциями Linux, и важно знать, где и как получить требуемую помощь.

Основным способом получения помощи в мире Linux являются списки почтовой рассылки и телеконференции, о которых было сказано выше. Ряд фирм также предоставляет коммерческую поддержку Linux. Подписавшись на нее, вы можете при возникновении проблем с Linux обращаться за помощью к консультантам. Ряд производителей дистрибутивов Linux предоставляет техническую поддержку через Интернет или по телефону, однако при наличии доступа к Usenet и электронной почте Интернета можно получить бесплатную помощь, которая окажется не хуже.

Следующие советы помогут освоить Linux и с большим успехом искать помощь в возникающих проблемах.

Сначала ищите решение во всей имеющейся документации. Столкнувшись с проблемами, прежде всего следует обратиться к различным источникам информации, перечисленным в предыдущем разделе и приложении А. Эти документы были старательно написаны для таких людей, как вы, то есть тех, кому нужна помощь по системе Linux. Даже книги, написанные по Unix, в общем применимы к Linux, и вам следует ими воспользоваться. Сколь бы невозможным это ни показалось, но в этой документации вы почти наверняка найдете решение своих проблем.

Если у вас есть доступ к Web, телеконференциям и относящимся к Linux спискам почтовой рассылки, сначала почитайте то, что в них есть, прежде чем писать письмо с изложением своих проблем и просьбой о помощи. Очень часто решения часто возникающих проблем нелегко найти в документации, но они оказываются хорошо освещены в телеконференциях и списках почтовой рассылки, посвященных Linux. Если вы будете только писать в эти конференции, но не читать их, то навлечете на себя неприятности.

Пользуйтесь поисковыми механизмами! Массу относящейся к Linux информации можно получить просто с помощью популярных поисковых машин. В Google есть даже специальная машина поиска, посвященная Linux, на *http://www.google.com/linux*. Хорошей отправной точкой может послужить архив телеконференций на Google (*http://groups.google.com/*). Вместо того чтобы бродить по веб-сайтам, посвященным Linux, рыться в HOWTO и архивах почтовых списков, часто результата гораздо быстрее можно достичь с помощью нескольких точных вопросов к поисковой машине, которой вы привыкли пользоваться.

Конечно, нужно научиться эффективно пользоваться поисковыми машинами: общий вопрос типа «Linux help» вряд ли возвратит именно то, что вам нужно. Гораздо лучше поискать «Linux Sony Vaio CD-ROM» (конечно, если вы ищете совета по своему Vaio CD-ROM!).

Привыкайте к самостоятельному техническому обслуживанию. Во многих случаях предпочтительно провести как можно больше самостоятельного изучения и исследования, прежде чем искать помощи со стороны. Помните, что Linux неразрывно связана с хакерством и самостоятельным решением проблем. Это не коммерческая операционная система, и она не стремится стать таковой. Вы не умрете, если займетесь хакерством. На практике в результате самостоятельного исследования и решения проб-

лем вы многое узнаете о системе, так что в один прекрасный день, возможно, сможете назвать себя гуру Linux. Научитесь ценить пользу хакерства системы и самостоятельного решения проблем. Нельзя рассчитывать на то, что законченная самодельная система Linux заработает без всякого приложения рук.

Сохраняйте спокойствие. Важно не терять присутствия духа при работе с системой. Вы ничего не добьетесь, набросившись в припадке гнева на свою Linux-систему с топором или, того хуже, мощным электромагнитом. Авторы открыли для себя, что приличных размеров боксерская груша или другой аналогичный неодушевленный предмет очень хорошо способствуют снятию время от времени возникающих стрессов. Когда Linux наберет зрелость и дистрибутивы станут более надежными, эта проблема, как мы надеемся, отпадет. Даже коммерческие версии Unix иногда могут преподносить сюрпризы. Если ничто другое не помогает, откиньтесь в кресле, сделайте несколько глубоких вдохов и, расслабившись, продолжите изучение проблемы. Ваш ум прояснится, а система ответит вам благодарностью. Не забывайте наш совет из Дзэн, данный в предисловии!

Воздержитесь от необоснованных почтовых отправлений. Многие совершают ошибку, поспешно отправляя письма в телеконференции и почтовые списки просьбы о помощи. Столкнувшись с проблемой, нельзя – повторяем, *нельзя* – бросаться к ближайшему терминалу, чтобы отправить послание в телеконференцию по Linux. Очень часто вы через пять минут сами обнаружите свою ошибку и окажетесь в глупой ситуации, доказывая в общественном месте, что вы находитесь в здравом уме. Прежде чем писать что-либо в любую телеконференцию или список рассылки, постарайтесь решить проблему сами и определите со всей четкостью, в чем она заключается. Ваша система не реагирует, когда вы пытаетесь ее включить? Возможно, вы забыли вставить шнур в розетку.

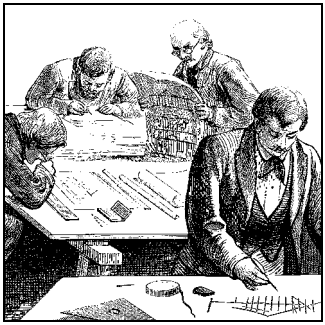
Если вы все же обращаетесь за помощью, пишите толково. Если другого выхода нет, можете послать просьбу о помощи в любой из электронных форумов, посвященных Linux, например телеконференцию Usenet или список рассылки. Помните при этом, что люди, читающие вашу почту, собрались там не для того, чтобы помогать вам. Сеть – это не ваша персональная служба консультаций. Поэтому старайтесь писать возможно более вежливо, сжато и информативно.

Как этого добиться? Во-первых, следует сообщить как можно более полные, но относящиеся к делу сведения о вашей системе и возникшей проблеме. Просто сообщив, что у вас не работает электронная почта, вы, вероятно, не стронетесь с этой точки, если не расскажете также, что за система у вас установлена, каким программным обеспечением вы пользуетесь, что вы уже предпринимали и какими были результаты. При сообщении технических данных неплохо указать общие сведения о версиях вашего программного обеспечения (например, номер версии ядра Linux), а также краткую информацию об аппаратной конфигурации. Однако не перестарайтесь – сообщение марки и типа вашего монитора, скорее всего, неуместно, если вы озабочены настройкой сетевого программного обеспечения.

Во-вторых, помните, что нужно хоть как-то попытаться самому решить свою проблему, прежде чем выходить в Сеть. Если, к примеру, вы никогда не устанавливали электронную почту и решили сначала поспрашивать народ в сети, как это делается, то вы совершаете большую ошибку. Существует целый ряд документов (см. предыдущий раздел «Источники информации по Linux»), в которых описано, как приступить к решению многих стандартных задач для Linux. Вы должны как можно дальше продвинуться самостоятельно и только *потом* просить о помощи, если вы застряли.

Помните также, что люди, читающие ваше послание, как бы доброжелательны они ни были, иногда приходят в отчаяние, раз за разом встречая одну и ту же проблему. Обязательно прочтите сначала все HOWTO, FAQ, материалы телеконференций и списков рассылки, прежде чем писать о своих проблемах. Очень часто бывает, что решение вашей проблемы уже неоднократно обсуждалось, и все, что требуется, – это полистать текущие сообщения.

В-третьих, при отправке писем в телеконференции и списки рассылки старайтесь быть предельно вежливы. Гораздо эффективнее выражаться вежливо, прямо и содержательно – больше людей захотят вам помочь, если вы изложите свою просьбу смиренным тоном. Несомненно, войны страстей – вид искусства, проникший во многие виды телекоммуникации, но не позволяйте этим страстям отнимать свое и чужое время. Сеть – это отличное место для получения помощи по связанным с Linux проблемам, но важно уметь пользоваться ею *эффективно*.



2

Подготовка к установке Linux

Эта глава представляет ваш первый шаг в установке Linux. Мы расскажем, как получить программное обеспечение для Linux в виде одного из укомплектованных дистрибутивов и как подготовить систему к установке. Мы расскажем, как создать дисковые разделы для совместной установки Linux с Windows, OS/2 или другой операционной системой.

Как уже говорилось, не существует единого «официального» дистрибутива Linux. На самом деле существует много дистрибутивов, каждый из которых служит определенной цели и набору задач. Эти дистрибутивы можно получить с анонимного FTP-сервера через Интернет, по почте в виде CD-ROM или DVD либо купить в магазине.

Дистрибутивы Linux

Поскольку Linux является свободно распространяемым программным обеспечением, нет единой организации или лица, ответственного за его выпуск и распространение. Поэтому любой желающий может собрать вместе и распространять программное обеспечение Linux, если при этом не нарушаются ограничения, накладываемые лицензией GPL. Результатом этого является наличие большого количества дистрибутивов Linux, распространяемых через анонимные FTP-серверы или по почте.

Сначала встает задача выбрать дистрибутив, который вам больше подходит. Между дистрибутивами есть различия. Многие содержат практически все программы, которые нужны, чтобы установить полную систему, и даже больше. Другие дистрибутивы – «маленькие» и рассчитаны на пользователей со скромным объемом дискового пространства.

Нужно также учесть, что дистрибутивы могут быть нацелены на разные группы пользователей. Одни предназначены, скорее, для предприятий, другие – для использования на домашних компьютерах. Одни более пригодны для установки сервера, другие – для создания рабочих станций.

В HOWTO Linux Distribution содержится список дистрибутивов Linux, которые можно получить через Интернет, а также заказать по почте.

Как сделать свой выбор среди этих дистрибутивов? Если у вас есть доступ к телеконференциям Usenet или другой системе конференций, можно спросить личное мнение тех, кто уже установил Linux. Еще лучше обратиться за помощью и советом к своим знакомым, уже установившим Linux. На практике большинство распространенных дистрибутивов Linux содержит примерно одинаковый набор программ, поэтому можно выбирать дистрибутив с достаточной степенью произвола.

Получение носителя с Linux по почте или иным образом

При отсутствии доступа к Интернету можно получить дистрибутив Linux, заказав CD-ROM или DVD по почте. Многие компании, распространяющие дистрибутивы, принимают кредитные карты и заказы из-за рубежа, поэтому можно получить Linux вне зависимости от вашего места жительства.

Linux является свободно распространяемым программным обеспечением, но согласно GPL дистрибуторы могут взимать за нее плату. Поэтому заказ Linux по почте может обойтись вам от 5 до 150 долларов США в зависимости от выбранного дистрибутива. Однако если у вас есть знакомые, которые приобрели или загрузили Linux, можно позаимствовать или скопировать их программное обеспечение для собственного использования. Производители дистрибутивов не имеют права ограничивать лицензию или дальнейшее распространение программного обеспечения. Например, если вы намерены установить Linux на целую группу машин, то достаточно приобрести одну копию дистрибутива, который можно использовать для установки Linux на все машины. Есть, впрочем, одно исключение из этого правила: чтобы добавить стоимость к своему дистрибутиву, некоторые поставщики включают в него коммерческие пакеты, устанавливать которые на нескольких машинах может быть запрещено. В таких случаях об этом должно быть явно сказано в пакете.

Многие группы пользователей Linux предлагают свои дистрибутивы; поинтересуйтесь, нет ли такой группы поблизости от вас. Для отдельных платформ, таких как Alpha, очень полезно получить Linux от группы пользователей.

Получение Linux из Интернета

Если у вас есть доступ в Интернет, проще всего получить Linux с анонимного FTP.[1] Один такой большой сервер FTP есть на *ftp://ftp.ibiblio.org*; в каталоге */pub/Linux/distributions* находятся различные дистрибутивы Linux.

При загрузке Linux обязательно используйте для передачи файлов двоичный режим (в большинстве FTP-клиентов этот режим включается командой *binary*).

Небольшие проблемы могут возникнуть при загрузке файлов для одной системы (например, Linux) с помощью другой системы (например, Windows), поскольку эти системы не всегда разумно обращаются с файлами другой сис-

[1] Если отсутствует прямой доступ в Интернет, но есть возможность обмениваться интернет-почтой, можно получить Linux через службу FTPMAIL.

темы. Однако советы, помещенные в этой главе, должны дать вам возможность все же завершить процедуру инсталляции.

Некоторые дистрибутивы распространяются через анонимные FTP в виде набора образов дисков. Это значит, что дистрибутив состоит из ряда файлов, каждый из которых содержит двоичный образ гибкого диска. Чтобы скопировать двоичный образ на дискету, можно использовать программу *RAWRITE.EXE* под Windows. Эта программа поблочно копирует содержимое файла на дискету, игнорируя его формат. *RAWRITE.EXE* можно взять на многих FTP-серверах Linux, в частности на *ftp://ftp.ibiblio.org* в каталоге */pub/Linux/system/Install/rawwrite*.

Сразу предупреждаем, что это трудоемкий способ установки Linux: дистрибутив вполне может насчитывать более 50 дисков.

При таком способе нужно загрузить набор образов гибких дисков и поочередно применять *RAWRITE.EXE* к каждому образу, чтобы создать набор дискет. После начальной загрузки машины с так называемого «загрузочного диска» (boot floppy) вы готовы к установке. Установка обычно производится с гибких дисков, хотя некоторые дистрибутивы позволяют осуществлять ее с раздела Windows жесткого диска, а другие позволяют производить установку по сети TCP/IP. В каждом дистрибутиве должна быть документация, описывающая эти методы установки, если они возможны.

Существуют дистрибутивы, устанавливаемые с комплекта дискет, отформатированных в MS-DOS. Например, дистрибутив Slackware требует *RAWRITE.EXE* только для дискет boot и root. Остальная часть гибких дисков копируется на форматированные в MS-DOS диски с помощью команды MS-DOS COPY. Программное обеспечение устанавливается непосредственно с дискет MS-DOS. Это избавляет вас от необходимости применения *RAWRITE.EXE* к многочисленным образам дисков, зато требует машины с MS-DOS для создания дискет.

Если у вас есть Unix-станция с дисководом для гибких дисков, можно прямо копировать файл образа на дискету с помощью команды *dd*. Например, на рабочей станции Sun команда `dd of=/dev/rfd0 if=foo bs=18k` произведет «сырую запись» содержимого файла *foo* на гибкий диск. Проконсультируйтесь у своих местных знатоков Unix относительно гибких дисков в вашей системе и использования *dd*.

Каждый дистрибутив Linux, полученный с анонимного FTP, должен содержать файл *README*, в котором описана процедура загрузки и подготовки гибких дисков для установки. Обязательно прочтите всю документацию по используемой вами версии дистрибутива.

В последнее время некоторые большие дистрибутивы Linux распространяются в виде ISO-образов, которые можно прожечь на CD-ROM или DVD. Ввиду огромного объема данных загрузка таких образов возможна только при наличии большого жесткого диска и интернет-соединения с большой пропускной способностью.

Подготовка к установке Linux

После получения дистрибутива нужно подготовить машину к установке Linux. Этот процесс требует некоторого планирования, особенно если у вас уже установлена другая ОС. В следующих разделах мы расскажем, как планировать установку Linux.

Общие сведения об установке

Хотя все дистрибутивы Linux различны, общий метод установки следующий.

1. *Заново разбейте жесткий диск (диски) на разделы.* Если у вас уже установлены другие ОС, вам понадобится заново разбить диски, чтобы освободить место для Linux. Эта процедура описана в разделе «Переразбиение жестких дисков» далее в этой главе. В некоторых дистрибутивах это действие интегрировано в процедуру установки. Проверьте в документации, так ли это в вашем случае. В любом случае не помешает разбить жесткий диск на разделы заранее.
2. *Загрузитесь с установочного диска.* У каждого дистрибутива есть установочный носитель, которым обычно являются загрузочная дискета или загрузочный компакт-диск. Загрузка с него вызовет какую-нибудь программу, которая пошагово проведет вас через процесс установки либо позволит осуществить установку вручную.
3. *Создайте разделы Linux.* После переразбиения диска с целью выделения места под Linux нужно создать в этом свободном пространстве разделы Linux. Это выполняется с помощью программы Linux *fdisk*, о которой рассказано в разделе «Создание разделов в Linux» главы 3, либо с помощью какой-нибудь другой специфической для данного дистрибутива программы вроде *Disk Druid*, поступающей с Red Hat Linux.
4. *Создайте файловые системы и раздел подкачки (свопинга).* На этом этапе на вновь созданных разделах создаются одна или несколько файловых систем для хранения файлов. Кроме того, если вы собираетесь использовать свопинг (а это необходимо, если только на вашей машине не установлена оперативная память гигантского размера), нужно выделить пространство для свопинга на одном из разделов Linux. Это описано в разделах «Создание пространства для свопинга» и «Создание файловых систем» главы 3.
5. *Установите программное обеспечение на новые файловые системы.* Наконец, вы устанавливаете программное обеспечение на вновь созданные файловые системы. Об этом рассказывается в разделе «Установка программного обеспечения» главы 3. Далее в разделе «Устранение неполадок» главы 3 мы расскажем, что делать, если что-то не получается. Ну а если все прошло хорошо, начинается спокойное плавание.

Те, кто хочет переключаться между различными ОС, иногда находятся в сомнении, какую систему устанавливать сначала – Linux или другую? Мы свидетельствуем, что при установке Windows 95/98/ME после Linux могут воз-

никнуть неприятности. Windows 95/98/ME имеет привычку при установке затирать данные в загрузочном секторе, поэтому безопаснее сначала установить Windows, а затем Linux, используя сведения, приведенные в этой главе. Похоже, что Windows NT/2000 ведет себя более терпимо в отношении уже существующих данных в загрузочном секторе. Можно предположить, что это распространяется и на Windows XP, ведущую происхождение от Windows 2000, но нельзя пока сослаться в этом на какой-либо личный опыт.

Во многих дистрибутивах Linux есть программа установки, которая проводит через процесс установки и автоматизирует один или несколько из описанных этапов. Читая эту и следующую главу, помните, что в зависимости от дистрибутива некоторое число описанных выше шагов может быть автоматизировано.

При подготовке к установке Linux очень полезно вести записи в течение всего процесса. Записывайте все, что вы делаете, все, что вводите с клавиатуры, и все, что может показаться вам ненормальным. Идея проста: если (или когда) случится неприятность, вам потребуется проследить свои шаги, чтобы понять, что произошло. Установить Linux несложно, но нужно помнить многие детали. Вам потребуется регистрировать все эти детали, чтобы при необходимости попробовать другие методы. Иметь записи о процессе установки полезно и в том случае, если потребуется обратиться к кому-нибудь за помощью, например послать сообщение в одну из телеконференций по Linux. А возможно, вы когда-нибудь захотите показать свой дневник внукам.[1]

Принципы создания разделов

Как правило, жесткие диски разбиваются на *разделы* (*partitions*), и каждой операционной системе отводится один или несколько разделов. Например, на одном диске можно иметь несколько разделов: один, скажем, для Windows, другой для OS/2 и еще два для Linux.

Если у вас на компьютере уже установлено какое-то программное обеспечение, вам может потребоваться изменить размеры разделов, чтобы освободить место для Linux. Затем вы создадите на освободившемся месте один или несколько разделов Linux для хранения программного обеспечения Linux и свопинга. Мы называем этот процесс *переразбиением* (*repartitioning*).

Во многих системах Windows использует единственный раздел, занимающий весь диск. В Windows этот раздел называется `C:`. Если разделов несколько, Windows называет их `D:`, `E:` и т. д. В некотором смысле каждый раздел ведет себя как отдельный жесткий диск.

---

[1] Мэтт стыдливо признается, что записывал в дневник обо всех своих неприятностях с Linux в течение первых нескольких месяцев работы с системой. Сейчас дневник пылится на полке.

В первом секторе диска находятся *главная загрузочная запись* (*master boot record*) и *таблица разделов* (*partition table*). Главная загрузочная запись, как и говорит ее название, используется для загрузки системы. Таблица разделов содержит данные о расположении и размерах дисковых разделов.

Есть три вида разделов: *первичный* (*primary*), *расширенный* (*extended*) и *логический* (*logical*). Чаще всего используются первичные разделы. Однако, поскольку размер таблицы разделов ограничен, на каждом жестком диске можно иметь только четыре первичных раздела. Это следствие плохой продуманности MS-DOS и Windows, в других ОС, даже относящихся к той же эпохе, таких ограничений нет.

Обойти ограничение на четыре раздела можно с помощью расширенного раздела. Расширенный раздел сам по себе не содержит никаких данных, а служит «контейнером» для логических разделов. Поэтому можно создать расширенный раздел, охватывающий весь диск, а в нем создать много логических разделов. Однако на каждом диске можно создать только один расширенный раздел.

Требования Linux к дисковым разделам

Прежде чем выяснять, как произвести переразбиение дисков на разделы, нужно получить представление о том, сколько места требуется для Linux. О том, как создавать эти разделы, мы расскажем позднее в разделе «Создание разделов в Linux» главы 3.

В Unix-системах файлы хранятся в *файловой системе*, которая, в сущности, является частью жесткого диска (или другого носителя, например компакт-диска или гибкого диска), отформатированной для хранения файлов. Каждая файловая система связана с конкретной частью дерева каталогов. Например, во многих системах существует файловая система для всех файлов в каталоге */usr*, другая файловая система для каталога */tmp* и т. д. *Корневая* (*root*) файловая система является главной и соответствует самому верхнему каталогу /.

В Linux каждая файловая система существует в отдельном разделе жесткого диска. Например, если есть одна файловая система для каталога / и другая для */usr*, нужны два дисковых раздела для хранения этих двух файловых систем.

Перед установкой Linux нужно подготовить файловые системы для хранения программного обеспечения. У вас должна быть как минимум одна файловая система (корневая), а значит, один раздел диска, отведенный под Linux. Многие пользователи Linux решают хранить все свои файлы в корневой файловой системе, что во многих случаях проще, чем разбираться с несколькими файловыми системами и разделами.

Однако при желании, можно создать для Linux несколько файловых систем. Например, можно использовать разные файловые системы для */usr* и */home*. Читатели, имеющие опыт администрирования Unix-систем, знают, как можно творчески подойти к использованию нескольких файловых систем.

В разделе «Создание файловых систем» главы 6 мы обсуждаем использование нескольких разделов и файловых систем.

Для чего может понадобиться несколько файловых систем? Чаще всего они организуются по соображениям надежности: если по какой-либо причине одна из файловых систем оказывается поврежденной, остальные (обычно) остаются в целости. Напротив, если хранить все файлы в корневой файловой системе, то при ее повреждении теряются сразу все файлы. Однако это случается довольно редко. Если регулярно проводить резервное копирование, то можно чувствовать себя в безопасности.

С другой стороны, использование нескольких файловых систем имеет то преимущество, что можно легко обновить систему, не подвергая опасности свои бесценные данные. Можно завести раздел для личных каталогов пользователей и при обновлении системы, не трогая этого раздела, очистить остальные и установить Linux с нуля. Конечно, в современных дистрибутивах есть развитые процедуры обновления, но иногда возникает желание «начать все сначала».

Несколько файловых систем используют также для того, чтобы распределить дисковую память между несколькими жесткими дисками. Если у вас, скажем, свободны 100 Мбайт на одном жестком диске и 2 Гбайт – на другом, то можно создать 100-мегабайтную корневую систему на первом диске и 2-гигабайтную файловую систему */usr* – на втором диске. Существует также возможность охватить одной файловой системой несколько дисков, для чего используется так называемый администратор логических дисков (*Logical Disk Manager* – LVM), но его сложно настроить, если входящая в дистрибутив программа установки не делает этого автоматически.

Итак, для Linux требуется, по крайней мере, один раздел – для корневой файловой системы. Если вы хотите создать несколько файловых систем, для каждой из дополнительных файловых систем нужен свой раздел на диске. Некоторые дистрибутивы Linux автоматически создают разделы и файловые системы, так что вообще не приходится беспокоиться об этих проблемах.

При планировании разделов следует также подумать о свопинге. *Пространство свопинга* (*swap space*) – это часть диска, используемая системой для временного хранения фрагментов программ, загруженных пользователем, но не используемых в данный момент. Linux не требует обязательного выделения пространства для свопинга, но если объем установленной в машине физической оперативной памяти меньше 64 Мбайт, настоятельно рекомендуется организовать свопинг.

Есть две возможности. Первая – использовать *файл свопинга*, располагаемый в одной из файловых систем Linux. Файл свопинга для использования в качестве виртуальной памяти создается после установки программного обеспечения. Вторая возможность – создать *раздел для свопинга* (*подкачки*) в качестве отдельного раздела, используемого только с этой целью. В большинстве случаев для свопинга используется раздел, а не файл.

Отдельный файл или раздел для свопинга может занимать до 2 Гбайт.[1] При желании использовать для свопинга более 2 Гбайт (что едва ли может потребоваться) можно создать несколько файлов или разделов свопинга, общим количеством до восьми.

О создании раздела подкачки рассказано в разделе «Создание пространства для свопинга» главы 3, а о создании файла подкачки – в разделе «Администрирование пространства для свопинга» главы 6.

Таким образом, для Linux обычно создается как минимум два раздела: один для использования в качестве корневой файловой системы, а другой – для использования в качестве раздела подкачки. Конечно, возможны различные варианты выделения разделов, но этот – минимальный.

Естественно, нужно знать, каким должен быть *размер* этих разделов. Размер файловых систем Linux (содержащих собственно программное обеспечение) сильно зависит от объема устанавливаемого программного обеспечения и используемого дистрибутива Linux. Весьма вероятно, что в документации вашего дистрибутива сказано о примерных размерах необходимого дискового пространства. Небольшая Linux-система может использовать 60 Мбайт или менее. Более крупная система займет от 500 Мбайт до 2 Гбайт или даже больше. Учтите, что помимо пространства, необходимого для самого программного обеспечения, необходимо выделить дополнительное пространство для каталогов пользователей, место для расширения системы в будущем и т. д.

Если создано несколько разделов, то для корневого каталога можно отвести довольно скромное пространство. Достаточно 128 Мбайт, но добавьте к ним еще хотя бы 30–50 Мбайт, если в том же разделе вы храните каталог */var*, как это обычно делается. Напротив, для */usr* вы, вероятно, захотите выделить большой раздел.

Размер раздела подкачки (если вы решите его создать) зависит от того, сколько виртуальной памяти вам требуется. Эмпирическое правило – использовать для свопинга вдвое больший размер, чем объем вашей оперативной памяти. Например, если у вас 64 Мбайт оперативной памяти, для свопинга достаточно раздела в 128 Мбайт. Разумеется, это чистая прикидка: фактический размер необходимого для свопинга пространства зависит от программного обеспечения, с которым вы будете работать. Если у вас достаточно много оперативной памяти (256 Мбайт или больше), свопинг вам, возможно, вообще не нужен.

Ограничения BIOS иногда не позволяют загружаться с разделов, использующих цилиндры с номерами, большими 1023. Поэтому при выделении места на диске для Linux имейте в виду, что в таких разделах может оказаться невозможным размещение корневой файловой системы. *Использовать* разделы с цилиндрами с номерами свыше 1023 можно, но *загрузить* Linux с такого раздела, возможно, не удастся. Этот совет может показаться несвоевре-

[1] Эта величина относится только к процессорам Intel. В других архитектурах она может быть как больше, так и меньше.

менным, но данное обстоятельство следует учитывать при планировании дискового пространства, а сегодня у многих есть диски с числом цилиндров, превышающим 1023. Есть некоторые новые средства, позволяющие обойти это ограничение, но мы рекомендуем пользоваться ими только в случае исключительной необходимости.

Если нужно использовать для корневой системы Linux раздел с номерами цилиндров выше 1023 и нет других вариантов, можно загружать Linux с гибкого диска. На самом деле это не так уж плохо и требует всего нескольких лишних секунд по сравнению с загрузкой с жесткого диска.

Переразбиение жестких дисков

В этом разделе мы расскажем, как изменить размер текущих разделов (если они есть), чтобы освободить место для Linux. Если вы устанавливаете Linux на «чистый» диск, можете пропустить этот раздел и перейти к главе 3.

Обычный способ изменить размер существующего раздела – это удалить его (при этом разрушив все находящиеся в нем данные) и создать заново. Прежде чем изменять размеры разделов, *сделайте резервную копию данных*. После изменения размеров разделов вы сможете восстановить свои программы с резервной копии. Есть, однако, несколько программ для Windows, осуществляющих неразрушающее изменение размеров разделов. Одна из них называется FIPS, и ее можно найти на многих FTP-серверах Linux.

Не забывайте также, что поскольку вы уменьшите свои исходные разделы, для восстановления всех исходных программ может оказаться недостаточно места. В этом случае вам потребуется удалить ненужные программы, чтобы оставшиеся могли поместиться на уменьшившихся разделах.

Для создания и назначения разделов используется программа под названием *fdisk*. В каждой операционной системе используется свой вариант этой программы. Например, в Windows есть команда FDISK. Следует обратиться к документации по операционной системе, которой вы в настоящее время пользуетесь, чтобы получить сведения о переразбиении диска. Ниже мы расскажем, как изменить размеры разделов Windows с помощью *fdisk*, но эти данные легко экстраполировать на другую операционную систему.

В любой ОС программа *fdisk* читает таблицу разделов данного жесткого диска и обрабатывает ее, чтобы добавить или удалить разделы. Однако некоторые версии *fdisk* выполняют дополнительные действия, например добавляют в начало нового раздела данные, позволяющие использовать его определенной ОС. Поэтому обычно следует создавать разделы для некоторой операционной системы только с помощью той версии *fdisk*, которая вместе с ней поставляется. Нельзя создавать разделы Windows с помощью *fdisk* для Linux: созданные таким образом разделы не могут правильно использоваться Windows (на самом деле, если вы хорошо понимаете суть вопроса, то вам, возможно, удастся создать разделы для Windows из Linux, но мы бы не советовали этого делать). Аналогично, *fdisk* для Windows может не распознать разделы Linux. (Не во всех системах эта программа называется *fdisk*, иногда она называется «disk manager» или «volume manager».)

Позднее, в разделе «Создание разделов в Linux» главы 3, мы покажем, как создавать разделы Linux, но пока наша задача – изменить размеры уже существующих.

Пожалуйста, посмотрите документацию по своей текущей ОС, прежде чем переразбивать свой жесткий диск. В данном разделе дан общий обзор процедуры, и есть много тонкостей, о которых мы здесь не говорим. Если вы неправильно произведете переразбиение диска, то можете потерять все имеющиеся программы.

Допустим, что в машине установлен один жесткий диск, целиком выделенный под Windows. Ваш диск, следовательно, состоит из одного раздела Windows, обычно обозначаемого `C:`. Поскольку этот способ переразбиения уничтожит на нем все данные, нужно создать загрузочный «системный диск» Windows, содержащий все необходимое, чтобы запустить *fdisk* и восстановить программное обеспечение с резервной копии по завершении переразбиения.

Часто для этой цели можно использовать инсталляционные диски MS-DOS. Однако если нужно создать собственный системный диск, отформатируйте дискету командой

```
FORMAT /s A:
```

Скопируйте на этот гибкий диск все необходимые утилиты Windows (обычно большую часть программ из каталога */DOS* жесткого диска), а также программы *FORMAT.COM* и *FDISK.EXE*. Теперь вы должны быть в состоянии загрузиться с этой дискеты и начать работу с *fdisk*, выполнив команду

```
FDISK C:
```

Работа с *fdisk* должна быть самоочевидна, но по поводу деталей можно обратиться к документации по Windows. В начале работы с *fdisk* используйте пункт меню, выводящий таблицу разделов, и запишите результаты. Важно сохранить записи о первоначальной установке на тот случай, если вы решите вернуться к состоянию до установки Linux.

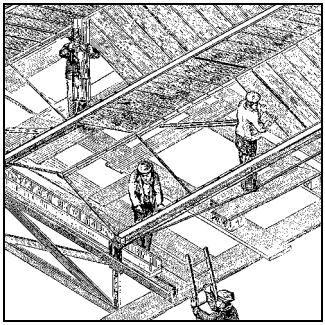
Чтобы удалить существующий раздел, выберите в меню пункт «`Delete an MS-DOS Partition or Logical DOS Drive`». Укажите тип раздела, который вы хотите удалить (первичный, расширенный или логический) и номер раздела. Внимательно прочтите все предупреждения и – вперед!

Чтобы создать новый (меньший) раздел, выберите в меню пункт «`Create an MS-DOS Partition or Logical DOS Drive`». Укажите тип создаваемого раздела (первичный, расширенный или логический) и его размер в мегабайтах. *fdisk* должен создать раздел.

Закончив работу с *fdisk*, выйдите из программы и переформатируйте новые разделы. Например, если вы изменили размер первого раздела DOS на диске (`C:`), нужно выполнить команду:

```
FORMAT /s C:
```

Теперь можно восстановить исходные программы с резервной копии.



3

Установка и начальное конфигурирование

К этому моменту у вас должен быть дистрибутив Linux и выделенное для Linux дисковое пространство. В этой главе мы дадим общий обзор процесса установки. В каждом дистрибутиве есть свои инструкции по установке, но представленные здесь сведения должны дать вам возможность установить любой дистрибутив. Если вы окажетесь в полной растерянности, то в приложении А перечислены источники, из которых можно получить сведения по установке и другую помощь.

Различные дистрибутивы Linux записывают файлы в разные каталоги, что затрудняет описание администрирования Linux. Например, в Red Hat, SuSE и Debian могут быть одни и те же файлы, но в одном случае в каталоге */etc*, а в другом – в каталоге */sbin*. Постепенно поставщики стандартизируют список каталогов в соответствии с документом, носящим название Filesystem Hierarchy Standard (FHS, стандарт иерархии файловой системы), но в данной книге мы постараемся справиться с остающимися расхождениями, перечислив расположение наиболее важных файлов в проверенных нами версиях основных дистрибутивов.

Установка программного обеспечения Linux

После изменения размеров разделов с целью выделения пространства под Linux можно приступать к установке программного обеспечения. Вот краткое описание этой процедуры:

1. Загрузитесь с установочного диска Linux.
2. Запустите в Linux программу *fdisk*, чтобы создать разделы Linux.
3. Запустите программы *mke2fs* и *mkswap*, чтобы создать файловые системы Linux и пространство для свопинга.
4. Установите программное обеспечение Linux и сконфигурируйте его.
5. Наконец, установите на жесткий диск начальный загрузчик LILO или создайте загрузочную дискету для Linux.

Как уже говорилось, некоторые из этих шагов могут автоматически выполняться процедурой установки вашего дистрибутива (или быть объединены с ней). Соответствующие инструкции можно найти в документации дистрибутива.

Начальная загрузка Linux

Сначала нужно загрузиться с инсталляционного носителя Linux. В большинстве случаев это загрузочная дискета с маленькой системой Linux или загрузочный компакт-диск. При загрузке с дискеты или компакт-диска выводится то или иное меню, позволяющее пошагово провести установку. В некоторых дистрибутивах при загрузке с дискеты выводится приглашение ввести регистрационное имя. Для того чтобы начать установку, обычно нужно зарегистрироваться как пользователь `root` или `install`.

В поступающей с вашим дистрибутивом документации должно быть сказано, как загрузиться с инсталляционного носителя.

В большинстве дистрибутивов используется загрузочная дискета, которая позволяет в ответ на приглашение ввести параметры, инициирующие поиск различных устройств. Например, если при загрузке с дискеты не был обнаружен ваш контроллер SCSI, нужно загрузиться заново и в ответ на приглашение указать адрес ввода/вывода и номер прерывания IRQ. Или, например, машины IBM PS/1, ThinkPad и ValuePoint не запоминают в CMOS геометрию дисков, поэтому ее нужно задавать при загрузке.

Загрузочное приглашение часто автоматически выводится при загрузке с дискеты или компакт-диска. Так происходит, например, в дистрибутиве Red Hat. Если дистрибутив не выводит приглашение, то во время загрузки нужно удерживать в нажатом состоянии клавиши <Shift> или <Ctrl> либо нажать <Scroll Lock>, чтобы ввести параметры в ответ на загрузочное приглашение. В случае успеха выводится приглашение

```
boot:
```

и, возможно, другие сообщения. То, что вы при этом видите, это приглашение, выводимое LILO (LInux LOader) – программой, использующейся для загрузки ОС Linux и задания параметров обнаружения оборудования при загрузке. Установив Linux, вы можете установить себе на жесткий диск LILO, что позволит выбирать Linux или другую ОС (например, Windows) во время загрузки системы.

Дальше возникает несколько возможностей. Можно нажать <Enter> и просто загрузить Linux с гибкого диска без каких-либо параметров. (Сначала так и надо поступить, и, возможно, установка пройдет гладко. Если из внешней памяти у вас есть только жесткий диск IDE и CD-ROM, скорее всего, вам ничего не придется задавать.) Либо можно просто подождать продолжения процесса установки: в современных дистрибутивах устанавливается тайм-аут, по истечении которого загрузка продолжается, если вы ничего не ввели. Если вы не уверены в том, что нужно ввести, нажмите любую клавишу (например, пробел); в результате тайм-аут будет отменен, и у вас появится время, чтобы подумать.

Если не удается нормально запустить Linux без указания параметров, может потребоваться ввести в ответ на приглашение начальной загрузки параметры для обнаружения аппаратных устройств, чтобы заставить систему правильно идентифицировать установленные в вашей машине устройства. Но общее правило следующее: используйте сначала значения по умолчанию и посмотрите, не будет ли этого достаточно. Лишь в том случае, если они не сработают, нужно начинать подбирать параметры. Однако, скорее всего, это не потребуется.

Если на этом этапе вы не хотите использовать какие-либо параметры определения аппаратуры, в ответ на загрузочное приглашение нажмите <Enter>. Следите за сообщениями, выводимыми системой во время начальной загрузки. Если, например, у вас есть контроллер SCSI, вы должны увидеть список обнаруженных хостов SCSI. Если вы увидите сообщение:

```
SCSI: 0 hosts
```

это означает, что ваш SCSI-контроллер не обнаружен и вам придется использовать процедуру обнаружения аппаратных устройств, которую мы вскоре опишем.

Во многих новых дистрибутивах часто используется другой способ выбора аппаратуры. На загрузочном диске находится минимальное ядро, а со второй дискеты или компакт-диска загружаются так называемые *модули ядра* (*kernel modules*). В этом случае вам, вероятно, будет предложено некоторое меню, в котором можно выбрать дополнительные модули для проверки. Даже задание модулей в значительной мере автоматизировано: вы просто просите программу установки проверить SCSI-адаптеры и смотрите, находит ли она тот, который у вас установлен. То же относится к картам Ethernet и другим устройствам, которые необходимы для процесса установки. Устройства, которые при установке не нужны, например звуковые платы, на этом этапе, скорее всего, не будут обнаружены, и возможность сконфигурировать их будет дана вам позднее. Здесь также следует сначала проверить, не будут ли работать значения по умолчанию.

Если процедуры автоматического обнаружения аппаратуры окажутся безуспешными (что обычно случается, только если у вас очень старое, очень новое или очень редкое оборудование), вам придется немного помочь Linux, инициировав принудительное обнаружение аппаратуры.

Для принудительного обнаружения устройств нужно ввести соответствующие параметры в ответ на загрузочное приглашение, используя следующий синтаксис:

```
linux parameters
```

Таких параметров существует много, и некоторые из них перечислены ниже. Не требуется, чтобы вы понимали смысл или назначение всех их, нужно лишь уметь определить, какие параметры соответствуют вашей системе. Мы приводим здесь сразу весь достаточно полный список, который может вам позднее пригодиться.

Например, если у вас есть SCSI-контроллер типа AHA152x и вы знаете, что в Windows нужно сконфигурировать эту плату на определенные адрес I/O и IRQ, можно использовать соответствующий параметр (`aha152x=`). На практике многие из этих параметров загрузки при начальной установке просто не нужны.

Еще один совет: запишите и запомните параметры загрузки, которые вы используете для запуска системы. После установки Linux нужно будет использовать те же самые параметры, чтобы при каждой загрузке ваша система правильно определяла имеющиеся устройства. Если установить на жесткий диск загрузчик LILO, то можно сконфигурировать его так, чтобы эти параметры не приходилось вводить каждый раз вручную.

`nosmp`

Сообщает ядру, настроенному для симметричной многопроцессорной обработки (для нескольких ЦПУ), о необходимости выполняться в режиме однопроцессорного ядра.

`root=`*`device`*

Указывает устройство, используемое в качестве корневой файловой системы при начальной загрузке. При первоначальной установке использовать не следует. После установки можно использовать для переопределения расположения корневой файловой системы Linux.

`ro`

Монтирует корневую файловую систему в режиме только для чтения. Используется при обслуживании системы.

`lock`

Сохраняет параметры загрузки на будущее, так что их не придется вводить при каждой загрузке системы.

`rw`

Монтирует корневую файловую систему в режиме для чтения и записи. Используется при обслуживании системы.

`debug`

Требует от ядра вывода на консоль текстовых отладочных сообщений во время работы системы.

`ramdisk=`*`kilobytes`*

Требует от системы зарезервировать указанное количество килобайт памяти для электронного диска. Часто используется установочными загрузочными дискетами для загрузки в память всего образа файловой системы. При установке этот параметр не нужен, но с его помощью можно потом поэкспериментировать с электронными дисками.

`mem=`*`size`*

Во многих ПК системный BIOS сообщает не более чем о 64 Мбайт установленной оперативной памяти. Linux использует эти данные для определения размера установленной памяти. Если у вас больше 64 Мбайт памяти

и старое ядро (версии ниже 2.2), может понадобиться задать этот параметр, чтобы использовать всю память. Здесь параметр *size* – это число с окончанием k или M; например mem=96M указывает, что в системе установлено 96 Мбайт RAM. Если вы зададите больше памяти, чем реально есть в системе, произойдут «нехорошие вещи».

hd=*cylinders*,*heads*,*sectors*

Задает геометрию диска для приводов IDE и стандарта ST-506 (не SCSI). Требуется на таких системах, как IBM PS/1, ValuePoint и старых ThinkPad. Например, если у вашего диска 683 цилиндра, 16 головок и 32 сектора на дорожке, задайте следующие значения для этого параметра:

```
ramdisk hd=683,16,32
```

Можно использовать этот параметр в виде hda=, hdb=, hdc= или hdd= для описания геометрии конкретного диска IDE. Параметр hd= может быть обязательным на некоторых старых машинах для большого диска IDE с числом цилиндров, большим 1024. Если у Linux возникнут проблемы с распознанием геометрии вашего диска (о чем вы узнаете при попытке создать разделы для Linux), попробуйте использовать этот параметр.

max_scsi_luns=*num*

Если *num* равен 1, система не будет пытаться определить SCSI-устройства с логическим номером (LUN), отличным от нуля. Этот параметр требуется для некоторых плохо спроектированных SCSI-устройств, которые блокируются при проверке с LUN, отличным от 0. Обратите внимание, что этот параметр не имеет отношения к ID SCSI-устройства. LUN позволяет обращаться к нескольким логическим устройствам или подустройствам одного устройства SCSI, таким как диск.

aha152x=*iobase*,*irq*,*scsi-id*,*reconnect*,*parity*

Задает параметры для интерфейсов Adaptec AHA151x, AHA152x, AIC6260, AIC6230 и SB16-SCSI. *iobase* должен указываться в шестнадцатеричном виде, например 0x340. Все аргументы, кроме *iobase*, необязательны.

aha1542=*iobase*

Задает в шестнадцатеричном виде адрес I/O для SCSI-интерфейсов Adaptec AHA154x.

aic7xxx=*extended*,*no-reset*

Задает параметры для интерфейсов Adaptec AHA274x, AHA284x и AIC7xxx SCSI. Ненулевое значение *extended* разрешает расширенное транслирование для больших дисков. Если значение *no-reset* отлично от 0, драйвер не будет сбрасывать шину SCSI при конфигурировании адаптера во время загрузки.

buslogic=*iobase*

Задает в шестнадцатеричном виде адрес I/O для SCSI-интерфейсов Buslogic.

`tmc8xx=mem-base,irq`

Задает в шестнадцатеричном виде адрес отображаемой в память области ввода/вывода и IRQ для SCSI-интерфейсов Future Domain TMC-8xx и TMC-950.

`pas16=iobase,irq`

Задает в шестнадцатеричном виде адрес ввода/вывода и IRQ для SCSI-интерфейсов Pro Audio Spectrum.

`st0x=mem-base,irq`

Задает в шестнадцатеричном виде адрес отображаемой в память области ввода/вывода и IRQ для SCSI-интерфейсов Seagate ST-0x.

`t128=mem-base,irq`

Задает в шестнадцатеричном виде адрес отображаемой в память области ввода/вывода и IRQ для SCSI-интерфейсов Trantor T128.

`aztcd=iobase`

Задает в шестнадцатеричном виде адрес ввода/вывода для интерфейсов CD-ROM Aztech.

`cdu31a=iobase,irq,pas`

Задает в шестнадцатеричном виде адрес ввода/вывода и IRQ для интерфейсов Sony CD-ROM CDU-31A и CDU-33A. Эти параметры используются некоторыми звуковыми картами Pro Audio Spectrum и платами Sony. Параметры *irq* и *pas* необязательны. Если плата не поддерживает прерывания, то *irq*=0 (относится к некоторым платам). Единственное допустимое значение для параметра *pas* — это PAS, что указывает на использование карты Pro Audio Spectrum.

`soncd535=iobase,irq`

Задает в шестнадцатеричном виде адрес ввода/вывода и IRQ (необязателен) для интерфейсов Sony CDU-535.

`gscd=iobase`

Задает в шестнадцатеричном виде адрес ввода/вывода для интерфейсов GoldStar CD-ROM.

`mcd=iobase,irq`

Задает в шестнадцатеричном виде адрес ввода/вывода и IRQ (необязателен) для стандартных интерфейсов CD-ROM Mitsumi.

`optcd=iobase`

Задает в шестнадцатеричном виде адрес ввода/вывода для интерфейсов CD-ROM Optics Storage Interface.

`cm206=iobase,irq`

Задает в шестнадцатеричном виде адрес ввода/вывода и IRQ для интерфейсов CD-ROM Philips CM206.

`sjcd=iobase,irq,dma`

Задает в шестнадцатеричном виде адрес ввода/вывода, IRQ и канал DMA для интерфейсов Sanyo CD-ROM. Параметры *irq* и *dma* необязательны.

`sbpcd=iobase,type`

Задает в шестнадцатеричном виде адрес ввода/вывода для интерфейсов CD-ROM SoundBlaster Pro и совместимых. Параметр *type* может иметь значение `SoundBlaster`, `LaserMate` или `SPEA` в зависимости от того, какая у вас плата. Учтите, что это параметр *только* для интерфейса CD-ROM, а не звуковых устройств на плате. Применяется к очень старым приводам CD-ROM (до ATAPI) и не требуется большинству пользователей.

`ether=irq,iobase,parameters...`

Задает IRQ и адрес I/O для плат Ethernet. Если возникают проблемы с определением карты Ethernet и она участвует в процессе установки (например, при установке через FTP или NFS), почитайте Linux Ethernet HOWTO, где различные загрузочные параметры для карт Ethernet описаны подробнее. Их слишком много, чтобы описывать их тут полностью.

`floppy=thinkpad`

Сообщает драйверу гибких дисков, что у вас старая машина ThinkPad; необходим для доступа к гибким дискам на старых машинах ThinkPad.

`floppy=0,thinkpad`

Сообщает драйверу гибких дисков, что ваша машина не ThinkPad, если он ошибается.

`bmouse=irq`

Задает IRQ для интерфейса шинной мыши.[1]

`msmouse=irq`

Задает IRQ для интерфейса шинной мыши Microsoft.

Есть еще много других параметров – здесь описаны лишь те, которые необходимы для нормальной работы системы. (Например, мы опустили многочисленные параметры для драйверов звуковых карт; настоятельно рекомендуем прочесть соответствующие HOWTO, если использование звуковой карты является для вас вопросом жизни и смерти.)

Для использования этих параметров нужно ввести *linux*, а затем – требуемые вам параметры.

Если у вас возникли вопросы по поводу использования этих параметров загрузки, почитайте LinuxBootprompt HOWTO, Linux SCSI HOWTO и Linux CD-ROM HOWTO. Эти три документа должны быть на всех FTP-серверах Linux (и на большинстве компакт-дисков), и в них более подробно описаны параметры загрузчика LILO.

---

[1] Шинная мышь (busmouse) – это мышь, подключаемая к системной шине, а не к последовательному порту или порту мыши типа PS/2.

Диски и дисковые разделы в Linux

Во многих дистрибутивах Linux требуется вручную создать разделы Linux с помощью программы *fdisk*. Другие дистрибутивы создают разделы автоматически. Ниже приводятся сведения о разделах Linux и именах устройств, которые необходимо знать в любом случае. (Они относятся только к системам Intel и Alpha, загружаемым через AlphaBIOS; в других системах, таких как PowerPC, SPARC и m68k, нет логических и расширенных разделов.)

Диски и разделы в Linux именуются не так, как в других операционных системах. В Windows гибкие диски имеют имена `A:` и `B:`, а разделы жестких дисков называются `C:`, `D:` и т. д. В Linux используются совершенно иные соглашения об именах.

Драйверы устройств, находящиеся в каталоге */dev*, применяются для обмена данными с устройствами системы, такими как жесткие диски, мышь и т. д. Например, если в системе есть мышь, доступ к ней происходит через драйвер */dev/mouse*. Дисководы для гибких дисков, жесткие диски и отдельные разделы имеют собственные драйверы устройств. Сейчас вас не должен беспокоить интерфейс между устройством и драйвером; чтобы использовать различные устройства, важно лишь понимать, как они именуются. В разделе «Файлы устройств» главы 6 об устройствах рассказано подробнее.

В табл. 3.1 перечислены имена различных драйверов устройств, причем разные имена могут создаваться с помощью возрастающих номеров (0, 1 и т. д.). В качестве примера в таблице приведено несколько имен.

Таблица 3.1. Имена разделов Linux

Устройство	Имя
Первый гибкий диск (A:)	*/dev/fd0*
Второй гибкий диск (B:)	*/dev/fd1*
Первый жесткий диск (весь диск) или CD-ROM	*/dev/hda*
Первый жесткий диск, первичный раздел 1	*/dev/hda1*
Первый жесткий диск, первичный раздел 2	*/dev/hda2*
Первый жесткий диск, первичный раздел 3	*/dev/hda3*
Первый жесткий диск, первичный раздел 4	*/dev/hda4*
Первый жесткий диск, логический раздел 1	*/dev/hda5*
Первый жесткий диск, логический раздел 2	*/dev/hda6*
⋮	
Второй жесткий диск (весь диск) или CD-ROM	*/dev/hdb*
Второй жесткий диск, первичный раздел 1	*/dev/hdb1*
⋮	
Первый жесткий диск SCSI (весь диск)	*/dev/sda*

Устройство	Имя
Первый жесткий диск SCSI, первичный раздел 1	*/dev/sda1*
⋮	
Второй жесткий диск SCSI (весь диск)	*/dev/sdb*
Второй жесткий диск SCSI, первичный раздел 1	*/dev/sdb1*
⋮	
Первый SCSI CD-ROM	*/dev/scd0*
Второй SCSI CD-ROM	*/dev/scd1*
⋮	
Первое родовое устройство SCSI (например, сканер, пишущий CDR и т. д.). Обратите внимание, что в новых системах используются цифры вместо букв, например */dev/sg0* вместо */dev/sga*	*/dev/sga*
Второе родовое устройство SCSI	*/dev/sgb*
⋮	

Несколько замечаний по поводу этой таблицы: */dev/fd0* соответствует первому гибкому диску (A: в Windows), а */dev/fd1* соответствует второму гибкому диску (B:).

Кроме того, жесткие диски SCSI именуются иначе, чем другие диски. К дискам IDE, MFM и RLL доступ организован через устройства */dev/hda*, */dev/hdb* и т. д. Отдельные разделы на диске */dev/hda* суть */dev/hda1*, */dev/hda2* и т. д. То же справедливо для приводов CD-ROM ATAPI и IDE. Однако диски SCSI называются */dev/sda*, */dev/sdb* и т. д., а разделы на них – */dev/sda1* и */dev/sda2*.

Конечно, в большинстве систем нет четырех первичных разделов. Но имена от */dev/hda1* до */dev/hda4* для них все равно зарезервированы, и их нельзя использовать для именования логических разделов.

Приведем пример. Допустим, что есть единственный жесткий диск IDE с тремя первичными разделами. Первые два раздела выделены под Windows, а третий является расширенным разделом, в котором есть два логических. Названия устройств для этих разделов будут такими:

Устройство	Имя
Первый раздел Windows (C:)	*/dev/hda1*
Второй раздел Windows (D:)	*/dev/hda2*
Расширенный раздел	*/dev/hda3*
Первый логический раздел Linux	*/dev/hda5*
Второй логический раздел Linux	*/dev/hda6*

Обратите внимание, что имя */dev/hda4* пропущено: оно соответствует четвертому первичному разделу, которого в этом примере нет. Логические разделы именуются последовательно, начиная с */dev/hda5*.

Создание разделов в Linux

Теперь вы готовы к тому, чтобы создать разделы Linux с помощью команды *fdisk*. Обычно нужно создать хотя бы один раздел собственно для Linux и еще один раздел для свопинга.

Мы опишем основной текстовый режим *fdisk*, который должен быть доступен в любом дистрибутиве. Во многих дистрибутивах теперь есть более дружественный интерфейс с *fdisk*. Хотя он и не такой гибкий, как «чистый» *fdisk*, но может помочь вам сделать правильный выбор. Каким бы средством вы ни пользовались, этот раздел поможет вам понять основополагающие вещи. В конечном итоге, все программы делают примерно одно и то же, только в одних пилюля подслащена больше, чем в других. Приводимые ниже сведения можно также использовать для исправления или проверки, если при работе с графическим средством возникли подозрения, что не все сделано правильно.

После загрузки с инсталляционного носителя запустите *fdisk* путем ввода:

```
fdisk drive
```

где `drive` – это имя устройства, к которому вы хотите добавить разделы (см. табл. 3.1). Например, если вы хотите запустить *fdisk* для первого SCSI-диска своей системы, введите команду:

```
# fdisk /dev/sda
```

Если не указать диск, то значением по умолчанию будет */dev/hda* (первый диск IDE).

Если вы создаете разделы Linux на нескольких дисках, запустите *fdisk* по одному разу для каждого диска:

```
# fdisk /dev/hda

Command (m for help):
```

Теперь *fdisk* ждет команду; можно ввести `m`, чтобы получить список возможных команд:

```
Command (m for help): m
Command action
   a   toggle a bootable flag          переключить флаг bootable (загрузочный)
   d   delete a partition              удалить раздел
   l   list known partition types      вывести список известных типов разделов
   m   print this menu                 вывести это меню
   n   add a new partition             добавить раздел
   p   print the partition table       вывести таблицу разделов
   q   quit without saving changes     выйти без сохранения изменений
   t   change a partition's system id  изменить системный id раздела
```

```
   u   change display/entry units          изменить отображаемые единицы
   v   verify the partition table          проверить таблицу разделов
   w   write table to disk and exit        сохранить таблицу на диске и выйти
   x   extra functionality (experts only)  дополнительные функции (только
                                           для опытных пользователей)

Command (m for help):
```

Команда n используется для создания нового раздела. Большая часть остальных команд вам не понадобится. Чтобы завершить работу *fdisk* без сохранения изменений, воспользуйтесь командой q. Чтобы выйти из *fdisk* и записать измененную таблицу разделов на диск, воспользуйтесь командой w. Стоит повторить: если вы закончите работу командой q без записи, можете делать в *fdisk* что хотите, не рискуя потерять данные. Только при вводе w может произойти катастрофа, если вы в чем-то ошибетесь.

Первым делом нужно вывести текущую таблицу разделов и записать результат, который может в дальнейшем понадобиться. Это можно сделать с помощью команды p. Неплохо записывать в тетрадь каждое изменение, проведенное в таблице разделов. Если по какой-либо причине таблица разделов будет повреждена, данные на диске станут недоступными, несмотря на то, что они останутся на месте. Но с помощью своих записей во многих случаях можно восстановить таблицу разделов и вернуть данные, снова запустив *fdisk* и удалив и воссоздав разделы с теми параметрами, которые вы записали ранее. Не забудьте по окончании работы сохранить восстановленную таблицу разделов.

Вот пример вывода таблицы разделов с указанием блоков, секторов и цилиндров, описывающих структуру диска:

```
Command (m for help): p

Disk /dev/hda: 16 heads, 38 sectors, 683 cylinders
Units = cylinders of 608 * 512 bytes
   Device   Boot   Begin   Start      End     Blocks    Id    System
/dev/hda1      *       1       1      203      61693     6    DOS 16-bit >=32M

Command (m for help):
```

В этом примере есть один раздел Windows на */dev/hda1*, в котором 61 693 блока (около 60 Мбайт).[1] Раздел начинается с цилиндра 1 и заканчивается цилиндром 203. Всего на этом диске 683 цилиндра, поэтому остается 480 цилиндров, на которых можно создать разделы Linux.

Чтобы создать новый раздел, воспользуйтесь командой n. В этом примере мы создадим для Linux два новых первичных раздела (*/dev/hda2* и */dev/hda3*):

```
Command (m for help): n
Command action
  e   extended
  p   primary partition (1-4)
  p
```

[1] В Linux блок составляют 1024 байта.

Здесь *fdisk* спрашивает, какого типа раздел нужно создать – расширенный или первичный. В нашем примере создаются только первичные разделы, поэтому мы выбираем p:

```
Partition number (1-4):
```

Затем *fdisk* спрашивает номер создаваемого раздела; поскольку раздел 1 уже есть, первый раздел Linux будет иметь номер 2:

```
Partition number (1-4): 2
First cylinder (204-683):
```

Теперь мы укажем номер начального цилиндра раздела. Поскольку не использованы цилиндры с 204 по 683, мы используем первый свободный цилиндр (с номером 204). Нет оснований оставлять между разделами пустое пространство:

```
First cylinder (204-683): 204
Last cylinder or +size or +sizeM or +sizeK (204-683):
```

fdisk запрашивает размер создаваемого раздела. Можно задать номер последнего цилиндра либо размер в байтах, килобайтах или мегабайтах. Поскольку мы хотим создать раздел размером 80 Мбайт, укажем +80M. При таком задании размера раздела *fdisk* округляет фактический размер раздела до ближайшего числа цилиндров:

```
Last cylinder or +size or +sizeM or +sizeK (204-683): +80M

Warning: Linux cannot currently use 33090 sectors of this partition
```
(Предупреждение: Linux в настоящее время не может использовать 33 090 секторов этого раздела.)

Если вы увидите такое сообщение, можете его проигнорировать. *fdisk* выводит его, потому что это старая программа из тех времен, когда разделам Linux не позволялось быть больше 64 Мбайт.

Теперь можно создать второй раздел Linux. В целях наглядности сделаем его размером в 10 Мбайт:

```
Command (m for help): n
Command action
   e   extended
   p   primary partition (1-4)
p
Partition number (1-4): 3
First cylinder (474-683): 474
Last cylinder or +size or +sizeM or +sizeK (474-683): +10M
```

Наконец, выведем таблицу разделов. Опять-таки, запишем все эти данные, особенно число блоков в новых разделах. Размеры разделов понадобятся при создании файловых систем. Проверьте также, чтобы разделы не перекрывали друг друга:

```
Command (m for help): p

Disk /dev/hda: 16 heads, 38 sectors, 683 cylinders
```

```
Units = cylinders of 608 * 512 bytes
   Device   Boot   Begin   Start     End    Blocks     Id    System
/dev/hda1     *       1       1     203     61693      6    DOS 16-bit >=32M
/dev/hda2           204     204     473     82080     83    Linux native
/dev/hda3           474     474     507     10336     83    Linux native
```

Как видно, */dev/hda2* является теперь разделом размером 82 080 блоков (что соответствует примерно 80 Мбайт), а */dev/hda3* содержит 10 336 блоков (около 10 Мбайт).

Обратите внимание, что в большинстве дистрибутивов необходимо выполнить в *fdisk* команду `t`, чтобы изменить тип раздела для свопинга на «Linux swap», который имеет номер 82. С помощью команды `L` можно вывести список кодов известных типов разделов, а затем командой `t` установить тип раздела для свопинга в тот, который соответствует «Linux swap».

Благодаря этому программа установки сможет автоматически найти разделы для свопинга по типу. Если окажется, что программа установки не может найти раздел для свопинга, нужно снова запустить *fdisk* и установить тип нужного раздела командой `t`.

В нашем примере оставшиеся на диске цилиндры с номерами от 508 до 683 не используются. Можно оставить на диске свободное пространство, чтобы позднее создать дополнительные разделы.

Наконец, с помощью команды `w` запишем изменения на диск и выйдем из *fdisk*:

```
Command (m for help): w
#
```

Запомните, что все изменения, которые вы производите во время работы с *fdisk*, вступают в силу только после выполнения команды `w`, поэтому можно попробовать другие конфигурации и только в конце сохранить их. Можно также в любой момент прекратить работу с *fdisk* без сохранения изменений с помощью команды `q`. Не забывайте, что нельзя изменять разделы других операционных систем с помощью программы *fdisk* для Linux.

Загрузка Linux с раздела, в котором есть цилиндры с номерами, большими 1023, может оказаться невозможной. Поэтому постарайтесь создать корневой раздел Linux в диапазоне цилиндров ниже 1024, что почти всегда можно сделать (например, создав маленький корневой раздел на цилиндрах с номерами до 1024). Если по какой-либо причине вы не можете или не хотите этого сделать, можно просто загружать Linux с гибкого диска.

В некоторых дистрибутивах Linux требуется перезагрузить систему после работы *fdisk*, чтобы изменения в таблице разделов вступили в силу до установки программного обеспечения. Более новые версии *fdisk* автоматически обновляют в ядре данные о разделах, и перезагрузка не нужна. Для надежности лучше после работы с *fdisk* перезагрузиться с инсталляционного носителя и только тогда продолжать работу.

Создание пространства для свопинга

Если вы собираетесь использовать для виртуальной памяти раздел свопинга, теперь можно его подготовить.[1] В разделе «Управление пространством для свопинга» главы 6 мы расскажем, как подготовить своп-файл, если вы не хотите использовать отдельный раздел.

Во многих дистрибутивах перед установкой программного обеспечения нужно создать и активировать пространство для свопинга. Если у вас мало оперативной памяти, установка может не выполниться, если не выделить какой-то объем пространства под свопинг.

Подготовка пространства для свопинга осуществляется командой *mkswap*, которая имеет следующий формат:

```
mkswap -c partition
```

где *partition* — имя раздела для свопинга. Например, если для свопинга выделен раздел */dev/hda3*, нужно выполнить команду

```
# mkswap -c /dev/hda3
```

В старых версиях *mkswap* требовалось указать размер раздела, что было опасно, поскольку один неверно введенный знак мог разрушить диск логически.

Параметр *-c* требует, чтобы *mkswap* при создании пространства для свопинга проверила наличие в разделе дефектных боков, т. е. участков магнитного носителя, которые не могут правильно хранить данные. На современных дисках это бывает редко, но если такие блоки есть и вы об этом не знаете, могут возникнуть бесчисленные неприятности. Всегда используйте параметр *-c*, чтобы проверить диск на наличие плохих блоков. Они будут автоматически исключены из использования.

Если для свопинга отведено несколько разделов, команду *mkswap* нужно выполнить с каждым из них.

После форматирования пространства для свопинга нужно дать системе возможность его использовать. Обычно система автоматически активирует пространство для свопинга при загрузке, однако, поскольку вы еще не установили Linux, это нужно сделать вручную.

Команда, активирующая пространство для свопинга, называется *swapon*. Она имеет следующий формат:

```
swapon partition
```

После выполнения команды *mkswap* активируйте пространство для свопинга на разделе */dev/hda3* следующей командой:

```
# swapon /dev/hda3
```

---

[1] Опять-таки, некоторые дистрибутивы Linux создают пространство для свопинга автоматически или через пункт меню при установке.

Создание файловых систем

Для того чтобы записывать файлы в разделы Linux, нужно создать на них файловые системы. Создание файловой системы аналогично форматированию раздела в Windows или другой операционной системе. Мы кратко обсуждали файловые системы в разделе «Требования к разделам Linux» главы 2.

В Linux поддерживается несколько типов файловых систем. У каждой файловой системы свой формат и набор характеристик, таких как длина имени файла, максимальный размер файла и т. д. Linux поддерживает также несколько типов файловых систем других разработчиков, например файловую систему Windows.

Чаще всего используется *«вторая расширенная файловая система» – Second Extended Filesystem*, или *ext2fs*. *ext2fs* является одной из наиболее эффективных и гибких файловых систем; она допускает имена файлов длиной до 256 символов и размер файлов до 32 терабайт. В разделе «Типы файловых систем» главы 6 мы расскажем о различных типах файловых систем, доступных в Linux. Для начала же мы предлагаем использовать файловую систему *ext2fs*.

Создать файловую систему *ext2fs* можно командой

```
mke2fs -c partition
```

где `partition` – это имя раздела. Например, чтобы создать файловую систему в разделе */dev/hda2*, нужно выполнить команду

```
# mke2fs -c /dev/hda2
```

Чтобы использовать в Linux несколько файловых систем, нужно для каждой из них выполнить соответствующую команду *mke2fs*.

Если в этом месте у вас возникли проблемы, см. раздел «Устранение неполадок» далее в этой главе.

Установка программного обеспечения

Теперь вы можете установить на своем компьютере программное обеспечение. В каждом дистрибутиве для этого существуют свои методы. Во многих дистрибутивах есть специальная программа, которая проведет вас через все этапы установки. В других дистрибутивах нужно *монтировать* (*mount*) ваши файловые системы в определенном каталоге (например, */mnt*) и копировать в них программное обеспечение вручную. Дистрибутивы на CD-ROM предоставляют возможность установить часть программного обеспечения на жесткий диск и оставить большую часть на CD-ROM. Часто это называется «живой файловой системой» (live filesystem). Такая файловая система удобна для того, чтобы испытать работу в Linux, прежде чем решиться полностью установить ее на диск.

Некоторые дистрибутивы предлагают несколько способов установки программного обеспечения. Например, может быть предложено установить программное обеспечение прямо с раздела Windows на жестком диске, а не с

дискет, или даже провести установку по сети TCP/IP через FTP или NFS. Посмотрите подробности в документации к дистрибутиву.

Дистрибутив Slackware, например, требует следующих действий:

1. Создать разделы с помощью *fdisk*.
2. Создать, по желанию, пространство для свопинга с помощью *mkswap* и *swapon* (если у вас меньше 16 Мбайт RAM).
3. Запустить программу *setup* для установки программного обеспечения. *setup* проводит вас по понятной системе меню.

Конкретный способ установки программного обеспечения Linux сильно зависит от дистрибутива.

Разнообразие устанавливаемого программного обеспечения может показаться ошеломляющим. Современные дистрибутивы Linux вполне могут содержать тысячи пакетов, разбросанных по нескольким CD-ROM. Есть три основных метода отбора программных пакетов:

Отбор по назначению машины

Для новичков это самый простой способ отбора. Не нужно думать о том, требуется ли вам тот или иной пакет, просто решите, должна ли ваша Linux-машина выступать в роли рабочей станции, машины разработчика или сетевого маршрутизатора, и программа установки подберет для вас нужные пакеты. В любом случае вы можете потом уточнить свой выбор вручную или снова вернуться к программе установки.

Отбор пакетов по сериям

При таком механизме отбора все пакеты группируются по сериям, например «сетевое взаимодействие», «разработка программ» или «графика». Вы можете пробежаться по всем сериям и выбрать в них отдельные пакеты. При этом приходится принимать больше решений, чем при выборе по назначению машины, поскольку для каждого пакета нужно решать, нужен он вам или нет, но можно опустить целую серию, если вы уверены, что предлагаемые ею функции вам не нужны.

Выбор пакетов, отсортированных в алфавитном порядке

Этот способ полезен, только если вы заранее знаете, какие пакеты нужно установить. В противном случае вы не увидите леса из-за деревьев.

Выбор одного из этих методов не исключает использования остальных. В большинстве дистрибутивов предлагаются два или более из вышеупомянутых механизмов отбора.

Все же может оказаться затруднительным выбрать нужные пакеты. В хороших дистрибутивах на экран выводится краткое описание каждого пакета, что облегчает выбор, но если вы не чувствуете уверенности, то наш совет таков: если есть сомнения, оставьте пакет в покое! Позднее вы всегда сможете вернуться и добавить его.

В современных дистрибутивах есть отличная функция, так называемое *отслеживание зависимостей*. Некоторые пакеты работают только тогда, когда

установлены некоторые другие пакеты (например, средству просмотра графики могут понадобиться специальные графические библиотеки, из которых импортируются файлы). При отслеживании зависимостей программа установки может сообщать о таких зависимостях, позволяя автоматически выбирать нужный вам пакет вместе с теми, от которых он зависит. Если только вы не уверены абсолютно в правильности своих действий, такое предложение следует всегда принимать, иначе пакет может потом оказаться неработоспособным.

Программы установки могут и другими способами помочь вам сделать выбор и избежать ошибок. Например, программа установки может отказаться начать установку, если вы не выберете пакет, который совершенно необходим для запуска системы хотя бы в минимальном объеме (скажем, базовую структуру каталогов). Или она может проверить наличие взаимоисключающих выборов в случаях, когда можно установить один из пакетов, но не оба вместе.

С некоторыми дистрибутивами, например SuSE, поставляется большая книга, в которой перечислены все пакеты с краткими описаниями. Неплохо хотя бы просмотреть эти описания, чтобы представлять, что у вас есть в запасе, чтобы не оказаться в положении, когда в результате вашего выбора на машине окажется две дюжины текстовых редакторов.

Создание загрузочной дискеты и установка LILO

Каждый дистрибутив предлагает тот или иной способ начальной загрузки Linux после установки программного обеспечения. Во многих случаях программа установки предлагает создать загрузочную дискету, содержащую ядро Linux, сконфигурированное для использования вновь созданной вами корневой файловой системы. Для начальной загрузки Linux можно использовать эту дискету. После загрузки управление будет передано жесткому диску. В других дистрибутивах загрузочной является сама установочная дискета.

Многие дистрибутивы предлагают установить на жесткий диск LILO – программу, которая помещается в главную загрузочную запись вашего диска. Она загружает несколько операционных систем, в том числе Windows и Linux, и позволяет выбрать одну из них в момент загрузки.

Чтобы успешно установить загрузчик LILO, нужно сообщить ему ряд сведений о конфигурации жесткого диска, например, в каких разделах какие операционные системы установлены, как загружать каждую операционную систему и т. д. Многие дистрибутивы при установке LILO стараются «угадать» правильные параметры вашей конфигурации. Изредка случается, что автоматическая установка LILO ошибается и превращает в руины главную загрузочную запись (хотя едва ли при этом будут также повреждены фактические данные на жестком диске). В частности, при использовании загрузчика OS/2 Boot Manager *нельзя* устанавливать LILO с помощью автоматической процедуры. Есть специальные инструкции по использованию LILO совместно с Boot Manager, которые приведены в главе 5.

Часто бывает лучше воспользоваться загрузочной дискетой, если нельзя сконфигурировать LILO самостоятельно. Но если вы абсолютно уверены, то можете прибегнуть к автоматической установке LILO, когда такая возможность предоставляется вашим дистрибутивом.

В разделе «Использование LILO» главы 5 мы подробно расскажем о том, как сконфигурировать и установить LILO в вашем конкретном случае.

Помимо LILO существуют другие начальные загрузчики, например Grand Unified BootLoader (GRUB). Однако в большинстве дистрибутивов используется LILO.

Если все прошло хорошо, примите поздравления! Вы только что установили Linux на свою машину. Выпейте чашечку чаю или чего-нибудь еще, вы это заслужили.

Если вы столкнулись с какими-либо неприятностями, то в разделе «Устранение неполадок» далее в этой главе описаны наиболее часто встречающиеся затруднения при установке Linux и способы их преодоления.

Дополнительные процедуры установки

В некоторых дистрибутивах Linux есть дополнительные процедуры установки, позволяющие сконфигурировать различные пакеты, например сетевые программы TCP/IP, X Window System и т. д. Если во время установки вам предоставляется такая возможность, то далее в этой книге можно прочесть, как конфигурировать эти программы. В противном случае следует отложить эти процедуры установки до тех пор, пока у вас не возникнет полное понимание того, как конфигурировать это программное обеспечение.

Решайте сами. Если нет иного выхода, отдайтесь течению и посмотрите, что из этого выйдет. Допустив на этом этапе какие-либо ошибки, вы, скорее всего, сможете исправить их в будущем (постучите по дереву).

Послеустановочные процедуры

После завершения установки программного обеспечения Linux вы должны быть в состоянии перезагрузить машину, зарегистрироваться как `root` и начать изучение системы. (В каждом дистрибутиве для этого есть свой способ, следуйте инструкциям в дистрибутиве.)

Прежде чем начать самостоятельную работу, нужно сделать несколько вещей, чтобы уберечь себя от будущих невзгод. Некоторые из этих задач тривиальны, если у вас подходящие аппаратная часть и дистрибутив Linux. Другие задачи потребуют от вас некоторых исследований, и вы можете отложить их на будущее.

Создание учетной записи пользователя

Чтобы начать использование системы, необходимо создать для себя учетную запись (user account). Если вы предполагаете, что на вашей машине будут

работать другие пользователи, нужно будет создать учетные записи и для них. Но для начала нужна хотя бы одна учетная запись.

Для чего это нужно? В каждой системе Linux есть несколько предустановленных учетных записей, например root (суперпользователь). Однако учетная запись root предназначена исключительно для административных целей. Суперпользователь обладает всевозможными привилегиями, и ему доступны все файлы.

Однако работать в системе как пользователь root опасно, особенно если у вас нет опыта работы в Linux. Поскольку нет никаких ограничений на действия суперпользователя, очень легко неправильно ввести команду, случайно удалить файлы, повредить файловую систему и т. д. Работать как root следует только при выполнении задач системного администрирования, например изменения файлов конфигурации, установки нового программного обеспечения и т. д. Подробности можно прочесть в разделе «Работа в системе» главы 5.[1]

Для обычной работы нужно создать стандартную учетную запись пользователя. Unix-системы располагают встроенной системой защиты, которая предотвращает удаление файлов других пользователей и повреждение важных ресурсов, таких как системные файлы конфигурации. Работая как обычный пользователь, вы защитите себя от собственных ошибок. Это особенно относится к пользователям, не имеющим опыта администрирования Unix-систем.

Во многих дистрибутивах Linux есть средства для создания новых учетных записей. Обычно эти программы имеют название *useradd* или *adduser*. Зарегистрировавшись как root и вызвав одну из этих команд, вы получите справку по применению команды, и создание учетной записи должно оказаться вполне очевидным.

В большинстве современных дистрибутивов есть общий инструмент системного администрирования для решения различных задач, одной из которых является создание учетных записей для новых пользователей.

В других дистрибутивах, таких как SuSE Linux, Red Hat Linux и Caldera Open Linux, установка и система администрирования интегрированы в одной программе: *yast* или *yast2* в SuSE Linux и *lisa* в Caldera Open Linux.

Если нет другого способа, можно создать учетную запись вручную. Обычно для создания учетной записи необходимо сделать следующее:

1. Отредактировать файл */etc/passwd*, чтобы добавить нового пользователя. (Если воспользоваться для этого *vipw*, а не редактировать файл напрямую, можно застраховаться от параллельного изменения файла паролей, но *vipw* есть не во всех дистрибутивах.)
2. При необходимости отредактировать файл */etc/shadow* для задания «теневого пароля» (shadow password) нового пользователя.

---

[1] Попутное замечание: в системах Windows 95/98/ME пользователь всегда обладает правами, равносильными правам root, независимо от того, требуются ли такие права в действительности.

3. Создать рабочий каталог пользователя.
4. Скопировать в рабочий каталог нового пользователя шаблоны файлов конфигурации, например *.bashrc*). Иногда они находятся в каталоге */etc/skel*.

Мы не будем здесь вдаваться в особые детали: подробности того, как создаются новые учетные записи, можно найти практически в каждой книге по системному администрированию Unix (см. библиографию). О создании пользователей мы еще поговорим в разделе «Администрирование учетных записей пользователей» главы 5. Вы наверняка найдете утилиту, которая позаботится за вас обо всех деталях.

Запомните, что для установки или изменения пароля для новой учетной записи используется команда *passwd*. Например, чтобы изменить пароль пользователя `duck`, выполните команду

```
# passwd duck
```

В результате вы получите приглашение установить или изменить пароль для `duck`. Если вы выполните команду *passwd*, будучи зарегистрированным как `root`, вам не будет предложено ввести старый пароль. Благодаря этому, если вы забыли свой пароль, но можете зарегистрироваться как `root`, то сможете установить пароль заново.

Получение оперативной помощи

Linux предоставляет оперативную помощь в виде электронных страниц руководства, или «man pages». На протяжении всей книги мы будем отправлять вас к страницам руководства за получением дополнительных сведений об отдельных командах. Страницы руководства подробно описывают имеющиеся в системе программы и приложения, и важно знать, как получить доступ к этой оперативной документации в случае затруднений.

Для получения оперативной помощи по конкретной команде нужно использовать команду *man*. Например, для получения помощи по команде *passwd* введите следующее:

```
$ man passwd
```

В результате должна быть показана страница руководства для *passwd*.

Обычно страницы руководства поставляются как дополнительный пакет в дистрибутиве. Чтобы получить к ним доступ, этот пакет должен быть установлен. Мы настоятельно рекомендуем установить страницы руководства, без помощи которых вы слишком часто будете заходить в тупик.

Надо сказать, что некоторые страницы руководства могут быть неполными или отсутствовать в вашей системе. Все зависит от того, насколько полный у вас дистрибутив и насколько новое руководство.

В страницах руководства Linux описаны также системные вызовы, библиотечные функции, форматы файлов конфигурации и структура ядра. В разделе «Страницы руководства» главы 4 более подробно описывается их использование.

Помимо обычных страниц руководства есть так называемые инфо-страницы (info pages). Прочесть их можно с помощью текстового редактора Emacs, команды *info* или какого-либо из многочисленных графических средств чтения страниц Info.

Во многих дистрибутивах есть также документация в формате HTML, которую можно читать любым броузером типа Konqueror или редактором Emacs.

Наконец, есть файлы документации, являющиеся простыми текстовыми файлами. Их можно читать с помощью любого текстового редактора или просто команды *more*.

Если не удается найти документацию по некоторой команде, можно попробовать выполнить ее с параметром *-h* или *--help*, при этом большая часть команд кратко сообщает о том, как ими пользоваться.

Редактирование /etc/fstab

Чтобы обеспечить доступность всех файловых систем Linux после перезагрузки, может потребоваться отредактировать файл */etc/fstab*, описывающий файловые системы на машине. Многие дистрибутивы автоматически создают файл */etc/fstab* во время установки, тогда все просто. Однако если у вас есть другие файловые системы, которые не использовались во время установки, может потребоваться добавить их в */etc/fstab* для того, чтобы они стали доступны. В */etc/fstab* должны быть также включены разделы для свопинга.

Для того чтобы иметь доступ к файловой системе, ее нужно *монтировать* в системе. Монтирование файловой системы связывает ее с определенным каталогом. Например, корневая файловая система монтируется на /, файловая система */usr* – на */usr*, и т. д. (Если вы не создали отдельной файловой системы для */usr*, все файлы под */usr* будут храниться в корневой файловой системе.)

Мы не хотим сейчас перегружать вас техническими подробностями, но прежде чем начать исследовать систему, необходимо понять, как сделать доступными файловые системы. Более подробно о монтировании файловых систем см. в разделе «Монтирование файловых систем» главы 6 или в любой книге по системному администрированию Unix.

Корневая файловая система автоматически монтируется на / при загрузке Linux. Однако остальные файловые системы надо монтировать отдельно. Обычно это делается командой

```
# mount -av
```

Ее помещают в один из стартовых файлов системы в */etc/rc.d* или другом месте, куда ваш дистрибутив записывает файлы конфигурации. Эта команда *mount* монтирует все файловые системы, перечисленные в */etc/fstab*, поэтому для автоматического монтирования файловых систем во время загрузки необходимо включить их в */etc/fstab*. (Конечно, всегда можно вручную монтировать файловые системы после загрузки командой *mount*, но это лишняя работа.)

Ниже приведен пример файла */etc/fstab*. Он сокращен за счет того, что в каждой строке опущены два последние параметра, которые не являются обязательными и не относятся к нашему обсуждению. В данном примере корневая файловая система находится на */dev/hda1*, файловая система */home* находится на */dev/hdb2*, а раздел для свопинга – на */dev/hdb1*:

```
# /etc/fstab
# device      directory   type   options
#
/dev/hda1     /           ext2   defaults
/dev/hdb2     /home       ext2   defaults
/dev/hdb1     none        swap   sw
/proc         /proc       proc   defaults
```

Строки, начинающиеся с символа «#», являются комментариями. Обратите также внимание на дополнительную строку для */proc*. Устройство */proc* является «виртуальной файловой системой», используемой для сбора данных о процессах командами типа *ps*.

Как вы видите, */etc/fstab* состоит из ряда строк. Первое поле каждой строки – имя устройства раздела, например */dev/hda1*. Второе поле – точка монтирования, т. е. каталог, в котором монтируется файловая система. Третье поле – тип файловой системы. Для файловых систем Linux *ext2fs* в этом поле нужно указать `ext2`, для разделов свопинга – `swap`. В четвертом поле указываются параметры монтирования. Для файловых систем в нем нужно указывать `defaults`, а для свопинга – `sw`.

Используя этот пример в качестве образца, вы сможете добавить строки для любых файловых систем, которых еще нет в файле */etc/fstab*.

Как добавлять строки в файл? Проще всего, зарегистрировавшись как `root`, отредактировать его с помощью таких редакторов, как *vi* или Emacs. Сейчас мы не станем касаться использования текстовых редакторов – о них рассказывается в начале главы 9.

После редактирования файла нужно выполнить команду

```
# /bin/mount -a
```

или перезагрузиться, чтобы изменения возымели действие.

Если вы завязли в этом месте, не тревожьтесь. Мы рекомендуем новичкам в Unix немного почитать об основах использования Unix и системном администрировании. В оставшейся части книги мы предполагаем знакомство читателя с этими основами, поэтому не говорите, что мы вас не предупреждали.

Останов системы

Никогда нельзя перезагружать или останавливать систему путем нажатия кнопки перезагрузки или выключения питания. Как и большинство систем Unix, Linux кэширует в памяти запись на диск. Поэтому при внезапной перезагрузке системы без предварительного корректного останова можно повредить данные на дисках. Заметьте, что «судорожное движение» (одновремен-

ное нажатие Ctrl-Alt-Del) обычно безопасно: ядро перехватывает последовательность нажатий клавиш и инициирует корректный останов системы (как описано в разделе «Файлы init, inittab и rc» главы 5). В настройках вашей системы комбинация Ctrl-Alt-Del может быть зарезервирована для использования только системным администратором, чтобы обычные пользователи не могли остановить сетевой сервер, от которого зависит работа целого подразделения. Чтобы ограничить использование этой комбинации клавиш, создайте файл */etc/shutdown.allow* и перечислите в нем имена всех пользователей, которым разрешается останавливать машину.

Проще всего остановить систему с помощью команды *shutdown*. Например, для немедленного останова и перезагрузки системы выполните, зарегистрировавшись как `root`, команду

```
# shutdown -r now
```

Тем самым система будет корректно перезагружена. На странице руководства для *shutdown* описаны другие аргументы командной строки. Вместо `now` можно задать время, когда система должна быть перезагружена. В большинстве дистрибутивов есть также команда *halt*, вызывающая *shutdown now*. В некоторых дистрибутивах есть еще команда *poweroff*, которая останавливает компьютер и выключает его питание. Будет ли она работать, зависит не от Linux, а от аппаратной части, которая должна поддерживать APM.

Устранение неполадок

При первой попытке установить Linux почти все сталкиваются с какими-либо препятствиями. По большей части проблемы возникают просто от недопонимания. Однако иногда могут быть более серьезные ситуации – промах разработчиков или ошибка в программе.

В этом разделе описаны некоторые наиболее часто возникающие проблемы и способы их решения. Здесь также говорится о некоторых неожиданных сообщениях об ошибках, появляющихся во время установки, которая выглядит успешной.

Нормальная последовательность загрузки обычно следующая:

1. После загрузки с приглашения LILO система должна загрузить с дискеты ядро. Для этого может понадобиться несколько секунд; вы можете убедиться в том, что процесс идет нормально, если лампочка привода гибких дисков продолжает светиться.
2. Во время загрузки ядра должны быть проверены устройства SCSI. Если у вас нет устройств SCSI, система «зависнет» примерно на 15 секунд, пока производится поиск устройств SCSI; обычно это происходит после вывода строки

   ```
   lp_init: lp1 exists (0), using polling driver
   ```

3. По окончании загрузки ядра управление передается загрузочным файлам системы на гибком диске. В конце будет выведено приглашение ре-

гистрации или запущена программа установки. Если показано приглашение регистрации в виде:

```
Linux login:
```

нужно зарегистрироваться (обычно как `root` или `install`, что зависит от дистрибутива, который вы используете). После ввода имени пользователя система может замолчать секунд на 20 или более, пока с дискеты загружается программа установки или оболочка. При этом должна гореть лампочка привода гибких дисков. Не думайте, что система зависла.

Проблемы с загрузочным и установочным носителем

При первой попытке загрузки с установочного носителя можно столкнуться с рядом проблем. Отметим, что описываемые проблемы *не* относятся к загрузке только что установленной системы Linux. Об этих проблемах читайте в разделе «Проблемы, возникающие после установки Linux».

При попытке загрузки возникает ошибка обращения к дискете или носителю. Самой частой причиной такой ошибки является повреждение загрузочной дискеты. Либо дискета повреждена физически, и тогда вам нужно создать новый загрузочный диск на другой дискете, либо испорчены данные на дискете, и тогда нужно проверить правильность загрузки данных и копирования их на дискету. Во многих случаях проблема решается повторным созданием загрузочной дискеты. Повторите операции по созданию дискеты и попробуйте еще раз.

Если вы получили загрузочную дискету по почте или у какого-либо дистрибьютора, а не загружали данные из Интернета и создавали дискету сами, попросите дистрибьютора предоставить вам новую загрузочную дискету, но только убедившись сначала, что проблема состоит именно в дискете. Конечно, это может быть нелегко, но если привод гибких дисков издает необычные звуки или появляются сообщения типа «невозможно прочесть сектор», весьма вероятно, что поврежден носитель.

Машина зависает во время загрузки или после нее. После начала загрузки установочной дискеты ядро выводит ряд сообщений, указывающих на то, какие устройства были обнаружены и сконфигурированы. После этого обычно следует приглашение регистрации, позволяющее продолжить установку (некоторые дистрибутивы вместо этого сразу запускают программу установки). На этом этапе может показаться, что система зависла. Будьте терпеливы, загрузка программ с дискеты происходит очень медленно. Часто система вовсе не зависла, просто ей требуется время. Прежде чем решить, что система зависла, убедитесь, что активности дисков или компьютера не отмечается в течение нескольких минут.

Каждая из перечисленных в начале раздела операций может вызвать задержку, которая заставит предположить, что система встала. Однако система может и действительно зависнуть во время загрузки по ряду причин. Во-первых, в системе может оказаться недостаточно оперативной памяти для загрузки инсталляционного носителя. (См. в следующем

пункте, как отключить электронный диск для освобождения оперативной памяти.)

Причиной зависания во многих случаях может быть несовместимость аппаратного обеспечения. Даже если имеющиеся аппаратные средства поддерживаются системой, могут возникнуть проблемы с несовместимостью аппаратных конфигураций, которые вызовут зависание системы. Ниже в разделе «Проблемы с аппаратным обеспечением» рассказывается об аппаратной несовместимости. В разделе «Требования к аппаратному обеспечению» главы 10 перечислены поддерживаемые в настоящее время наборы микросхем видеосистемы, с которыми связаны основные проблемы работы Linux в графическом режиме.

При начальной загрузке или установке программного обеспечения система сообщает о нехватке памяти (out-of-memory). Эта проблема связана с объемом оперативной памяти, установленной в машине. Самой системе Linux требуется для работы не менее 4 Мбайт памяти. Практически все современные дистрибутивы требуют 8 Мбайт или более. Если в машине 8 Мбайт памяти или меньше, могут возникнуть неприятности при загрузке установочного носителя или установке самого программного обеспечения. Это происходит из-за того, что во многих дистрибутивах используется *электронный диск* (*ramdisk*), который представляет собой файловую систему, загружаемую непосредственно в оперативную память и используемую для действий с установочным носителем. Например, на электронный диск может быть загружен весь образ установочной дискеты, для чего нужно больше 1 Мбайт памяти.

Решение проблемы состоит в том, чтобы отключить возможность использования электронного диска на время загрузки установочного носителя. В каждом дистрибутиве для этого существует своя процедура. Сведения о ней должны быть в документации, сопровождающей дистрибутив.

Сообщения об ошибке «out of memory» во время загрузки и установки можно и не увидеть. Система просто неожиданно зависнет или прекратит загрузку. Если система зависла, и причины этого не связываются с объяснениями предыдущего раздела, попробуйте отключить электронный диск.

Во время загрузки система сообщает об ошибке, например: «Permission denied» (в доступе отказано) или «File not found» (файл не найден). Это свидетельствует о повреждении инсталляционного носителя. При загрузке с инсталляционного носителя (если вы все делаете правильно) таких сообщений быть не должно. Свяжитесь со своим дистрибьютором программного обеспечения Linux и сообщите о проблеме; при необходимости получите другую копию загрузочного носителя. Если вы загрузили диск из Интернета сами, попробуйте его создать заново, возможно, это решит ваши проблемы.

При загрузке система сообщает об ошибке: «VFS: Unable to mount root». Это сообщение означает, что корневая файловая система (находящаяся на самом загрузочном носителе) не найдена, и свидетельствует о том, что ваш загрузочный носитель поврежден или вы неправильно загружаете систему.

Например, многие дистрибутивы на CD-ROM требуют, чтобы во время загрузки компакт-диск находился в приводе. Проверьте также, включен ли привод CD-ROM и есть ли к нему обращения. Возможно, система не может найти привод CD-ROM во время загрузки. Более подробные сведения есть в следующем разделе «Проблемы с аппаратным обеспечением».

Если вы уверены, что правильно производите загрузку, возможно, ваш загрузочный носитель действительно поврежден. Это происходит редко, поэтому попробуйте найти другие решения, прежде чем пытаться использовать другой загрузочный диск или ленту. В дистрибутиве RedHat есть полезная новая функция *mediacheck*, с помощью которой можно проверить целостность компакт-диска.

Проблемы с аппаратным обеспечением

Наиболее часто встречающаяся проблема при установке или работе Linux – несовместимость аппаратного обеспечения. Даже если все ваше аппаратное обеспечение поддерживается Linux, неправильная настройка его или конфликт аппаратуры могут приводить к необычным результатам: устройства не определяются при загрузке или система виснет.

Важно идентифицировать эти проблемы с аппаратурой, если есть подозрение, что они могут быть причиной возникших проблем. В следующих разделах мы опишем некоторые распространенные проблемы с аппаратным обеспечением и способы их решения.

Идентификация проблем, связанных с аппаратным обеспечением

Если возникла проблема, которая, предположительно, связана с аппаратным обеспечением, надо ее, во-первых, идентифицировать. Это значит, что нужно исключить все возможные переменные составляющие и, как часто бывает, разобрать систему по кусочкам, пока не удастся выделить зловредный элемент аппаратуры.

Все не так страшно, как может показаться. Обычно нужно убрать из машины все несущественные устройства (предварительно выключив машину из сети), а затем определить, которое из устройств вызывает проблемы – возможно, путем поочередной установки каждого устройства. Это означает, что нужно убрать все устройства, кроме контроллера гибкого диска и видеоадаптера и, конечно, клавиатуры. Даже такое невинно выглядящее устройство, как контроллер мыши, может оказаться нарушителем спокойствия, если вы сочтете его малозначительным. Поэтому на время проведения экспериментов на всякий случай удалите все, что не является абсолютно необходимым для начальной загрузки, и собирайте потом машину заново, добавляя устройства по одному.

Допустим, например, что система виснет при обнаружении платы Ethernet во время загрузки. Вы предполагаете, что существует конфликт или проблема с картой Ethernet, установленной в вашей машине. Простой и быстрый способ выяснить это состоит в том, чтобы вынуть карту Ethernet и попытаться снова загрузиться. Если это удастся, то либо ваша карта Ethernet не под-

держивается Linux, либо она вызывает конфликт по адресам или IRQ. Кроме того, некоторые скверные сетевые карты (в основном это относится к клонам NE2000 на шине ISA, которые, к счастью, сейчас вымирают) могут подвесить всю систему во время автоматической проверки. В таком случае лучше всего вынуть сетевую карту на время установки и вернуть ее на место позднее либо при загрузке передать ядру необходимые параметры, чтобы избежать автоматической проверки сетевой карты. Самое верное средство – избавиться от этой карты и приобрести новую у производителя, который более тщательно разрабатывает свои изделия.

Вы можете спросить, что означает «Конфликт адресов или IRQ»? Все устройства в вашей машине используют *линию запроса прерывания*, или IRQ, с помощью которой сообщают системе о том, что она должна что-то для них сделать. Можете представлять себе IRQ как веревочку, за которую дергает устройство, когда ему нужно, чтобы система выполнила какое-нибудь возникшее у него требование. Если за одну и ту же веревочку дергает несколько устройств, система не может определить, которое из устройств она должна обслужить. Мгновенно возникает каша.

Поэтому обеспечьте использование всеми установленными у вас устройствами, если они не на шине PCI, уникальных линий IRQ. Обычно IRQ для устройства можно установить перемычками на карте. Как это сделать, должно быть описано в документации по конкретному устройству. Для некоторых устройств IRQ вообще не требуется, но вам предлагается сконфигурировать их для использования некоторого IRQ, если это возможно (примерами таких устройств являются SCSI-контроллеры Seagate ST01 и ST02). Шина PCI спроектирована более удачно, и подключаемые к ней устройства могут использовать линии прерывания совместно.

В некоторых случаях ядро, находящееся на установочном носителе, сконфигурировано для использования определенных IRQ с определенными устройствами. Например, в некоторых дистрибутивах Linux ядро заранее настроено на использование IRQ 5 для SCSI-контроллера TMC-950, контроллера CD-ROM Mitsumi и драйвера шинной мыши. Если нужно использовать два или более из этих устройств, следует сначала установить Linux с одним из этих устройств, а затем перекомпилировать ядро, чтобы изменить значение IRQ по умолчанию одного из этих устройств. (О перекомпиляции ядра см. раздел «Сборка нового ядра» в главе 7.)

Другими областями, где могут возникать аппаратные конфликты, являются каналы *прямого доступа к памяти* (DMA), адреса ввода/вывода и адреса совместно доступной памяти. Все эти термины описывают механизмы взаимодействия системы с аппаратными устройствами. Например, некоторые карты Ethernet используют для взаимодействия с системой как адреса совместно доступной памяти, так и IRQ. Если они вступают в конфликт с другими устройствами, система может вести себя непредсказуемым образом. Вы должны с помощью перемычек изменить у этих устройств канал DMA, адреса ввода/вывода и совместно доступной памяти. (К несчастью, некоторые устройства не позволяют это сделать.)

В документации к вашим аппаратным устройствам должны быть указаны IRQ, канал DMA, адреса ввода/вывода и совместно доступной памяти, используемые устройством, а также способ их настройки. Проблема, разумеется, в том, что некоторые из этих установок нельзя узнать, не собрав систему, а потому их может не быть в документации. Простой способ обойти эти проблемы все тот же – временно отключить конфликтующие устройства и разобраться с причиной возникшей проблемы позднее.

В табл. 3.2 приведен список каналов IRQ и DMA, используемых различными «стандартными» устройствами в большинстве компьютеров. Эти устройства есть почти во всех машинах, поэтому избегайте установки тех же IRQ или DMA в других устройствах.

Таблица 3.2. Распространенные настройки устройств

Устройство	**Адрес I/O**	**IRQ**	**DMA**
ttyS0 (*COM1*)	3f8	4	n/a
ttyS1 (*COM2*)	2f8	3	n/a
ttyS2 (*COM3*)	3e8	4	n/a
ttyS3 (*COM4*)	2e8	3	n/a
lp0 (*LPT1*)	378 – 37f	7	n/a
lp1 (*LPT2*)	278 – 27f	5	n/a
fd0, *fd1* (гибкие диски 1 и 2)	3f0 – 3f7	6	2
fd2, *fd3* (гибкие диски 3 и 4)	370 – 377	10	3

Проблемы при распознавании жесткого диска или контроллера

В начале загрузки Linux вы видите на экране ряд сообщений, например:

```
Console: colour VGA+ 80x25
Floppy drive(s): fd0 is 1.44M
ttyS00 at 0x03f8 (irq = 4) is a 16450A
...
```

Здесь ядро сообщает об обнаружении в вашей машине различных аппаратных устройств. В некоторый момент должна появиться строка:

```
Partition check:
```

за которой следует список обнаруженных разделов, например:

```
Partition check:
  hda: hda1 hda2
  hdb: hdb1 hdb2 hdb3
```

Если по какой-либо причине ваши диски или разделы окажутся нераспознанными, вы никак не сможете ими воспользоваться.

Это может произойти в нескольких ситуациях:

Жесткий диск или контроллер не поддерживаются

Если ваш контроллер жесткого диска (IDE, SCSI или иной) не поддерживается Linux, то ядро не сможет распознать разделы во время загрузки.

Диск или контроллер неправильно сконфигурированы

Даже если Linux поддерживает ваш контроллер, он может оказаться неверно настроенным. (Обычно эта проблема возникает со SCSI-контроллерами, большая часть других контроллеров должна работать без дополнительной настройки.)

Для решения такого рода проблем обратитесь к документации по вашему жесткому диску и контроллеру. В частности, на многих жестких дисках нужно установить перемычку, если он будет использоваться как «slave», т. е. в качестве второго жесткого диска. Проверкой в такой ситуации будет загрузка Windows или другой операционной системы, относительно которой известно, что она может работать с вашим диском и контроллером. Если диск и контроллер доступны из другой операционной системы, то проблема не в аппаратной конфигурации.

Почитайте о разрешении возможных конфликтов между устройствами в предыдущем разделе «Идентификация проблем с аппаратным обеспечением» и о конфигурировании SCSI-устройств – в следующем разделе «Решение проблем, возникающих с SCSI-контроллерами и устройствами».

Контроллер сконфигурирован правильно, но не обнаруживается

Некоторые SCSI-контроллеры без BIOS требуют, чтобы пользователь ввел сведения о контроллере во время загрузки. В следующем разделе «Решение проблем, возникающих с SCSI-контроллерами и устройствами» описано, как обеспечить обнаружение аппаратных устройств для таких контроллеров.

Не распознается геометрия диска

В некоторых системах, таких как IBM PS/ValuePoint, данные о геометрии не записываются в память CMOS, где Linux рассчитывает их обнаружить. Кроме того, некоторые SCSI-контроллеры требуют, чтобы им указали, где находятся данные о геометрии диска, и тогда Linux сможет распознать структуру вашего диска.

В большинстве дистрибутивов есть параметр загрузки, позволяющий задать геометрию диска. Обычно при загрузке с инсталляционного носителя можно задать геометрию диска в ответ на приглашение загрузки LILO командой

```
boot: linux hd=cylinders,heads,sectors
```

где *cylinders*, *heads* и *sectors* определяют число цилиндров, головок и секторов соответственно на дорожке вашего жесткого диска.

После установки программного обеспечения можно установить LILO, что позволит загружать систему с жесткого диска. В этот момент можно задать геометрию диска в процедуре установки LILO, что избавит от необхо-

димости вводить геометрию при каждой загрузке. Более подробно о LILO см. в разделе «Использование LILO» главы 5.

Решение проблем, возникающих с SCSI-контроллерами и устройствами

Ниже описываются некоторые наиболее распространенные проблемы, возникающие с SCSI-контроллерами и устройствами, такими как CD-ROM, жесткие диски и ленточные приводы. Если возникли проблемы с распознаванием Linux диска или контроллера, прочтите этот раздел. Еще раз отметим, что большинство дистрибутивов использует модульное ядро, и, возможно, было необходимо загрузить модуль, поддерживающий ваше устройство, на ранней стадии процесса установки. Это может выполняться и автоматически.

В Linux SCSI HOWTO содержится много полезных дополнительных сведений о SCSI-устройствах. SCSI-устройства иногда конфигурируются весьма замысловатым образом.

Например, использование дешевых шлейфов может оказаться ложной экономией, особенно для wide SCSI. Дешевые кабели – основной источник проблем; они могут вызывать все виды отказов. Если у вас есть устройства SCSI, используйте надлежащие кабели.

Вот наиболее часто встречающиеся проблемы и их возможные решения:

Устройство SCSI определяется на всех возможных ID. Эта проблема возникает, когда система назначает устройству тот же адрес, который имеет контроллер. Необходимо изменить положение перемычек, чтобы диск использовал адрес, отличный от адреса контроллера.

Linux сообщает об отсутствии доступа, хотя известно, что устройства исправны. Это может быть вызвано плохими шлейфами или терминаторами. Если шина SCSI не имеет терминаторов на обоих концах, то при доступе к устройствам SCSI могут возникать ошибки. Если у вас есть сомнения, обязательно проверьте шлейфы. Нередко источником ошибок являются неприсоединенные или имеющие низкое качество шлейфы.

Устройства SCSI сообщают об ошибках тайм-аута. Обычно это вызвано конфликтом IRQ, DMA или адресов устройств. Проверьте также правильность установки прерываний контроллера.

Не определяются SCSI-контроллеры, использующие BIOS. Обнаружение контроллеров, использующих BIOS, может быть безуспешным, если BIOS отключен или «сигнатура» контроллера не распознается ядром. Более подробно читайте об этом в Linux SCSI HOWTO.

Не работают контроллеры, использующие отображаемый в память ввод/вывод. Это происходит, когда отображаемые в память порты ввода/вывода неправильно кэшируются. Пометьте в XCMOS адресное пространство платы как некэшируемое либо вообще отключите кэш.

При создании разделов выводится предупреждение «cylinders > 1024», или невозможно загрузить систему с раздела, использующего цилиндры с номерами, большими 1023. BIOS ограничивает количество цилиндров числом 1024, и любой раздел, использующий цилиндры с номерами, большими этого числа, недоступен через BIOS. Что касается Linux, это оказывает влияние только на загрузку. Если система загрузилась, вы должны получить доступ к такому разделу. Решение – загружать Linux либо с загрузочной дискеты, либо с раздела, использующего цилиндры до номера 1024. См. раздел «Создание загрузочной дискеты или инсталляция LILO» выше в этой главе.

Во время загрузки не распознается привод CD-ROM или другого съемного носителя. Попробуйте загрузиться при находящемся в приводе компакт-диске (или дискете). Для некоторых устройств это необходимо.

Если не распознается SCSI-контроллер, может потребоваться осуществить принудительное определение устройства во время загрузки. Это особенно важно для SCSI-контроллеров без BIOS. Большинство дистрибутивов позволяют задать IRQ контроллера и адрес совместно используемой памяти при загрузке с установочного носителя. Например, при использовании контроллера TMC-8xx можно в ответ на приглашение загрузки LILO ввести:

```
boot: linux tmx8xx=interrupt,memory-address
```

где *interrupt* – это IRQ контроллера, а *memory-address* – адрес совместно используемой памяти. Наличие такой возможности зависит от того, какой дистрибутив вами используется, поэтому изучите поставляемую с дистрибутивом документацию.

Проблемы при установке программного обеспечения

Если вам повезет, то установка программного обеспечения пройдет без помех. Проблемы, с которыми вы можете столкнуться, могут быть связаны с повреждением инсталляционного носителя или недостатком места в файловых системах Linux. Вот перечень наиболее часто встречающихся проблем:

Во время установки программного обеспечения появляется сообщение «Read error, file not found» или другое сообщение об ошибке. Это указывает на проблемы с инсталляционным носителем. При установке с дискет имейте в виду, что дискеты весьма подвержены такого рода ошибкам носителя. Используйте новые, заново отформатированные дискеты. Если на вашем диске есть раздел Windows, то можно воспользоваться тем, что многие дистрибутивы Linux позволяют устанавливать программное обеспечение с жесткого диска. Это может оказаться более быстрым и надежным, чем установка с дискет.

При установке с компакт-диска убедитесь в отсутствии царапин, пыли или других помех, способных вызвать ошибки носителя.

Причиной проблемы может также быть неверный формат носителя. Например, при использовании дискет многие дистрибутивы Linux требуют, чтобы дискеты были отформатированы в формате Windows с высокой

плотностью записи. (Исключением является загрузочная дискета: в большинстве случаев она имеет формат, отличный от Windows.) Если ничто другое не помогает, получите новый комплект дискет или заново их создайте, используя новые дискеты, если вы самостоятельно загрузили программное обеспечение из сети.

Система выводит сообщение «tar: read error» или «gzip: not in gzip format». Обычно эти ошибки связаны с искажением файлов на инсталляционном носителе. Иными словами, сама дискета цела, но данные на ней каким-то образом искажены. Например, если вы загрузили программное обеспечение в текстовом, а не в двоичном режиме, то ваши файлы окажутся искажены и не смогут быть прочтены программой установки. При загрузке по FTP выполните команду *binary*, чтобы установить правильный режим, прежде чем запрашивать передачу файлов.

Во время установки система выводит сообщение «device full» (нет места на устройстве) или аналогичное. Это четкое указание на то, что у вас недостаточно места для установки программного обеспечения. При переполнении диска не все дистрибутивы могут корректно восстановить работу, и в результате процесса установки вы не получите работающей системы.

Решение проблемы обычно состоит в повторном создании файловых систем командой *mke2fs*, которая удалит частично установленное программное обеспечение. Затем можно попытаться еще раз установить программное обеспечение, на этот раз отобрав для установки меньше программ. Если же вам совершенно необходимы эти программы, то может потребоваться начать установку с самого начала и переопределить размеры разделов и файловых систем.

При обращении к жесткому диску система выводит сообщение типа «read_intr: 0x10». Обычно это свидетельствует о наличии на жестком диске сбойных блоков. Однако если такие сообщения возникают во время работы *mkswap* или *mke2fs*, может оказаться, что системе не удается обратиться к жесткому диску из-за ошибок устройства (см. раздел «Проблемы с аппаратной частью» ранее в этой главе) или неправильно указанной геометрии диска. Если при загрузке используется параметр

```
hd=cylinders,heads,sectors
```

для принудительного определения геометрии диска, и он задан неправильно, то можно получить такое сообщение. То же может произойти при неправильном задании геометрии диска в системном CMOS.

Система выводит сообщение «file not found» (файл не найден) или «permission denied» (доступ запрещен). Это может произойти, если на инсталляционном носителе присутствуют не все файлы или программа установки столкнулась с проблемами прав доступа. Известно, например, что в некоторых дистрибутивах Linux есть ошибки в самой программе установки. Такие ошибки быстро исправляются и встречаются довольно редко. Если вы подозреваете, что в программном обеспечении дистрибу-

тива есть ошибки, и уверены, что все делали правильно, свяжитесь с тем, кто осуществляет сопровождение дистрибутива, и сообщите об ошибке.

Если во время установки Linux возникают другие непонятные ошибки (особенно если вы самостоятельно загрузили программное обеспечение), проверьте, действительно ли вы загрузили все необходимые файлы.

Например, некоторые используют для загрузки программного обеспечения по FTP команду

```
mget *.*
```

В результате загружаются только файлы, в именах которых содержится «.», а если есть файлы без «.», то они не будут получены. Правильная команда должна быть следующей:

```
mget *
```

Лучше всего в случае неприятностей повторить все действия заново. Вам может казаться, что вы все сделали правильно, а в действительности где-то по дороге вы пропустили маленький, но важный шаг. Во многих случаях достаточно заново загрузить или снова установить программное обеспечение, чтобы решить проблему. Не стоит слишком долго биться головой о стену!

Кроме того, если Linux неожиданно зависает во время установки, это может быть результатом тех или иных проблем с аппаратурой. Соответствующие советы см. в разделе «Проблемы с аппаратным обеспечением».

Проблемы, возникающие после установки Linux

Вы провели целый день за установкой Linux. Чтобы освободить для нее место, вы затерли разделы Windows и OS/2 и со слезами удалили SimCity 2000 и Railroad Tycoon 2. Вы перезагружаете систему, и ничего не происходит. Или, того хуже, *что-то* происходит, но не то, что должно произойти. Что делать?

В разделе «Проблемы при загрузке с инсталляционного носителя» этой главы мы рассказали о наиболее часто встречающихся проблемах при загрузке инсталляционного носителя Linux. Описанные причины могут действовать и в данной ситуации. Кроме того, вы могли стать жертвой следующих напастей.

Проблемы при загрузке Linux с дискеты

При загрузке Linux с дискеты может потребоваться задать расположение корневого раздела Linux. Это особенно касается случая, когда для загрузки используется сама установочная дискета, а не специальная загрузочная дискета, созданная во время установки.

При загрузке с дискеты удерживайте нажатой клавишу <Shift> или клавишу <Ctrl>. Это должно привести к появлению меню загрузки. Нажмите <Tab>, чтобы вывести список возможных вариантов. Например, многие дистрибутивы позволяют загрузиться с дискеты после ввода следующей строки:

```
boot: linux root=partition
```

где *partition* – имя корневого раздела Linux, например */dev/hda2*. Дистрибутив SuSE Linux предлагает меню на ранней стадии выполнения установочной программы, которая загружает вновь созданную вами систему Linux с установочной загрузочной дискеты. Подробности смотрите в документации дистрибутива.

Проблемы при загрузке Linux с жесткого диска

Если вы решили установить LILO, а не создавать загрузочную дискету, у вас должна быть возможность загрузить Linux с жесткого диска. Однако используемая во многих дистрибутивах автоматизированная процедура установки LILO не всегда совершенна. Она может сделать неправильные предположения о структуре вашего раздела, и тогда вам понадобится переустановить LILO, чтобы все заработало. Установка загрузчика LILO описана в разделе «Использование LILO» главы 5.

Вот некоторые часто возникающие проблемы:

Система выводит сообщение «Drive not bootable-Please insert system disk.» (Диск не загрузочный – вставьте, пожалуйста, системный диск.) Это сообщение об ошибке появляется при повреждении главной загрузочной записи жесткого диска. В большинстве случаев это не опасно, и все остальные данные на диске целы. Есть несколько способов справиться с этой проблемой:

- При разбиении диска на разделы с помощью *fdisk* вы могли удалить раздел, помеченный как «active» (активный). Windows и другие операционные системы пытаются осуществлять загрузку с активного раздела (Linux обычно не обращает внимания, является ли раздел активным, но некоторые дистрибутивы, например Debian, устанавливают главную загрузочную запись, которая ищет активный раздел). Можно загрузить MS-DOS с дискеты и, запустив *fdisk*, установить флаг активности на разделе MOS-DOS, и тогда все пойдет нормально.

  Другая команда, которую можно использовать (с MS-DOS 5.0 и выше, а также в Windows начиная с Windows 95):

  ```
  FDISK /MBR
  ```

  Эта команда попытается переписать главную загрузочную запись жесткого диска для загрузки Windows, затерев LILO. Если Windows на жестком диске у вас не осталась, нужно загрузить Linux с дискеты и попробовать установить LILO позже.

- Такую ошибку можно получить, если создать раздел Windows с помощью Linux-версии *fdisk*, или наоборот. Разделы Windows можно создавать только с помощью версии *fdisk* для Windows. (То же относится и к операционным системам, отличным от Windows.) Самое лучшее решение – начать все с самого начала и правильно разбить диск на разделы либо просто удалить плохие разделы и создать их заново с помощью корректной версии *fdisk*.

- Процедура установки LILO могла пройти неудачно. В этом случае нужно перезагрузиться с загрузочной дискеты Linux (если она есть) или с исходного инсталляционного носителя. В любом случае вы получите возможность указать во время загрузки корневой раздел Linux. Во время загрузки удерживайте нажатой клавишу <Shift> или клавишу <Ctrl> и нажмите <Tab>, находясь в загрузочном меню, чтобы вывести список возможных вариантов.

При загрузке системы с жесткого диска вместо Linux запускается Windows (или другая операционная система). Во-первых, проверьте, действительно ли вы установили LILO во время установки программного обеспечения Linux. Если нет, то система будет по-прежнему загружать Windows (или другую имеющуюся операционную систему) при попытке загрузки с жесткого диска. Чтобы загружать с жесткого диска Linux, необходимо установить LILO (см. раздел «Использование LILO» главы 5).

Если же LILO действительно *установлен*, но вместо Linux загружается другая операционная система, значит, вы сконфигурировали LILO таким образом, что по умолчанию загружается другая операционная система. Во время загрузки системы удерживайте нажатой клавишу <Shift> или <Ctrl> и нажмите <Tab>, находясь в загрузочном меню. Вам должен быть выведен список операционных систем, которые можно загрузить. Для загрузки Linux выберите соответствующий пункт меню (обычно просто `linux`).

Если вы хотите, чтобы Linux стала операционной системой, загружаемой по умолчанию, необходимо перенастроить LILO.

Возможно также, что вы пытались установить LILO, но процедура установки по какой-то причине не завершилась. См. предыдущий пункт об установке.

Проблемы при входе в систему

После загрузки Linux должно выводиться приглашение для входа в систему:

```
Linux login:
```

Что делать дальше, говорится в документации дистрибутива или сообщает сама система. Во многих дистрибутивах вы регистрируетесь просто как `root` без пароля. Другими возможными именами пользователя могут быть `guest` или `test`.

Большинство дистрибутивов Linux запрашивает начальный пароль суперпользователя `root`. Надеемся, что вы запомнили пароль, который вводили при установке, – он вам теперь снова нужен. Если при установке дистрибутива пароль не запрашивался, можно попробовать ввести пустой пароль.

Если вы просто не можете зарегистрироваться, посмотрите документацию дистрибутива: необходимые имя пользователя и пароль должны быть в ней указаны. Имя пользователя и пароль могли быть даны вам во время процедуры установки или они могут быть выведены на баннере регистрации.

Возможной причиной неудачи с паролем может быть проблема с установкой файлов регистрации и инициализации Linux. В этом случае может потребоваться переустановка программного обеспечения (или только какой-то его части) либо нужно загрузиться с инсталляционного носителя Linux и попытаться устранить проблему вручную.

Проблемы при работе с системой

Если регистрация прошла успешно, должно быть выдано приглашение оболочки (например, # или $), и можно начать путешествие по системе. В этом случае следующим шагом должны стать процедуры, описанные в главе 4. Однако иногда всплывают некоторые начальные проблемы использования системы.

Самой распространенной проблемой начальной настройки являются некорректные права доступа к файлам или каталогам. В этом случае после регистрации может появиться сообщение об ошибке:

```
Shell-init: permission denied (доступ запрещен)
```

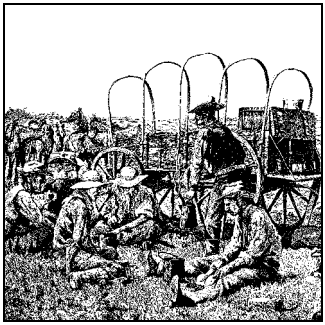
В действительности в любом случае, когда вы видите сообщение `permission denied`, можно быть уверенным, что проблема в правах доступа к файлам.

Часто достаточно просто воспользоваться командой *chmod*, чтобы исправить права доступа к соответствующим файлам или каталогам. Например, в некоторых дистрибутивах Linux когда-то использовался неверный режим файла `0644` для корневого каталога (/). Для решения проблемы нужно, зарегистрировавшись как `root`, выполнить команду

```
# chmod 755 /
```

(О правах доступа к файлам рассказано в разделе «Владение файлами и права доступа» главы 4.) Однако для того чтобы выполнить эту команду, необходимо загрузиться с инсталляционного носителя и смонтировать корневую файловую систему вручную – это сложная задача для большинства новичков.

Во время работы в системе вы вполне можете столкнуться с тем, что права доступа к некоторым файлам и каталогам некорректны или программное обеспечение работает не в соответствии с настройками. Добро пожаловать в Linux! Хотя большинство дистрибутивов вполне работоспособно, нельзя ожидать от них идеальной работы. Мы не хотим здесь рассказывать обо всех проблемах, а вместо этого на протяжении всей книги будем помогать избавиться от многих из этих проблем настройки, показывая, как можно их обнаружить и решить самостоятельно. В главе 1 мы более или менее подробно обсуждали эту философию. В главе 5 мы дадим советы по решению многих из часто встречающихся проблем настройки.



4

Основные команды и понятия Unix

Если до Linux вы работали в Windows или другой операционной системе, отличной от Unix, то вам предстоит многое изучить. Скажем откровенно: Unix – это целый самостоятельный мир.

В этой главе мы познакомим с основами Unix тех читателей, которые никогда не сталкивались с этой операционной системой. Если вы раньше использовали Microsoft Windows или иную среду, то сведения, приведенные в этой главе, для вас совершенно необходимы. В отличие от других операционных систем, Unix вовсе не воспринимается интуитивно. Многие команды имеют странные имена или синтаксис, и причины этого закладывались в первые годы существования системы Unix. И хотя многие команды выглядят похожими на свои аналоги в Windows, между ними существуют важные различия.

Есть десятки книг об основах использования Unix. Вы можете пойти в отдел компьютерной литературы любого большого книжного магазина и обнаружить на полке несколько таких книг. (Некоторые из тех, которые нам нравятся, перечислены в библиографии.) Однако в большинстве этих книг говорится о Unix с точки зрения человека, сидящего за рабочей станцией или терминалом, подключенным к мэйнфрейму, а не того, кто запустил собственную Unix-систему на персональном компьютере. Популярное введение в работу с Unix, где говорится также о Linux, можно найти в книге «Learning the Unix Operating System», написанной Грейсом Тодино (Grace Tjdino), Джоном Стренгом (John Strang) и Джерри Пиком (Jerry Peek), O'Reilly.

Кроме того, в этих книгах часто занимаются достаточно обыденными аспектами Unix: скучными командами обработки текста, такими как *awk*, *tr* и *sed*, которые по большей части никогда вам не понадобятся, если только вы не займетесь в Unix чем-то мудреным. Во многих книгах по Unix рассказывается о древнем редакторе строк *ed*, давно устаревшем с появлением *vi* и Emacs. Поэтому, хотя во многих из имеющихся в настоящее время книг по Unix есть масса полезной информации, многие страницы в них заполнены скучным материалом, который в данный момент меньше всего вам нужен.

В этой главе мы стремимся избежать обсуждения вопросов обработки текста, синтаксиса команд оболочки и т. п., и вместо этого представим основные команды, которые позволят быстро освоить систему, если вы не знакомы со средой Unix. Эта глава далека от полного изложения материала: настоящее введение в Unix заняло бы целую книгу. Мы надеемся, что вы найдете в этой главе достаточно материала для того, чтобы продолжить свое путешествие по Linux, и что в случае необходимости вы не поскупитесь на одну из вышеперечисленных книг по Unix. Мы дадим достаточную подготовку по Unix, чтобы вы смогли работать за терминалом, следить за выполнением заданий и вводить основные команды.

Глава 5 содержит материал по администрированию и сопровождению системы. Это, несомненно, самая важная глава для каждого, кто работает в собственной системе Linux. Если вы ничего не знаете о Unix, то после изучения данного начального руководства должны легко понять материал главы 5.

Одна из существенных тем, не освещаемых в данной главе, – это редактирование файлов. Эта тема должна быть одной из первых при изучении любой операционной системы. О двух наиболее популярных в Linux редакторах, *vi* и Emacs, рассказано в начале главы 9.

Регистрация пользователя в системе

Допустим, что установка прошла гладко, и на экране появилось следующее приглашение:

```
Linux login:
```

Не всем пользователям Linux так везет: иногда приходится серьезно повозиться, чтобы вывести систему из сырого состояния и однопользовательского режима. Но сейчас мы говорим о регистрации в действующей Linux-системе.

Регистрация, конечно, позволяет отличить одного пользователя от другого. Благодаря этому на машине могут работать одновременно несколько пользователей и исключается доступ посторонних к вашим файлам.

Возможно, вы установили Linux у себя дома и теперь думаете: «Очень хорошо. Никто, кроме меня, системой не пользуется и мне пока не нужно регистрироваться». Однако регистрация с личной учетной записью обеспечивает некоторую степень защиты: ваша учетная запись не позволит уничтожить или переместить важные системные файлы. Для таких деликатных операций используется учетная запись администратора (о чем говорится в следующей главе).

Если вы подключаете свой компьютер к Интернету даже с помощью модема, следите за тем, чтобы во всех учетных записях использовались нетривиальные пароли. Используйте в них специальные символы, знаки и строки, не являющиеся словами или именами.

Следует учесть, что в некоторых дистрибутивах для регистрации сразу вызывается графический администратор регистрации, и вы не увидите таинственного приглашения `login:`, написанного белыми буквами на черном фоне.

Вместо этого вам предложат красивый графический экран, возможно, с показом имен пользователей, у которых есть учетные записи на вашей машине, и различных режимов регистрации. Тем не менее основная процедура регистрации остается той же самой: вы вводите свое имя и пароль.

Возможно, во время установки вам было предложено создать личную учетную запись. Если такая запись есть, введите выбранное вами имя в ответ на приглашение `Linux login:`. Если у вас еще нет учетной записи, введите `root`, поскольку эта запись обязана существовать. В некоторых дистрибутивах создана учетная запись `install` или какое-либо другое имя, позволяющее зарегистрироваться при первоначальной установке системы.

Введя имя учетной записи, вы увидите приглашение

```
Password:
```

и должны будете ввести правильный пароль. Во время этой операции терминал отключает обычное отображение вводимых символов на экране, чтобы никто не мог увидеть ваш пароль, глядя на экран. Если такой подсказки не появилось, нужно установить пароль, чтобы защитить себя от постороннего вмешательства. О том, как это сделать, мы скажем позднее.

Между прочим, как имя, так и пароль чувствительны к регистру. Проверьте состояние клавиши <Caps Lock>, потому что нельзя вводить `ROOT` вместо `root`.

После успешной регистрации вы увидите приглашение. Если вы зарегистрировались как `root`, это может быть просто

```
#
```

Для других пользователей приглашением обычно является знак `$`. В приглашении может также содержаться имя, которое вы присвоили машине, или каталог, в котором вы в данное время находитесь. Что бы там ни появилось, теперь можно вводить команды. Мы говорим, что сейчас вы находитесь на «уровне оболочки», а приглашение, которое вы видите, – это «приглашение оболочки», поскольку вы сейчас работаете в программе, называемой оболочкой (shell), и которая обрабатывает ваши команды (поэтому иногда shell называют «интерпретатором команд» или «командным процессором»). Пока что мы можем не думать об оболочке, но позднее в данной главе мы выясним, что оболочка может делать ряд полезных вещей.

В этой главе при показе команд мы будем изображать приглашение просто как `$`. Поэтому если вы видите:

```
$ pwd
```

это означает, что оболочка выводит `$`, а вы должны ввести `pwd`.

Установка пароля

Если у вас еще нет пароля, советуем его установить. Просто введите команду *passwd*. Вам будет предложено ввести пароль, а затем повторить ввод, чтобы застраховаться от ошибок при вводе.

Есть стандартные правила подбора пароля, чтобы посторонним было труднее его угадать. Некоторые системы даже проверяют ваш пароль и отклоняют его, если он не удовлетворяет некоторым минимальным требованиям. Например, часто требуется, чтобы пароль содержал не менее шести символов. Более того, регистр символов должен быть смешанным или должен включать символы, отличные от букв и цифр.

Чтобы изменить пароль, снова введите команду *passwd*. Она предлагает ввести старый пароль (чтобы убедиться, что вы действительно хозяин этой учетной записи), а затем позволяет его изменить.

Виртуальные консоли

Будучи многозадачной средой, Linux предоставляет ряд интересных способов одновременного выполнения нескольких задач. Вы можете начать долгую установку программного обеспечения, а затем переключиться на чтение электронной почты или компиляцию программы, и все это будет выполняться одновременно.

Большинство пользователей Linux, которым нужен такой асинхронный доступ, использует систему X Window. Но прежде чем запустить X, можно сделать нечто аналогичное с помощью виртуальных консолей. Такая возможность существует в некоторых других версиях Unix, хотя не во всех.

Чтобы переключиться на виртуальную консоль, нажмите любую из функциональных клавиш от <F1> до <F8> при нажатой левой клавише <Alt>. При нажатии каждой функциональной клавиши вы видите новый экран с приглашением регистрации. Можно регистрироваться в различных виртуальных консолях, как если бы вы были разными людьми, и переключаться при этом между консолями для выполнения различных действий. Можно даже в каждой консоли запустить свой сеанс X. X Window System по умолчанию использует виртуальную консоль номер 7. Поэтому, запустив X и переключившись затем в одну из текстовых виртуальных консолей, вы можете вернуться в X, нажав <Alt>+<F7>. Если вы обнаружите, что нажатие комбинации <Alt>+ <функциональная клавиша> вместо переключения виртуальных консолей вызывает меню X или некоторую другую функцию, используйте комбинацию <Ctrl>+<Alt>+<функциональная клавиша>.

Часто используемые команды

Количество команд в типовой системе Unix таково, что можно заполнить несколько сот страниц справочного руководства. Кроме того, можно добавлять новые команды. Команд, о которых мы вам расскажем, достаточно, чтобы перемещаться по каталогам и смотреть, что имеется в системе.

Каталоги

Так же как в Windows и практически любой современной компьютерной системе, файлы Unix организованы в иерархическую структуру каталогов.

Unix не накладывает ограничений на то, где должны располагаться файлы, но за годы его использования возникли некоторые соглашения. Так, например, в Linux вы найдете каталог с именем */home*, в который помещаются файлы всех пользователей. У каждого пользователя в */home* имеется подкаталог. Поэтому если ваше имя в системе – `mdw`, то ваши личные файлы расположены в */home/mdw*. Этот каталог называется вашим домашним (home) каталогом. Конечно, вы можете создавать в нем другие подкаталоги.

Если вы работали раньше в Windows, то использование косой черты (/) в качестве разделителя в пути может показаться странным, поскольку вы привыкли к обратной косой черте (\). Ничего загадочного в обычной косой черте нет, и она использовалась в качестве разделителя в пути задолго до того, как появились MS-DOS и Windows. Обратная косая черта имеет в Unix другое назначение (отключает особое значение следующего за ней символа, если таковое у него есть).

Как можно видеть, составляющие каталога разделяются косой чертой. Для такого разделенного косыми чертами списка часто используется термин *путь* (*pathname*).

В каком каталоге находится */home*? Разумеется, в каталоге с именем /. Он называется корневым каталогом. Мы уже говорили о нем при установке файловых систем.

При регистрации система помещает вас в домашний каталог. Чтобы убедиться в этом, используйте команду «вывести рабочий каталог» (print working directory), или *pwd*:

```
$ pwd
/home/mdw
```

Система подтверждает, что вы находитесь в */home/mdw*.

Конечно, находиться все время в одном каталоге не очень весело. Попробуем использовать другую команду, *cd*, чтобы перейти в другой каталог:

```
$ cd /usr/bin
$ pwd
/usr/bin
$ cd
```

Где теперь мы находимся? Команда *cd* без аргументов возвращает нас в наш домашний каталог. Между прочим, домашний каталог часто обозначается символом «тильда» (~). Поэтому строка *~/programs* означает, что *programs* находится прямо в вашем домашнем каталоге.

Раз уж на то пошло, давайте создадим каталог с именем *~/programs*. Находясь в домашнем каталоге, можно ввести либо:

```
$ mkdir programs
```

либо полный путь:

```
$ mkdir /home/mdw/programs
```

Теперь перейдем в новый каталог:

```
$ cd programs
$ pwd
/home/mdw/programs
```

Специальная последовательность символов «..» относится к каталогу, расположенному непосредственно выше текущего, или «родительскому». Поэтому в домашний каталог можно вернуться, если ввести следующее:

```
$ cd ..
```

Противоположностью *mkdir* служит команда *rmdir*, которая удаляет каталоги:

```
$ rmdir programs
```

Аналогично команда *rm* удаляет файлы. Мы не говорим о ней сейчас, поскольку не показали еще, как создавать файлы. Для этого обычно используются редакторы *vi* или Emacs (см. главу 9), но создавать файлы можно и с помощью некоторых команд, о которых будет рассказано далее в этой главе. Выполненная с ключом *-r* (recursive), команда *rm* удаляет весь каталог и его содержимое (будьте осторожны!).

Вывод списка файлов

Чтобы посмотреть содержимое каталога, введите *ls*. Команда *ls* без параметров показывает содержимое текущего каталога. Задав аргумент, можно увидеть содержимое другого каталога:

```
$ ls /home
```

В некоторых системах команда *ls* причудлива и отображает особые файлы, такие как каталоги и исполняемые файлы, полужирным шрифтом или даже различными цветами. Если вы хотите изменить цвета, используемые по умолчанию, отредактируйте файл */etc/DIR_COLORS* или создайте в своем исходном каталоге его копию с именем *.dir_colors* и отредактируйте ее.

Как и для большинства команд Unix, можно управлять *ls* с помощью параметров, начинающихся с дефиса (-). Не забывайте вводить перед дефисом пробел. Полезным параметром является *-a* (all), который позволит вам увидеть в своем домашнем каталоге богатства, о которых вы и не подозревали:

```
$ cd
$ ls -a
.                    .bashrc              .fvwmrc
..                   .emacs               .xinitrc
.bash_history        .exrc
```

Одна точка означает ссылку на текущий каталог, а две точки – на каталог, находящийся уровнем выше. Но что это еще за файлы, начинающиеся с точки? Они называются скрытыми. Точка перед именем предотвращает их показ при выполнении обычной команды *ls*. Многие программы используют скрытые файлы для определения пользовательских параметров – описания

их поведения по умолчанию, которое можно изменить. Например, в файл *.Xdefaults* можно поместить команды, которые изменяют характер выполнения программ, использующих X Window System. Поэтому о существовании этих файлов обычно можно забыть, но при конфигурировании системы они оказываются очень важны. Некоторые из них мы позднее перечислим.

Другим полезным параметром *ls* является *-l* (long). Он позволяет вывести дополнительные сведения о файлах. На рис. 4.1 показан типичный вывод с объяснением смысла всех полей.

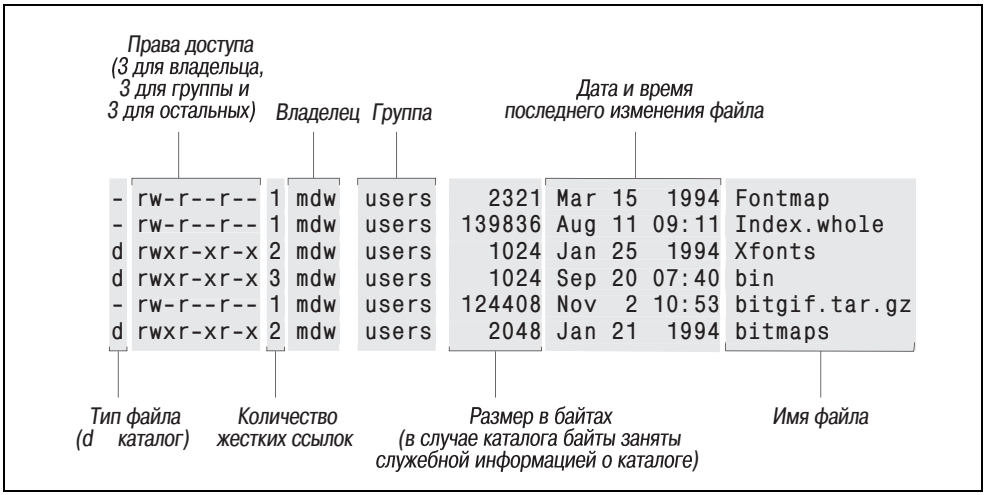


Рис. 4.1. *Результат выполнения команды ls -l*

О полях «Права доступа», «Владелец» и «Группа» мы расскажем ниже в разделе «Владение файлами и права доступа». Команда *ls* показывает также размер каждого файла и время его последнего изменения.

Просмотр файлов, команды more и less

Один из способов увидеть файл – это вызвать редактор, например:

```
$ emacs .bashrc
```

Однако если вам нужно лишь просмотреть файл, а не редактировать его, это можно сделать быстрее с помощью других команд. Проще всего использовать команду со странным именем *cat* (от *concatenate*, поскольку ее можно также использовать для конкатенации, т. е. соединения, нескольких файлов в один):

```
$ cat .bashrc
```

Длинный файл промелькнет слишком быстро, чтобы его можно было разглядеть, поэтому вместо *cat* чаще используется команда *more* («более»):

```
$ more .bashrc
```

Эта команда выводит часть содержимого файла, умещающуюся на целом экране, и ждет нажатия клавиши «пробел» для вывода очередной порции.

У команды *more* много мощных параметров. Например, можно осуществлять поиск строки в файле: нажмите клавишу «косая черта» (/), введите строку и нажмите <Enter>.

Популярной разновидностью команды *more* является команда *less* («менее»). У нее еще более мощные функции. Например, она позволяет пометить в файле некоторое место и позднее вернуться туда.

Символические ссылки

Иногда удобно хранить файл в одном месте так, чтобы казалось, будто он находится в другом. Чаще всего это делает не пользователь, а системный администратор. Например, у вас может быть несколько версий некоторой программы с именами *prog.0.9*, *prog.1.1* и т. д., но обращаться к используемой в данное время версии нужно по имени *prog*. Либо вы можете поместить файл в некоторый дисковый раздел, где есть свободное пространство, а использующей этот файл программе нужно, чтобы он находился в другом месте, поскольку путь к нему жестко зашит в коде.

Для таких случаев в Unix предусмотрены *ссылки* (*links*). В этом разделе мы рассмотрим *символические ссылки*, являющиеся наиболее гибким и распространенным типом ссылок. Символическая ссылка – это, в своем роде, фиктивный файл, просто указывающий на другой файл. Если вы пытаетесь редактировать, читать или выполнять символическую ссылку, то система поймет, что вам нужна не сама ссылка, а реальный файл. Символические ссылки во многом похожи на ярлыки в Windows 95/98, но обладают значительно большей мощью.

Вернемся к примеру с *prog*. Нужно создать ссылку с именем *prog*, указывающую на реальный файл с именем *prog.1.1*. Введите команду:

```
$ ln -s prog.1.1 prog
```

В результате создается новый файл с именем *prog*, являющийся в некотором роде фиктивным: при его запуске в действительности запустится *prog.1.1*. Посмотрим, что *ls -l* может сказать об этом файле:

```
$ ls -l prog
lrwxrwxrwx   2 mdw      users           8 Nov 17 14:35 prog -> prog.1.1
```

Символ `l` в начале строки показывает, что файл является ссылкой, а стрелка `->` обозначает реальный файл, на который ссылка указывает.

Пользоваться символическими ссылками очень просто, если освоиться с идеей о том, что один файл может указывать на другой. При установке программных пакетов приходится постоянно сталкиваться со ссылками.

Оболочки

Как уже говорилось, после входа в систему в консольном режиме вы попадаете в оболочку. Если ваша система сконфигурирована так, чтобы использовать графический менеджер регистрации, то после входа в систему вы по-

падаете в графический интерфейс, в котором для получения доступа к оболочке можно открыть окно *xterm* или аналогичное. Оболочка интерпретирует и выполняет все ваши команды. Прежде чем продолжить, взглянем на различные имеющиеся оболочки, поскольку они связаны с частью последующего материала.

Наличие в Unix многих различных оболочек может привести в замешательство, но к этому нужно относиться как к результату эволюционного развития. Можете поверить, что вам не захотелось бы остановиться на самой первой разработанной для Unix оболочке, оболочке Борна (Bourne shell). Хотя для своего времени (середина 70-х) это был очень мощный интерфейс пользователя, в нем недоставало многих полезных функций для интерактивного использования, в том числе показанных в этом разделе. Поэтому со временем были разработаны другие оболочки, и сегодня можно выбрать из них ту, которая более всего подходит вашему стилю работы.

Ниже перечислены некоторые оболочки, имеющиеся в Linux:

bash

Bourne Again shell (еще одна оболочка Борна). Наиболее часто используемая и самая мощная оболочка в Linux. POSIX-совместимая, совместимая с оболочкой Борна, создана и распространяется в рамках проекта GNU (Фонд свободного программного обеспечения). Поддерживает редактирование в командной строке, подстановку строк из истории и совместимость с оболочкой Борна.

csh

Оболочка C. Разработана в Беркли. В основном совместима с оболочкой Борна при интерактивном использовании, но имеет совершенно другой интерфейс программирования. Нет редактирования в командной строке, но имеется очень сложная альтернатива, называемая подстановкой истории. В Linux *csh* служит просто другим именем для более новой *tcsh*.

ksh

Оболочка Корна (Korn shell). Вероятно, наиболее популярная на Unix-системах в целом и первая, в которой в оболочку Борна были введены современные технологии оболочки (в том числе заимствованные из оболочки C). Совместима с оболочкой Борна. Поддерживает редактирование в командной строке.

sh

Оболочка Борна (Bourne shell). Первоначально разработанная оболочка. Не поддерживает редактирование в командной строке.

tcsh

Расширенная оболочка C. Поддерживает редактирование в командной строке.

zsh

Оболочка Z. Самая новая из оболочек. Совместима с оболочкой Борна. Поддерживает редактирование в командной строке.

Определить, какая оболочка запущена, можно с помощью следующей команды, которая покажет полный путь к месту нахождения оболочки (не пропустите знак доллара):

```
$ echo $SHELL
```

Скорее всего, у вас работает оболочка Борна, *bash*. Если какая-либо другая, то самое время перейти на *bash*. Она мощная, POSIX-совместимая, хорошо поддерживается и очень популярна на Linux. Чтобы сменить оболочку, выполните команду *chsh*:

```
$ chsh
Enter password: Здесь нужно ввести свой пароль - это в целях безопасности
Changing the login shell for mdw
Enter the new value, or press return for the default

        Login Shell [/bin/sh]:/bin/bash
```

Чтобы пользователь мог выбрать некоторую оболочку, запускаемую после регистрации, ее нужно установить, а системный администратор должен сделать ее доступной, поместив в */etc/shells*.

Оболочки можно разбить на классы, пользуясь разными критериями. Во-первых, можно разделить оболочки на Борн-совместимые и *csh*-совместимые. Такое разделение представляет интерес, когда вы начинаете программировать в оболочке, т. е. создавать «сценарии оболочки». Оболочки Борна и С имеют различные языковые конструкции. По мнению большинства Борн-совместимые оболочки предпочтительнее, и многие утилиты Unix признают только оболочку Борна.

Иначе можно разделить оболочки на те, которые поддерживают редактирование в командной строке (все новейшие), и те, которые не поддерживают. Оболочки *sh* и *csh* не поддерживают этой полезной функции.

При объединении обоих критериев – совместимости с оболочкой Борна и редактирования в командной строке – лучшим выбором становятся *bash*, *ksh* или *zsh*. Попробуйте несколько оболочек, прежде чем остановиться на одной из них. Полезно ознакомиться с несколькими оболочками на тот случай, если когда-нибудь вы столкнетесь с машиной, на которой выбор оболочек ограничен.

Полезные клавиши и как ими пользоваться

При вводе команды нажатие клавиши <Backspace> удаляет последний символ. Комбинация <Ctrl>+<U> удаляет строку от курсора до ее начала, поэтому при нахождении курсора в конце строки удаляется вся строка.[1] Если вы закончили ввод команды и она начала выполняться, то при нажатии <Ctrl>+<C> ее выполнение должно быть прекращено, а при нажатии

[1] <Ctrl>+<U> означает, что, удерживая в нажатом состоянии <Ctrl>, нужно нажать клавишу <u>.

<Ctrl>+<Z> – приостановлено. При желании возобновить выполнение приостановленной программы введите *fg* (от «foreground», передний план).

<Ctrl>+<S> останавливает вывод на экран, пока он не будет продолжен с помощью <Ctrl>+<Q>. Сегодня это, возможно, не так полезно, поскольку в большинстве эмуляций терминалов есть средства прокрутки экрана, но об этой комбинации нужно помнить на тот случай, если вы случайно нажмете <Ctrl>+<S> и терминал внезапно «замрет»; тогда нужно просто нажать <Ctrl>+<Q>, и он снова оживет.

Если некоторые из этих комбинаций не работают, значит, в силу каких-то причин ваш терминал неверно сконфигурирован. Исправить это можно командой *stty*, используя следующий синтаксис:

```
stty function key
```

где *function* – то, что вы хотите сделать, а *key* – клавиша, которую нужно нажать. Клавиша <Ctrl> обозначается диакритическим знаком «крышечка» (^) перед клавишей.

Вот группа команд для установки описанных выше функций:

```
$ stty erase ^H
$ stty kill ^U
$ stty intr ^C
$ stty susp ^Z
```

Первая показанная клавиша, ^H, представляет ASCII-код, генерируемый клавишей <Backspace>.

Кстати, список текущих установок терминала можно получить, введя команду *stty -a*. Разобраться в выводе не так просто: *stty* – сложная команда, имеющая многочисленные применения, и для некоторых из них требуются обширные познания в терминалах.

Сокращенный ввод с клавиатуры

Если вы повторяете команды этого руководства, работая за терминалом, то, вероятно, уже устали неоднократно вводить одно и то же. Особенно неприятно сделать ошибку и начать все сначала. Здесь оболочка может облегчить жизнь. Она не упрощает Unix до интерфейса «укажи и щелкни», но может помочь быстрой работе в командной среде.

В этом разделе рассказывается о редактировании командной строки. Эти советы помогут, если у вас оболочка *bash*, *ksh*, *tcsh* или *zsh*. Редактирование командной строки рассматривает последние 50 строк (или около того) как буфер в редакторе. Можно перемещаться по этим строкам и изменять их так же, как при редактировании документа. При каждом нажатии клавиши <Return> оболочка выполняет текущую строку.

Завершение слов

Сначала попробуем что-нибудь простое, но сберегающее массу времени. Введите следующие символы, не нажимая клавишу <Return>:

```
$ cd /usr/inc
```

Теперь нажмите клавишу <Tab>. Оболочка добавит `lude` до полного имени каталога */usr/include*. Теперь можно нажать <Return>, и команда будет выполнена.

Критерием определения имени файла является «минимальная полнота». Введите достаточное количество символов, чтобы имя было отличным от всех других в этом каталоге. Оболочка сможет найти имя и завершить его – вплоть до косой черты, если имя является каталогом.

Можно использовать завершение и с командами. Например, если ввести

```
$ ema
```

и нажать клавишу <Tab>, оболочка допишет `cs` до команды `emacs` (если только какая-либо другая команда в вашем каталоге не начинается с `ema`).

А что происходит, если есть несколько файлов, соответствующих введенным символам? Если все они начинаются с одних и тех же символов, оболочка допишет слово до того места, где имена файлов начнут различаться. Больше, как правило, оболочки ничего не делают. В *bash* есть изящное усовершенствование: при двойном нажатии <Tab> выводятся все возможные полные слова. Например, вы вводите

```
$ cd /usr/l
```

и дважды нажимаете <Tab>, тогда *bash* выводит что-нибудь типа

```
lib         local
```

Перемещение по командам

Нажмите клавишу «стрелка вверх», и появится предыдущая введенная вами команда. «Стрелка вверх» перемещает вас по буферу команд назад, в то время как «стрелка вниз» перемещает вперед. Если нужно изменить символ в текущей строке, используйте левую или правую стрелку.

Предположим, вы попытались выполнить команду:

```
$ mroe .bashrc
bash: mroe: command not found
```

Ясно, вы напечатали `mroe` вместо `more`. Чтобы исправить команду, вызовите ее снова стрелкой вверх. Затем нажимайте левую стрелку, пока курсор не переместится на `o` в `mroe`. Можно удалить `o` и `r` клавишей <Backspace> и правильно ввести их заново. Есть еще более изящная комбинация: нажмите <Ctrl>+<T>. В результате `o` и `r` поменяются местами, и можно нажать <Return>, чтобы выполнить команду.

Для редактирования командной строки есть много других комбинаций клавиш, но даже показанные основы значительно облегчат вам жизнь. Если вы изучите редактор Emacs, то обнаружите, что большая часть его клавиш одинаково работает в оболочке. А если вы любитель *vi*, то можете настроить оболочку так, что она будет использовать привязку клавиш *vi*, а не Emacs. Для этого в *bash*, *ksh* или *zsh* нужно ввести команду

```
$ export VISUAL=vi
```

В *tcsh* введите:

```
$ setenv VISUAL vi
```

Расширение имен файлов

Еще один способ сократить время при вводе – это использование специальных символов для сокращения имен файлов. С помощью этих символов можно задать сразу много файлов. Эта функция оболочки иногда называется «globbing» (универсализация файловых имен).

В интерпретаторе командной строки Windows есть несколько функций такого типа, но они менее развиты. В нем можно использовать знак вопроса, означающий «любой символ», и звездочку, означающую «любую строку символов». В Unix тоже есть эти символы-заместители, но они используются более ясно и строго.

Допустим, у вас есть каталог, содержащий следующие файлы с исходными текстами на С:

```
$ ls
inv1jig.c   inv2jig.c   inv3jig.c   invinitjig.c   invpar.c
```

Чтобы перечислить три файла, содержащие в именах цифры, можно ввести:

```
$ ls inv?jig.c
inv1jig.c   inv2jig.c   inv3jig.c
```

Оболочка ищет единственный символ, которым можно заменить знак вопроса. Поэтому она покажет *inv1jig.c*, *inv2jig.c* и *inv3jig.c*, но не *invinitjig.c*, поскольку в последнем имени слишком много символов.

Если второй файл вас не интересует, можно получить нужные файлы с помощью квадратных скобок:

```
$ ls inv[13]jig.c
inv1jig.c   inv3jig.c
```

Если после подстановки какого-либо символа из скобок полученный файл соответствует имеющемуся, то имя этого файла выводится. Можно задавать в скобках диапазон символов:

```
$ ls inv[1-3]jig.c
inv1jig.c   inv2jig.c   inv3jig.c
```

У нас снова выводятся три файла. Дефис означает «совпадение с любым символом от 1 до 3 включительно». Можно потребовать совпадения с любой цифрой, указав `0-9`, и с любой буквой, указав `[a-zA-Z]`. В последнем случае нужно задать два диапазона, поскольку оболочка различает верхний и нижний регистры. Кстати, используемое упорядочение основано на наборе символов ASCII.

Предположим, что вы хотите увидеть и файл *init*. Тогда можно воспользоваться звездочкой, поскольку требуется соответствие с любым числом символов между `inv` и `jig`:

```
$ ls inv*jig.c
inv1jig.c   inv2jig.c   inv3jig.c   invinitjig.c
```

Звездочка фактически означает «ноль или больше символов», поэтому, если бы существовал файл с именем *invjig.c*, он тоже был бы показан.

В отличие от интерпретатора командной строки Windows, оболочки Unix позволяют использовать специальные и обычные символы в любом сочетании. Скажем, вы хотите увидеть все исходные (*.c*) или объектные (*.o*) файлы, содержащие в имени цифру. В следующем шаблоне соединены вместе все шаблоны, изученные нами в этом разделе:

```
$ ls *[0-9]*.[co]
```

Расширения имен файлов очень удобно использовать в сценариях оболочки (программах), когда точно не известно, сколько существует файлов. Пусть, например, нужно обработать несколько файлов журналов с именами *log001*, *log002* и т. д. Сколько бы их ни было, все они соответствуют шаблону *log**.

Расширения имен файлов – это не то же самое, что регулярные выражения, используемые во многих утилитах для задания групп строк. Рассмотрение регулярных выражений выходит за рамки этой книги, но они описаны во многих книгах, рассказывающих об утилитах Unix. Мы столкнемся с регулярными выражениями в главе 9.

Сохранение выводимых данных

По экрану компьютера системного администратора (да и простого человека) проносится масса важных сообщений. Часто желательно сохранить эти сообщения, чтобы позднее изучить или, нередко, послать другу, который сможет разобраться, что же произошло. Поэтому мы немного расскажем в этой главе о перенаправлении – мощной функции, поддерживаемой оболочками Unix. Если вы знакомы с MS-DOS, то, вероятно, сталкивались с подобным, но более ограниченным типом перенаправления.

Если после любой команды поместить знак «больше» (>) и имя файла, то данные, выводимые этой командой, будут посланы в этот файл. Например, чтобы перехватить вывод *ls*, можно ввести:

```
$ ls /usr/bin > ~/Binaries
```

Перечень содержимого каталога */usr/bin* будет записан в файл *Binaries* в вашем исходном каталоге. Если *Binaries* уже существует, то > сотрет его содержимое и заменит выводом *ls*. Перезапись существующего файла является частой ошибкой пользователей. Если вы работаете в оболочке *csh* или *tcsh*, перезапись можно предотвратить командой

```
$ set noclobber
```

В *bash* того же можно достичь, введя:

```
$ noclobber=1
```
Необязательно 1, годится любое ненулевое число

Другим и более полезным способом предотвращения перезаписи является добавление новых данных к файлу. Например, сохранив содержание каталога */usr/bin*, мы хотим добавить к тому же файлу содержание */bin*. Можно дописать его в конец файла *Binaries*, введя два знака «больше»:

```
$ ls /bin >> ~/Binaries
```

Прием перенаправления вывода оказывается очень полезным, когда нужно неоднократно выполнить одну и ту же утилиту и сохранить вывод для анализа ошибок.

В большинстве программ Unix есть два потока вывода. Один называется стандартным выводом (standard output), а другой – стандартным выводом ошибок (standard error). Если вы программируете на C, то сразу их узнаете: стандартный поток вывода ошибок – это файл *stderr*, в который выводятся сообщения.

Символ > не перенаправляет поток вывода ошибок. И это удобно, если вам нужно сохранить нормальный вывод, не портя файла с сообщениями об ошибках. Но что если нужно сохранить как раз сообщения об ошибках, что часто бывает при поиске неполадок? Решением является использование знака «больше» с амперсандом (эта конструкция работает почти во всех современных оболочках). При этом перенаправляются как поток стандартного вывода, так и поток вывода ошибок. Например:

```
$ gcc invinitjig.c >& error-msg
```

Эта команда сохраняет все сообщения компилятора *gcc* в файле с именем *error-msg*. В оболочке Борна и *bash* тот же результат можно получить немного иначе:

```
$ gcc invinitjig.c &> error-msg
```

Теперь представим, что у нас возник такой каприз: сохранить сообщения об ошибках, а не обычный вывод, т. е. поток вывода ошибок, а не поток стандартного вывода. В Борн-совместимых оболочках это можно сделать, введя:

```
$ gcc invinitjig.c 2> error-msg
```

Оболочка присваивает номер 1 потоку стандартного вывода и номер 2 потоку вывода ошибок. Поэтому описанная команда сохраняет только поток вывода ошибок.

Наконец, допустим, что вы захотели подавить поток стандартного вывода – не дать выводу появиться на экране. Решение состоит в том, чтобы перенаправить его в особый файл с именем */dev/null*. (Вы слышали выражения типа «Пошлите свою критику в */dev/null*»? Вот отсюда идет эта фраза.) Каталог */dev* – это место, где Unix хранит специальные файлы, связанные с терминалами, ленточными приводами и другими устройствами. Но */dev/null* уникально: это место, где объекты исчезают, как в черной дыре. Например, следующая команда сохраняет поток вывода ошибок и подавляет поток стандартного вывода:

```
$ gcc invinitjig.c 2>error-msg >/dev/null
```

Теперь вы можете выделить именно тот вывод, который вам нужен.

Если вас заинтересовало, имеет ли знак «меньше» какой-либо смысл для оболочки, то ответ утвердительный. Этот знак заставляет команды принимать свои входные данные из файла. Однако большинство команд и так позволяет задавать входные файлы в командной строке, поэтому в «перенаправлении ввода» редко возникает необходимость.

Иногда требуется, чтобы одна утилита обрабатывала данные, выводимые другой утилитой. Например, с помощью команды *sort* можно упорядочить выдачу других команд. Грубый способ добиться этого – сохранить вывод команды в файле и обработать этот файл командой *sort*. Например:

```
$ du > du_output
$ sort -n du_output
```

Unix предоставляет более краткий и эффективный способ выполнения таких операций, используя *канал* (*pipe*). Нужно лишь поместить вертикальную черту между первой и второй командами:

```
$ du | sort -n
```

Оболочка пошлет весь вывод программы *du* программе *sort*.

В этом примере команда *du* означает «disk usage» (использование диска) и показывает, сколько блоков занимает каждый файл в текущем каталоге. Обычно ее вывод имеет отчасти случайный порядок:

```
$ du
10        ./zoneinfo/Australia
13        ./zoneinfo/US
9         ./zoneinfo/Canada
4         ./zoneinfo/Mexico
5         ./zoneinfo/Brazil
3         ./zoneinfo/Chile
20        ./zoneinfo/SystemV
118       ./zoneinfo
298       ./ghostscript/doc
183       ./ghostscript/examples
3289      ./ghostscript/fonts
          .
          .
          .
```

Поэтому мы решили пропустить ее через команду *sort* с параметрами *-n* и *-r*. Параметр *-n* означает «сортировать в числовом (numerical) порядке», а не в порядке ASCII, используемом по умолчанию. Параметр *-r* означает «сортировать в обратном (reverse) порядке», чтобы вначале появлялись самые большие числа. В результате мы получим вывод, который быстро покажет, какие каталоги и файлы заняли больше всего места:

```
$ du | sort -rn
34368     .
16005     ./emacs
16003     ./emacs/20.4
13326     ./emacs/20.4/lisp
4039      ./ghostscript
3289      ./ghostscript/fonts
          .
          .
          .
```

Поскольку файлов очень много, лучше использовать второй канал для команды *more* (одно из наиболее частых применений каналов):

```
$ du | sort -rn | more
34368     .
16005     ./emacs
16003     ./emacs/20.4
13326     ./emacs/20.4/lisp
4039      ./ghostscript
3289      ./ghostscript/fonts
          .
          .
          .
```

Альтернативой *more* может быть команда *head*, показывающая только несколько первых строк (по умолчанию 10). Конечно, если есть *head*, должна быть и команда *tail*, которая показывает только несколько последних строк.

Что такое команда?

Мы говорили, что в Unix много команд и можно добавлять к ним новые команды. Это в корне отличает Unix от большинства операционных систем, содержащих строго ограниченный набор команд. Так что же такое команды Unix и как они хранятся? В Unix команда – это просто файл. Например, команда *ls* является исполняемым файлом, расположенным в каталоге *bin*. Поэтому можно ввести не *ls*, а полное имя, известное как *абсолютный путь*:

```
$ /bin/ls
```

Благодаря этому Unix является очень мощной и гибкой системой. Чтобы предоставить новую утилиту, системный администратор может просто установить ее в стандартный каталог, где размещаются команды. Могут существовать различные версии команды – например, можно предложить для тес-

тирования новую версию утилиты, оставив старую версию в другом месте, а пользователи выберут ту, которую захотят.

При этом часто возникает проблема: вы вводите команду, которая, как вы рассчитываете, есть в системе, но получаете такое сообщение, как «Not found» (не найдена). Проблема может быть в том, что команда размещается в таком каталоге, где оболочка ее не ищет. Список каталогов, где оболочка ищет команды, называется *путем поиска*. Введите следующую команду, чтобы увидеть свой путь поиска (не забудьте про символ доллара, иначе вы увидите не содержимое переменной окружения, а ее имя, которое вам и так известно!):

```
$ echo $PATH
/usr/local/bin:/usr/bin:/usr/X11R6/bin:/bin:/usr/lib/java/bin:\
/usr/games:/usr/bin/TeX:.
```

На эту выдачу нужно внимательно посмотреть. Она представляет собой ряд путей, разделенных двоеточиями. Первый путь у данного пользователя – */usr/local/bin*. Второй – */usr/bin* и т. д. Поэтому если существуют две версии команды, одна в */usr/local/bin*, а другая в */usr/bin*, выполнится та, которая находится в */usr/local/bin*. Последний путь в этом примере представлен просто точкой, что означает ссылку на текущий каталог. В отличие от командного интерпретатора Windows, Unix не выполняет поиск в вашем текущем каталоге автоматически. Для этого нужно дать явное указание, как здесь и показано. Некоторые считают, что разрешать поиск в текущем каталоге не очень хорошо по соображениям безопасности. (Нарушитель, получивший доступ к вашей учетной записи, может скопировать вредную программу в один из ваших рабочих каталогов.) Однако это относится в основном к суперпользователю, а обычным пользователям об этом обычно не нужно беспокоиться.

Если команда не найдена, нужно установить, где она находится, и добавить соответствующий каталог к своему пути. О местонахождении команды должно быть сказано на страницах руководства. Допустим, вы обнаружили ее в */usr/sbin*, где находится ряд команд системного администрирования. Вам ясно, что требуется доступ к этим командам системного администрирования, поэтому введите следующую команду (обратите внимание, перед первым словом PATH нет символа доллара, а перед вторым есть):

```
$ export PATH=$PATH:/usr/sbin
```

Эта команда добавляет */usr/sbin*, но делает его последним каталогом, в котором осуществляется поиск. Эта команда означает: «Сделать мой путь равным прежнему пути плюс */usr/sbin*».

Такая команда работает не во всех оболочках. Для большинства пользователей Linux, работающих в Борн-совместимых оболочках типа *bash*, она действует прекрасно, но в *csh* или *tcsh* вместо нее нужно ввести следующую команду:

```
set path = ( $PATH /usr/sbin )
```

Наконец, есть несколько команд, не являющихся файлами, и одной из них является *cd*. В большинстве своем эти команды оказывают воздействие на саму оболочку и потому должны быть поняты и выполнены самой оболочкой. Поскольку они являются частью оболочки, то называются встроенными командами.

Запуск команды в фоновом режиме

До появления системы X Window, с помощью которой стало легко запустить сразу несколько программ, пользователи Unix применяли функции многозадачности Unix, помещая в конце команды амперсанд, как показано в следующем примере:

```
$ gcc invinitjig.c &
[1] 21457
```

Амперсанд запускает команду в фоновом режиме, который означает, что приглашение оболочки возвращается, и вы можете снова выполнять другие команды, пока *gcc* компилирует вашу программу. Число `[1]` – это номер задания, присвоенный вашей команде. Число `21457` – это идентификатор процесса, о котором мы поговорим позже. Номера заданий присваиваются фоновым командам по порядку, и потому их легче запоминать и вводить, чем ID процессов.

Разумеется, многозадачность достигается не бесплатно. Чем больше команд запущено в фоновом режиме, тем медленнее работает система, поскольку чередует их выполнение.

Не следует запускать команду в фоновом режиме, если она должна получать данные, вводимые пользователем. Если сделать это, появится сообщение об ошибке:

```
Stopped (tty input)
```

Решить эту проблему можно, переведя задачу обратно в приоритетный режим командой *fg*. Если в фоновом режиме выполняется много команд, то выбрать одну из них можно, указав номер задания или ID процесса. Для нашей долгоиграющей команды *gcc* эквивалентны следующие команды:

```
$ fg %1
$ fg 21457
```

Не забудьте поставить знак процента перед номером задания: он позволяет отличать номера заданий от идентификаторов процесса.

Прекратить выполнение команды в фоновом режиме можно командой *kill*:

```
$ kill %1
```

Страницы руководства

Ваши возможности больше всего расширяются сведениями о том, как выполнять собственные исследования. Следуя этой заповеди, мы расскажем

вам сейчас об интерактивной системе помощи, которая встраивается в Unix-системы. Она называется страницами руководства (man pages).

В действительности страницы руководства не являются верхом совершенства, поскольку они кратки и предполагают наличие существенной подготовки в Unix. Каждая страница посвящена отдельной команде и редко способствует получению представления о том, зачем ее нужно использовать. Тем не менее страницы очень важны. В каждой Unix-системе команды могут иметь некоторые особенности, и страницы руководства – наиболее надежный способ выяснить, что делает именно ваша система. Участники Проекта документирования Linux заслуживают большой благодарности за то, что потратили огромное количество времени на создание страниц руководства. Справку о команде можно получить, введя, например, следующее:

```
$ man ls
```

Страницы руководства состоят из нескольких разделов в зависимости от задач. Команды пользователя находятся в разделе 1, системные вызовы Unix – в разделе 2 и т. д. Наибольший интерес должны вызывать разделы 1, 5 (форматы файлов) и 8 (команды системного администрирования). При интерактивном просмотре страниц номера разделов принципиально важны, так как их можно указывать при поиске команды:

```
$ man 1 ls
```

При чтении печатной копии руководства вы обнаружите, что оно фактически разделено на части согласно схеме нумерации. Иногда в различных разделах встречаются статьи с одинаковыми названиями. (Например, *chmod* является командой и системным вызовом.) Поэтому можно иногда увидеть название страницы, за которым в скобках следует номер раздела, например *ls(1)*.

Существует ситуация, в которой номер раздела необходимо вводить в командной строке, а именно, если для одного ключевого слова есть несколько страниц, например для команды с некоторым именем и системной функции с тем же именем. Предположим, что вы хотите посмотреть библиотечный вызов, а система показывает вам описание команды, поскольку по умолчанию сначала ищутся команды. Чтобы увидеть страницу руководства по системному вызову, нужно указать номер раздела, в котором она находится.

Взгляните на верхнюю часть страницы руководства. Первый заголовок – NAME (Название). Под ним расположено краткое описание темы в одну строку. Эти описания полезны, если вы не вполне уверены в том, что разыскиваете. Выберите какое-нибудь слово, связанное с тем, что вы ищете, и укажите его в команде *apropos*:

```
$ apropos edit
```

Эта команда показывает все страницы руководства, так или иначе связанные с редактированием. Алгоритм очень простой: *apropos* просто выводит все строки названий, содержащие запрошенную строку.

Многие другие утилиты, в особенности предлагаемые графическими средами, обсуждаемыми в главе 11, выводят страницы руководства в более привлекательном виде.

Как и команды, страницы руководства иногда располагаются в неожиданных местах. Например, некоторые специфические для сайта программы могут быть установлены в каталог */usr/local*, а их страницы руководства помещены в */usr/local/man*. Команда *man* не станет автоматически осуществлять поиск в */usr/local/man*, поэтому при запросе страницы руководства можно получить сообщение «No manual entry». Чтобы исправить положение, нужно задать все каталоги верхнего уровня для руководств в переменной MANPATH. Например, вам нужно вставить фактические каталоги, в которых размещены страницы руководств в вашей системе:

```
$ export MANPATH=/usr/man:/usr/local/man
```

Синтаксис такой же, как в PATH, описанной выше: каталоги разделены двоеточиями. Если вы работаете в оболочке *csh* или *tcsh*, тогда нужно ввести:

```
$ setenv MANPATH /usr/man:/usr/local/man
```

Есть еще одна переменная окружения, которую, возможно, желательно установить, – MANSECT. Она определяет порядок разделов при поиске статьи руководства. Например,

```
$ export MANSECT=«2:3:1:5:4:6:7:8:n:9»
```

устанавливает, что поиск начинается с раздела 2.

Вы прочли несколько страниц руководства и все же пребываете в недоумении? Они не предназначены быть введением в незнакомые темы. Достаньте хорошую книгу по Unix для начинающих и возвращайтесь к страницам руководства постепенно, по мере ознакомления с системой. Со временем они станут для вас незаменимыми.

Страницы руководства – не единственный источник информации по Unix. У программ проекта GNU часто есть страницы Info, которые читаются программой *info*. Например, чтобы прочесть страницы Info для команды *find*, нужно ввести:

```
info find
```

Программа *info* загадочна и имеет массу функций для навигации. Чтобы разобраться в ней, лучше всего, находясь в программе, нажать <Ctrl>+<H> и прочесть весь экран помощи. К счастью, есть программы, которые позволяют читать страницы Info с большей легкостью, например *tkinfo* и *kdehelp*. Эти команды имеют графический интерфейс в X Window System. Можно читать страницы Info и в Emacs (см. раздел «Редактор Emacs» в главе 9) либо использовать команду *pinfo*, имеющуюся в некоторых дистрибутивах Linux и работающую сходным с веб-броузером Lynx образом.

В последнее время документация все чаще поставляется в виде страниц HTML. Их можно читать любым веб-броузером (см. главу 16). Например,

в веб-броузере Konqueror вы выбираете «Open Location...» в меню «Location» и нажимаете кнопку с изображением папки, в результате чего открывается обычное окно диалога выбора файла, в котором можно выбрать нужный файл документации.

Владение файлом и права доступа

Понятия «владелец файла» и «права доступа» – центральные в вопросах обеспечения защиты. Их надо правильно устанавливать, даже если вы единственный пользователь, иначе могут произойти неприятности. С файлами, которые пользователи создают и используют ежедневно, эти атрибуты обычно работают, не требуя особых размышлений (хотя все же полезно иметь о них понятие). При администрировании системы все не так просто. Стоит назначить неверные права владения и доступа и можно попасть в обескураживающую ситуацию, не будучи, к примеру, в состоянии прочесть собственную почту. Как правило, сообщение

```
Permission denied
```

означает, что кто-то назначил права владения и доступа, ограничивающие доступ больше, чем нужно.

Права доступа

Права доступа (*permissions*) означают способ, которым кто-либо может использовать файл. В Unix есть три таких права:

- Право на *чтение* (*read*) означает разрешение просмотреть содержимое файла.
- Право на *запись* (*write*) означает разрешение изменить или удалить файл.
- Право на *исполнение* (*execute*) означает разрешение выполнять файл как программу.

При создании каждого файла система назначает некоторые права по умолчанию, которых достаточно в большинстве случаев. Например, вы получаете права на чтение и запись, но остальные получают только право на чтение. Если вы одержимы паранойей, то можете сделать так, что у остальных не будет вообще никаких прав.

Кроме того, большинство утилит умеет устанавливать права доступа. Например, компилятор при создании исполняемой программы автоматически устанавливает для нее право на исполнение.

Однако бывают случаи, когда установок по умолчанию недостаточно. Например, если вы создаете сценарий оболочки или программу на Perl, то должны присвоить себе право на их выполнение, иначе вы не сможете их запустить. Ниже в этой главе мы покажем, как это делать, но сначала разберемся с основными понятиями.

Для каталога права доступа имеют другой смысл:

- Право на чтение означает возможность вывода содержания каталога.

- Право на запись означает возможность добавления и удаления файлов в этом каталоге.
- Право на исполнение означает разрешение вывода сведений о файлах в этом каталоге.

Не ищите разницу между правом на чтение и на выполнение для каталога. Обычно они идут в связке: присвойте оба или не присваивайте ни одного.

Обратите внимание, что, дав кому-либо право добавлять файл в каталог, вы тем самым даете ему и право удалять оттуда файлы. Эти привилегии появляются совместно при присвоении права записи. Есть, однако, способ дать пользователям право совместного доступа к каталогу и помешать им удалять файлы друг друга. См. раздел «Обновление другого программного обеспечения» в главе 7.

В Unix помимо каталогов и обычных файлов, о которых мы до сих пор говорили, есть и другие файлы. Это специальные файлы (устройства), сокеты, символические ссылки и др., причем у каждого типа существуют свои правила в отношении прав доступа. Но знание каждого типа в подробностях от вас не требуется.

Владельцы и группы

А кто же получает эти права доступа? Чтобы позволить людям совместно работать, в Unix есть три уровня прав доступа: владелец, группа и все остальные. «Остальные» охватывает всех, у кого есть доступ к системе, но кто не является владельцем или членом группы.

Идея создания групп состоит в том, чтобы обеспечить некоторому числу пользователей, например группе программистов, доступ к файлу. Скажем, программист, создающий исходный код, может оставить за собой право на запись, но остальным членам своей группы дать право на чтение, используя разрешение для группы. Что касается «остальных», то у них может вообще не быть прав. (Вы уверены, что ваш код *этого* достоин?)

У каждого файла есть владелец и группа. Обычно владельцем является тот, кто создал файл. Каждый пользователь принадлежит также к некоторой группе по умолчанию, и эта группа назначается каждому файлу, создаваемому пользователем. Можно, однако, создать много групп и приписать каждого пользователя к нескольким группам. Изменяя группу, назначенную файлу, можно дать доступ любой совокупности пользователей по вашему желанию. Мы еще поговорим о группах, когда доберемся до раздела «Файл групп» в главе 5.

Теперь у нас есть все элементы системы безопасности: три права доступа (на чтение, на запись и на исполнение) и три уровня (пользователь, группа, остальные). Посмотрим на некоторые типичные файлы с точки зрения прав доступа.

На рис. 4.2 показана типичная исполняемая программа. Мы получили этот вывод, выполнив *ls* с параметром *-l*.

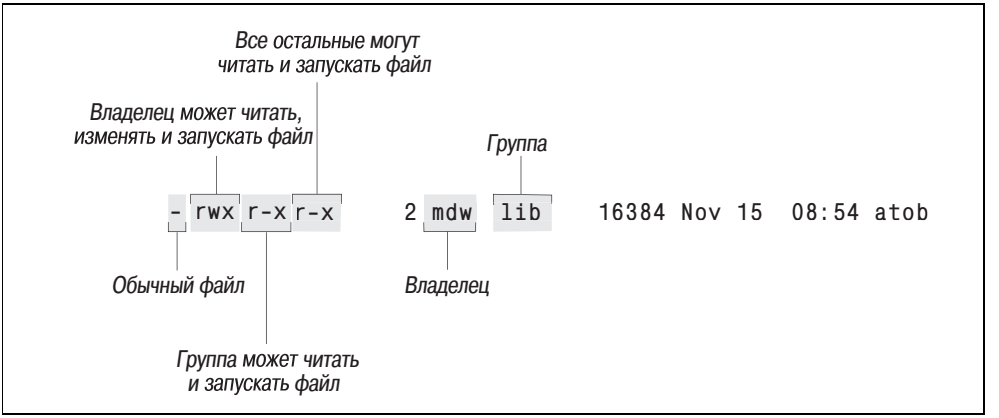


Рис. 4.2. *Отображение сведений о владельце и правах доступа*

Два полезных факта сразу обращают на себя внимание: владелец файла является автором данной книги и вашим верным спутником, `mdw`, в то время как группой является `lib` (возможно, группа, созданная для программистов, работающих над библиотеками). Но главная информация о правах доступа зашифрована в виде набора букв в левой части экрана.

Первый символ – дефис, указывающий на то, что это обычный файл. Следующие три позиции относятся к владельцу; как можно было ожидать, у `mdw` есть все три разрешения. Очередные три позиции относятся к членам группы: они могут читать файл (не очень полезное свойство для двоичного файла) и выполнять его, но не могут записывать в него, поскольку в поле, которое должно содержать `w`, помещен дефис. И последние три поля относятся ко «всем остальным»; в данном случае у них те же права, что и у группы.

Еще один пример: запросим «длинный» вывод информации о файле с исходным кодом C:

```
$ ls -l
-rw-rw-r--   1 kalle    kalle       12577 Apr 30 13:13 simc.c
```

Вывод показывает, что владелец, как и группа, обладает правами чтения и записи (`rw`). У всех остальных есть только право на чтение.

Предположим теперь, что мы скомпилируем файл и получим исполняемую программу. Компилятор *gcc* создаст файл *simc*:

```
$ gcc -osimc simc.c
$ ls -l
total 36
-rwxrwxr-x   1 kalle    kalle       19365 Apr 30 13:14 simc.c
-rw-rw-r--   1 kalle    kalle       12577 Apr 30 13:13 simc.c
```

В дополнение к битам разрешения чтения и записи *gcc* установил бит исполнения (`x`) для владельца, группы и остальных для исполняемого файла. Это нужно сделать, чтобы файл можно было запустить:

```
$ ./simc
(вывод программы)
```

Еще пример – типичный каталог:

```
drwxr-xr-x   2 mdw      lib           512 Jul 17 18:23 perl
```

Самый левый символ теперь – d, что указывает на каталог (directory). Разрешения на выполнение даны, поскольку вы хотите, чтобы все могли посмотреть, что лежит в каталоге.

Файлы иногда могут находиться в не вполне понятных состояниях, о которых здесь речь не идет; подробности можно прочесть на странице руководства, относящейся к *ls*. А сейчас пора посмотреть, как можно изменять владение и права доступа.

Изменение владельца, группы и прав доступа

Как было сказано, чаще всего можно обойтись правами по умолчанию, устанавливаемыми системой. Но всегда есть исключения, особенно для системных администраторов. Возьмем простой пример. Предположим, вы создаете каталог для нового пользователя в */home*. Вы должны создавать его будучи суперпользователем, но, закончив создание, передать владение пользователю, иначе тот не сможет пользоваться файлами! (К счастью, есть возможность воспользоваться командой *adduser*, которая описывается в разделе «Создание учетных записей» главы 5, – она позаботится за вас о владении.)

Аналогично, есть некоторые утилиты, такие как UUCP и News, у которых есть собственные пользователи. Никто не входит в систему под именем UUCP или News, но такие пользователи и группы должны существовать, чтобы утилиты могли работать, соблюдая защиту. Обычно последним шагом при установке программного обеспечения является изменение владельца, группы и прав доступа согласно указаниям в документации.

Команда *chown* изменяет владельца файла, а команда *chgrp* изменяет группу. В Linux только root может использовать *chown* для изменения владельца файла, но любой пользователь может поменять группу на другую, к которой принадлежит сам.

Итак, после установки некой программы с именем *sampsoft* можно заменить как владельца, так и группу на bin, выполнив команды:

```
# chown bin sampsoft
# chgrp bin sampsoft
```

Можно сделать это за один шаг, используя точечную нотацию:

```
# chown bin.bin sampsoft
```

Синтаксис изменения прав доступа более сложен. Права доступа можно назвать «режимом» файла, и команда, которая его изменяет, называется *chmod*. Начнем изучение этой команды с простого примера. Скажем, вы на-

писали на Perl или Tcl хорошую программу с именем *header* и хотите иметь возможность выполнить ее. Вы должны набрать следующую команду:

```
$ chmod +x header
```

Знак «плюс» означает «добавить разрешение», а `x` указывает, какое именно разрешение.

Если вы хотите снять разрешение на выполнение, используйте вместо плюса минус:

```
$ chmod -x header
```

Эта команда присваивает права доступа на всех уровнях – владельца, группы и остальных. Допустим, что вы тайно копите программы и не хотите, чтобы командой пользовался еще кто-либо, кроме вас. Нет, это слишком жестоко; скажем лучше, что вы боитесь, что в сценарии слишком много ошибок, и хотите защитить других пользователей от неприятностей, пока сами не испробуете его. Можно присвоить право выполнения только себе самому командой

```
$ chmod u+x header
```

То, что находится перед плюсом, обозначает уровень права доступа, а то, что после него – тип права доступа. Разрешение для пользователя (для себя) обозначается `u`, для группы – `g`, для остальных – `o`. Поэтому для присвоения права доступа себе и группе введите:

```
$ chmod ug+x header
```

Можно присваивать несколько прав доступа:

```
$ chmod ug+rwx header
```

Есть еще несколько сокращений, о которых вы можете узнать из страницы руководства для *chmod* и поразить кого-нибудь, глядящего через плечо на вашу работу, но они не добавляют функциональности помимо той, которую мы вам показали.

Каким бы таинственным ни казался синтаксис аргумента режима, есть другой синтаксис, еще более сложный. Придется, однако, по некоторым причинам описать и его. Во-первых, есть ситуации, которые не охватываются только что продемонстрированным синтаксисом, называемым *символическим режимом*. Во-вторых, в документации часто используется другой синтаксис, называемый *абсолютным режимом*. В-третьих, иногда абсолютный режим может оказаться более удобным.

Чтобы понять абсолютный режим, нужно думать на языке битов и восьмеричной нотации. Не пугайтесь, это не очень сложно. В обычном режиме есть три символа, относящиеся к уровням прав доступа (пользователь, группа и остальные). Эти уровни показаны на рис. 4.3. Внутри каждого уровня есть три бита, соответствующие правам на чтение, запись и исполнение.

ПОЛЬЗОВАТЕЛЬ			ГРУППА			ОСТАЛЬНЫЕ		
чтение	запись	исполнение	чтение	запись	исполнение	чтение	запись	исполнение
400	200	100	40	20	10	4	2	1

Рис. 4.3. *Биты в абсолютном режиме*

Допустим, вы хотите дать себе право на чтение, а всем остальным не давать никаких прав. Для этого нужно задать бит, представляемый числом 400, поэтому команда будет такой:

```
$ chmod 400 header
```

Чтобы дать всем право на чтение, выберите нужный бит для каждого уровня: 400 для себя самого, 40 для группы и 4 для остальных. Полностью команда выглядит так:

```
$ chmod 444 header
```

Эта команда аналогична использованию режима `+r`, за исключением того, что одновременно снимаются все права записи и выполнения. (Если быть точным, это все равно, что режим `=r`, о котором мы раньше не говорили. Знак равенства означает «присвоить эти права и никакие больше».)

Чтобы дать всем права на чтение и исполнение, нужно добавить биты чтения и исполнения. Например, 400 плюс 100 дает 500.

Поэтому соответствующая команда будет следующей:

```
$ chmod 555 header
```

Это то же самое, что `=rx`. Для того чтобы дать кому-либо полный доступ, нужно задать соответствующую цифру равной 7 – сумме 4, 2 и 1.

И под конец еще один трюк: задание режима по умолчанию, присваиваемого каждому создаваемому вами (с помощью текстового редактора, оператора перенаправления `>` и т. п.) файлу. Это делается командой *umask*, которую можно поместить и в ваш стартовый файл оболочки. Последний файл может называться *.bashrc*, *.cshrc* или как-либо иначе в зависимости от используемой оболочки (мы поговорим о стартовых файлах в следующем разделе).

Использование аргумента команды *umask* схоже с абсолютным режимом *chmod*, но значение битов инвертировано. Нужно определить тип доступа, который вы хотите установить для владельца, группы и остальных, и вычесть каждую цифру из 7. В результате получится маска из трех цифр.

Например, вы хотите себе предоставить все права (7), своей группе – право чтения и выполнения (5), а остальным – никаких прав (0). Вычтите каждую цифру из 7 и получите 0 – для себя, 2 – для своей группы, и 7 – для остальных. Поэтому в стартовый файл нужно поместить команду:

```
umask 027
```

Необычный прием, но работает. Команда *chmod* при интерпретации вашего режима смотрит на маску. Например, если при создании файла вы присваиваете ему исполняемый режим, право на исполнение будет присвоено вам, вашей группе и никому другому, поскольку маска не позволяет остальным иметь какой-либо доступ.

Стартовые файлы

Возможность настройки является сильной стороной Unix. Корни этого, возможно, лежат в двух характерных для хакеров чертах: они стремятся к полному контролю над своей средой и минимизации нажатий на клавиши и прочих необходимых телодвижений. Поэтому все основные утилиты в Unix – редакторы, почтовые программы, клиенты X Window System – имеют файлы, с помощью которых можно в значительной степени переопределить поведение по умолчанию. Имена этих файлов часто оканчиваются на *rc* (от *resource configuration* – конфигурации ресурсов).

Стартовые файлы обычно находятся в домашнем каталоге пользователя. Их имена начинаются с точки, благодаря чему команда *ls* обычно их не показывает. Наличие стартовых файлов не является необходимым: все связанные с ними программы используют значения по умолчанию, если файл не существует. Но иметь стартовые файлы всегда удобно. Вот некоторые распространенные стартовые файлы:

.bashrc

Для оболочки *bash*. Файл является сценарием оболочки, т. е. содержит команды и некоторые программные конструкции. Вот очень короткий стартовый файл, который мог быть помещен в ваш домашний каталог программой, создавшей вашу учетную запись:

```
PS1='\u$'              # В приглашение включается имя для входа в систему.

HISTSIZE=50            # Хранить 50 команд для вызова нажатием стрелки вверх.

# Список каталогов, в которых ищутся команды.
PATH=/usr/local/bin:/usr/bin:/bin:/usr/bin/X11

# Чтобы не допустить случайного окончания сеанса регистрации,
# отключить использование <Ctrl>+<D> для выхода.
IGNOREEOF=1

 stty erase "^H"       # Обеспечить стирание при нажатии backspace.
```

.bash_profile

Для оболочки *bash*. Это тоже сценарий оболочки. Различие между этим сценарием и *.bashrc* в том, что *.bash_profile* выполняется только при входе в систему. Он был разработан для того, чтобы можно было отделить интерактивные оболочки от тех, которые выполняются фоновыми обработчиками типа *cron* (описывается в главе 8). Но на современных компьютерах с X Window System он не очень полезен, поскольку при открытии ок-

на *xterm* выполняется только *.bashrc*. Если вы открываете окно командой *xterm -ls*, то *.bash_profile* тоже выполняется.

.cshrc

Для оболочки C или *tcsh*. Файл является сценарием, использующим языковые конструкции оболочки C.

.login

Для оболочки C или *tcsh*. Файл является сценарием, использующим языковые конструкции оболочки C. Как и *.bash_profile* в оболочке *bash*, он выполняется только при входе в систему. Вот несколько команд, которые могут быть в *.cshrc* или *.login*:

```
set prompt='% '        # Приглашение в виде %.

set history=50         # Хранить 50 команд, для вызова нажатием стрелки вверх.

# Список каталогов, в которых ищутся команды.
set path=(/usr/local/bin /usr/bin /bin /usr/bin/X11)

# Чтобы не допустить случайного окончания сеанса регистрации,
# отключить использование <Ctrl>+<D> для выхода.
set ignoreeof

# Обеспечить стирание при нажатии backspace.
stty erase "^H"
```

.emacs

Для редактора Emacs. Состоит из функций LISP. См. раздел «База данных ресурсов X» в главе 11.

.exrc

Для редактора *vi* (другое название – *ex*). Каждая строка является командой редактора. См. раздел «Основы настройки X» в главе 11.

.newsrc

Для чтения телеконференций. Содержит список всех телеконференций, имеющихся на сайте.

.Xdefaults

Для программ, использующих X Window System. Каждая строка задает ресурс (обычно имя программы и некоторое ее свойство), а также значение, которое этот ресурс должен принимать. Этот файл описан в разделе «База данных ресурсов X» главы 11.

.xinitrc

Для X Window System. Состоит из команд оболочки, которые выполняются при регистрации в сеансе X. Детали использования этого файла см. в разделе «Основные настройки X» главы 11.

.kde/share/config

Фактически это целый каталог с файлами конфигурации для графической среды K Desktop Environment (KDE). Там находится множество фай-

лов, имена которых начинаются с имени программы, настройки которой они содержат, а оканчиваются *rc*. Обычно не требуется редактировать эти файлы вручную: все соответствующие программы поставляются со своими утилитами настройки. В зависимости от версии KDE этот путь может начинаться с *.kde2* или *.kde3*.

.gnome

Как и в предыдущем случае, это целый каталог файлов конфигурации, на этот раз для графической среды GNOME.

Важные каталоги

Вам уже известен каталог */home*, в котором хранятся файлы пользователей. Для вас как системного администратора и программиста будут важны и некоторые другие каталоги. Вот некоторые из них и то, что в них содержится:

/bin

Основные команды Unix , такие как *ls*.

/usr/bin

Другие команды. Различие между */bin* и */usr/bin* произвольно; в старых системах Unix с маленькими дисками было удобно разделить команды.

/usr/sbin

Команды, используемые суперпользователем для системного администрирования.

/boot

Место, где иногда хранятся ядро и другие файлы, используемые при загрузке.

/etc

Файлы, используемые такими подсистемами, как сетевое взаимодействие, NFS и электронная почта. Обычно в них содержатся таблицы сетевых сервисов, монтируемых дисков и т. д. Многие файлы из этого каталога используются для загрузки системы или отдельных служб, и мы еще встретимся с ними.

/var

Административные файлы, такие как журналы, используемые различными утилитами.

/var/spool

Временное хранилище для файлов, передаваемых на печать, пересылаемых по UUCP и т. д.

/usr/lib

Стандартные библиотеки, например *libc.a*. При сборке программы компоновщик ищет здесь файлы, указанные в параметрах *-l*.

/usr/lib/X11

Дистрибутив X Window System. Содержит используемые клиентами X библиотеки, а также шрифты, образцы файлов ресурсов и другие важные части пакета X. Этот каталог обычно является символической ссылкой на */usr/X11R6/lib/X11*.

/usr/include

Стандартное местоположение заголовочных файлов программ на C, таких как *<stdio.h>*.

/usr/src

Расположение исходных текстов программ, собираемых в системе.

/usr/local

Программы и файлы данных, локально добавленные системным администратором.

/etc/skel

Примеры стартовых файлов, которые можно помещать в исходные каталоги новых пользователей.

/dev

Этот каталог содержит так называемые файлы устройств – интерфейс между файловой системой и аппаратурой (например, */dev/modem* представляет в системе модем).

/proc

Подобно тому как */dev* представляет интерфейс между файловой системой и аппаратурой, */proc* представляет интерфейс между файловой системой и выполняющимися процессами, ЦПУ и памятью. Содержащиеся здесь файлы (это не настоящие, а виртуальные файлы, генерируемые динамически, когда нужно их просмотреть) могут предоставить информацию об окружении определенного процесса, состоянии и конфигурации ЦПУ, настройке портов ввода/вывода и т. д.

Обслуживающие программы

Мы включили в книгу этот раздел, поскольку вам пора заинтересоваться тем, что же выполняется в системе неведомо для вас.

В наше время компьютер обычно производит более сложную работу, чем просмотр файла или другого статического ресурса. Иногда эта работа требует взаимодействия между работающими процессами.

Возьмем, например, протокол FTP, которым вы, возможно, пользовались при загрузке относящихся к Linux документов и программ. Когда вы подключаетесь по FTP к другой системе, на ней должна работать другая программа, принимающая ваше подключение и интерпретирующая ваши команды. Поэтому на той системе работает некоторая программа по имени *ftpd*. Буква *d* в имени происходит от *daemon* (*демон*), что является причудливым

термином Unix для обозначения сервера, постоянно работающего в фоновом режиме. Большинство демонов занято поддержкой работы в сети.

Вы, наверное, слишком часто слышали навязшее в зубах выражение «клиент/сервер», но это как раз то, что здесь применяется, более того, применяется десятки лет.

Демоны запускаются при загрузке системы. Чтобы увидеть, как они запускаются, посмотрите файлы */etc/inittab* и */etc/inetd.conf* и специфические для дистрибутива файлы конфигурации. Их формат мы сейчас не будем обсуждать. Но в каждой строке этих файлов указана программа, запускаемая при старте системы. Специфические для дистрибутива файлы можно найти, посмотрев поставляемую с системой документацию или обратив внимание на имена путей, чаще всего встречающихся в */etc/inittab*. Обычно они указывают на дерево каталогов, в которых ваш дистрибутив хранит стартовые файлы системы.

Чтобы увидеть пример того, как ваша система использует */etc/inittab*, взгляните на строки, содержащие `getty` или `agetty`. Это программа, которая слушает терминал (tty) и ждет регистрации пользователя. Это программа, которая выводит приглашение `login :`, о котором мы говорили в начале главы.

Файл */etc/inetd.conf* представляет собой более сложный способ запуска программ – еще один уровень перенаправления. Идея использования */etc/inetd.conf* состоит в том, что было бы напрасной тратой ресурсов системы, если бы десяток или более демонов крутились без дела, ожидая, пока поступит запрос из сети. Вместо этого система запускает один демон с именем *inetd*. Этот демон ждет поступления запросов на соединение от клиентов на других машинах и при осуществлении входящего соединения запускает для его обработки соответствующий демон. Например, при осуществлении входящего FTP-соединения *inetd* запускает FTP-демон (*ftpd*) для управления соединением. Благодаря этому выполняются только те сетевые демоны, которые действительно используются.

В следующем разделе мы покажем, как узнать, какие демоны действительно работают в системе. Для каждого сервиса, предлагаемого системой другим машинам в сети, есть свой демон: *fingerd* – для обработки удаленных запросов *finger*, *rwhod* – для обработки запросов *rwho*, и т. д. Некоторые демоны поддерживают несетевые сервисы, например *kerneld* занимается автоматической загрузкой модулей в ядро. (В версиях 2.4 и выше эту задачу решает *kmod*, который теперь к тому же не процесс, а поток ядра.)

Процессы

Понятие процесса является центральным в Unix. Освоение этого понятия поможет вам как пользователю управлять своим сеансом работы. Если вы также являетесь системным администратором, это понятие еще более важно.

Процесс – это независимо выполняющаяся программа со своими ресурсами. Например, выше мы показывали, как можно переназначить вывод програм-

мы в файл, в то время как оболочка продолжает вывод на экран. Возможность того, что оболочка и другая программа могут посылать выходные данные в разные места, обусловлена тем, что они представляют собой разные процессы. В Unix ограниченные ресурсы системы, такие как оперативная память и диски, управляются одной всемогущей программой, называемой ядром. Все остальное в системе есть процесс.

Таким образом, пока вы не вошли в систему, ваш терминал управляется процессом *getty*. После регистрации процесс *getty* умирает (при выходе из сеанса ядро запускает новый процесс), и ваш терминал управляется оболочкой, являющейся другим процессом. Оболочка создает новый процесс при каждом вводе команды. Создание нового процесса называется *ветвлением* (*forking*), поскольку один процесс расщепляется на два.

Если вы работаете в X Window System, то каждый процесс открывает одно или более окон. Так, владельцем окна, в котором вы вводите команды, является процесс *xterm*. Этот процесс разветвляет оболочку, чтобы она выполнялась в окне. А эта оболочка порождает дополнительные процессы, когда вы вводите команды.

Чтобы увидеть процессы, которые у вас выполняются, введите команду *ps*. На рис. 4.4 показаны типичные результаты ее работы и объяснено значение полей. Вас может удивить, как много процессов выполняется, особенно если вы работаете в X. Команда *ps* сама является одним из процессов, который, конечно, умирает, как только вывод данных осуществлен.

```
$   ps
    PID    TTY    STAT    TIME    COMMAND
   1663    pp3    S       0:01    -bash
   1672    pp3    T       0:07    emacs
   1676    pp3    R       0:00    ps
```

PID – *идентификатор процесса (используется для принудительного завершения процесса)*

TTY – *управляющий терминал*

TIME – *используемое процессорное время*

STAT – *состояние*

COMMAND – *запущенная команда*

Рис. 4.4. *Результат работы команды ps*

Первым полем вывода *ps* является уникальный идентификатор процесса. Если у вас есть вышедший из-под контроля процесс, от которого нельзя избавиться с помощью <Ctrl>+<C> или другим способом, можно убить его, перейдя в другую виртуальную консоль или окно X и введя:

```
$ kill process-id
```

Поле TTY показывает, на каком терминале работает процесс, если таковой имеется. (Все, что запускается из оболочки, использует, конечно, терминал, но у демонов, запущенных в фоновом режиме, нет терминала.)

Поле STAT показывает состояние процесса. Оболочка в данный момент находится в режиме ожидания (suspended), поэтому в этом поле стоит S. Выполняется сеанс редактирования в Emacs, но он приостановлен нажатием <Ctrl>+<Z>. Это отмечено знаком T в поле STAT. Последним показан процесс *ps*, создающий эту выдачу. Его состояние, конечно, R (running), поскольку он выполняется.

Поле TIME показывает, сколько процессорного времени использовали процессы. Поскольку *bash* и Emacs являются интерактивными, они мало используют процессор.

Можно посмотреть не только свои процессы, но и все процессы системы. Взглянем на все процессы в системе. Параметр *a* задает все процессы, а параметр *x* позволяет включить процессы, у которых нет управляющего терминала (такие как демоны, запущенные во время работы):

```
$ ps ax | more
```

Теперь вы можете увидеть демоны, о которых говорилось в предыдущем разделе.

В последних версиях команды *ps* есть приятный дополнительный параметр. Если вы ищете определенный процесс и знаете его имя или хотя бы часть его, можно воспользоваться параметром *-C*, указав за ним это имя, и увидеть только те процессы, имена которых соответствуют заданному:

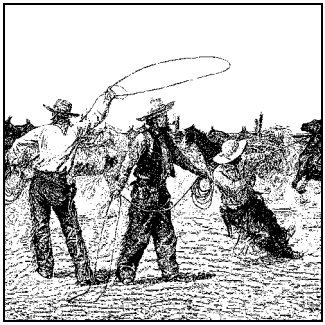
```
$ ps -C httpd
```

И на этом месте, глядя на захватывающую дыхание картину всей системы Unix за работой, мы заканчиваем эту главу (строки ограничены 76 колонками; если вы хотите увидеть их в полном блеске, добавьте к команде *ps* параметр *-w*):

```
kalle@tigger: > ps aux
kalle@owl:~ > ps aux
USER       PID %CPU %MEM   VSZ  RSS TTY      STAT START   TIME COMMAND
root         1  0.0  0.0   416   64 ?        S    Mar23   0:57 init [3]
root         2  0.0  0.0     0    0 ?        SW   Mar23  30:05 [kflushd]
root         3  0.0  0.0     0    0 ?        SW   Mar23  13:26 [kupdate]
root         4  0.0  0.0     0    0 ?        SW   Mar23  11:39 [kswapd]
root         5  0.0  0.0     0    0 ?        SW<  Mar23   0:00 [mdrecoveryd]
root         9  0.0  0.0     0    0 ?        SW   Mar23   0:00 [khubd]
bin       1512  0.0  0.0  1272    0 ?        SW   Mar23   0:00 [portmap]
root      1530  0.0  0.0  1328  216 ?        S    Mar23   0:17 /sbin/syslogd
root      1534  0.0  0.0  1728  180 ?        S    Mar23   0:03 /sbin/klogd -c 1
root      1560  0.0  0.0  1236    0 ?        SW   Mar23   0:00 [usbmgr]
root      1570  0.0  0.0  1728    0 ?        SW   Mar23   0:00 [nlservd]
at        1578  0.0  0.0  1388  108 ?        S    Mar23   0:00 /usr/sbin/atd
nobody    1585  0.0  0.0  5592   24 ?        S    Mar23   0:00 [in.identd]
nobody    1586  0.0  0.0  5592   24 ?        S    Mar23   0:49 [in.identd]
nobody    1587  0.0  0.0  5592   24 ?        S    Mar23   0:00 [in.identd]
nobody    1588  0.0  0.0  5592   24 ?        S    Mar23   0:00 [in.identd]
root      1618  0.0  0.0  1676   96 ?        S    Mar23   0:42 /usr/sbin/rpc.mou
```

```
root      1621  0.0  0.0  1696   92 ?        S    Mar23   0:41 /usr/sbin/rpc.nfs
root      1649  0.0  0.1  2352  340 ?        S    Mar23   0:23 sendmail: accepti
root      1657  0.0  0.1  2156  404 ?        S    Mar23   0:47 /usr/sbin/nmbd -D
root      1664  0.0  0.0  2652  196 ?        S    Mar23   0:00 [smbd]
root      1681  0.0  0.0  1408  164 ?        S    Mar23   0:15 /usr/sbin/cron
root      1689  0.0  0.0  1328   32 ?        S    Mar23   0:00 /usr/sbin/lpd
root      1713  0.0  0.1 11724  352 ?        S    Mar23   0:03 /usr/sbin/nscd
root      1731  0.0  0.0  1280  168 ?        S    Mar23   0:00 /usr/sbin/inetd
root      1795  0.0  0.0  1996    0 tty1     SW   Mar23   0:00 [login]
root      1796  0.0  0.0  1996    0 tty2     SW   Mar23   0:00 [login]
root      1798  0.0  0.0  1228    0 tty4     SW   Mar23   0:00 [mingetty]
root      1799  0.0  0.0  1228    0 tty5     SW   Mar23   0:00 [mingetty]
root      1800  0.0  0.0  1228    0 tty6     SW   Mar23   0:00 [mingetty]
kalle     2302  0.0  0.0  2584    0 tty1     SW   Mar23   0:00 [bash]
root      2309  0.0  0.0  2428    0 tty2     SW   Mar23   0:00 [bash]
root      2356  0.0  0.0  1292   56 ?        S    Mar23  19:32 /usr/sbin/gpm -t
root      5860  0.0  0.0  2108   88 ?        S    Mar23   0:01 [sshd]
root     21391  0.0  0.0  2004    0 tty3     SW   Mar30   0:00 [login]
kdeuser  21394  0.0  0.0  2548    0 tty3     SW   Mar30   0:00 [bash]
kalle    17613  0.0  0.0  1660    0 ?        SW   Mar31   0:00 [ssh-agent]
kalle    20007  0.0  0.1  2628  512 tty1     S    Apr03   0:00 /bin/bash
kalle    27278  0.0  0.4  2384 1140 tty1     S    Apr22   0:00 sh /usr/X11R6/bin
kalle    27279  0.0  0.2  1552  636 tty1     S    Apr22   0:00 tee /home/kalle/.
kalle    27291  0.0  0.3  2728  824 tty1     S    Apr22   0:00 xinit /home/kalle
root     27292 25.2  7.2 45724 19080 ?       S    Apr22 2531:10 X :0 -auth /home
kalle    27308  0.0  0.4  2376 1124 tty1     S    Apr22   0:00 bash /home/kalle/
kalle    27316  0.0  0.4  2500 1296 tty1     S    Apr22   0:00 bash --login /opt
kalle    27374  0.0  0.4 14048 1212 ?        S    Apr22   0:08 kdeinit: dcopserv
kalle    27376  0.0  0.6 14540 1812 ?        S    Apr22   0:01 kdeinit: klaunche
kalle    27378  0.0  0.7 13964 1852 ?        S    Apr22   5:36 kdeinit: kded
kalle    27381  0.1  2.3 18564 6252 ?        S    Apr22  12:03 kdeinit: kdesktop
kalle    27388  0.0  0.6 14164 1716 ?        S    Apr22   0:00 kdeinit: kxmlrpcd
kalle    27399  0.1  2.0 15324 5468 ?        S    Apr22  13:10 kdeinit: klipper
kalle    27401  0.0  1.4 14316 3920 ?        S    Apr22   0:02 kdeinit: khotkeys
kalle    27403  0.0  0.1 13608  296 ?        S    Apr22   0:00 kdeinit: Running.
kalle    27404  0.0  1.6 14516 4204 ?        S    Apr22   0:01 kdeinit: kwrited
kalle    27406  0.0  0.1  1440  520 pts/0    S    Apr22   0:00 /bin/cat
kalle    27409  0.0  2.1 13680 5664 tty1     S    Apr22   0:06 knotify
kalle    27413  0.0  1.8 10476 4844 tty1     S    Apr22   0:03 ksmserver --resto
kalle    27415  0.3  1.0 15032 2728 ?        S    Apr22  30:35 kdeinit: kwin -se
kalle    27426  0.0  0.9 15496 2436 ?        S    Apr22   0:29 kdeinit: konsole
kalle    27429  0.0  0.9 15456 2424 ?        S    Apr22   0:03 kdeinit: konsole
kalle    27430  0.0  0.5  2672 1568 pts/2    S    Apr22   0:00 /bin/bash
kalle    27431  0.0  0.6  2908 1776 pts/1    S    Apr22   0:01 /bin/bash
kalle    27489  0.0  2.1 15616 5648 ?        S    Apr22   0:06 kdeinit: kio_uise
kalle    27492  0.0  1.9 14636 5092 ?        S    Apr22   0:03 kdeinit: kcookiej
kalle    27494  0.0  0.7  9896 1916 ?        S    Apr22   0:05 kdesud
kalle    30812  0.0  1.1 17620 2900 pts/1    S    Apr23   6:01 kicker
kalle    30893  0.0  0.6 15556 1784 ?        S    Apr23   0:22 kdeinit: konsole
kalle    30894  0.0  0.6  2680 1580 pts/3    S    Apr23   0:00 /bin/bash
kalle    30902  0.0  2.7 13912 7132 pts/3    S    Apr23   0:26 xemacs
```

```
kalle    30924  0.0  7.7 36564 20152 ?       S    Apr23   1:29 konqueror -mimety
root      1573  0.0  1.2  4740 3320 ?        S    Apr23   1:07 /usr/sbin/smbd -D
kalle     5812  0.0  0.9 15600 2408 ?        S    Apr24   1:35 kdeinit: konsole
kalle     5813  0.0  0.6  2836 1720 pts/4    S    Apr24   0:00 /bin/bash
kalle     5820  0.1  5.3 18904 13908 pts/4   S    Apr24  12:13 xemacs
kalle     6100  0.0  0.1  1660  400 ?        S    Apr24   0:00 ssh-agent
kalle    11670  0.0  6.2 26836 16376 ?       S    Apr25   0:16 konqueror -mimety
kalle    13390  0.0  0.6 15596 1812 ?        S    Apr25   0:20 kdeinit: konsole
kalle    13391  0.0  0.6  2684 1592 pts/5    S    Apr25   0:00 /bin/bash
kalle    15039  0.0  0.1  1656  396 ?        S    Apr25   0:00 ssh-agent
kalle    15337  0.0  0.9 15836 2560 ?        S    Apr25   0:03 designer
kalle    18660  0.0  6.4 24592 16976 pts/5   S    Apr26   4:29 xemacs
kalle    22903  0.2 13.2 42784 34756 ?       S    Apr26  11:59 kmail -caption KM
kalle     8744  0.0  3.3 15336 8712 ?        S    10:09   0:22 ksnapshot
kalle    12077  4.3  4.4 14796 11756 ?       R    20:42   0:34 xemacs
kalle    12130  0.0  1.2 13692 3204 ?        S    20:50   0:00 kdeinit: kio_file
kalle    12132  0.0  1.2 13692 3204 ?        S    20:50   0:00 kdeinit: kio_file
kalle    12152  1.2  0.5  2560 1340 pts/6    S    20:55   0:00 /bin/bash -i
kalle    12162  0.0  0.3  2688  988 pts/6    R    20:56   0:00 ps aux
kalle@owl:~ >
/usr/sbin/
```



5

Основы управления системой

При запуске собственной Linux-системы вскоре возникает необходимость изучения способов системного администрирования. Вам не удастся долго обходиться без осуществления некоторых видов системного сопровождения, обновления программного обеспечения и просто мелочей, необходимых для поддержания дел в порядке.

Работа в Linux-системе не лишена сходства с ездой на мотоцикле и уходом за ним.[1] Многие любители мотоциклетной езды предпочитают сами заботиться о своей технике: регулярно чистить, заменять изношенные детали и т. д. Linux дает возможность осуществлять такого же рода практический уход за сложной операционной системой.

Хотя одержимый администратор может бесконечно настраивать систему, чтобы улучшить ее работу, на практике заниматься администрированием необходимо только в случае значительных событий: установки нового диска, появления нового пользователя или выхода системы из строя при неожиданном отказе питания. Все эти ситуации мы обсудим на протяжении последующих четырех глав.

Linux удивительно доступна во всех отношениях – от будничных задач обновления совместно используемых библиотек до таких, доступных лишь посвященным, задач, как копание в ядре. Ввиду доступности всего кода – а основная масса разработчиков и пользователей Linux всегда была из хакерской породы – системное сопровождение не только является частью повседневной работы, но и имеет большое познавательное значение. Поверьте нам: ничто не может сравниться с рассказом друзьям о том, как вы менее чем за полчаса подняли систему с XFree86 3.3.6 до XFree86 4.0.3 и при этом перекомпилировали ядро для поддержки файловой системы ISO 9660. (Они могут понятия не иметь, о чем вы им говорите; в таком случае подарите им экземпляр этой книги.)

---

[1] По крайней мере один из авторов свидетельствует о сильном сходстве, которое имеют между собой системное администрирование Linux и «Дзен-буддизм и искусство ухода за мотоциклом» Роберта Пирсига. Присуща ли Linux многоликость Будды?

В нескольких последующих главах мы будем исследовать вашу систему Linux с точки зрения механика: покажем, что находится под капотом, и объясним, как за всем этим ухаживать, включая обновление программного обеспечения, управление пользователями, файловыми системами и другими ресурсами, изготовление резервных копий и действия в экстренных ситуациях.

Если вы сделали нужные изменения в стартовых файлах, Linux, по большей части, будет работать самостоятельно. Пока вас устраивает системная конфигурация и программы, работающие в системе, особых трудозатрат не требуется. Однако мы хотим побудить пользователей Linux к экспериментам со своей системой и настройке ее согласно своим вкусам. Твердокаменность не свойственна Linux, и если что-либо работает не так, как вам того хотелось бы, наверняка есть возможность это изменить. Например, если вы предпочитаете мигающий зеленый цвет на голубом фоне, а не обычный белый на черном, мы покажем, как этого добиться. (Если вы обещаете никому не говорить, кто вас этому научил.) Но мы покажем вам кое-что более важное. После установки дистрибутива Linux обычно оказывается, что в системе работает масса ненужных служб (например, веб-сервер). Любая из этих служб может представлять угрозу безопасности, поэтому может потребоваться поковыряться в стартовых файлах, чтобы оставить работать только те службы, которые совершенно необходимы.

Следует отметить, что в Linux часто включаются замысловатые средства, упрощающие решение многих задач системного администрирования. В число таких средств входят YaST в системах SuSE, COAS в системах Caldera и ряд утилит в системах Red Hat. С их помощью можно делать все – от управления учетными записями пользователей до создания файловых систем и сборки мусора. Эти утилиты могут как облегчить, так и затруднить вашу жизнь, в зависимости от того, как на них взглянуть. В этой и последующих главах мы представим «нутро» системного администрирования, показав средства, которые должны иметься в любой системе Linux и в действительности почти в любой Unix-системе. Они составляют основу инструментального ящика администратора: его молоток, отвертку и торцевой ключ, благодаря которым можно рассчитывать справиться с задачей. Если вы хотите воспользоваться 40-сильной дисковой пилой, это ваше право, но всегда полезно уметь пользоваться ручным инструментом на случай отключения электроэнергии. При желании подробнее изучить другие темы по администрированию Unix мы рекомендуем книги Эви Немета (Evi Nemeth) и др. «Unix System Administration Handbook» (Справочник системного администратора Unix), Prentice-Hall, и Элин Фриш (Æleen Frisch) «Essential System Administration» (Основы системного администрирования), O'Reilly.

Сопровождение системы

Работа системного администратора любой Unix-системы в определенной мере требует чувства ответственности и осторожности. В равной степени это применимо к Linux, даже если вы – единственный пользователь системы.

Многие из задач системного администрирования выполняются пользователем root. В Unix-системах эта учетная запись обладает особыми правами, а именно: обычные права доступа к файлам и механизмы системы безопасности просто не применяются к root. Таким образом, root может обратиться к любому файлу системы вне зависимости от его принадлежности и модифицировать его. В то время как обычный пользователь не может повредить систему (например, путем повреждения файловой системы или действий с файлами других пользователей), root таких ограничений не имеет.

Почему в Unix защита стоит на первом месте? Наиболее очевидная причина этого в том, чтобы разрешить пользователям самим определять порядок доступа к своим файлам. Изменяя биты разрешения на доступ к файлу (командой *chmod*), пользователи могут указать, что некоторые файлы доступны только для чтения, записи или выполнения определенным группам других пользователей или недоступны вообще никому. Права доступа позволяют обеспечить конфиденциальность и целостность данных; например, вам бы не хотелось, чтобы другие пользователи могли читать вашу личную почту или редактировать неведомо для вас исходный код важной программы.

Механизмы защиты Unix предотвращают также повреждение системы пользователями. Система ограничивает прямой доступ ко многим файлам устройств (доступным через */dev*; подробнее об этом см. в разделе «Файлы устройств» главы 6), связанным с аппаратными устройствами, такими как жесткие диски. Если бы обычные пользователи имели возможность чтения и записи непосредственно на дисковые устройства, это могло бы привести к большим неприятностям, когда, например, содержимое диска оказалось бы полностью затертым. Вместо этого система требует, чтобы обычные пользователи обращались к диску через файловую систему, и при этом поддерживалась система защиты через биты прав доступа, описанных ранее.

Важно отметить, что не все виды «ущерба», который может быть нанесен, обязательно являются зловредными. Система безопасности скорее призвана защитить пользователей от собственных случайных ошибок и недосмотров, а не устанавливать полицейские порядки. И действительно во многих системах система защиты довольно слаба. Система безопасности Unix создана с целью обеспечить совместный доступ к данным группе пользователей, которые могут, например, вместе работать над некоторым проектом. Система позволяет включать пользователей в группы, а права доступа к файлу могут устанавливаться для группы в целом. Например, в некотором разрабатываемом проекте у одних пользователей могут быть права чтения и записи для ряда файлов, а другим пользователям может быть запрещено модифицировать эти файлы. В отношении личных файлов вы сами должны решить, какую степень открытости или закрытости должны обеспечивать разрешения.

Механизм системы защиты Unix не разрешает также обычным пользователям производить некоторые действия, например обращаться в программе к определенным системным вызовам. Например, существует системный вызов, выполняющий останов системы и используемый такими программами, как *shutdown* (подробнее об этом ниже), для перезагрузки системы. Если бы обыч-

ные пользователи могли обращаться к этой функции из своих программ, они могли бы в любой момент случайно (или преднамеренно) остановить систему.

Во многих случаях требуется обойти механизмы защиты Unix, чтобы выполнить операции по сопровождению системы или обновлению программного обеспечения. Для этого служит учетная запись root. Поскольку на нее не накладываются никакие ограничения, знающий системный администратор может без труда выполнить задачу, не заботясь об обычных правах доступа к файлам или других ограничениях. Обычно для регистрации в качестве root используется команда *su*. Эта команда позволяет действовать от лица другого пользователя, например

```
su andy
```

выведет приглашение для ввода пароля, принадлежащего andy, и если пароль верный, установит ID пользователя andy. Суперпользователю часто нужно временно зарегистрироваться в качестве другого пользователя, чтобы исправить проблемы с файлами этого пользователя или по каким-то аналогичным причинам. Если не указывать имя пользователя, то *su* пригласит ввести пароль для root, проверяя, являетесь ли вы пользователем root. Закончив пользоваться учетной записью root, вы обычным способом завершаете сеанс работы и возвращаетесь к собственному смертному обличью.

Почему просто не зарегистрироваться как root через обычное приглашение регистрации? Как мы увидим, в некоторых случаях это желательно, но, как правило, лучше зарегистрироваться под своим именем, а затем воспользоваться командой *su*. В системе со многими пользователями обращение к *su* создает в системных журналах, таких как */var/log/messages*, запись типа:

```
Nov  1 19:28:50 loomer su: mdw on /dev/ttyp1
```

(о файлах системных журналов мы поговорим позднее). Это сообщение говорит о том, что пользователь mdw успешно выполнил команду *su*, в данном случае для root. При непосредственной регистрации в качестве root такой записи в журналах не было бы, и какой пользователь регистрировался в качестве root, сказать было бы невозможно. Это важно, если на машине несколько администраторов: часто желательно знать, кто и когда пользовался *su*.

Есть еще один небольшой фокус, связанный с командой *su*. Выполнив ее так, как описано выше, вы измените свой ID пользователя, но при этом не получите все настройки, связанные с этим ID. У каждого пользователя могут быть особые файлы настройки (об их создании еще будет рассказано), но при таком использовании *su* они не будут выполняться. Чтобы эмулировать реальную регистрацию с выполнением всех файлов конфигурации, нужно добавить символ «-»:

```
su - andy
```

или

```
su -
```

чтобы стать root и выполнить файлы конфигурации root.

Учетную запись root можно считать волшебной палочкой – полезным и потенциально опасным средством. Неумелое обращение с волшебными словами, пока у вас в руках эта палочка, может принести несказанный вред системе. Например, всего лишь восемь символов rm -rf / уничтожат все файлы системы, если вы зарегистрированы как root и невнимательны. Эта проблема кажется искусственной? Отнюдь. Вы могли попытаться удалить старый каталог, такой как */usr/src/oldp*, и случайно вставили пробел после первой косой черты:

```
rm -rf / usr/src/oldp
```

Проблемы могут создаваться и каталогами, имена которых содержат пробелы. Допустим, что имеются каталоги *Dir\ 1* и *Dir\ 2*, где обратная косая черта означает, что *Dir\ 1* – это одно имя файла, в котором есть пробел. Пусть теперь вы захотите удалить оба каталога, но по ошибке снова вставите лишний пробел:

```
rm -rf Dir\  *
```

Между обратным слэшем и звездочкой оказалось два пробела. Первый из них защищен обратной косой чертой, а второй – нет, поэтому он разделяет аргументы и делает звездочку новым аргументом. Увы, текущий каталог и все, что в нем находится, будут уничтожены.

Другой частой ошибкой является путаница в аргументах команд, таких как *dd*, которая часто используется для перемещения больших объемов данных из одного места в другое. Например, чтобы сохранить первые 1024 байта данных с устройства */dev/hda* (в которых содержатся загрузочная запись и таблица разделов этого диска), можно выполнить команду:

```
dd if=/dev/hda of=/tmp/stuff bs=1k count=1
```

Однако если в этой команде поменять местами if и of, то произойдет совершенно иное: содержимое файла */tmp/stuff* будет записано поверх */dev/hda*. Скорее всего, вам только что удалось убить свою таблицу разделов и, возможно, суперблок файловой системы. Добро пожаловать в чудесный мир системного администрирования!

Все это сказано к тому, что прежде чем выполнять какую-либо команду как root, посидите некоторое время, ничего не делая. Посмотрите внимательно на команду в течение минуты, прежде чем давить <Enter>, и убедитесь, что она верная. Если вы не уверены в аргументах и синтаксисе команды, быстренько посмотрите страницы руководства или проверьте ее в безопасном окружении, прежде чем запускать на выполнение. Иначе вы можете получить суровый урок. Ошибки, совершенные от имени root, могут быть катастрофическими.

Полезный совет: можно воспользоваться командой *alias*, чтобы сделать некоторые команды, выполняемые root, менее опасными. Например, можно выполнить команду

```
alias rm="rm -i"
```

Опция *-i* означает *интерактивность* и приводит к тому, что команда *rm* станет запрашивать подтверждение перед удалением каждого файла. Правда, это не защитит вас от жестокой ошибки, показанной выше: опция *-f* (от *force*) просто отменит *-i*, поскольку использована позднее.

Часто приглашение для учетной записи `root` выглядит не так, как для обычных пользователей. В классическом случае приглашение для `root` содержит символ `#`, в то время как для обычных пользователей это `$` или `%`. (Разумеется, вам решать, использовать ли это соглашение, однако оно используется во многих Unix-системах.) Хотя приглашение может напомнить о том, что у вас в руках волшебная палочка `root`, но пользователи часто забывают об этом или по ошибке вводят команду в неверном окне или виртуальной консоли.

Как и любым мощным инструментом, учетной записью `root` можно злоупотребить. Системному администратору необходимо защитить пароль для `root`, и если вы вообще кому-нибудь его раскрываете, делайте это исключение только для тех пользователей, которым вы доверяете (или с которых можно спросить за их действия в системе). Если вы единственный пользователь системы, то эта информация к вам не относится, если только вы не подключены к сети или не разрешаете доступ по телефонной линии.

Основное преимущество единоличного использования учетной записи `root` состоит не в уменьшении возможностей злоупотребления, хотя и это важно. Еще более важно, что если вы один можете использовать учетную запись `root`, то находитесь в полном ведении того, как сконфигурирована система. Если кто-либо еще обладает правами, скажем, модифицировать важные системные файлы (о чем мы расскажем в этой главе), то системная конфигурация может быть изменена без вашего ведома, и ваши предположения о том, как работает система, оказываются неверными. Наличие единственного администратора, выступающего судьей конфигурации системы, означает, что есть человек, который всегда знает, что происходит.

Кроме того, если пароль `root` знает кто-то другой, возникает возможность того, что, в конце концов, он сделает ошибку при использовании учетной записи `root`. Даже если всем, кто знает пароль суперпользователя, можно доверять, никто не застрахован от ошибок. Если вы – единственный администратор системы, то остается винить только себя за неизбежные ошибки при использовании учетной записи `root`.

После всего сказанного займемся реальными задачами администрирования под Linux. Пристегните ремни.

Начальная загрузка системы

Есть несколько способов загрузки Linux на машине. Чаще всего загрузка происходит с жесткого диска или загрузочной дискеты. Во многих случаях процедура установки настраивает один или оба эти способа. В любом случае важно знать, как самостоятельно сконфигурировать загрузку.

Использование загрузочной дискеты

Обычно загрузочная дискета Linux просто содержит образ ядра, который загружается в память, когда вы вставляете дискету и запускаете машину.[1]

Многие дистрибутивы Linux создают загрузочную дискету во время установки системы. С помощью загрузочной дискеты легко запустить систему, если вы не хотите мучиться с загрузкой с жесткого диска. (Например, начальный загрузчик Windows NT/2000 не так просто настроить для загрузки Linux. Об этом мы поговорим в следующем разделе.) Как только ядро загружено с дискеты, можно использовать привод гибких дисков для других целей.

Мы поместили в книгу некоторые технические сведения, чтобы объяснить процедуру начальной загрузки, но можете поверить, что в большинстве случаев достаточно вставить в машину дискету, и система загрузится. Все же чтение последующих разделов поможет вам лучше разобраться в своей машине.

Образ ядра обычно сжат с использованием того же алгоритма, что в программе сжатия *gzip* или *bzip2* (подробнее об этом см. в разделе «Сборка ядра» главы 7). Сжатие позволяет уместить ядро размером более 1 Мбайт в несколько сот килобайт. Часть кода ядра не сжата; в ней содержатся программы, необходимые для разархивирования ядра из дискового образа и загрузки в память. Поэтому ядро фактически загружает себя, разархивируясь в память.

В образе ядра содержится ряд параметров. Среди них имя устройства, используемого в качестве корневой файловой системы после загрузки ядра. Другим параметром является текстовый режим, используемый системной консолью. Все эти параметры можно изменить с помощью команды *rdev*, которую мы будем обсуждать ниже в этом разделе.

После запуска ядро пытается монтировать файловую систему на корневое устройство, жестко прошитое в самом образе ядра. Она будет служить корневой файловой системой, т. е. файловой системой на /. В разделе «Управление файловыми системами» главы 6 о файловых системах рассказано более подробно; сейчас вам достаточно знать, что в образе ядра должно содержаться имя устройства вашей корневой файловой системы. Если ядро не может монтировать на это устройство файловую систему, оно сдается и выводит сообщение ядра «panic». (По сути, *kernel panic* – это неустранимая ошибка, о которой сообщает само ядро. Паника происходит, когда ядро находится в полном замешательстве и не может продолжать работу. Например, если в ядре есть программная ошибка, то паника может произойти при попытке доступа к несуществующей памяти. О панике ядра мы еще поговорим в разделе «Действия в аварийной ситуации» главы 8.)

---

[1] Возможен другой вариант: загрузочная дискета Linux содержит загрузочную запись LILO, которая заставляет систему загрузить ядро с жесткого диска. Мы обсудим это в следующем разделе, когда будем говорить о LILO.

Корневое устройство, записанное в образе ядра, – это устройство вашей корневой файловой системы на жестком диске. Это значит, что после загрузки ядро монтирует раздел жесткого диска в качестве корневой файловой системы, и все управление переходит к жесткому диску. Будучи загруженным в память, ядро остается в ней – доступа к загрузочной дискете больше не потребуется (если, конечно, вы не перезагрузите систему).

Если у вас есть образ ядра, вы можете создать собственную загрузочную дискету. Во многих системах Linux само ядро хранится в файле */boot/vmlinuz*.[1] Однако это не общепринятое соглашение; прочие системы Linux хранят ядро в */vmlinuz* или */vmlinux*, или в таком файле, как */Image*. (Если у вас несколько образов ядра, можно с помощью LILO выбрать нужное для загрузки. См. следующий раздел.) Учтите, что вновь установленная система Linux может не иметь на жестком диске образа ядра, если программа установки создала загрузочную дискету. В любом случае вы можете скомпилировать собственное ядро. Обычно это правильное решение: вы можете «настроить» ядро, чтобы оно включало только те драйверы, которые необходимы для ваших устройств. Подробности см. в разделе «Сборка ядра» главы 7.

Отлично. Допустим, у вас есть образ ядра в файле */boot/vmlinuz*. Для создания загрузочной дискеты нужно первым делом с помощью команды *rdev* установить корневое устройство на вашу корневую систему Linux. (Если вы сами компилировали ядро, то оно уже имеет правильное значение, хотя не повредит проверить его с помощью *rdev*.) О том, как создать корневое устройство, мы говорили в разделах «Диски и разделы в Linux» и «Создание разделов в Linux» в главе 3.

Зарегистрировавшись как `root`, введите команду *rdev -h*, чтобы посмотреть сообщение об использовании команды. Как вы увидите, поддерживается много параметров, позволяющих задать корневое устройство (наша текущая задача), устройство для свопа, размер электронного диска и т. д. Эти параметры вас сейчас в основном не должны волновать.

Если мы выполним команду *rdev /boot/vmlinuz*, то будет выведено корневое устройство, закодированное в ядре, находящемся в */boot/vmlinuz*:

```
courgette:/# rdev /boot/vmlinuz
Root device /dev/hda1
```

Если это неверно, и корневая файловая система Linux фактически лежит на */dev/hda3*, следует выполнить команду:

```
courgette:/# rdev /boot/vmlinuz /dev/hda3
courgette:/#
```

---

[1] Откуда взялось такое странное название? Во многих системах Unix ядро хранится в файле */vmunix*, где *vm* происходит от «virtual memory» (виртуальная память). Естественно, Linux должна быть оригинальной и называет образы своего ядра *vmlinux*, помещая их в каталог */boot*, чтобы не оставлять в корневом каталоге. Имя *vmlinuz* было принято для отличия сжатых образов от несжатых. На практике имя и местонахождение ядра не играют никакой роли, если у вас есть загрузочная дискета с ядром или LILO знает, где найти его образ.

У *rdev* характер сильный и немногословный: когда устанавливается корневое устройство, то не выводится никаких сообщений, поэтому снова выполните команду *rdev /boot/vmlinuz*, чтобы проверить правильность установки.

Теперь вы можете создать загрузочную дискету. Лучше всего использовать новую отформатированную дискету. Можно отформатировать дискету под Windows или с помощью *fdformat* в Linux[1]; при этом будет записана информация о секторах и дорожках, чтобы система могла автоматически определить размер дискеты. (См. раздел «Управление файловыми системами» в главе 6 об использовании дискет.)

Чтобы создать загрузочную дискету, воспользуйтесь командой *dd*, которая скопирует на дискету образ ядра:

```
courgette:/# dd if=/boot/vmlinuz of=/dev/fd0 bs=8192
```

Если вас заинтересовала команда *dd*, можете почитать о ней на страницах руководства. Если сказать кратко, то она копирует входной файл (параметр *if*) с именем */boot/vmlinuz* в выходной файл (параметр *of*) с именем */dev/fd0* (первый привод гибких дисков), используя блок размером (*bs*) 8192 байт. Конечно, можно использовать плебейскую *cp*, но системные администраторы Unix любят использовать таинственные команды для выполнения сравнительно простых задач. Это то, что отличает нас от простых смертных.

Теперь ваша загрузочная дискета должна быть готова к работе. Вы можете остановить систему (см. раздел «Останов системы» далее в этой главе) и загрузиться с дискеты, и если все пойдет хорошо, ваша Linux-система должна загрузиться как обычно. Неплохо сделать про запас еще одну загрузочную дискету, а в разделе «Действия в аварийной ситуации» главы 8 мы опишем способы использования загрузочных дискет при восстановлении после катастрофы.

Использование LILO

LILO является менеджером загрузки общего назначения, способным загружать любые операционные системы, установленные на вашей машине, в том числе Linux. Есть десятки способов конфигурирования LILO. Мы хотим изложить два наиболее часто используемых метода: установку LILO в главную загрузочную запись жесткого диска и установку LILO как вторичного начального загрузчика только для Linux.

LILO чаще всего используется для загрузки Linux с жесткого диска. (Под загрузкой с жесткого диска мы подразумеваем, что само ядро хранится на жестком диске и загрузочная дискета не нужна. Но не забывайте, что даже если вы пользуетесь загрузочной дискетой, как только ядро загружено в память, контроль передается жесткому диску.) Если загрузчик LILO установлен в главную загрузочную запись вашего диска, MBR, то это первая программа, которая выполняется при загрузке жесткого диска. Затем LILO мо-

[1] В некоторых версиях дистрибутива Debian нет команды *fdformat*; используйте вместо нее удачно названную команду *superformat*.

жет загрузить другие операционные системы, например Linux или Windows, и позволить вам делать выбор во время загрузки.

Следует отметить, что LILO – не единственный менеджер загрузки, доступный для запуска Linux. Есть альтернативы, такие как *grub* (Grand Unified Bootloader), который работает так же хорошо. Однако из-за того что большинство дистрибутивов используют LILO, именно это мы здесь и рассматриваем.

Однако в Windows NT и последующих версиях Windows есть собственные начальные загрузчики, располагающиеся в MBR. Если вы пользуетесь одной из этих систем, можно установить LILO в качестве «вторичного» начального загрузчика только для Linux. В этом случае LILO устанавливается в загрузочную запись только раздела Linux, а начальный загрузчик Windows NT/2000 берет на себя задачу запустить оттуда LILO, когда вы хотите загрузить Linux.

Однако, как мы увидим, начальные загрузчики Windows NT/2000 не очень расположены к сотрудничеству, когда дело касается загрузки LILO. Это слабое архитектурное решение, и если вам абсолютно необходимо использовать один из этих начальных загрузчиков, проще загружать Linux с дискеты. Либо, если вы действительно готовы больше не расставаться с Linux, используйте LILO для загрузки Windows NT/2000 и вообще забудьте о начальных загрузчиках Windows.

Напротив, использовать LILO с Windows 95/98/ME/2000 совсем просто. Нужно лишь сконфигурировать LILO для загрузки Windows 95/98/ME/2000 (см. следующий раздел). Однако если вы установите Windows 95/98/ME/2000 после установки LILO, вам понадобится переустановить LILO, поскольку процедура установки Windows 95/98/ME/2000 перепишет MBR первичного жесткого диска. Позаботьтесь о том, чтобы иметь на руках загрузочную дискету Linux для загрузки Linux и повторного запуска LILO.

Прежде чем продолжить, следует отметить, что ряд дистрибутивов Linux может устанавливать и конфигурировать LILO во время первоначальной установки Linux. Однако часто лучше всего самостоятельно конфигурировать LILO, чтобы гарантировать правильность установки.

Файл /etc/lilo.conf

Первым шагом в конфигурировании LILO является установка конфигурационного файла LILO, который обычно находится в */etc/lilo.conf*. (Иногда этот файл можно найти в */boot/lilo.conf* или */etc/lilo/config*.)

Мы рассмотрим образец файла *lilo.conf*. Вы можете использовать его в качестве основы для собственного *lilo.conf* и отредактировать для использования в своей системе.

В первой части этого файла устанавливаются некоторые базовые параметры:

```
boot = /dev/hda
compact
install = /boot/boot.b
map = /boot/map
```

Строка boot определяет имя устройства, на котором загрузчик LILO должен установить себя в загрузочной записи. В данном случае мы хотим установить LILO в главную загрузочную запись */dev/hda* первого не являющегося SCSI жесткого диска. Если вы загружаетесь с жесткого диска SCSI, используйте такое имя устройства, как, например, */dev/sda*. Если вы укажете имя устройства раздела, а не диска (например, */dev/hda2*), LILO будет установлен в качестве вторичного загрузчика в названном разделе. Мы поговорим об этом подробнее дальше.

Строка compact требует от LILO проведения некоторой оптимизации. Всегда задавайте ее, если только не занимаетесь серьезным хакерством с конфигурированием LILO.[1] Точно так же всегда используйте строки install и map, как это показано в примере. В install указан файл, содержащий загрузочный сектор, который нужно использовать в MBR, а map указывает на «файл отображения», который LILO создает после установки. Во многих дистрибутивах (например, SuSE) эти файлы должны находиться в каталоге */boot*, хотя в некоторых системах их можно найти в */etc/lilo*. Файл */boot/map* будет создан только тогда, когда вы в первый раз установите LILO.

Теперь для каждой операционной системы, которую вы хотите загружать с помощью LILO, добавьте строфу в */etc/lilo.conf*. Например, строфа для Linux может выглядеть так:

```
# Строфа для Linux с корневым разделом в /dev/hda2.
   image = /boot/vmlinuz   # Местонахождение ядра
   label = linux           # Название OS (для меню загрузки LILO)
   root = /dev/hda2        # Местонахождение корневого раздела
   vga = ask               # Запрос текстового режима VGA при загрузке
```

Строка image определяет имя образа ядра. В число полей входят: label для указания имени, используемого в загрузочном меню LILO (подробнее об этом ниже); root для указания корневого раздела Linux; vga для задания текстового режима VGA, используемого в системной консоли.

Допустимыми значениями для vga являются normal (стандартный вывод 80x25), extended (для расширенного режима, обычно 132x44 или 132x60), ask (для вывода запроса режима во время загрузки) или целое число 1, 2 или 3. Это число соответствует номеру режима, выбираемого при значении ask. Имеющиеся режимы зависят от типа видеокарты. Чтобы получить их список, используйте vga = ask.

Если у вас не одно, а несколько ядер Linux, которые вам хотелось бы загружать – например, вы отлаживаете ядро, – можно для каждого из них добавить строфу с образом. Единственное обязательное поле в строфе образа – label. Если вы не зададите root или vga, будут использованы значения по умолчанию, записанные в образе ядра с помощью *rdev*. Если вы все же укажете root или vga, то их значения переопределят те, которые могли быть установлены *rdev*. Поэтому при загрузке Linux с помощью LILO нет необходимости ис-

[1] В некоторых случаях требуется параметр linear, который не совместим с compact. Дополнительные сведения ищите в документации по LILO.

пользовать *rdev*: параметры загрузки устанавливаются с помощью конфигурационного файла LILO.

Строфа для загрузки Windows 95/98/ME/2000 будет выглядеть так:

```
# Строфа для раздела Win 95/Win 98/ Win ME/ Win 2000 на /dev/hda1.
   other = /dev/hda1    # Местонахождение  раздела
   table = /dev/hda     # Местонахождение таблицы разделов для /dev/hda2
   label = windows      # Название ОС (для меню загрузки LILO)
```

Аналогичная строфа используется для загрузки OS/2 из LILO (разумеется, с другой строкой `label`).

Если вы хотите загрузиться с раздела Windows 95/98/ME/2000, размещенного на втором диске, то надо добавить строку:

```
   loader = /boot/any_d.b
```

к строфе Windows `other`.

Для конфигурирования LILO есть много других параметров. Сам дистрибутив LILO, который можно найти на большинстве FTP-серверов и дистрибутивов Linux, содержит пространное руководство, в котором они все описаны. Однако предыдущих примеров в большинстве случаев достаточно.

Когда файл */etc/lilo.conf* готов, можно выполнить команду

```
/sbin/lilo
```

войдя в систему как `root`. При этом должны быть выведены данные типа следующих:

```
courgette:/# /sbin/lilo
Added linux
Added windows
courgette:/#
```

С помощью параметра *-v* можно заставить *lilo* вывести больше диагностической информации, если возникнут неполадки. Кроме того, параметр *-C* позволяет задать файл конфигурации, отличный от */etc/lilo.conf*.

Когда все это сделано, можно остановить систему (опять-таки, более подробно об этом см. в разделе «Останов системы» далее в этой главе), перезагрузиться и посмотреть, что получится. По умолчанию загружается первая операционная система из перечисленных в */etc/lilo.conf*. Чтобы выбрать другое ядро или операционную систему из перечисленных в */etc/lilo.conf*, держите нажатой клавишу <Shift> или <Ctrl> либо просто нажмите <Scroll Lock> во время начальной загрузки. В результате должно появиться приглашение начальной загрузки LILO:

```
boot:
```

Теперь можно нажать <Tab> и получить список имеющихся вариантов загрузки:

```
boot: tab-key
linux msdos
```

Это имена, указанные в строках `label` файла */etc/lilo.conf*. Введите соответствующее имя, и загрузится нужная операционная система. В данном примере ввод `windows` вызывает загрузку Windows из */dev/hda1*, как указано в файле *lilo.conf*.

Следует заметить, что некоторые дистрибутивы добавляют к LILO красивый графический интерфейс (обычно при этом создается фон с изображением Tux, пингвина Linux). Однако конфигурирование в этих случаях должно быть таким же, как для обычных прежних текстовых LILO.

Использование LILO в качестве вторичного загрузчика

Если вы используете начальный загрузчик Windows NT/2000, устанавливаете дистрибутив Linux Debian или не хотите, чтобы LILO поселился в главной загрузочной записи вашего диска, можно так сконфигурировать LILO, что он будет вторичным загрузчиком, расположенным в загрузочной записи корневого раздела Linux.

Для этого просто поставьте в строке `boot = …` файла */etc / lilo.conf* имя корневого раздела Linux. Например,

```
boot = /dev/hda2
```

установит LILO в загрузочную запись раздела */dev/hda2*, чтобы загружать только Linux. Учтите, что это возможно только для первичных разделов жесткого диска, но не для расширенных или логических. Это ограничение не распространяется на дистрибутив Debian, в котором MBR может загружать операционную систему и с расширенного (но не логического) раздела. Чтобы загрузить Linux таким способом, корневой раздел Linux должен быть помечен в таблице разделов как «active». Это можно сделать с помощью *fdisk* под Linux или Windows. При начальной загрузке системы BIOS читает загрузочную запись первого «активного» раздела и запускает Linux.

При использовании начального загрузчика Windows NT/2000 нужно установить LILO именно таким способом, а затем сообщить начальному загрузчику о загрузке другой операционной системы из этого раздела жесткого диска. Как это сделать, зависит от конкретного загрузчика (см. соответствующую документацию).

Задание параметров начальной загрузки

Когда вы устанавливали Linux, то, скорее всего, загружались с дискеты или компакт-диска, получая при этом знакомое уже приглашение начальной загрузки LILO. В ответ на это приглашение можно ввести несколько параметров загрузки, например

```
hd=cylinders,heads,sectors
```

для указания геометрии жесткого диска. Может оказаться необходимым задавать эти параметры при каждой загрузке Linux, чтобы устройства правильно определялись, как это описано в разделе «Начальная загрузка Linux» главы 3. Если вы используете LILO для загрузки Linux с жесткого дис-

ка, можно задать эти параметры в файле */etc/lilo.conf* вместо того, чтобы каждый раз вводить их с клавиатуры. Просто добавьте к строфе Linux файла *lilo.conf* такую строку :

```
append = "hd=683,16,38"
```

В результате система будет вести себя так, как если бы вы ввели строку `hd=683,16,38` в ответ на приглашение загрузки LILO. При необходимости ввести несколько параметров загрузки можно сделать это одной строкой:

```
append = "hd=683,16,38 hd=64,32,202"
```

В этом случае указывается геометрия первого и второго жестких дисков соответственно.

Обратите внимание, что такие параметры загрузки нужно вводить, только если ядро не может определить ваши устройства при загрузке, что бывает, когда устройства очень старые или нестандартные. Такая необходимость может выясниться уже в процессе установки Linux. В общем случае строка `append` должна использоваться в *lilo.conf*, только если параметры загрузки пришлось указывать при первой загрузке с инсталляционного носителя.

Есть ряд других параметров, которые задаются на этапе загрузки. По большей части они касаются обнаружения устройств, о чем уже говорилось в главе 3. Однако могут оказаться полезными следующие дополнительные параметры:

`single`

Загружает систему в однопользовательском режиме; опускаются все параметры конфигурации и на консоли запускается оболочка для `root`. Советы по использованию такого режима см. в разделе «Действия в аварийной ситуации» главы 8.

`root=`*`partition`*

Монтирует указанный раздел *`partition`* в качестве корневой файловой системы Linux. Тем самым переопределяется любое значение из */etc/lilo.conf*.

`ro`

Монтирует корневую файловую систему в режиме только для чтения. Обычно это делается для запуска *fsck*; см. раздел «Проверка и исправление файловых систем» главы 6.

`ramdisk=`*`size`*

Задает размер электронного диска в байтах. Тем самым переопределяется любое значение в */etc/lilo.conf*. Большинству пользователей не нужно беспокоиться об электронном диске; он полезен, главным образом, при установке.

`vga=`*`mode`*

Устанавливает режим для VGA-дисплея. Тем самым переопределяется любое значение в */etc/lilo.conf*. Допустимы режимы `normal`, `extended`, `ask` или целое число. Этот параметр эквивалентен использованию `vga =` значениям, используемым в *lilo.conf* (см. раздел «Файл /etc/lilo.conf» ранее в главе).

`mem=size`

Сообщает ядру о размере имеющейся оперативной памяти. Если объем памяти равен 64 Мбайт или меньше, ядро может получить эти сведения из BIOS, но если у вас старое ядро и памяти больше, придется сообщить ядру точный размер, или оно будет использовать только первые 64 Мбайт. Например, если у вас 128 Мбайт, задайте `mem=128m`. Если ядро достаточно новое, в этом нет необходимости.

Любой из этих параметров можно задать вручную в ответ на приглашение загрузки LILO или указать в параметре `append` в */etc/lilo.conf*.

По LILO есть полная документация, в которой описаны все параметры конфигурации. Во многих системах Linux эту документацию можно найти в */usr/src/lilo*. В системах Debian она находится в */usr/share/doc/lilo/Manual.txt.gz*. Если вам не удается ничего найти, скачайте дистрибутив LILO с одного из архивных сайтов Linux или попросите вашего поставщика Linux предоставить источники и документацию по LILO. В этой документации есть руководство, в котором подробно описаны все понятия начальной загрузки и использования LILO, а также файл *README*, содержащий отрывки из этого руководства в простом текстовом формате.

Удаление LILO

Если LILO установлен в MBR, то проще всего удалить его с помощью *fdisk* Windows. Команда

```
FDISK /MBR
```

запускает *fdisk* и записывает в MBR правильную загрузочную запись Windows.

LILO сохраняет резервные копии исходной загрузочной записи в файлах */boot/boot.0300* (для приводов IDE) и */boot/boot.0800* (для приводов SCSI). В этих файлах содержится MBR диска, существовавшая до установки LILO. С помощью команды *dd* можно заменить загрузочную запись диска этой резервной копией. Например,

```
dd if=/boot/boot.0300 of=/dev/hda bs=446 count=1
```

копирует первые 446 байт файла */boot/boot.0300* на */dev/hda*. Даже если файлы имеют размер 512 байт, скопировать в MBR нужно только первые 446 байт.

Будьте осторожны при использовании этой команды! Это один из тех случаев, когда слепое следование книге без понимания производимых действий может причинить реальные неприятности. Используйте этот метод как последнее прибежище и только при уверенности, что файл */boot/boot.0300* или */boot/boot.0800* содержит нужную вам загрузочную запись. Многие дистрибутивы Linux имеют ненастоящие версии этих файлов. Может потребоваться их удалить, прежде чем устанавливать LILO.

В документации к LILO есть дальнейшие указания по удалению LILO и отладке его конфигурации.

Запуск системы и инициализация

В этом разделе мы расскажем, что именно происходит при загрузке системы. Понимание этого процесса и участвующих в нем файлов важно для осуществления разного рода настроек системы.

Сообщения ядра при загрузке

На первом шаге загружается ядро. Как сказано в предыдущем разделе, загрузка может происходить с дискеты или жесткого диска. По мере загрузки в память ядро выводит различные сообщения на системную консоль, а также обычно записывает их в системные журналы. Зарегистрировавшись как `root`, вы всегда можете проверить файл */var/log/messages* (в котором содержатся также и сообщения ядра, выдаваемые во время работы). Команда *dmesg* выводит последние сообщения, содержащиеся в кольцевом буфере сообщений ядра. Сразу после загрузки вы, естественно, получите сообщения загрузки.

В нескольких последующих разделах мы разберем ряд наиболее интересных сообщений и разъясним их смысл. Все эти сообщения выводит само ядро по ходу инициализации каждого драйвера устройства. Точный текст сообщений зависит от того, какие драйверы включены в ядро и какие устройства есть на машине. Вполне возможно, что у вас сообщений будет больше, или меньше, или они будут не такими. Мы остановим свое внимание на тех сообщениях, которые встречаются очень часто.

Строка

```
Linux version 2.4.10-64GB-SMP (root@SMP_X86.suse.de) \
(gcc version 2.95.3 20010315) #1 SMP Fri Sep 28 17:26:36 GMT 2001
```

сообщает номер версии ядра, имя машины, время и компилятор, с помощью которого оно было собрано.

Затем ядро сообщает о том, какие процессоры оно обнаружило и как эти процессоры будут вместе работать (вывод для машины с двумя процессорами):

```
Intel MultiProcessor Specification v1.1
    Virtual Wire compatibility mode.
OEM ID: OEM00000 Product ID: PROD00000000 APIC at: 0xFEE00000
Processor #0 Pentium(tm) Pro APIC version 17
    Floating point unit present.
    Machine Exception supported.
    64 bit compare & exchange supported.
    Internal APIC present.
    Bootup CPU
Processor #1 Pentium(tm) Pro APIC version 17
    Floating point unit present.
    Machine Exception supported.
    64 bit compare & exchange supported.
```

```
    Internal APIC present.
...
I/O APIC #2 Version 17 at 0xFEC00000.
Processors: 2
mapped APIC to ffffe000 (fee00000)
mapped IOAPIC to ffffd000 (fec00000)
Detected 498673 kHz processor.
```

Затем ядро сообщает, какой шрифт оно выбрало для консоли и какой тип консоли обнаружен:

```
Console: colour VGA+ 80x25
```

Обратите внимание, что это относится только к текстовому режиму, используемому ядром, а не к возможностям видеокарты. (Видеокарта SVGA отображается как VGA+, если речь идет о текстовом режиме консоли.)

Затем ядро собирает сведения о шине PCI и проверяет наличие в системе карт PCI:

```
PCI: PCI BIOS revision 2.10 entry at 0xfb140, last bus=1
PCI: Using configuration type 1
PCI: Probing PCI hardware
PCI: Using IRQ router PIIX [8086/7110] at 00:07.0
PCI: Found IRQ 14 for device 00:07.2
PCI: Sharing IRQ 14 with 00:0b.0
Limiting direct PCI/PCI transfers.
```

Следующее выведенное сообщение – расчет «BogoMips» для вашего процессора:

```
Calibrating delay loop... 996.14 BogoMIPS
Calibrating delay loop... 996.14 BogoMIPS
Total of 2 processors activated (1992.29 BogoMIPS).
```

Это совершенно фальшивое измерение скорости процессора (отсюда и название), используемое для получения оптимальной производительности в циклах задержки нескольких драйверов устройств. Ядро выводит также сведения о системной памяти:

```
Memory: 770672k/786368k available (1390k kernel code, 15308k reserved, 392k data,
128k init, 0k highmem)
```

Как видим, система может использовать 770 672 Кбайт RAM. Само ядро использует 1390 Кбайт.

Затем ядро определяет различные структуры памяти и свойства процессора. Например, строка

```
CPU serial number disabled
```

сообщает, что ядро Linux просто отключило печально известную функцию серийного номера Pentium III.

Затем инициализируются сетевые возможности, порт мыши и драйвер последовательных портов. Строка

```
tty00 at 0x03f8 (irq = 4) is a 16550A
```

означает, что первое последовательное устройство (*/dev/tty00*, или COM1) было обнаружено по адресу 0x03f8, IRQ 4 и использует функции UART 16550A. Затем идет поиск некоторых других аппаратных устройств, например датчика истинного времени и привода гибких дисков:

```
Real Time Clock Driver v1.10e
...
Floppy drive(s): fd0 is 1.44M
FDC 0 is a post-1991 82077
loop: loaded (max 8 devices)
ide-floppy driver 0.97.sv
...
```

Немного позже идет проверка системой хоста адаптера SCSI. Ядро печатает сведения обо всех найденных устройствах SCSI. Они пространные и не стоят того, чтобы воспроизводить их здесь. Строка

```
Adding Swap: 120480k swap-space (priority -42)
```

сообщает, какого размера пространство для свопинга, обнаруженное ядром. В число других задач, осуществляемых при типичной загрузке, входят обнаружение и конфигурирование параллельного порта (`lp1`), обнаружение и конфигурирование сетевой карты и, наконец, настройка подсистемы USB.

Файлы init, inittab и rc

Когда драйверы устройств инициализированы, ядро выполняет программу *init*, находящуюся в */etc*, */bin* или */sbin* (в большинстве систем это */sbin/init*). Программа *init* является многоцелевой; она порождает новые процессы и перезапускает некоторые программы, когда они заканчивают работу. Например, для каждой виртуальной консоли *init* запускает процесс *getty*. После регистрации процесс *getty* заменяется другим, а после выхода из системы *init* запускает новый *getty*, позволяющий опять зарегистрироваться.

Программа *init* отвечает также за запуск ряда программ и сценариев при загрузке системы. Вся работа *init* управляется файлом */etc/inittab*. Чтобы понять его содержимое, нужно сначала разобраться с концепцией *уровней исполнения* (*runlevels*).

Уровень исполнения – это число или буква, обозначающая текущее состояние системы по отношению к *init*. Например, когда уровень исполнения изменяется на 3, будут выполнены все строки в */etc/inittab*, содержащие цифру 3 в колонке уровня исполнения. Уровни исполнения являются удобным способом группировки строк в */etc/inittab*. Например, вам бы хотелось, чтобы на уровне выполнения 1 выполнялся только самый минимум сценариев конфигурации, на уровне 2 выполнялось все то же, что на уровне 1, плюс конфигурирование сети, на уровне 3 – все, что на уровнях 1 и 2, плюс удаленный вход по коммутируемым линиям, и т. д. Дистрибутив Red Hat в настоящее время сконфигурирован так, что на уровне исполнения 5 автоматичес-

ки стартует графический интерфейс X Window System. Дистрибутив SuSE делает это на уровне исполнения 3, а дистрибутив Debian – на уровне 2, при условии, что вы установили X.

Чаще всего уровни исполнения не должны вас беспокоить. При загрузке система входит в уровень исполнения по умолчанию (устанавливаемый в */etc/inittab*, как мы скоро покажем). В большинстве систем это уровень 2 или 3. После обсуждения обычной загрузки мы покажем, как выйти на другой уровень исполнения, который иногда требуется, – уровень исполнения 1, или однопользовательский режим.

Посмотрим на пример файла */etc/inittab*:

```
# Установить уровень исполнения по умолчанию 3
id:3:initdefault:

# Выполнить /etc/rc.d/rc.sysinit при начальной загрузке системы
si:S:sysinit:/etc/rc.d/rc.sysinit

# Запустить /etc/rc.d/rc с уровнем исполнения в качестве аргумента
l0:0:wait:/etc/rc.d/rc 0
l1:1:wait:/etc/rc.d/rc 1
l2:2:wait:/etc/rc.d/rc 2
l3:3:wait:/etc/rc.d/rc 3
l4:4:wait:/etc/rc.d/rc 4
l5:5:wait:/etc/rc.d/rc 5
l6:6:wait:/etc/rc.d/rc 6

# выполняется при нажатии ctrl-alt-delete
ca::ctrlaltdel:/sbin/shutdown -t3 -rf now

# Запустить agetty для виртуальных консолей с 1 по 6
c1:12345:respawn:/sbin/agetty 38400 tty1
c2:12345:respawn:/sbin/agetty 38400 tty2
c3:45:respawn:/sbin/agetty 38400 tty3
c4:45:respawn:/sbin/agetty 38400 tty4
c5:45:respawn:/sbin/agetty 38400 tty5
c6:45:respawn:/sbin/agetty 38400 tty6
```

Поля разделяются двоеточием. Легче всего узнать последнее поле: это команда, которую *init* выполняет для этой строки. Первое поле является произвольным идентификатором (неважно, каким, лишь бы он был уникален в этом файле), а второе поле определяет, на каком уровне исполнения должна быть запущена команда. Третье поле сообщает *init*, как обрабатывать строку; например, выполнить команду один раз или перезапускать после каждого завершения.

Точное содержание файла */etc/inittab* зависит от вашей системы и установленного вами дистрибутива Linux.

В нашем примере мы видим сначала, что уровень исполнения по умолчанию установлен в 3. Поле *action* этой строки содержит `initdefault`, что влечет установление данного уровня исполнения в качестве значения по умолчанию. Это уровень исполнения, обычно действующий при загрузке системы. Вы

можете переопределить значение по умолчанию, запустив *init* вручную (что можно сделать при отладке конфигурации) и передав желаемый уровень выполнения в качестве аргумента. Например, следующая команда завершает все текущие процессы и стартует с уровня выполнения 5 (прежде чем это сделать, предупредите всех пользователей о необходимости выйти из системы!):

```
tigger# init 5
```

LILO тоже может загружать систему в однопользовательском режиме (обычно это уровень исполнения 1) – см. раздел «Задание параметров времени загрузки» выше в этой главе.

Следующая строка сообщает *init* о необходимости выполнить при загрузке системы сценарий */etc/rc.d/rc.sysinit*. (В поле *action* стоит sysinit, что указывает на необходимость выполнения этой строки, когда *init* впервые запускается при загрузке системы.) Этот файл является просто сценарием оболочки, содержащим команды для осуществления базовой инициализации системы. Например, включается свопинг, проверяются и монтируются файловые системы, и системные часы синхронизируются с таймером CMOS. Вы можете посмотреть, что содержится в этом файле на вашей машине. Подробнее мы поговорим о содержащихся в нем командах в разделах «Управление файловыми системами» и «Управление пространством для свопинга» главы 6. В различных дистрибутивах этот файл может находиться в разных местах, например, в SuSE это */sbin/init.d/boot*, что соответствует стандарту иерархии файловой системы Linux (Filesystem Hierarchy Standard, FHS).

Далее мы видим, что система выполняет сценарий */etc/rc.d/rc*, когда входит на любой из уровней выполнения от 0 до 6 с соответствующим уровнем выполнения в качестве аргумента. *rc* является родовым стартовым сценарием, выполняющим другие сценарии в соответствии с уровнем выполнения. Поле *action* содержит wait, что сообщает *init* о необходимости выполнить данную команду и дождаться окончания ее выполнения, прежде чем идти дальше.

Файлы rc

Linux хранит стартовые команды в файлах, имена которых содержат *rc*, согласно старинному соглашению Unix. Эти команды делают все необходимое для получения полностью функционирующей системы, например запускают серверы и демоны, о которых говорилось в главе 4. Благодаря этим командам система готова выполнять задачи регистрации, электронной почты, веб-сервера или другие, в зависимости от того, какое программное обеспечение вы установили и пожелали запустить. Как было разъяснено в предыдущем разделе, файлы запускаются из */etc/inittab*. Команды являются стандартными командами оболочки, и можно, просмотрев различные файлы *rc*, увидеть, что они делают.

В этом разделе мы описываем структуру файлов *rc*, чтобы вы могли понять, откуда все запускается, и запускать или останавливать серверы вручную в тех редких случаях, когда они не хотят делать то, что вы от них требуете. В качестве модели мы будем использовать Red Hat, однако, получив пред-

ставление о том, что нужно искать, вы сможете найти соответствующие файлы в любом дистрибутиве Linux. Приводить Red Hat в качестве примера одновременно хорошо и плохо, потому что Red Hat нарушает правила FHS. Linux FHS представляет собой не связанную с каким-либо дистрибутивом инициативу, имеющую целью определить стандартные имена каталогов и файлов для важнейших системных файлов. Этому стандарту должен следовать каждый добропорядочный дистрибутив Linux. Компания Red Hat решила – и не впервые – не быть добропорядочной в этом отношении, поэтому имена путей и файлов в Red Hat являют собой пример разнообразия, с которым можно столкнуться при поиске системных файлов. SuSE и Debian служат примерами дистрибутивов, следующих стандарту FHS.

В Red Hat rc-сценарий верхнего уровня – это */etc/rc.d/rc*. В других дистрибутивах путь несколько отличается (*/etc/init.d/rc* в SuSE, к примеру), но содержание аналогично. В предыдущем разделе вы могли видеть, как */etc/inittab* запускает сценарий в различных обстоятельствах с различными числами от 0 до 6 в качестве аргумента. Числа соответствуют уровням исполнения, и каждый из них заставляет rc-файлы запускать разные наборы сценариев. Поэтому нашим следующим шагом будет найти сценарии, соответствующие каждому уровню исполнения.

В Red Hat сценарии каждого уровня исполнения хранятся в каталоге */etc/rc.d/rcN.d*, где *N* – номер запускаемого уровня исполнения. Так, для уровня исполнения 3 будут использоваться сценарии из */etc/rc.d/rc3.d*. Опять же, в других дистрибутивах, как правило, есть маленькие отличия. Например, в Debian каждому уровню исполнения соответствует каталог */etc/rcN.d/*.

Загляните в какой-нибудь из этих каталогов – вы увидите ряд файлов вида *Snnxxxx* или *Knnxxxx*, где *nn* – число от 00 до 99, а *xxxx* – имя какого-либо системного сервиса. */etc/rc.d/rc* сначала выполняет сценарии с именами, начинающимися на K, которые убивают работающие сервисы. Затем выполняются сценарии с именами, начинающимися на S, которые запускают новые сервисы.

Числа *nn* в именах используются для установления порядка выполнения сценариев: сначала выполняются сценарии с меньшими номерами. Имя *xxxx* используется просто для того, чтобы помочь определить, какому из системных сервисов соответствует сценарий. Это соглашение по именованию может показаться странным, но оно позволяет легко добавлять и удалять сценарии из этих каталогов и помогает */etc/rc.d/rc* автоматически выполнять их в нужный момент. Для редактирования стартовых сценариев удобно использовать графический редактор уровней исполнения, такой как *ksysv* в KDE.

Например, сценарий для инициализации сетевого взаимодействия может называться *S10network*, а сценарий для останова демона системной регистрации может называться *K70syslog*. Если эти сценарии расположены в соответствующих каталогах */etc/rc.d/rcN.d*, они будут запущены */etc/rc.d/rc* в порядке номеров при запуске или останове системы. Если уровень выполнения по умолчанию 3, см. в */etc/rc.d/rc3.d*, какие сценарии выполняются при нормальной загрузке системы.

Поскольку одни и те же сервисы запускаются или останавливаются на различных уровнях исполнения, дистрибутив Red Hat использует символические ссылки вместо повторения одного и того же сценария в разных местах. Так, каждый S- или K-файл является символической ссылкой, указывающей на центральный каталог, в котором хранятся сценарии запуска и останова всех сервисов. В Red Hat этот центральный каталог называется */etc/rc.d/init.d*, а в SuSE и Debian это */etc/init.d*. В Debian и SuSE этот каталог содержит сценарий с именем *skeleton*, который вы можете адаптировать для запуска и останова любых демонов, которые напишете сами.

Знать, где находится сценарий запуска или останова, полезно, если вы не хотите полностью перезагружаться или входить в другой уровень выполнения, но вам нужно запустить или остановить некоторый сервис. Найдите в каталоге *init.d* сценарий с соответствующим именем и выполните его, передав параметр `start` или `stop`. Например, в SuSE, если вы хотите запустить веб-сервер Apache, но ваша система находится на уровне исполнения, в который не входит Apache, введите следующую строку:

```
tigger # /sbin/init.d/apache start
```

Другой важный сценарий системной конфигурации – */etc/rc.d/rc.local*, выполняемый после запуска остальных сценариев инициализации системы. (Как это достигается? Обычно в каждом из каталогов */etc/rc.d/rcN.d* создается символическая ссылка с именем *S99local*, указывающая на *rc.local*. Поскольку 99 – самый большой возможный номер S-сценариев, он выполняется последним.) Вы можете редактировать файл *rc.local* так, чтобы выполнять любые специфические или в каком-то отношении неуместные команды во время загрузки, или если вы не уверены, в каком месте их следует выполнять. В Debian нет эквивалента сценарию *rc.local*, но ничто не мешает вам добавить его вызов из *rc*, если вы привыкли, чтобы он у вас был.

Следующая строка с меткой `ca` выполняется при нажатии на консоли комбинации клавиш Ctrl-Alt-Del. Эта комбинация вызывает прерывание, которое обычно перезагружает систему. В Linux это прерывание перехватывается и посылается *init*, которая выполняет запись с `ctrlaltdel` в поле *`action`*. Показанная здесь команда, */sbin/shutdown -t3 -rf now*, осуществляет «безопасную» перезагрузку системы. (См. раздел «Останов системы» ниже в этой главе.) Таким образом мы защищаем систему от случайной перезагрузки при нажатии Ctrl-Alt-Del.

И наконец, в файле *inittab* содержатся записи, которые запускают */sbin/agetty* для первых шести виртуальных консолей. *agetty* является одним из нескольких вариантов *getty*, имеющихся в Linux. Эти программы позволяют регистрироваться на терминалах; без них терминал был бы, в сущности, мертв и не реагировал на нажатие пользователем клавиши или кнопки мыши. Различные команды *getty* открывают терминальное устройство (например, виртуальную консоль или последовательную линию), устанавливают различные параметры драйвера терминала и выполняют */bin/login* для начала сеанса регистрации на этом терминале. Поэтому для того чтобы на данной виртуальной консоли можно было зарегистрироваться, на ней должна

быть запущена *getty* или *agetty*. В ряде систем Linux используется *agetty*, в остальных используется *getty*, имеющая несколько отличающийся синтаксис. О *getty* и *agetty* в вашей системе см. на страницах руководства.

agetty принимает два аргумента: скорость обмена и имя устройства. Порты для виртуальных консолей Linux называются */dev/tty1*, */dev/tty2* и т. д. *agetty* предполагает, что имя устройства задается относительно */dev*. Скорость обмена для виртуальных консолей обычно должна быть 38 400.

Обратите внимание, что для команды *agetty* в поле `action` стоит параметр `respawn`. Это значит, что *init* необходимо перезапустить команду, указанную в записи, когда процесс *agetty* умирает, что происходит каждый раз при завершении сеанса пользователем.

Вы познакомились с *init*, но различные файлы и команды в */etc/rc.d*, которые делают всю работу, остались тайной. Чтобы разобраться в этих файлах, нужно получить представление о некоторых дополнительных задачах системного администрирования, таких как управление файловыми системами. Мы рассмотрим эти задачи в нескольких следующих главах, и в итоге все прояснится.

Однопользовательский режим

Большую часть времени система работает в многопользовательском режиме, и пользователи могут регистрироваться в системе. Однако есть особое состояние, называемое *однопользовательским режимом*, когда Unix работает, но приглашения регистрации нет. В однопользовательском режиме вы, по существу, являетесь суперпользователем `root`. Вам может понадобиться войти в этот режим во время установки, если возникнут неполадки. Однопользовательский режим важен для некоторых рутинных административных задач, например проверки поврежденных файловых систем. (Это нешуточное дело. Постарайтесь не повредить свою файловую систему. Например, всегда останавливайте систему командой останова перед отключением питания, как описано в следующем разделе.)

В однопользовательском режиме система почти бесполезна – используется малая часть конфигурации, файловые системы не монтируются и т. д. Это необходимо для восстановления после возникновения некоторых системных проблем (см. подробнее в разделе «Действия в аварийной ситуации» главы 8).

Обратите внимание, что Unix остается многозадачной системой даже в однопользовательском режиме. Одновременно могут выполняться несколько программ. Серверы могут выполняться в фоновом режиме, благодаря чему могут действовать некоторые специальные функции, такие как сетевое взаимодействие. Но если система поддерживает более одного терминала, можно использовать только консоль. Система X Window тоже не запустится.

Останов системы

К счастью, останов системы выполняется значительно проще, чем загрузка и запуск. Тем не менее это не просто нажатие кнопки reset. Linux, как и все системы Unix, буферизует в памяти дисковое чтение и запись. Это значит, что запись на диск откладывается до того времени, когда она становится абсолютно необходимой, а чтение одного и того же блока диска осуществляется непосредственно из RAM. Это значительно увеличивает производительность, поскольку диски работают гораздо медленнее, чем процессор.

Проблема заключается в том, что если система внезапно отключается от питания или перезагружается, находящиеся в памяти буферы не будут записаны на диск, что может привести к потере или повреждению данных. */sbin/update* – это программа, запускаемая в большинстве систем из */etc/rc.d/boot*; она сбрасывает «грязные» буферы (те, которые изменились после чтения с диска) обратно на диск каждые 5 секунд, чтобы предотвратить серьезные повреждения в случае краха системы. Однако для полной надежности система должна до перезагрузки осуществить «безопасный» останов. Это не только обеспечит правильную синхронизацию дисковых буферов, но и позволит корректно завершиться всем выполняющимся процессам.

Команда *shutdown* является общей многоцелевой командой, используемой для останова или перезагрузки системы. Зарегистрировавшись как `root`, вы можете выполнить команду

```
/sbin/shutdown -r +10
```

чтобы заставить систему перезагрузиться через 10 минут. Параметр *-r* указывает на то, что система после останова должна быть перезагружена, а `+10` задает время ожидания до останова в минутах. Система выведет предупреждающее сообщение на все терминалы и начнет отсчет времени останова. Вы можете добавить собственное сообщение, включив его в командную строку:

```
/sbin/shutdown -r +10 "Перезагрузка для проверки нового ядра"
```

Можно задать и абсолютное время останова:

```
/sbin/shutdown -r 13:00
```

для перезагрузки в 1 час дня. Аналогично можно дать команду

```
/sbin/shutdown -r now
```

чтобы немедленно перезагрузиться (после процесса безопасного останова).

Если вместо параметра *-r* использовать *-h*, то система будет просто остановлена без перезагрузки; после этого можно отключить питание, не опасаясь потери данных. Если не заданы ни *-h*, ни *-r*, система переходит в однопользовательский режим.

Как мы видели в разделе «Файлы init, inittab и rc», можно заставить *init* перехватывать комбинацию Ctrl-Alt-Del и в ответ на нее выполнять команду *shutdown*. Если вам привычнее перезагружать систему таким способом, неплохо проверить, что */etc/inittab* содержит запись `ctrlaltdel`. Обратите внима-

ние, что никогда нельзя перезагружать систему выключением питания или нажатием кнопки перезагрузки на передней панели машины. Если только система не зависла намертво (что бывает редко), всегда нужно использовать останов. Замечательное свойство многозадачной системы состоит в том, что если одна программа зависла, почти всегда можно переключиться в другое окно или виртуальную консоль, чтобы исправить положение.

Команда *shutdown* имеет ряд других параметров. Ключ *-c* прекратит текущий процесс останова. (Конечно, всегда можно убить процесс с помощью *kill*, но *shutdown -c* может быть проще.) Ключ *-k* заставит вывести предупредительные сообщения, но не осуществит фактический останов системы. См. страницы руководства *shutdown*(8), если вы заинтересованы в деталях.

Файловая система /proc

Системы Unix проделали долгий путь в отношении обеспечения единообразия интерфейсов различных частей системы. Как вы узнаете из следующей главы, аппаратные устройства представлены в Unix в виде файлов особого типа. Есть, однако, особая файловая система */proc*, которая делает дальнейший шаг: она унифицирует файлы и процессы.

С точки зрения пользователя или системного администратора файловая система */proc* выглядит как любая другая файловая система; можно перемещаться по ней с помощью команды *cd*, выводить содержимое каталогов командой *ls* и просматривать содержимое файлов командой *cat*. Однако ни один из этих каталогов и файлов не занимает пространства на жестком диске. Ядро перехватывает обращения к файловой системе */proc* и на лету создает содержимое каталогов и файлов. Иными словами, всякий раз, когда вы выводите содержимое каталога или файла в файловой системе */proc*, ядро динамически создает содержимое, которое вы хотите увидеть.

Чтобы сделать это менее абстрактным, рассмотрим несколько примеров. Следующий пример – это список файлов каталога верхнего уровня файловой системы */proc*:

```
owl # ls /proc
1      1618  17613  27191  27317  2859   8929         kcore_elf   rtc
11120  1621  1795   27192  27320  2860   9            kmsg        scsi
11121  1649  1796   27204  27324  28746  bus          ksyms       self
11153  1657  1798   27205  27326  28747  cmdline      loadavg     slabinfo
15039  1664  1799   27221  27374  28754  config.gz    locks       stat
1512   1681  1800   27229  27377  29877  cpuinfo      lvm         swaps
1530   1689  2      27287  27379  29878  devices      mdstat      sys
1534   1703  20007  27289  27380  29944  dma          meminfo     tty
1560   1708  21391  27292  27381  3      fb           misc        uptime
1570   1709  21394  27297  27397  4      filesystems  modules     version
1578   1710  2302   27308  27515  5      fs           mounts
1585   1711  2309   27310  27518  5841   ide          mtrr
1586   1712  2356   27312  27521  5842   interrupts   net
1587   1713  27182  27314  2786   5860   ioports      partitions
1588   1731  27183  27315  28536  6100   kcore        pci
```

В вашей системе числа будут иными, но общая структура сохранится. Все эти числа являются каталогами, представляющими выполняющиеся в вашей системе процессы. Например, взглянем на информацию о процессе с ID 1534:

```
owl # ls /proc/1534
cmdline  environ  fd      mem     stat    status
cwd      exe      maps    root    statm
```

Вы видите ряд файлов, каждый из которых содержит данные об этом процессе. Например, файл *cmdline* показывает командную строку, которая запустила этот процесс. *status* дает сведения о внутреннем состоянии процесса, а *cwd* содержит ссылку на текущий рабочий каталог этого процесса.

Возможно, данные об устройствах покажутся вам еще более интересными, чем данные о процессах. Все сведения, которые ядро собрало о ваших устройствах, собраны в файловой системе */proc*, хотя найти в ней требуемое может быть не просто.

Начнем с проверки памяти вашей машины. Она представлена в файле */proc/meminfo*:

```
owl # cat /proc/meminfo
        total:    used:    free:  shared: buffers:  cached:
Mem:  267919360 255311872 12607488        0 40587264 77791232
Swap: 123371520  5861376 117510144
MemTotal:    261640 kB
MemFree:      12312 kB
MemShared:        0 kB
Buffers:      39636 kB
Cached:       75968 kB
BigTotal:         0 kB
BigFree:          0 kB
SwapTotal:   120480 kB
SwapFree:    114756 kB
```

Если вы выполните команду *free*, то увидите ту же самую информацию, только формат чисел несколько другой. *free*, в сущности, читает */proc/meminfo* и несколько преобразует вывод.

Большая часть системных утилит, сообщающих данные об аппаратных средствах, делает то же самое. Файловая система */proc* является переносимым и простым способом получения этих данных. Особенно полезны сведения при добавлении к системе новых устройств. Например, аппаратные платы, как правило, требуют нескольких адресов ввода/вывода для связи с процессором и операционной системой. Если вы сконфигурировали две платы так, что они используют одни и те же адреса ввода/вывода, катастрофа неминуема. Избежать ее можно, если проверить, какие адреса ввода/вывода ядро уже определило как используемые:

```
tigger # more /proc/ioports
0000-001f : dma1
0020-003f : pic1
```

```
0040-005f : timer
0060-006f : keyboard
0080-009f : dma page reg
00a0-00bf : pic2
00c0-00df : dma2
00f0-00ff : npu
01f0-01f7 : ide0
0220-022f : soundblaster
02e8-02ef : serial(auto)
0388-038b : OPL3/OPL2
03c0-03df : vga+
03f0-03f5 : floppy
03f6-03f6 : ide0
03f7-03f7 : floppy DIR
03f8-03ff : serial(auto)
0530-0533 : WSS config
0534-0537 : MSS audio codec
e000-e0be : aic7xxx
e400-e41f : eth0
```

Теперь вы можете определить свободные адреса ввода/вывода. Конечно, ядро может показать адреса ввода/вывода только для плат, которые обнаружило и распознало, но в нормально сконфигурированной системе это должно быть правилом для всех плат.

Можно также использовать файловую систему */proc* для получения другой информации, необходимой для конфигурирования новых устройств: в файле */proc/interrupts* перечислены занятые линии прерываний (IRQ), а в */proc/dma* – используемые каналы прямого доступа к памяти (DMA).

Управление учетными записями пользователей

Понимание того, как управлять учетными записями пользователей необходимо, даже если вы единственное человеческое существо, использующее вашу Linux-систему, а если пользователей несколько, то тем более.

Учетные записи пользователей служат ряду целей в Unix-системах. В первую очередь они позволяют различать пользователей системы для идентификации и безопасности. У каждого пользователя есть личная учетная запись с отдельным именем пользователя и паролем. Как обсуждалось в разделе «Владение файлами и права доступа» главы 4, пользователи могут устанавливать права доступа для своих файлов, разрешая или ограничивая доступ к ним со стороны других пользователей. Каждым файлом системы «владеет» конкретный пользователь, который может установить для него права доступа. Учетные записи пользователей служат для авторизации доступа к системе: доступ к машине могут получить только те, у кого есть учетные записи. Кроме того, учетные записи служат для идентификации пользователей, ведения системных журналов, указания отправителя в сообщениях электронной почты и т. д.

Помимо личных учетных записей в системе есть пользователи, обеспечивающие административные функции. Как мы видели, администратор системы использует учетную запись `root` для осуществления сопровождения системы, но обычно не для личного пользования ею. Доступ к таким учетным записям осуществляется с помощью команды *su*, позволяющей обратиться к другой учетной записи после регистрации через личную учетную запись.

Некоторые учетные записи в системе могут быть вовсе не предназначены для взаимодействия с пользователем. Такие учетные записи обычно используются системными демонами, которые должны иметь доступ к файлам системы через ID пользователя, отличного от `root` или личных учетных записей пользователей. Например, если вы конфигурируете систему для получения новостей из телеконференций с другого сервера, демон новостей должен записывать статьи телеконференций в каталог спулинга, доступ к которому может иметь каждый, но только пользователь-демон новостей может осуществлять в него запись. С учетной записью `news` не ассоциируется никакая личность; это «мнимый» пользователь, созданный только для демона новостей.

Одним из битов разрешения, которые можно установить на выполняемые файлы, является бит *setuid*, заставляющий программу выполняться с правами владельца файла. Например, если владельцем демона новостей является пользователь `news`, и на выполняемом файле установлен бит *setuid*, демон работает так, как если бы был запущен пользователем `news`. `news` будет иметь право записи в каталог спулинга новостей, а все остальные пользователи будут иметь право чтения имеющихся там статей. Это особенность системы защиты. Программы чтения новостей предоставят пользователям лишь необходимые права в отношении каталога спулинга новостей, и никому не будет позволено безнаказанно резвиться в нем.

В качестве системного администратора вы должны создавать на своей машине учетные записи всех пользователей, реальных и виртуальных, и управлять ими. В большинстве случаев это безболезненная и легкая задача, но важно понять, как она решается.

Файл passwd

Для каждой учетной записи в системе есть запись в файле */etc/passwd*. В этом файле есть по одной строке на каждого пользователя, в которой указан ряд атрибутов учетной записи, таких как имя пользователя, его настоящее имя и т. д.

Каждая запись в этом файле имеет формат:

```
username:password:uid:gid:gecos:homedir:shell
```

Ниже объясняется значение этих полей:

username

Уникальная строка символов, идентифицирующая учетную запись. Для личных учетных записей это имя, под которым пользователь регистрируется в системе. В большинстве систем оно ограничено восемью алфавитно-цифровыми символами, например `larry` или `kirsten`.

password

Зашифрованное представление пароля пользователя. Это поле устанавливается программой *passwd*; используется однонаправленное шифрование, которое трудно (но можно) расколоть. Его не надо устанавливать вручную, это делает за вас программа *passwd*. Обратите внимание, что если первый символ в поле *password* – звездочка (*), то учетная запись «отключена»; система не позволит этому пользователю регистрироваться. См. раздел «Создание учетных записей» далее в настоящей главе.

uid

Идентификатор пользователя (ID), уникальное целое число, присваиваемое системой учетной записи. Система использует поле *uid* внутри себя для задач, связанных с правами доступа к процессам и файлам. С целыми числами обращаться проще и удобнее, чем со строками символов. Таким образом, каждая учетная запись идентифицируется как *uid*, так и именем пользователя *username*: *uid* для системы более важен, в то время как имя пользователя удобнее для людей.

gid

ID группы, целое число, указывающее на группу пользователя по умолчанию, находящуюся в файле */etc/group*. См. ниже раздел «Файл групп».

gecos

Различные сведения о пользователе, в том числе его настоящее имя, необязательные адресные сведения типа служебного адреса или номера телефона. Эти данные используют такие программы, как *mail* и *finger*, для идентификации пользователей в системе; мы еще поговорим об этом позднее. Между прочим, название *gecos* – историческое (появилось в 1970-е годы) и происходит от *General Electric Comprehensive Operating System*. GECOS не имеет никакого отношения к Unix, кроме того, что первоначально это поле было добавлено к */etc/passwd* для обеспечения совместимости с некоторыми ее сервисами.

homedir

Домашний каталог пользователя, предназначенный для личного использования; подробнее о нем ниже. При входе пользователя в систему оболочка считает текущим рабочим каталогом указанный домашний каталог *homedir*.

shell

Имя программы, запускаемой при регистрации пользователя; как правило, это полный путь к оболочке, например */bin/bash* или */bin/tcsh*.

Многие поля являются необязательными; обязательными полями являются только *username*, *uid*, *gid* и *homedir*. В большей части учетных записей заполнены все поля, однако «мнимые» или административные учетные записи могут использовать только некоторые.

Вот пример двух записей в */etc/passwd*:

```
root:ZxPsI9ZjiVd9Y:0:0:The root of all evil:/root:/bin/bash
aclark:BjDf5hBysDsii:104:50:Anna Clark:/home/aclark:/bin/bash
```

Первая запись относится к учетной записи `root`. Прежде всего, обратите внимание на то, что `uid` для `root` равен нулю. Это то, что делает `root` суперпользователем: система знает, что `uid`, равный 0, является «особым» и у него нет обычных ограничений прав доступа. `gid` для `root` является также нулем, что является обычным соглашением. Владельцем многих файлов системы являются `root` и группа `root`, для которых `uid` и `gid` суть нули. Подробнее о группах чуть позже.

Во многих системах `root` использует исходный каталог */root* или просто /. Обычно это не важно, поскольку чаще всего вы используете *su* для доступа к `root` из своей учетной записи. Традиционным является также использование какого-нибудь варианта оболочки Борна (в данном случае */bin/bash*) в качестве оболочки для учетной записи `root`, хотя при желании можно воспользоваться оболочкой C. (Оболочки обсуждаются в разделе «Оболочки» главы 4.) Однако будьте осторожны: у оболочек Борна и C разный синтаксис, и переключение между ними при использовании `root` может привести к неясностям и ошибкам.

Вторая строка относится к реальному пользователю-человеку с именем пользователя `aclark`. В нашем случае *uid* равен 104. Технически поле *uid* может быть любым целым числом. Во многих системах учетные записи пользователей получают номера 100 и выше, а административные учетные записи – в диапазоне до 100. gid равен 50, это означает, что пользователь `aclark` принадлежит группе, имеющей номер 50 в файле */etc/group*. О группах будет сказано в разделе «Файл групп» далее в этой главе.

Исходные каталоги часто находятся в */home* и называются по имени владельца. Как правило, это удобное соглашение, позволяющее избежать путаницы при поиске исходного каталога конкретного пользователя, но технически исходный каталог можно расположить в любом месте. Следует, однако, придерживаться структуры каталогов, принятой в вашей системе.

В качестве системного администратора вам обычно не придется непосредственно редактировать файл */etc/passwd*. Есть несколько программ, позволяющих создавать и поддерживать учетные записи пользователей; см. далее раздел «Создание учетных записей».

Теневые пароли

Возможность видеть зашифрованные пароли в файле */etc/passwd*, которая есть у всех имеющих доступ к системе, создает некоторую угрозу безопасности. Есть специальные программы для взлома, перебирающие огромное число возможных паролей и сравнивающие результат их шифрования с заданным паролем.

Для преодоления такой потенциальной угрозы безопасности были придуманы *теневые пароли* (*shadow passwords*). При использовании теневых паро-

лей поле `password` в файле */etc/passwd* содержит лишь `x` или `*`, чего никогда не может быть в зашифрованном варианте пароля. Для хранения паролей используется второй файл с именем */etc/shadow*. В нем содержатся записи, очень похожие на записи в */etc/passwd*, но содержащие реальные зашифрованные пароли в поле password. */etc/shadow* может быть прочтен только суперпользователем, поэтому у обычных пользователей нет доступа к зашифрованным паролям. Остальные поля, кроме username и password, тоже есть в */etc/shadow*, но обычно содержат фиктивные значения или пусты.

Учтите, что для использования теневых паролей нужны специальные версии программ, имеющих доступ или модифицирующих данные о пользователях, таких как *passwd* или *login*. Сейчас большинство дистрибутивов поставляется с уже установленными теневыми паролями, так что это не должно вызвать проблем. Пользователи Debian должны задать параметр «shadowconfig on», чтобы включить поддержку теневых паролей.

Есть два средства для преобразования «нормальных» учетных записей в теневые и обратно. Утилита *pwconv* берет файл */etc/passwd*, ищет в нем записи, которых еще нет в */etc/shadow*, создает для них теневые записи и объединяет с уже имеющимися в файле */etc/shadow*.

Утилита *pwunconv* используется редко, поскольку не усиливает защиту, а ослабляет ее. Она работает подобно *pwconv*, но создает обычные записи в */etc/passwd*, которые могут работать без соответствующих им в */etc/shadow*.

PAM и другие способы аутентификации

Если вы решили, что двух способов аутентификации, */etc/passwd* и */etc/shadow*, достаточно для выбора, то вы ошибаетесь. Есть ряд других методов аутентификации с необычными именами, такими как аутентификация Kerberos (по имени собаки в греческой мифологии, охранявшей вход в ад). Хотя мы считаем, что теневые пароли в большинстве случаев предоставляют достаточную защиту, все зависит от того, какая степень защиты вам действительно нужна и в какой степени вы страдаете паранойей.

Проблема с этими методами аутентификации состоит в том, что нет простого способа переключения с одного метода на другой, поскольку вместе с этими средствами всегда должен устанавливаться набор программ типа *login* и *passwd*. Для решения этой проблемы придумана система *подключаемых методов аутентификации* (*Pluggable Authentification Methods*, *PAM*). Если у вас есть набор средств, поддерживающих PAM, можете изменять используемый в системе метод аутентификации путем переконфигурирования PAM. Эти средства автоматически получают программный код, необходимый для требуемых процедур аутентификации, из динамически загружаемых библиотек совместного доступа.

Установка и применение PAM выходят за рамки данной книги, но необходимые сведения вы можете получить на *http://www.kernel.org/pub/linux/libs/pam/*. Большинство современных дистрибутивов устанавливает PAM.

Файл group

Группы пользователей являются удобным способом логической организации наборов учетных записей пользователей и предоставляют пользователям возможность иметь общий доступ к файлам внутри одной или нескольких групп. У каждого файла в системе есть пользователь и группа, которые им владеют. Для каждого конкретного файла вы можете посмотреть владельца и группу с помощью команды *ls -l*, например:

```
rutabaga% ls -l boiler.tex
-rwxrw-r--   1 mdw      megabozo    10316 Oct  6 20:19 boiler.tex
rutabaga%
```

Владельцем этого файла является пользователь `mdw`, и принадлежит он группе `megabozo`. Права доступа говорят о том, что пользователь `mdw` имеет права на чтение, запись и выполнение файла; все члены группы `megabozo` имеют права на чтение и запись; все остальные пользователи имеют только право на чтение.

Это не означает, что пользователь `mdw` является членом группы `megabozo`; просто доступ к файлу согласно битам разрешений предоставлен всем членам группы `megabozo` (в состав которой может входить или не входить пользователь `mdw`).

Таким образом, организован совместный доступ к файлам групп пользователей, а права доступа могут отдельно указываться для владельца, группы, к которой относится файл, и всех остальных. С правами доступа можно ознакомиться в разделе «Владение файлами и права доступа» главы 4.

Каждый пользователь принадлежит, по крайней мере, к одной группе, указанной в поле *gid* файла */etc/passwd*. Однако пользователь может быть членом нескольких групп. В файле */etc/group* содержится по одной строке для каждой группы в системе, что очень похоже на */etc/passwd*. Формат этого файла такой:

```
groupname:password:gid:members
```

Здесь *groupname* является строкой символов, идентифицирующей группу; это то имя группы, которое выводится такими командами, как *ls -l*.

password является необязательным паролем, присвоенным группе, который дает возможность пользователям, не принадлежащим этой группе, присоединяться к группе с помощью команды *newgrp*. Об этом будет рассказано ниже.

gid является ID группы, используемым системой при ссылке на группу; это число, используемое в поле *gid* файла */etc/passwd* для указания группы пользователя по умолчанию.

members – это список разделенных запятыми имен пользователей (без пробелов между ними), указывающий пользователей, являющихся членами этой группы, но имеющих другие значения *gid* в файле */etc/passwd*. Это означает, что в данный список не нужно включать имена тех пользователей, для которых эта группа указана как группа по умолчанию в */etc/passwd*, т. е. это список дополнительных членов группы.

Например, */etc/group* может содержать такие записи:

```
root:*:0:
bin:*:1:root,daemon
users:*:50:
bozo:*:51:linus,mdw
megabozo:*:52:kibo
```

Первые записи, для групп root и bin, являются административными группами, сходными, по сути, с «мнимыми» учетными записями в системе. Владельцами многих файлов являются группы, такие как root и bin. Другие группы предназначены для учетных записей пользователей. Подобно ID пользователей, значения ID групп пользователей часто принимают значения, большие 50 или 100.

Поле *password* файла групп представляет известный интерес. Используется оно не часто, но вместе с программой *newgrp* позволяет пользователям, не являющимся членами данной группы, принять ID этой группы при условии знания пароля. Например, команда

```
rutabaga% newgrp bozo
Password: пароль для группы bozo
rutabaga%
```

запускает новую оболочку с ID группы bozo. Если поле *password* пустое или первым символом является звездочка, то команда *newgrp* отказывает в доступе к этой группе.

Однако поле *password* файла групп используется редко и на самом деле не является необходимым. (Фактически большинство систем не имеет средств установки пароля для группы; можно воспользоваться *passwd*, чтобы установить пароль для фиктивного пользователя с именем, совпадающим с именем группы, в файле */etc/passwd* и скопировать зашифрованное поле пароля в файл */etc/group*.) Вместо этого можно сделать пользователя членом нескольких групп, просто включив имя пользователя в поле *members* для каждой дополнительной группы. В предыдущем примере пользователи linus и mdw являются членами группы bozo и той группы, к которой они приписаны в файле */etc/passwd*. Если мы хотим добавить пользователя linus еще и к группе megabozo, последнюю строку предыдущего примера нужно заменить такой:

```
megabozo:*:52:kibo,linus
```

Команда *groups* выводит список групп, к которым вы принадлежите, например:

```
rutabaga% groups
users bozo
```

Если передать команде *groups* список пользователей, она выведет списки групп, к которым принадлежит каждый из пользователей.

При входе в систему вам автоматически присваиваются ID группы из */etc/passwd*, а также дополнительных групп, в списки которых вы включены в файле */etc/group*. Это значит, что у вас есть «групповой доступ» ко всем файлам в системе, для которых ID группы содержится в вашем списке групп. В этом случае биты прав доступа к этому файлу для группы (установленные

командой *chmod g+...*) применимы к вам. (Если вы не являетесь его владельцем, то применяются биты прав доступа владельца.)

Теперь вам все известно о группах. Как же следует пользоваться возможностью создания групп? Это дело вкуса и зависит от того, как будет использоваться ваша система. Если пользователь в системе один или их всего несколько, проще создать единственную группу, скажем, users, к которой принадлежат все учетные записи. Заметьте, что все системные группы, содержащиеся в файле */etc/group* после первоначальной установки, скорее всего, следует сохранить: от них могут зависеть различные демоны и программы.

Если пользователей на машине много, то есть несколько способов организовать их в группы. Например, в учебном заведении может быть несколько групп для студентов, кафедр и персонала. В компании, разрабатывающей программное обеспечение, для каждой бригады разработчиков может быть создана группа. В некоторых случаях каждого пользователя помещают в отдельную группу с именем, совпадающим с именем пользователя. В результате каждый живет сам по себе. Для файлов можно создавать специальные группы. Многие пользователи создают новые группы и делают их владельцами файлов, чтобы обеспечить совместный доступ пользователей к этим файлам. Однако при этом нужно вводить пользователей в эти новые группы, для чего обычно требуется вмешательство администратора (для редактирования файла */etc/group* или действий с утилитой *gpasswd* в дистрибутиве Debian). Решайте сами, как поступить.

Другая ситуация, в которой часто используются группы, – это разделение по правам доступа к аппаратным устройствам. Допустим, что у вас есть сканер, доступ к которому осуществляется через */dev/scanner*. Если вы не хотите, чтобы доступ к сканеру имели все, создайте группу с именем scanner, разрешите этой группе доступ к файлу */dev/scanner*, сделав его доступным для чтения этой группе и недоступным для всех остальных, и добавьте всех, кому разрешено пользоваться сканером, в группу scanner в файле */etc/groups*.

Создание учетных записей

Создание учетной записи пользователя производится в несколько этапов: добавление записи в */etc/passwd*, создание исходного каталога пользователя и установка в нем файлов конфигурации пользователя по умолчанию (например, *.bashrc*). К счастью, не нужно делать это вручную; почти во всех системах Linux есть программа *adduser*, которая сделает это за вас.[1]

---

[1] В некоторых системах, например Red Hat или SuSE, для создания и удаления учетных записей используется другой набор средств. Если вам не подходит та последовательность действий, которая описана в данном разделе, почитайте документацию к своему дистрибутиву. Red Hat позволяет управлять учетными записями через сервис панели управления, а в SuSE для этого имеется *YaST*; в Debian есть неинтерактивный сценарий «adduser», автоматически создающий пользователей на основе файла конфигурации */etc/adduser.conf*. Кроме того, есть графические программы управления пользователями типа *kuser* в KDE (см. главу 11).

Работа с *adduser* от имени root выглядит так. В ответ на приглашение вы вводите запрашиваемые данные. Часто приглашение предлагает разумное значение по умолчанию, которое можно выбрать, нажав <Enter>.

```
Adding a new user. The username should not exceed 8 characters
in length, or you many run into problems later.
```
(Добавление нового пользователя. Имя пользователя не должно превышать в длину 8 символов, иначе вы можете столкнуться с трудностями.)

```
Enter login name for new account (^C to quit): norbert
```
(Введите имя регистрации для новой учетной записи)

```
Editing information for new user [norbert]
```
(Редактирование сведений о новом пользователе norbert)

```
Full Name: Norbert Ebersol
```
(Полное имя)

```
GID [100]: 117

Checking for an available UID after 500
First unused uid is 501
```
(Проверка свободных UID, больших 500
Первый неиспользуемый UID - 501)

```
UID [501]: (enter)
Home Directory [/home/norbert]: (enter)
Shell [/bin/bash]: (enter)
Password [norbert]: (norbert's password)

Information for new user [norbert]:
Home directory: [/home/norbert] Shell: [/bin/bash]
Password: [(norbert's password)] uid: [501] gid: [51]

Is this correct? [y/N]: y
```
(Верно ли это?)

```
Adding login [norbert] and making directory [/home/norbert]
Adding the files from the /etc/skel directory:
```
(Добавление учетной записи и создание каталога /home/norbert
Добавление файлов из каталога /etc/skel:)

```
./.emacs -> /home/norbert/./.emacs
./.kermrc -> /home/norbert/./.kermrc
./.bashrc -> /home/norbert/./.bashrc
...more files...
```

Здесь нет ничего необычного: просто вводите запрашиваемые данные или предлагаемые значения по умолчанию. Обратите внимание, что *adduser* использует 100 в качестве ID группы по умолчанию и ищет первый неиспользованный ID пользователя после 500 (500 используется как минимальное значение в SuSE и Red Hat, в Debian используется 1000). Принятие этих значений не должно вызвать конфликтов; в предыдущем примере мы ввели ID группы 117, поскольку решили, что это будет группа пользователя, и предложенный системой ID пользователя 501.

После создания учетной записи в исходный каталог пользователя копируются файлы из */etc/skel*. В */etc/skel* содержатся «скелеты» файлов для новой учетной записи, представляющие собой файлы конфигурации по умолчанию для нового пользователя (такие как *.emacs* и *.bashrc*). Можно поместить сюда и другие файлы, если они нужны для учетных записей новых пользователей.

После того как все это сделано, новая учетная запись готова к использованию. `Norbert` может входить в систему, используя пароль, заданный при работе *adduser*. В целях безопасности новый пользователь должен сменить пароль сразу после первого входа в систему, используя утилиту *passwd*.

`root` имеет возможность установить пароль для любого пользователя системы. Например, команда

```
passwd norbert
```

приглашает ввести новый пароль для пользователя `norbert`, не спрашивая при этом старый пароль. Имейте, конечно, в виду, что нужно знать пароль суперпользователя, чтобы поменять его. Если вы забудете пароль для `root`, можно загрузить Linux с аварийной дискеты и очистить в */etc/passwd* поле *password* записи для `root`. См. раздел «Действия в аварийной ситуации» главы 8.

В некоторых системах Linux вместо *adduser* есть управляемая из командной строки утилита *useradd*. Она требует задания всей необходимой информации в виде параметров командной строки. Если вы не можете найти *adduser* и завязли с *useradd*, обращайтесь за помощью к страницам руководства.

Удаление и блокировка учетных записей

Удалить учетную запись пользователя значительно легче, чем создать. Это хорошо известное понятие энтропии в действии. Для удаления записи нужно удалить запись о пользователе из */etc/passwd*, убрать все ссылки на пользователя из */etc/group* и удалить домашний каталог пользователя, а также другие файлы, владельцем или создателем которых пользователь является. Например, если у пользователя есть ящик для входящей электронной почты в */var/spool/mail*, его тоже нужно удалить.

Команда *userdel* (инь для ян – *useradd*) удаляет учетную запись и исходный каталог пользователя, например,

```
userdel -r norbert
```

удалит недавно созданную учетную запись для `norbert`. Параметр *-r* говорит о том, что надо удалить также домашний каталог. Прочие связанные с пользователем файлы, например почтовый ящик, файлы *crontab* и др., должны быть удалены вручную. Обычно они не важны и могут быть оставлены. К концу главы вы будете знать, где они находятся, если существуют. Быстро найти файлы, связанные с некоторым пользователем, можно с помощью команды

```
find / -user username -ls
```

Эта команда дает список в виде *ls -l* всех файлов, владельцем которых является *username*. Разумеется, для ее использования необходимо, чтобы учетная запись *username* все еще присутствовала в */etc/passwd*. Если вы уже удалили учетную запись, используйте параметр вида *-uid num*, где *num* – числовой ID в бозе почившего пользователя.

Еще проще при необходимости отключить запись пользователя на короткое (или не очень короткое) время. Можно удалить запись пользователя из */etc/passwd* (оставив нетронутыми домашний каталог и прочие его файлы) или добавить звездочку в качестве первого символа поля *password* записи в */etc/passwd*, например:

```
aclark:*BjDf5hBysDsii:104:50:Anna Clark:/home/aclark:/bin/bash
```

Тем самым будет блокирована регистрация по этой учетной записи. Если используются теневые пароли, то же самое нужно сделать в */etc/shadow*.

Изменение учетных записей пользователей

Для изменения атрибутов учетных записей пользователей и групп обычно достаточно отредактировать записи в файлах */etc/passwd* и */etc/group*. Во многих системах для этой цели есть такие команды, как *usermod* и *groupmod*. Обычно оказывается проще отредактировать файлы вручную.

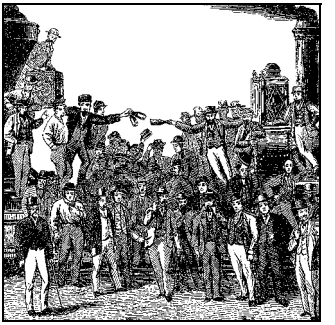
Для изменения пароля пользователя используется команда *passwd*, которая выдаст приглашение на ввод пароля, зашифрует его и запишет зашифрованный пароль в файл */etc/passwd* .

Если нужно изменить ID пользователя для существующей учетной записи, это можно сделать, непосредственно отредактировав поле *uid* в файле */etc/passwd*. Однако потребуется также изменить владельца файлов для этого пользователя на новый *uid*. Это можно сделать так:

```
chown -R aclark /home/aclark
```

При этом владельцем всех файлов в исходном каталоге, используемом aclark, снова станет aclark, если вы изменили *uid* для этой учетной записи. Если *ls -l* вместо имени пользователя выводит числовой ID, это значит, что нет имени пользователя, связанного с *uid*, владеющим файлами. Исправить положение можно с помощью команды *chown*.

6



Управление файловыми системами, свопингом и устройствами

Надо полагать, что вы создали файловые системы и пространство для свопинга во время первоначальной установки Linux (большинство дистрибутивов помогает сделать основную работу). Теперь есть возможность подвергнуть эти ресурсы тонкой настройке. Чаще всего такая работа производится вскоре после установки операционной системы, до того как начать загружать диски всякими интересными штучками. Но иногда требуется изменить уже работающую систему, чтобы добавить новое устройство или обновить пространство для свопинга после обновления RAM.

Управление файловыми системами

В Unix-системах файловая система представляет собой некоторое устройство (например, жесткий диск, гибкий диск или компакт-диск), отформатированное для хранения файлов. Файловые системы могут находиться на жестких дисках, гибких дисках, CD-ROM или других носителях, которые позволяют осуществлять произвольный доступ. (Замечание: магнитная лента позволяет осуществлять только последовательный доступ и потому не может содержать файловую систему как таковую.)

Точный формат и способы хранения файлов не имеют значения; система обеспечивает общий интерфейс для всех распознаваемых ею *типов файловых систем*. В Linux в число файловых систем входят «вторая расширенная файловая система», или *ext2fs*, в которой вы, возможно, уже храните файлы Linux (ее место постепенно занимает *ext3fs*), файловая система VFAT, позволяющая из Linux иметь доступ к файлам на разделах Windows 95/98/ME и гибких дисках, а также некоторые другие, в том числе файловая система ISO 9660, используемая на CD-ROM.

Все эти файловые системы имеют различные внутренние форматы хранения данных. Однако при доступе к любой файловой системе из Linux данные представляются в виде иерархии каталогов с находящимися в них файлами

вместе с идентификаторами владельцев и групп, битами прав доступа и прочими знакомыми вам характеристиками.

Фактически данные о владельцах, правах доступа и т. п. обеспечиваются только файловыми системами типов, предназначенных для хранения файлов Linux. Для файловых систем тех типов, которые не хранят такие данные, драйверы ядра, используемые для доступа к ним, «подделывают» их. Например, в файловой системе MS-DOS нет понятия владельца файла, поэтому все файлы представляются так, как если бы их владельцем был `root`. Благодаря этому, начиная с некоторого уровня все файловые системы выглядят одинаково, и у каждого файла есть некоторые связанные с ним атрибуты. Действительно ли эти данные используются внутри файловой системы, является уже другим вопросом.

В качестве системного администратора вы должны знать, как создавать файловые системы, когда вам требуется сохранить файлы Linux на гибком диске или добавить новые файловые системы на жестких дисках. Вы должны также знать, как пользоваться различными средствами для проверки и поддержки файловых систем, если данные окажутся повреждены. Кроме того, нужно знать команды и файлы, используемые для доступа к файловым системам, например на гибком диске или CD-ROM.

Типы файловых систем

В табл. 6.1 перечислены типы файловых систем, поддерживаемых ядром Linux по состоянию на версию 2.4.10. К системе постоянно добавляются новые файловые системы, и для некоторых файловых систем есть экспериментальные драйверы, здесь не отмеченные. Чтобы узнать, какие типы файловых систем поддерживает ваше ядро, взгляните на файл */proc/filesystems*. Файловые системы, которые необходимо поддерживать, можно выбрать при сборке ядра (см. раздел «Сборка ядра» главы 7).

Таблица 6.1. Типы файловых систем Linux

Файловая система	Тип	Описание
Second Extended	*ext2*	Наиболее часто используемая в Linux файловая система
Reiser filesystem	*reiserfs*	Журнальная файловая система для Linux
Third Extended	*ext3*	Еще одна журнальная файловая система для Linux, совместимая сверху вниз с *ext2*
Minix	*minix*	Первоначальная файловая система Minix; используется редко
ROM	*romfs*	Маленькая файловая система только для чтения; используется в основном с электронными дисками
CRAM	*cramfs*	Файловая система с компрессией, только для чтения, часто используется в PDA

Таблица 6.1 (продолжение)

Файловая система	Тип	Описание
Network File System (NFS)	NFS	Позволяет осуществлять доступ к удаленным файлам в сети
UMSDOS	*umsdos*	Устанавливает Linux на раздел MS-DOS
DOS-FAT	*msdos*	Предоставляет доступ к файлам MS-DOS
VFAT	*vfat*	Предоставляет доступ к файлам Windows 95/98
NT	*ntfs*	Предоставляет доступ к файлам Windows NT
HPFS	*hpfs*	Файловая система OS/2
/proc	*proc*	Обеспечивает сведения о процессах для *ps*
Device filesystem	*devfs*	Альтернативный способ представления файлов в каталоге */dev*
ISO 9660	*iso9660*	Используется в большинстве CD-ROM
Joliet	*iso9660*	Расширение файловой системы ISO 9660, способна работать с именами файлов в Unicode
UDF	*udf*	Самая современная файловая система для CD-ROM
System V	*sysv*	Предоставляет доступ к файлам клонов System V
Coherent	*coherent*	Предоставляет доступ к файлам Coherent
UFS	*ufs*	Предоставляет доступ к файлам файловых систем UFS, как в SunOS, BSD или True64 Unix
BFS	*bfs*	Доступ к файлам SCO Unixware
EFS	*efs*	Доступ к файлам старых версий IRIX
ADFS	*adfs*	Предоставляет доступ к файлам разделов Acorn
AFFS	*affs*	Предоставляет доступ к файлам стандартных разделов AmigaOS
Apple Mac	*hfs*	Предоставляет доступ к файлам Apple Macintosh
QNX4	*qnx4*	Предоставляет доступ к файлам разделов QNX4
JFFS	*jffs*	Файловая система для устройств флэш-памяти
Novell	*ncpfs*	Предоставляет доступ к файлам сервера Novell
SMB	*smbfs*	Предоставляет доступ к файлам Windows for Workgroups или Windows NT server
Coda	*coda*	Развитая сетевая файловая система, аналогичная NFS
RAM	*ramfs*	Файловая система для электронных дисков
Temporary filesystem	*tmpfs*	Еще одна файловая система, целиком располагающаяся в RAM

Файловая система каждого типа имеет свои атрибуты и ограничения. Например, файловая система MS-DOS ограничивает имена восемью символами и тремя символами расширения и должна использоваться только для доступа к имеющимся гибким дискам и разделам MS-DOS. Работая с Linux, вы, по большей части, будете использовать вторую расширенную файловую систему, разработанную в основном для Linux, поддерживающую 256-символьные имена файлов, размер до 32 терабайт и имеющую массу прочих достоинств. Можно также воспользоваться файловой системой Райзера (*reiserfs*) или третьей расширенной (*ext3*). Ранее в Linux использовались Extended filesystem (более не поддерживается) и файловая система Minix. (Первоначально по ряду причин использовалась файловая система Minix. Во-первых, Linux изначально кросс-компилировалась под Minix. Кроме того, Линус хорошо знал файловую систему Minix, и ее было просто реализовать в первых вариантах ядра.) Файловые системы Xia и Xenix, доступные в старых версиях ядра Linux, больше не поддерживаются.

Основное отличие второй расширенной файловой системы от файловой системы Райзера в том, что последняя является *журнальной*. Журналирование – это передовая технология, в которой отслеживаются изменения, выполненные в файловой системе, что значительно облегчает (и ускоряет!) восстановление поврежденных файловых систем (например, после краха системы или отказа питания). Другой журнальной файловой системой является третья расширенная, преемница второй расширенной. Ее достоинство в том, что она совместима сверху вниз со второй расширенной файловой системой, но поскольку она появилась недавно, то не так широко используется, как вторая расширенная и файловая система Райзера.

Файловая система ROM вам редко понадобится. Она очень маленькая, не поддерживает операции записи и предназначена для использования с электронными дисками при конфигурировании системы, во время запуска и даже в EPROMS. К той же категории относится файловая система Cram, которая также используется для ROM и сжимает свои данные. Она предназначена в основном для встроенных устройств, где ценится экономия памяти.

Файловая система UMSDOS используется для установки Linux в закрытый каталог на существующем разделе MS-DOS. Это хороший способ для новичков потренироваться с Linux, не производя разбиения диска на разделы. Напротив, файловая система DOS-FAT используется для прямого доступа к файлам MS-DOS. Доступ к файлам разделов, созданных с помощью Windows 95 или 98, может быть осуществлен с помощью файловой системы VFAT, а файловая система NTFS осуществляет доступ к файловым системам Windows NT. Файловая система HPFS используется для доступа к файловой системе OS/2.

Расширение CVF-FAT файловой системы DOS-FAT позволяет осуществлять доступ к разделам, сжатым с помощью DoubleSpace/DriveSpace производства Microsoft или Stacker производства Stac. Дополнительные подробности см. в файлах *Documentation/filesystems/fat_cvf.txt* в исходных текстах ядра Linux.

/proc является виртуальной файловой системой. Это означает, что с ней не связано реальное дисковое пространство. См. раздел «Файловая система /proc» главы 5.[1]

Файловая система *devfs* является виртуальной, как и */proc*. Она предназначена для замены текущего каталога */dev* (см. раздел «Файлы устройств» ниже в этой главе). Ее преимущество в том, что специальные файлы, которые в ней находятся, не нужно создавать вручную системному администратору: они создаются автоматически по запросу ядра.

Файловая система ISO 9660 (ранее известная как High Sierra Filesystem и сокращаемая как *hsfs* в других Unix-системах) используется на большинстве CD-ROM. Подобно MS-DOS, эта файловая система ограничивает длину имени файла и хранит лишь часть информации о каждом файле. Однако в большинстве CD-ROM применяется расширение Rock Ridge Extensions для ISO 9660, позволяющее драйверу файловой системы ядра присваивать каждому файлу длинные имена, права владения и доступа. В результате при доступе к ISO 9660 CD-ROM из MS-DOS получаются имена в формате 8.3, но при доступе из Linux получаются «подлинные» полные имена файлов.

Кроме того, Linux теперь поддерживает расширение Microsoft Joliet для ISO 9660, позволяющее обращаться с длинными именами в кодировке Unicode. В настоящее время это используется не широко, но в будущем может стать ценным достоинством, поскольку кодировка Unicode стала международно принятым стандартом кодировки сценариев.

Недавно в Linux была также включена поддержка UDF – файловой системы, предназначенной для использования с CD-RW и DVD.

Далее у нас есть шесть файловых систем для других платформ. Linux поддерживает форматы, принятые на этих платформах, чтобы осуществлять двойную загрузку и другие виды взаимодействия. Речь идет о системах UFS, EFS, BFS, System V и Coherent. (Последние две обрабатываются фактически одним и тем же драйвером ядра с несколько различающимися параметрами.) Если у вас есть файловые системы, созданные в одном из этих форматов в другой операционной системе, вы сможете иметь к ним доступ из Linux.

Наконец, есть масса файловых систем для доступа к данным на разделах, созданных в операционных системах, принадлежащих иным семействам, нежели DOS и Unix. Эти файловые системы поддерживают Acorn Disk Filing System (ADFS), файловые системы AmigaOS (без поддержки гибких дисков, за исключением компьютеров Amiga), Apple Mac HFS и QNX4. Большинство специальных файловых систем полезно только на особых аппаратных архи-

---

[1] Учтите, что файловая система */proc* имеет другой формат, нежели файловая система */proc* в SVR4 (скажем, в Solaris 2.x). В SVR4 каждый выполняющийся процесс имеет в */proc* один «файл», который можно открыть и производить с ним действия с помощью некоторых вызовов *ioctl*(), чтобы получить сведения о процессе. Напротив, Linux предоставляет большинство сведений в */proc* с помощью запросов *read*() и *write*().

тектурах. Например, на машине Intel вы не найдете жестких дисков, отформатированных в файловой системе Amiga FFS. Если вам понадобится один из этих драйверов, прочтите поступившую с ним информацию. Некоторые из них находятся в экспериментальной стадии.

Помимо этих файловых систем для доступа к локальным жестким дискам существуют сетевые файловые системы для доступа к удаленным ресурсам. Рассказ о них выходит за рамки данной книги. Адресуем вас к «Linux Network Administrator's Guide» Олафа Кирха (Olaf Kirch) и Терри Доусона (Terry Dawson), O'Reilly.

Монтирование файловых систем

Для доступа к любой файловой системе из Linux нужно монтировать ее в некотором каталоге, в результате чего ее файлы выглядят так, будто они находятся в заданном каталоге и доступны для вас.

Прежде чем рассказывать о том, как монтировать файловые системы, отметим, что некоторые дистрибутивы устанавливают автоматическое монтирование, при котором требуется лишь вставить дискету или CD в соответствующий привод, после чего можно работать с ними, как на других платформах. Однако бывают случаи, когда нужно уметь смонтировать и размонтировать носитель непосредственно.

Для осуществления этого используется команда *mount*, для выполнения которой обычно требуются привилегии суперпользователя. (Как мы увидим ниже, обычные пользователи могут выполнять команду *mount*, если устройство перечислено в файле */etc/fstab*.) Формат команды следующий:

```
mount -t type device mount-point
```

где *type* является типом файловой системы согласно табл. 6.1, *device* является физическим устройством, на котором находится файловая система (файл устройства в */dev*), а *mount-point* является каталогом, в который нужно монтировать файловую систему. Перед тем как использовать команду *mount*, этот каталог должен быть создан.

Например, если на разделе */dev/hda2* у вас есть вторая расширенная файловая система и вы хотите монтировать ее в каталог */mnt*, выполните команду:

```
mount -t ext2 /dev/hda2 /mnt
```

При благоприятном исходе вы сможете обращаться к файловой системе в каталоге */mnt*. Аналогично для монтирования гибкого диска, созданного в Windows и потому имеющего формат DOS, нужно использовать команду:

```
mount -t msdos /dev/fd0 /mnt
```

В результате файлы, имеющиеся на гибком диске в формате MS-DOS, будут доступны в каталоге */mnt*. Обратите внимание, что аргумент `msdos` означает использование старого формата DOS, допускающего для имен файлов 8 символов плюс 3 символа расширения. Если задать вместо него `vfat`, вы получите более новый формат, введенный в Windows 95. Конечно, гибкий или жесткий диск также должен быть записан в этом формате.

У команды *mount* много опций, которые можно задавать с ключом *-o*. Например, файловые системы MS-DOS и ISO 9660 поддерживают «автоконвертирование» текстовых файлов из формата MS-DOS (они содержат в конце каждой строки CR-LF) в формат Unix (в котором в конце каждой строки есть только символ перевода строки). Например, такая команда, как

```
mount -o conv=auto -t msdos /dev/fd0 /mnt
```

включает это преобразование для файлов, расширения которых не связываются с двоичными файлами (такими как *.exe*, *.bin* и т. д.).

При монтировании часто используется опция *-o ro* (что эквивалентно *-r*), в результате чего файловая система монтируется только для чтения. На все попытки записи в такую файловую систему возвращается сообщение об ошибке «permission denied». Монтирование файловой системы только для чтения необходимо для таких носителей, как CD-ROM, на который нельзя осуществлять запись. Можно успешно монтировать CD-ROM без параметра *-r*, но вы получите раздражающее предупредительное сообщение:

```
mount: block device /dev/cdrom is write-protected, mounting read-only
```

Лучше выполните команду

```
mount -t iso9660 -r /dev/cdrom /mnt
```

Это также необходимо и при монтировании гибкого диска с установленной на нем защитой от записи.

На странице руководства по *mount* перечислены все имеющиеся параметры. Не все из них представляют непосредственный интерес, но когда-нибудь они вам, возможно, понадобятся. Полезен вариант команды *mount -a*, который монтирует все файловые системы, перечисленные в */etc/fstab/*, исключая помеченные опцией `noauto`.

Обратной операцией по отношению к монтированию является, естественно, размонтирование. Размонтирование файловой системы влечет два последствия: происходит синхронизация системных буферов с фактическим содержанием файловой системы на диске, и файловая система перестает быть доступной через свою точку монтирования (mount point). После размонтирования вы можете монтировать в той же точке другую файловую систему.

Размонтирование производится командой *umount* (обратите внимание на отсутствие первого «n» в слове «unmount»), например

```
umount /dev/fd0
```

размонтирует файловую систему на */dev/fd0*. Аналогично, для размонтирования любой файловой системы, монтированной в данное время в некотором каталоге, используйте команду

```
umount /mnt
```

Важно отметить, что сменные носители, в том числе дискеты и CD-ROM, нельзя вынимать из дисковода или заменять другими, пока смонтирована файловая система. В результате такого действия система возникает рассинхронизация системной информации об устройстве с его фактическим состоя-

нием, что может привести к бесконечным проблемам. Если вы хотите поменять дискету или CD-ROM, сначала размонтируйте их командой *umount*, а затем вновь смонтируйте. Конечно, если речь идет о CD-ROM или защищенной от записи дискете, то самоустройство не может рассинхронизироваться, но могут возникнуть другие проблемы. Например, вы не сможете вынуть CD-ROM из привода, пока он не размонтирован.

Чтение и запись на гибкие диски буферизуются в памяти так же, как для жестких дисков. Это значит, что при чтении или записи можно сразу не увидеть работы дисковода. Система обрабатывает ввод/вывод на дискеты асинхронно и читает или записывает данные только тогда, когда это абсолютно необходимо. Поэтому если вы записываете на дискету маленький файл, и лампочка привода при этом не загорается, не впадайте в панику, в свое время данные будут записаны. Принудить систему записать на диск все буферы файловой системы можно командой *sync*, которая осуществит физическую запись буферизованных данных. То же происходит при размонтировании файловой системы.

Если вы хотите, чтобы и обычные пользователи могли производить монтирование и размонтирование некоторых устройств, есть два выхода. Во-первых, можно включить параметр `user` для этого устройства в файл */etc/fstab* (описывается ниже в этом разделе). Это позволяет любому пользователю пользоваться командами *mount* и *umount* для данного устройства. Другой способ – использование одного из интерфейсов монтирования более высокого уровня, имеющихся в Linux. Эти программы выполняются с установленным битом *setuid* и позволяют обычным пользователям монтировать некоторые устройства. Как правило, вы не позволите обычным пользователям монтировать и размонтировать разделы на жестких дисках, но к использованию CD-ROM и гибких дисков можно быть лояльнее.

При попытке монтирования файловой системы может возникнуть немало проблем. К сожалению, команда *mount* выдает одно и то же сообщение об ошибке при возникновении разных проблем:

```
mount: wrong fs type, /dev/cdrom already mounted, /mnt busy, or other error
```

`wrong fs type` означает, что вы, возможно, указали неверный тип файловой системы для монтирования. Если вы вообще не указали тип, *mount* пытается угадать тип файловой системы по суперблоку (это работает только для *minix*, *ext* и *iso9660*). Если *mount* все же не может определить тип файловой системы, она опробует те типы, драйверы которых включены в ядро (перечислены в */proc/filesystems*). Если и это не приведет к успеху, *mount* завершит работу с ошибкой. *device* `already mounted` означает, что устройство уже смонтировано в другом каталоге. Узнать, какие устройства смонтированы и где, можно с помощью команды *mount* без параметров:

```
rutabaga# mount
/dev/hda2 on / type ext2 (rw)
/dev/hda3 on /windows type vfat (rw)
/dev/cdrom on /cdrom type iso9660 (ro)
/proc on /proc type proc (rw,none)
```

Мы видим, что есть два раздела на жестких дисках, один типа *ext2*, а другой типа *vfat*, CD-ROM, смонтированный на */cdrom*, а также файловая система */proc*. Последнее поле каждой строки (например, `(rw)`) перечисляет параметры, с которыми смонтирована файловая система. Подробнее об этом будет рассказано далее. Обратите внимание на монтирование CD-ROM в */cdrom*. Если вы часто пользуетесь CD-ROM, то удобно создать каталог */cdrom* и монтировать в него устройство. */mnt* обычно используется для временного монтирования файловых систем, таких как гибкие диски.

Сообщение *mount-point* `busy` несколько необычно. В сущности, оно означает, что в точке монтирования (*mount-point*) происходит какая-то активность, не позволяющая монтировать в ней файловую систему. Обычно это связано с тем, что в этом каталоге есть открытый файл или у некоторого процесса рабочий каталог находится ниже точки монтирования. При использовании *mount* убедитесь, что текущий каталог в оболочке суперпользователя не находится внутри точки монтирования; выполните команду *cd /*, чтобы попасть в каталог верхнего уровня. Либо может оказаться, что в той же точке монтирования уже смонтирована другая файловая система. Выполните команду *mount* без аргументов, чтобы проверить, в чем дело.

Конечно, сообщение `other error` не очень содержательно. Есть и другие ситуации, в которых *mount* может завершиться с ошибкой. Если в рассматриваемой файловой системе есть ошибки данных или носителя, *mount* может сообщить о невозможности прочесть *суперблок* (*superblock*) файловой системы, который (в Unix-подобных системах) является частью файловой системы, хранящей данные о файлах и атрибутах для файловой системы в целом. При попытке монтировать CD-ROM или гибкий диск и отсутствии при этом в соответствующем приводе CD-ROM или дискеты вы получите сообщение об ошибке типа:

```
mount: /dev/cdrom is not a valid block device
```

Дискеты особенно склонны к физическим дефектам (больше, чем вам может изначально показаться), а CD-ROM страдают от пыли, царапин и отпечатков пальцев, а также вставки рабочей стороной вверх и т. п. (При попытке монтирования Stan Rogers CD в качестве ISO 9660 вы можете столкнуться со схожими проблемами.)

Убедитесь также в существовании точки монтирования (например, */mnt*), которую вы пытаетесь использовать. Если ее нет, можно просто создать ее командой *mkdir*.

Проблемы при монтировании или доступе к файловой системе могут быть вызваны повреждением данных. В Linux есть ряд программных средств, позволяющих исправлять некоторые файловые системы; см. раздел «Проверка и исправление файловых систем» далее в этой главе.

При начальной загрузке система автоматически монтирует несколько файловых систем. Это определяется файлом */etc/fstab*, в котором есть записи для всех файловых систем, которые должны монтироваться при начальной загрузке системы. Все строки этого файла имеют формат

```
device mount-point type options
```

Здесь *device*, *mount-point* и *type* эквивалентны значениям в команде *mount*, а *options* представляет собой разделенный запятыми список опций ключа *-o* для команды *mount*.

Вот образец файла */etc/fstab*:

```
# device        directory       type      options
/dev/hda2       /               ext2      defaults
/dev/hda3       /windows        vfat      defaults
/dev/cdrom      /cdrom          iso9660   ro
/proc           /proc           proc      none

/dev/hda1       none            swap      sw
```

В последней строке этого файла указан раздел для свопинга. Описание см. в разделе «Управление пространством для свопинга» далее в этой главе.

На странице руководства *mount*(8) перечислены возможные значения *option*. Если вы хотите задать более одной опции, перечислите их через запятую без пробелов, например:

```
/dev/cdrom      /cdrom          iso9660   ro,user
```

Опция user позволяет монтировать файловую систему пользователям, отличным от root. Если задана эта опция, пользователи могут монтировать устройства с помощью таких команд, как

```
mount /cdrom
```

Если вы указываете только устройство или точку монтирования (но не то и другое одновременно), производится поиск устройств или точек монтирования в */etc/fstab*, и устройство монтируется с заданными там параметрами. Это позволяет легко монтировать устройства, перечисленные в */etc/fstab*.

Для большинства файловых систем следует использовать опцию defaults, которая включает ряд других опций, таких как rw (доступ для чтения и записи), async (асинхронная буферизация в памяти ввода/вывода в файловой системе) и т. д. Если только у вас нет особой нужды модифицировать один из этих параметров, используйте для большинства файловых систем значение defaults и ro для устройств только для чтения, таких как CD-ROM. Другим полезным параметром является umask, позволяющий задать маску по умолчанию для битов прав доступа, что особенно полезно при использовании чужеродных файловых систем.

Команда *mount -a* монтирует все файловые системы, перечисленные в */etc/fstab*. Эта команда выполняется при начальной загрузке одним из сценариев, находящихся в */etc/rc.d*, например *rc.sysinit* (или другим, где ваш дистрибутив хранит конфигурационные файлы). Благодаря этому все файловые системы, перечисленные в */etc/fstab*, станут доступными после запуска системы: разделы жесткого диска, привод CD-ROM и др. – все они будут смонтированы.

Исключение составляет *корневая файловая система* (*root filesystem*). Корневая файловая система, монтируемая на /, обычно содержит файл */etc/fstab* и

сценарии из */etc/rc.d*. Для обеспечения доступа к ним ядро должно само смонтировать корневую файловую систему во время начальной загрузки. Устройство, содержащее корневую файловую систему, должно быть прошито в образе ядра, что можно сделать командой *rdev* (см. раздел «Использование загрузочной дискеты» в главе 5). Во время начальной загрузки ядро пытается установить это устройство в качестве корневой файловой системы, последовательно перебирая ряд типов файловых систем. При загрузке ядро может вывести сообщение типа

```
VFS: Unable to mount root fs
```

Оно может быть вызвано следующими причинами:

- В ядре прошито неверное корневое устройство.
- В ядре не скомпилирована поддержка того типа файловой системы, который имеет корневое устройство. (Подробности см. в разделе «Сборка ядра» главы 7. Обычно это происходит только при сборке собственного ядра.)
- Корневое устройство неким образом повреждено.

В любом из этих случаев ядро не может продолжить работу и впадает в панику. См. раздел «Действия в аварийной ситуации» в главе 8, чтобы узнать о том, что делать в такой ситуации. Если проблема состоит в повреждении файловой системы, обычно это поправимо; см. раздел «Проверка и исправление файловых систем» далее в этой главе.

Для того чтобы быть смонтированной, файловая система не обязана быть перечисленной в */etc/fstab*, но она должна быть перечислена там для «автоматического» монтирования командой *mount -a* или для использования при монтировании параметра `user`.

Автоматическое монтирование устройств

Если необходим доступ к нескольким файловым системам, особенно сетевым, вас может заинтересовать достаточно новое добавление к ядру Linux: *автомонтировщик* (*automounter*). Это комбинация функций ядра, демона и некоторых файлов конфигурации, автоматически определяющих потребность в доступе к определенной файловой системе и прозрачно монтирующих эту файловую систему. Если файловая система в течение некоторого времени не используется, *автомонтировщик* автоматически размонтирует ее, чтобы сберечь такие ресурсы, как оперативная память и пропускная способность сети.

При намерении пользоваться автомонтировщиком необходимо, прежде всего, включить эту функцию при сборке ядра. (См. раздел «Сборка ядра» в главе 7.) Понадобится также включить параметр NFS.

Далее необходимо запустить демон *automount*. Это совершенно новая функция, и в вашем дистрибутиве ее может не быть. Посмотрите в каталоге */usr/lib/autofs*, и если ее там нет, нужно получить пакет *autofs* из доступного вам архива Linux, скомпилировать и установить его согласно прилагаемым инструкциям.

Имейте в виду, что есть две версии поддержки автомонтировки: версия 3 и версия 4. В большинство дистрибутивов по-прежнему включается версия 3, поэтому ее мы и опишем.

Можно автоматически монтировать файловые системы в любом месте, но для простоты мы предполагаем, что вы хотите автоматически монтировать все файловые системы под одним каталогом, который мы назовем */automount*. Если вы хотите, чтобы точки автоматического монтирования были разбросаны по всей файловой системе, необходимо будет использовать несколько демонов *automount*.

Если вы сами скомпилировали пакет *autofs*, неплохо начать с копирования примеров файлов конфигурации, которые можно найти в каталоге *sample*, и приспособить их для своих нужд. Скопируйте файлы *sample/auto.master* и *sample/auto.misc* в каталог */etc*, а файл *sample/rc.autofs* под именем *autofs* в то место, где ваш дистрибутив хранит сценарии начальной загрузки. Допустим, что это */etc/init.d*.

Сначала нужно отредактировать файл конфигурации */etc/auto.master*. В нем перечислены все каталоги (так называемые *точки монтирования*), ниже которых автомонтировщик должен монтировать разделы. Поскольку в примере этой главы мы решили использовать только один раздел, нам нужно будет сделать только одну запись. Файл может выглядеть так:

```
/automount    /etc/auto.misc
```

Файл состоит из строк, в каждой из которых два элемента, разделенных пробелом. Первый элемент указывает на точку монтирования, а второй элемент – так называемый *файл соответствия* (*map file*), в котором указано, как и когда должны быть монтированы автоматически монтируемые устройства или разделы. Такой файл соответствия должен иметься для каждой точки монтирования.

В нашем случае файл */etc/auto.misc* выглядит так:

```
cd          -fstype=iso9660,ro      :/dev/scd0
floppy      -fstype=auto            :/dev/fd0
```

Этот файл состоит из элементов, каждый из которых задает устройство или раздел, которые должны автоматически монтироваться. Строки состоят из двух обязательных и одного дополнительного полей, разделенных пробелами. Первый элемент обязателен и указывает каталог, в который должны быть автоматически монтированы устройство или раздел. Это значение добавляется к точке монтирования, так что CD-ROM автоматически монтируется в */automount/cd*.

Второй элемент необязателен и задает флаги, которые должны быть применены при операции монтирования. Эти флаги эквивалентны используемым с командой *mount*, за исключением типа, который задается с помощью *-fstype=*, а не *-t*.

Наконец, третий элемент указывает на раздел или устройство, которые должны быть смонтированы. В нашем случае мы указываем первый привод

SCSI CD-ROM и первый гибкий диск соответственно. Двоеточие перед элементом необязательно. Оно отделяет часть узла от части устройство/каталог, как в *mount*. Поскольку эти два устройства находятся на локальной машине, слева от двоеточия пусто. При желании автоматически монтировать каталог *sources* с NFS-сервера sourcemaster нужно задать примерно следующее:

```
sources    -fstype=nfs,soft    sourcemaster:/sources
```

После редактирования файла конфигурации, чтобы привести его в соответствие с вашей системой, можно запустить демон автомонтирования, выполнив команду (замените путь соответствующим вашей системе):

```
tigger# /sbin/init.d/autofs start
```

Поскольку эта команда весьма неразговорчива, нужно проверить, действительно ли началось автомонтирование. Это можно сделать так:

```
tigger# /sbin/init.d/autofs status
```

Однако из вывода этой команды нелегко определить, действительно ли запущено автомонтирование, поэтому лучше всего проверить, существует ли процесс *automount*:

```
tigger# ps -aux | grep automount
```

Если эта команда показывает процесс *automount*, все должно быть в порядке. Если нет, нужно снова проверить конфигурационные файлы. Может случиться, что отсутствует поддержка в ядре: либо вообще не включена поддержка автомонтирования, либо она скомпилирована в виде модуля, который не установлен. В последнем случае можно поправить дело, выполнив команду

```
tigger# modprobe autofs
```

Если это не поможет, нужно выполнить:[1]

```
tigger# modprobe autofs4
```

Когда автомонтирование заработает так, как нужно, можно поместить вызовы *modprobe* и *autofs* в один из стартовых системных конфигурационных файлов, например */etc/rc.local*, */sbin/init.d/boot.local* или другой файл, используемый в вашем дистрибутиве.

Если все правильно настроено, то достаточно обратиться к какому-либо каталогу ниже точки монтирования, и автомонтировщик смонтирует вам соответствующее устройство или раздел. Например, если вы введете:

```
tigger$ ls /automount/cd
```

автомонтировщик смонтирует CD-ROM, чтобы *ls* могла вывести его содержание. Единственная разница между обычным и автоматическим монтированием состоит в том, что при автомонтировании заметна небольшая задержка перед выводом содержимого каталога.

---

[1] О команде *modprobe* будет рассказано в следующей главе.

С целью сбережения ресурсов автомонтировщик размонтирует устройство или раздел, если к нему нет обращения в течение некоторого времени (по умолчанию 5 минут).

У автомонтировщика есть несколько более тонких параметров. Например, вам необязательно читать из файла таблицу соответствия, но можно обращаться к системным базам данных или даже позволить автомонтировщику запустить программу, выдача которой будет использована в качестве таблицы соответствия. Подробности см. на страницах руководства *autofs*(5) и *automount*(8).

Создание файловых систем

Файловую систему можно создать командой *mkfs*. Создание файловой системы аналогично форматированию раздела или дискеты, позволяющему записывать на них файлы.

С каждой файловой системой связана своя команда *mkfs*, например файловые системы MS-DOS могут быть созданы с помощью *mkfs.msdos*, вторые расширенные файловые системы – с помощью *mkfs.ext2* и т. д. Сама программа *mkfs* является интерфейсом более высокого уровня, позволяющим создавать файловые системы любого типа, запуская соответствующие версии *mkfs*.[1]

При установке Linux вы, возможно, вручную создавали файловые системы с помощью такой команды, как *mke2fs*. (Если нет, то программа установки сделала это за вас.) В действительности *mke2fs* эквивалентна *mkfs.ext2*. Эти программы одинаковы (а во многих системах одна является символической ссылкой на другую), но файл с именем *mkfs.*`fs-type` облегчает для *mkfs* выполнение специфической для типа файловой системы программы. Если у вас нет *mkfs*, можете пользоваться непосредственно *mke2fs* или *mkfs.ext2*.

Если вы используете *mkfs*, то файловую систему можно создать с помощью следующей команды:

```
mkfs -t type device
```

где `type` – тип создаваемой файловой системы из указанных в табл. 6.1, а `device` является устройством, на котором создается файловая система (например, */dev/fd0* для гибкого диска).

Например, для создания файловой системы *ext2* на дискете выполните команду

```
mkfs -t ext2 /dev/fd0
```

Можно также создать дискету MS-DOS с помощью параметра *-t msdos*.

Теперь мы можем монтировать дискету, как указано в предыдущем разделе, скопировать на нее файлы и т. д. Не забудьте размонтировать дискету перед тем, как вынуть ее из дисковода.

---

[1] В Linux команда *mkfs* исторически создавала файловые системы Minix. На новых системах Linux *mkfs* создает любые файловые системы, а файловые системы Minix создаются с помощью *mkfs.minix*.

При создании файловой системы уничтожаются все данные в соответствующем физическом устройстве (дискете, в разделе жесткого диска и пр.). *mkfs* обычно не дает подсказки перед созданием файловой системы, поэтому тщательно контролируйте свои действия.

Создание файловой системы в разделе жесткого диска производится совершенно так же, как показано выше, за исключением того, что в качестве устройства нужно использовать имя раздела, например */dev/hda2*. Не пытайтесь создать файловую систему на устройстве, таком, например, как */dev/hda*. Это имя относится ко всему диску, а не к отдельному его разделу. Создать раздел можно с помощью *fdisk*, как описано в разделе «Создание разделов в Linux» главы 3.

Следует проявлять особую осторожность при создании файловых систем в разделах жестких дисков. Тщательно проверьте правильность аргументов `device` и `size`. При задании неправильного устройства `device` вы можете разрушить данные в текущих файловых системах, а если неверно задать `size`, то можно затереть данные других разделов. Убедитесь в том, что `size` соответствует размеру раздела, как он показан командой Linux *fdisk*.

При создании файловых систем на гибких дисках лучше всего сначала сделать форматирование низкого уровня, помещающее на дискету информацию о секторах и дорожках, с помощью которой автоматически определяется емкость дискеты при использовании устройств */dev/fd0* или */dev/fd1*. Один из способов произвести низкоуровневое форматирование – использовать команду MS-DOS FORMAT; другой способ – команда Linux *fdformat*.[1] Например, отформатировать дискету в первом дисководе можно командой

```
rutabaga# fdformat /dev/fd0
Double-sided, 80 tracks, 18 sec/track. Total capacity 1440 kB.
Formatting ... done
Verifying ... done
```

При использовании параметра *-n* команда *fdformat* пропускает этап верификации.

Каждая версия *mkfs*, специфическая для некоторой файловой системы, поддерживает несколько параметров, которые могут оказаться полезными. В большинстве типов поддерживается параметр *-c*, который указывает на необходимость проверки физического носителя на наличие сбойных блоков при создании файловой системы. Если обнаруживаются сбойные блоки, они помечаются и пропускаются при записи данных в файловую систему. Чтобы использовать параметры, специфические для типа файловой системы, включайте их в *mkfs* после параметра *-t type*, например:

```
mkfs -t type -c device blocks
```

Для того чтобы выяснить, какие параметры имеются, посмотрите страницы руководства по конкретной версии *mkfs*. (Например, для второй расширенной файловой системы см. *mke2fs*.)

---

[1] Пользователи Debian должны использовать вместо нее *superformat*.

У вас могут быть установлены не все имеющиеся типы *mkfs*. В этом случае *mkfs* не сможет создать файловую систему нужного вам типа, если для него не найдено *mkfs.type*. Для многочисленных типов файловых систем, поддерживаемых Linux, всегда где-то есть соответствующая программа *mkfs.type*.

Если возникают проблемы при использованием *mkfs*, они могут быть связаны с проблемами доступа Linux к физическому устройству. В случае дискеты это может просто означать, что она плохая. В случае жесткого диска проблемы могут быть более серьезными. Например, могут быть проблемы чтения диска у драйвера дискового устройства, находящегося в ядре. Это может быть связано с аппаратным устройством или просто неверным заданием геометрии диска. См. страницы руководства по различным версиям *mkfs* и разделы главы 3, посвященные проблемам, возникающим во время установки, они применимы и в данной ситуации.[1]

Проверка и исправление файловых систем

Иногда необходимо проверить целостность файловых систем Linux и исправить их при наличии ошибок или утерянных данных. Такие ошибки обычно происходят при крахе системы или выключении питания, когда ядро не может синхронизировать кэш файловой системы с содержимым диска. Такие ошибки происходят относительно редко. Однако при аварии системы во время записи большого файла этот файл может быть потерян, а связанные с ним блоки помечены как «используемые», хотя фактически не существует файла, который их использует. В других случаях ошибки могут быть вызваны случайной записью данных непосредственно в устройство жесткого диска (например, */dev/hda*) или один из его разделов.

Программа *fsck* используется для проверки файловых систем и исправления возникающих проблем. Подобно *mkfs*, программа *fsck* служит интерфейсом для специфических в отношении файловых систем программ *fsck.type*, таких как *fsck.ext2* для второй файловой системы. (Как и *mkfs.ext2*, *fsck.ext2* является символической ссылкой на *e2fsck*, и каждую из них можно запустить непосредственно, если *fsck* не установлена.)

Пользоваться *fsck* очень просто. Формат команды такой:

```
fsck -t type device
```

где *type* — тип файловой системы, которую нужно восстановить, согласно табл. 6.1, а *device* является устройством (разделом жесткого диска или гибким диском), на котором находится файловая система.

Например, для проверки файловой системы *ext2* на */dev/hda2* выполните команду

```
rutabaga# fsck -t ext2 /dev/hda2
Parallelizing fsck version 1.06 (7-Oct-96)
```

[1] Кроме того, процедура создания файловой системы ISO9660 для CD-ROM сложнее, чем простое форматирование и копирование файлов. Подробности см. в CD-Writing HOWTO.

```
e2fsck 1.06, 7-Oct-96 for EXT2 FS 0.5b, 95/08/09
/dev/hda2 is mounted.  Do you really want to continue (y/n)? y

/dev/hda2 was not cleanly unmounted, check forced.
Pass 1: Checking inodes, blocks, and sizes
Pass 2: Checking directory structure
Pass 3: Checking directory connectivity
Pass 4: Check reference counts.
Pass 5: Checking group summary information.

Free blocks count wrong for group 3 (3331, counted=3396).  FIXED
Free blocks count wrong for group 4 (1983, counted=2597).  FIXED
Free blocks count wrong (29643, counted=30341).  FIXED
Inode bitmap differences: -8280.  FIXED
Free inodes count wrong for group #4 (1405, counted=1406).  FIXED
Free inodes count wrong (34522, counted=34523).  FIXED

/dev/hda2: ***** FILE SYSTEM WAS MODIFIED *****
/dev/hda2: ***** REBOOT LINUX *****
/dev/hda2: 13285/47808 files, 160875/191216 blocks
```

Прежде всего, обратите внимание, что система запрашивает подтверждение перед тем как начать проверку смонтированной файловой системы. Если во время работы *fsck* найдены и исправлены ошибки, систему придется перезагрузить, если файловая система смонтирована. Это вызвано тем, что произведенные *fsck* изменения не затрагивают внутреннее представление системы о структуре файловой системы. Вообще говоря, проверять смонтированные файловые системы не рекомендуется.

Как можно видеть, было обнаружено и исправлено несколько проблем, и поскольку файловая система была смонтирована, система предупредила, что машину следует перезагрузить.

Как можно проверять файловую систему, не монтируя ее? За исключением корневой файловой системы, можно просто размонтировать любую файловую систему с помощью *umount*, прежде чем выполнить на ней *fsck*. Однако корневую файловую систему нельзя размонтировать, пока система работает. Один из способов проверить корневую файловую систему, пока она размонтирована, это использовать комбинацию загрузочной/корневой дискеты, например инсталляционных дискет, используемых вашим дистрибутивом. При этом корневая файловая система содержится на дискете, а корневая файловая система (на жестком диске) остается размонтированной, и появляется возможность ее проверки. Более подробно об этом см. в разделе «Действия в аварийной ситуации» главы 8.

Другой способ проверить корневую файловую систему – монтировать ее в режиме только для чтения. Это можно сделать с помощью параметра `ro` в приглашении загрузки LILO (см. раздел «Задание параметров времени загрузки» главы 5). Однако некоторые части системной конфигурации (например, программы, выполняемые */etc/init* во время загрузки) могут потребовать доступа к файловой системе для записи, поэтому нельзя нормально загрузить систему, иначе эти программы не выполнятся. Чтобы выполнить на-

чальную загрузку с монтированием корневой файловой системы в режиме только для чтения, можно загрузить систему в однопользовательском режиме (с помощью параметра загрузки `single`). Тем самым пропускается дополнительное конфигурирование системы при загрузке. После этого можно проверить корневую файловую систему и перезагрузить систему обычным образом.

Для того чтобы корневая файловая система была смонтирована в режиме только для чтения, можно либо использовать параметр загрузки `ro`, либо с помощью *rdev* установить флаг только для чтения в самом образе ядра.

Во многих системах Linux производится автоматическая проверка файловых систем во время начальной загрузки. Обычно это осуществляется путем запуска *fsck* из */etc/rc.d/rc.sysinit*. В этом случае система обычно сначала монтирует файловую систему только для чтения, запускает *fsck* для ее проверки, а затем выполняет команду

```
mount -w -o remount /
```

Параметр *-o remount* вызывает перемонтирование указанной файловой системы с новыми параметрами. В данном случае параметр *-w* (эквивалентный *-o rw*) заставляет монтировать файловую систему для чтения/записи. В итоге корневая файловая система заново монтируется в режиме доступа для чтения и записи.

При выполнении *fsck* во время начальной загрузки программа проверяет все файловые системы, кроме корневой, прежде чем монтировать их. По завершении работы *fsck* монтируются все прочие файловые системы с помощью команды *mount*. Посмотрите в файлах из каталога */etc/rc.d*, особенно в *rc.sysinit* (если он есть в вашей системе), как это делается. Если вы хотите отключить эту функцию в своей системе, закомментируйте строки в соответствующем файле */etc/rc.d*, которые запускают *fsck*.

Есть несколько параметров, которые можно передать конкретной версии *fsck*. Для большинства типов поддерживаются параметры: *-a*, автоматически подтверждающий все запросы, которые может делать *fsck.type; -c*, осуществляющий проверку сбойных блоков, как *mkfs*; *-v*, выводящий текстовые сообщения во время проверки. Эти параметры должны задаваться после задания типа файловой системы, например

```
fsck -t type -v device
```

запускает *fsck* с текстовыми сообщениями.

Более подробные сведения см. на страницах руководства по *fsck* и *e2fsck*.

Не для всех типов файловых систем, поддерживаемых Linux, есть свои версии *fsck*. Для проверки и исправления файловых систем MS-DOS следует использовать средства для MS-DOS, такие как Norton Utilities. Можно найти версии *fsck*, по крайней мере, для второй расширенной файловой системы, файловых систем Minix и Xia.

В разделе «Действия в аварийной ситуации» главы 8 мы даем дополнительные сведения о проверке файловых систем и восстановлении после аварий.

fsck не обязательно обнаружит и исправит все ошибки в ваших файловых системах, но с большинством стандартных проблем она справится. Нет простого способа восстановить уничтоженный важный файл, и *fsck* это не сделает. Ведутся работы по созданию утилиты «undelete» для восстановления удаленных файлов во второй расширенной файловой системе. Храните резервные копии или используйте команду *rm -i*, которая всегда запрашивает подтверждение перед удалением файла.

Управление пространством свопинга

Swap space является общим термином, обозначающим дисковую память, используемую для увеличения имеющейся в системе оперативной памяти. В Linux пространство для свопинга используется для *страничной* подкачки памяти – процесса, при котором страницы памяти (в системах Intel страница представляет собой 4096 байт, в других архитектурах значение может быть иным) записываются на диск при недостатке оперативной памяти и считываются в нее обратно при необходимости. Процесс, с помощью которого осуществляется страничная организация памяти, довольно сложен, но для некоторых ситуаций он оптимизирован. Подсистема виртуальной памяти в Linux разрешает совместный доступ к памяти различным работающим программам. Например, если у вас одновременно запущено несколько экземпляров Emacs, то фактически в памяти находится один экземпляр кода Emacs. Кроме того, текстовые страницы (содержащие код программы, а не данные) обычно имеют доступ только для чтения и потому при свопинге не записываются на диск. Такие страницы освобождаются из памяти и при повторном доступе считываются из исходного выполняемого файла.

Разумеется, пространство для свопинга не может полностью компенсировать недостаток оперативной памяти. Обращение к диску происходит значительно медленнее, чем к оперативной памяти, на несколько порядков. Поэтому свопинг используется преимущественно для того, чтобы одновременно запустить несколько программ, которые иначе не поместились бы в оперативной памяти. Если вы часто переключаетесь между этими программами, то заметите задержку, вызванную обменом страниц с диском.

В любом случае Linux поддерживает два вида пространства для свопинга: как отдельного дискового раздела для свопинга или как файла подкачки в файловых системах Linux. Можно иметь до 8 областей свопинга, при этом каждая область является дисковым файлом или разделом размером до 2 Гбайт (в системах, не являющихся Intel, эти значения могут быть иными). Тот, кто силен в математике, вычислит, что это дает до 16 Гбайт пространства для свопинга. (Если кто-либо практически использовал такой объем свопинга, авторам было бы приятно узнать об этом, вне зависимости от того, сильны вы в математике или нет.)

Учтите, что при использовании для свопинга дискового раздела, производительность может оказаться выше, поскольку блоки на диске располагаются непрерывно. Если же для свопинга используется файл, блоки могут быть

разбросаны по файловой системе, что в некоторых случаях может сильно снизить производительность. Часто своп-файл используется при необходимости временно расширить пространство для свопинга, когда система мечется от недостатка физической памяти и свопа. Своп-файлы – хороший способ расширять своп-пространство при необходимости.

Почти во всех Linux-системах используется того или иного вида пространство для свопинга, обычно в виде отдельного раздела. В главе 3 объяснялось, как создать своп-раздел во время установки Linux. В данном разделе мы объясним, как добавлять или удалять своп-файлы и разделы. Если у вас есть пространство для свопинга, которое вас удовлетворяет, этот раздел может не представлять для вас интереса.

Сколько у вас места для свопинга? Команда *free* сообщает сведения об использовании системной памяти:

```
rutabaga% free
            total                 used       free     shared    buffers     cached
Mem:        127888              126744       1144      27640       1884      51988
-/+         buffers/cache:       72872      55016
Swap:       130748               23916     106832
```

Все числа здесь представляют размер в блоках по 1024 байт. Здесь мы наблюдаем систему с 127 888 блоками (около 127 Мбайт) оперативной памяти, из которых в настоящее время используется 126 744 (около 126 Мбайт). Заметьте, что в вашей системе больше оперативной памяти, чем указано в колонке «total», поскольку в это количество не включена память, используемая самим ядром для различных нужд.

В колонке «shared» указан объем оперативной памяти, совместно используемой процессами. Здесь, как мы видим, затребовано около 27 Мбайт, что указывает на интенсивное использование памяти. В колонке «buffers» показан объем памяти, используемой в буферных кэшах ядра. Буферный кэш (о котором говорилось в предыдущем разделе) применяется для ускорения дисковых операций, позволяя обслуживать чтение и запись на диск с помощью оперативной памяти. Буферный кэш меняет свой размер в зависимости от использования памяти в системе. Если память нужна приложениям, она отбирается у кэша. Поэтому, хотя мы видим, что выделено 126 Мбайт системной памяти, не вся она (но большая часть) используется прикладными программами. В колонке «cache» указано, сколько страниц памяти помещено в кэш для последующего быстрого доступа к данным.

Поскольку память, используемую для буферов и кэша, можно передать приложениям, во второй строке (`-/+ buffers/cache`) указано количество памяти, которое реально используется приложениями (колонка «used») или доступно для них (колонка «free»). Для получения двух чисел во второй строке, суммарный объем, занятый буферами и кэшем согласно 1 строке, вычитается из общей использованной памяти и складывается с объемом свободной памяти.

В третьей строке указан общий объем свопа, 130 748 блоков (около 128 Мбайт). В нашем случае использована незначительная часть свопа, по-

скольку оперативной памяти много. Если запустить дополнительные приложения, для них будет использовано больше буферной памяти. К пространству для свопинга система обычно прибегает в последнюю очередь, когда невозможно получить физическую память другими способами.

Обратите внимание, что *free* указывает несколько меньший объем свопа, чем сумма размеров ваших файлов и разделов для свопинга. Это связано с необходимостью в каждой области использовать несколько блоков для хранения карты, показывающей, как будет использоваться каждая страница. Эти накладные расходы достаточно невелики и составляют лишь несколько килобайт в каждой области свопинга.

Если вы собираетесь создавать файл для свопинга, команда *df* сообщит вам сведения о свободном пространстве, которое есть в ваших файловых системах. Она выводит список файловых систем с указанием объема и текущего процента использования.

Создание пространства для свопинга

Чтобы добавить новое пространство для свопинга, нужно сначала создать файл или раздел, в котором будет размещаться своп. Если вы хотите создать раздел, это можно сделать утилитой *fdisk*, как описано в разделе «Создание разделов в Linux» в главе 3.

Чтобы создать файл свопинга, нужно открыть файл и записать в него столько байт, сколько должно быть в свопе, который вы хотите создать. Это можно легко сделать с помощью команды *dd*. Например, чтобы создать своп-файл размером 32 Мбайт, можно выполнить команду

```
dd if=/dev/zero of=/swap bs=1024 count=32768
```

В результате 32 768 блоков (32 Мбайт) данных с устройства */dev/zero* будут записаны в файл */swap*. (*/dev/zero* является особым устройством, для которого операция чтения всегда возвращает нулевые байты. Оно некоторым образом противоположно устройству */dev/null*.) После создания этого файла рекомендуется выполнить команду *sync*, чтобы синхронизировать файловые системы на случай краха машины.

Создав своп-файл или раздел, можно с помощью команды *mkswap* «отформатировать» область для свопинга. Как описано в разделе «Создание пространства для свопинга» главы 3, формат команды *mkswap* такой:

```
mkswap -c device size
```

где *device* является именем своп-раздела или файла, а *size* – размер своп-области в блоках (блок по-прежнему составляет 1 Кбайт). Обычно размер не нужно указывать, поскольку *mkswap* может сама определить размер раздела. Ключ *-c* не является обязательным, он указывает на необходимость выявления сбойных блоков в области во время форматирования.

Например, для своп-файла, созданного в предыдущем примере, нужно выполнить команду

```
mkswap -c /swap 32768
```

Если область для свопинга является разделом, нужно указать его имя (например, */dev/hda3*) и размер, также в блоках.

При использовании своп-файла (но не раздела) необходимо изменить права доступа к нему, например:

```
chmod 0600 /swap
```

Обработав своп-файл командой *mkswap*, выполните команду *sync*, чтобы гарантировать физическую запись данных форматирования в новый своп-файл. При форматировании своп-раздела выполнять *sync* не обязательно.

Активация пространства для свопинга

Чтобы начать использовать новое пространство для свопинга, нужно активировать его командой *swapon*. Например, создав предыдущий своп-файл с помощью *mkswap* и *sync*, можно выполнить команду

```
swapon /swap
```

Тем самым новая область свопинга добавляется к общему объему пространства для свопинга. С помощью команды *free* убедитесь, что это действительно произошло. Если вы используете для свопинга новый раздел с именем */dev/hda3*, активируйте его командой типа

```
swapon /dev/hda3
```

Подобно файловым системам, своп-области автоматически активируются при загрузке системы командой *swapon -a* в одном из стартовых файлов системы (обычно */etc/rc.d/rc.sysinit*). Эта команда просматривает файл */etc/fstab*, в котором, как вы помните из раздела «Монтирование файловых систем», содержатся данные о файловых системах и областях для свопинга. Команда *swapon -a* активирует все записи из */etc/fstab*, в поле `options` которых стоит `sw`.

Поэтому если */etc/fstab* имеет следующий вид:

```
# device     directory    type   options
/dev/hda3    none         swap   sw
/swap        none         swap   sw
```

то во время загрузки активируются две области для свопинга, */dev/hda3* и */swap*. Для каждой новой области свопинга нужно создать запись в */etc/fstab*.

Отключение пространства для свопинга

Как часто бывает, отменить действие легче, чем осуществить. Для отключения пространства для свопинга выполните команду

```
swapoff device
```

где `device` является именем своп-раздела или файла, который вы хотите отключить. Например, отключение свопинга на устройстве */dev/hda3* производится командой

```
swapoff /dev/hda3
```

Для отключения своп-файла можно просто удалить его командой *rm после* выполнения *swapoff*. Не удаляйте своп-файл до его отключения, это может вызвать катастрофические последствия.

Если вы отключили своп-раздел с помощью *swapoff*, то можете дальше использовать этот раздел по своему усмотрению или удалить с помощью *fdisk*.

Если своп-области соответствует запись в */etc/fstab*, удалите ее, иначе при очередной перезагрузке системы вы получите сообщения об ошибках из-за того, что области свопинга не будут обнаружены.

Файлы устройств

Файлы устройств позволяют программам пользователей обращаться к аппаратным устройствам через ядро системы. Они не являются «файлами» как таковыми, но с точки зрения программы выглядят как файлы: их можно читать, осуществлять в них запись, отображать в память при помощи системного вызова *mmap*() и т. д. При обращении к такому «файлу» устройства ядро распознает запрос ввода/вывода и передает его драйверу устройства, который осуществляет некоторую операцию, например чтение данных из последовательного порта или отправку данных звуковой карте.

Файлы устройств (несмотря на неудачное название, мы будем продолжать использовать этот термин) предоставляют удобный способ доступа к ресурсам системы без необходимости для прикладного программиста знать, как работает само устройство. В Linux, как и в большинстве Unix-систем, драйверы устройств сами являются частью ядра. В разделе «Сборка ядра» главы 7 мы покажем, как собрать собственное ядро, включив в него драйверы только тех устройств, которые есть в вашей системе.

Файлы устройств располагаются в каталоге */dev* почти во всех Unix-подобных системах. Для каждого устройства, имеющегося в системе, должен быть соответствующий элемент в */dev*. Например, */dev/ttyS0* соответствует первому последовательному порту, известному как COM1 в MS-DOS; */dev/hda2* соответствует второму разделу первого диска IDE. Фактически в */dev* будут записи для отсутствующих устройств. Файлы устройств создаются обычно во время установки системы и включают в себя все возможные драйверы устройств. Они не обязательно соответствуют фактически имеющимся в машине аппаратным устройствам.

В */dev* есть ряд псевдоустройств, не соответствующих какой-либо реальной периферии. Например, */dev/null* действует как сливная труба для байтов. Все запросы записи в */dev/null* проходят успешно, но записываемые данные игнорируются. Аналогично мы уже демонстрировали использование */dev/zero* для создания своп-файла: всякое чтение из */dev/zero* просто возвращает нулевые байты.

Если с помощью *ls -l* вывести файлы устройств в */dev*, то получится примерно следующее:

```
brw-rw----   1 root     disk       3,   0 May 19  1994 /dev/hda
```

Это устройство */dev/hda*, соответствующее первому диску IDE. Прежде всего, обратите внимание на первую букву `b` в поле прав доступа, означающую, что это файл блочного устройства. (Вспомним, что у обычных файлов в первой колонке находится символ «-», у каталогов – «`d`», и т. д.) Файлы устройств обозначаются либо символом `b` для блочных устройств, либо символом `c` для символьных устройств. Блочное устройство обычно является периферийным устройством, таким как жесткий диск: обмен данными с устройством производится целыми блоками (при этом размер блока определяется устройством; он может не равняться 1024 байтам, которым обычно равен «блок» в Linux), а доступ к устройству может осуществляться произвольно. Напротив, чтение и запись на символьные устройства обычно производятся последовательно, а ввод/вывод может осуществляться по одному байту. Примером символьного устройства является последовательный порт.

Обратите внимание также на то, что поле `size` в выводе *ls -l* заменено двумя числами, разделенными запятой. Первое число – *основной номер устройства* (*major device number*), второе число – *дополнительный номер устройства* (*minor device number*). При обращении к файлу устройства из программы ядро получает запрос ввода/вывода в терминах основного и дополнительного номеров устройства. Основной номер обычно указывает на конкретный драйвер ядра, а дополнительный – на конкретное устройство, с которым работает этот драйвер. Например, у всех последовательных портов один и тот же основной номер, но разные дополнительные номера. Ядро использует основной номер для перенаправления запроса ввода/вывода соответствующему драйверу, а драйвер использует дополнительный номер для определения того, к какому конкретно устройству нужно обратиться. В некоторых случаях дополнительные номера используются для доступа к особым функциям устройства.

Правила именования файлов в */dev* представляют собой, грубо говоря, полную неразбериху. Поскольку самому ядру безразлично, какие имена файлов используются в */dev* (ему нужны только основные и дополнительные номера), создатели дистрибутивов, прикладные программисты и разработчики драйверов устройств могут произвольно выбирать имена файлов устройств. Часто один разработчик драйвера предлагает для устройства некоторое имя, которое затем изменяется, чтобы позволить разместить другие аналогичные устройства. Это может вызывать путаницу и непоследовательность по мере развития системы. Будем надеяться, что вы не столкнетесь с этой проблемой, если только не станете работать с более новыми драйверами устройств, которые находятся в стадии тестирования.

В любом случае файлы устройств, включенные в ваш дистрибутив, должны быть корректными для включенных в этот дистрибутив версии ядра и драйверов устройств. При обновлении ядра или добавлении драйверов устройств (см. раздел «Сборка нового ядра» главы 7) вам может понадобиться добавить файл устройства с помощью команды *mknod*. Формат этой команды следующий:

```
mknod -m permissions name type major minor
```

где:

- *name* является полным именем пути для создаваемого устройства, например */dev/rft0*.
- *type* может принимать значение c для символьного устройства или b для блочного устройства.
- *major* является основным номером устройства.
- *minor* является дополнительным номером устройства.
- *-m permissions* является необязательным аргументом, устанавливающим биты прав доступа для нового устройства.

Допустим, например, что вы добавляете к ядру новый драйвер устройства, и в документации сказано, что нужно создать блочное устройство */dev/bogus* с основным номером 42 и дополнительным номером 0. Вам следует выполнить команду

```
mknod /dev/bogus b 42 0
```

Создание устройств еще более упрощается при использовании сценария оболочки */dev/MAKEDEV*, входящего во многие дистрибутивы: вам нужно задать только тип нужного устройства, а *MAKEDEV* определит для вас основной и дополнительный номера.

Если вы не указали аргумент *-m permissions*, то новое устройство получает права доступа, как для вновь созданного файла, модифицированные вашей текущей маской – обычно 0644. Для того чтобы изменить права доступа к устройству */dev/bogus* на 0660, выполните команду

```
mknod -m 660 /dev/bogus b 42 0
```

После того как файл устройства создан, изменить права доступа к нему можно командой *chmod*.

В чем важность прав доступа к устройствам? Как и для любого файла, права доступа к файлу устройства определяют, кто может обращаться к «сырому» устройству (т. е. обращаться напрямую к усторойству, минуя операционную систему) и как. Как видно из предыдущего примера, файл устройства для */dev/hda* имеет права доступа 0660. Это означает, что только владелец и группа файла (группой здесь является disk) могут непосредственно осуществлять чтение и запись на это устройство. (С правами доступа мы познакомили вас в разделе «Владение файлами и права доступа» главы 4.)

Как правило, нежелательно предоставлять любому пользователю право непосредственного чтения или записи на некоторые устройства, особенно устройства, относящиеся к дискам и разделам на них. В противном случае любой мог бы выполнить, скажем, *mkfs* на разделе диска и полностью уничтожить все данные системы.

Что касается дисков и разделов, то для того чтобы уничтожить таким способом данные, требуется право записи, но право чтения также является нарушением системы безопасности. Имея право на чтение к сырому файлу устройства, соответствующему дисковому разделу, пользователь может про-

сматривать чужие файлы. Аналогично файл устройства */dev/mem* соответствует оперативной памяти системы (он обычно используется лишь в исключительных случаях при отладке). Получив право чтения, особо умный пользователь может проследить за паролями регистрации других пользователей, в том числе суперпользователя, когда они вводятся при входе в систему.

Следите за тем, чтобы права доступа к любому устройству, добавляемому вами в систему, соответствовали тому, как могут и должны обращаться к нему пользователи. К таким устройствам, как последовательные порты, звуковые карты и виртуальные консоли, простые пользователи обычно могут обращаться без опасности причинить вред, однако к большинству других устройств доступ должен быть предоставлен только суперпользователю (и программам, выполняемым с его правами).

Многие файлы в */dev* являются в действительности символическими ссылками на другие файлы устройств, созданными обычным образом с помощью команды *ln -s*. Эти ссылки упрощают доступ к некоторым устройствам благодаря использованию более привычных имен. Например, если у вас есть мышь для последовательного порта, доступ к ней можно получить через один из файлов устройств */dev/ttyS0*, */dev/ttyS1*, */dev/ttyS2* или */dev/ttyS3*, в зависимости от того, к какому порту подключена мышь. Часто создают ссылку с именем */dev/mouse* на соответствующее устройство последовательного порта, например:

```
ln -s /dev/ttyS2 /dev/mouse
```

Благодаря этому можно обращаться к */dev/mouse*, а не пытаться вспомнить, на каком порту она находится. Это соглашение используется и для таких устройств, как *dev/cdrom* и */dev/modem*. Обычно эти файлы являются символическими ссылками на файлы устройств в */dev*, соответствующие фактическим устройствам CD-ROM или модема.

Чтобы удалить файл устройства, выполните команду *rm*, например:

```
rm /dev/bogus
```

Удаление файла устройства не означает, что соответствующий устройству драйвер удаляется из памяти или ядра. Просто вы теряете возможность общения с драйвером конкретного устройства. Аналогично при добавлении файла устройства в системе не появляется новый драйвер устройства. Можно добавлять файлы устройств даже для несуществующих драйверов. Файлы устройств предоставляют просто «привязку» к драйверу некоторого устройства, на случай если этот драйвер окажется в ядре.

7

Обновление программного обеспечения и ядра

В этой главе мы покажем, как обновлять программное обеспечение, в том числе компилировать и устанавливать новое ядро операционной системы. Хотя в большинстве дистрибутивов Linux есть автоматизированные средства установки, удаления и обновления отдельных программных пакетов системы, часто приходится устанавливать программное обеспечение вручную.

Не очень опытным пользователям проще всего устанавливать и обновлять программное обеспечение с помощью системы *пакетов*, предоставляемой большинством дистрибутивов. Если не пользоваться системой пакетов, то установка и обновление оказываются сложнее, чем в большинстве коммерческих систем. Даже если у вас есть скомпилированные двоичные файлы, возможно, потребуется декомпрессировать и распаковать их из архивного файла. Может также потребоваться создать символические ссылки или присвоить значения переменным окружения, чтобы двоичные файлы могли определить, где находятся необходимые им ресурсы. В других случаях требуется самостоятельная компиляция программного обеспечения из исходного кода.

Другой часто выполняемой в Linux работой является сборка ядра. Это важная задача по нескольким причинам. Во-первых, вам может понадобиться обновить ядро до новой версии, чтобы использовать новые функции или поддержку дополнительных аппаратных устройств. Во-вторых, самостоятельная сборка ядра позволяет выбрать только те функции, которые вы желаете иметь в скомпилированном ядре.

Что дает возможность выбора функций? Весь программный код ядра и его данные «заблокированы» в памяти, т. е. они не выгружаются на диск при свопинге. Например, если образ вашего ядра содержит драйверы для устройств, которых у вас нет или которыми вы не пользуетесь, занятая этими драйверами память не может употребляться пользовательскими приложениями. Настройка ядра позволяет сократить его до необходимых размеров.

Нужно отметить, что большинство современных дистрибутивов поставляет модульное ядро. Это означает, что устанавливаемое ими ядро содержит лишь

минимум функций, необходимый для работы ядра. Все сверх того содержится в *модулях*, которые предоставляют дополнительные функции по требованию. Ниже мы подробно обсудим модули.

Утилиты архивирования и сжатия

При установке или обновлении программного обеспечения на Unix-системах, прежде всего, необходимо уметь работать со средствами сжатия и архивирования файлов. Имеются десятки утилит такого рода. Некоторые из них (такие как *tar* и *compress*) относятся по времени создания к самым ранним дням Unix. Другие, такие как *gzip* и *bzip2*, являются относительно новыми. Главной задачей этих утилит является архивирование файлов (т. е. упаковка группы файлов в один для удобства перемещения и создания резервной копии) и сжатие файлов для уменьшения дискового пространства, занимаемого файлом или группой файлов.

В данном разделе мы обсудим наиболее часто встречающиеся форматы файлов и утилиты, с которыми вы, вероятнее всего, столкнетесь. Например, в мире Unix общепринято перемещать файлы или программное обеспечение в виде *tar*-архивов, сжатых с помощью *compress* или *gzip*. Для того чтобы иметь возможность самостоятельно создавать или распаковывать такие файлы, нужно уметь пользоваться соответствующими инструментами. Эти инструменты чаще всего применяются при установке нового программного обеспечения или создании резервных копий и рассматриваются в двух последующих разделах данной главы.

Использование gzip и bzip2

gzip является быстрой и эффективной программой, распространяемой в рамках проекта GNU. Основная функция *gzip* – сжать файл, сохранить сжатую версию в виде *filename.gz* и удалить исходный несжатый файл. Исходный файл удаляется только в случае успешного выполнения *gzip*. Случайно удалить при этом файл очень трудно. Конечно, *gzip*, являясь программным обеспечением GNU, имеет больше параметров, чем вы могли бы ожидать, и характер работы этой программы можно во многом модифицировать с помощью параметров командной строки.

Предположим, что у нас есть большой файл с именем *garbage.txt*:

```
rutabaga% ls -l garbage.txt
-rw-r--r--   1 mdw      hack       312996 Nov 17 21:44 garbage.txt
```

Для сжатия этого файла с помощью *gzip* мы просто выполним команду

```
gzip garbage.txt
```

В результате *garbage.txt* будет заменен сжатым файлом *garbage.txt.gz*. Конечный результат следующий:

```
rutabaga% gzip garbage.txt
rutabaga% ls -l garbage.txt.gz
-rw-r--r--   1 mdw      hack       103441 Nov 17 21:44 garbage.txt.gz
```

Обратите внимание на то, что по завершении работы *gzip* файл *garbage.txt* удаляется.

Можно передать *gzip* список имен файлов. Каждый файл списка будет сжат и получит расширение *.gz*. (В отличие от программы *zip* для Unix и MS-DOS, *gzip* по умолчанию не сжимает несколько файлов в один архив *.gz*. Для этой цели существует *tar*; см. следующий раздел.)

Эффективность сжатия файла зависит от его формата и содержания. Например, многие файлы графических форматов, такие как PNG и JPEG, уже в значительной мере сжаты, и применение *gzip* к таким файлам может не иметь эффекта. Обычно хорошо сжимаются текстовые файлы и такие двоичные файлы, как исполняемые модули и библиотеки. Сведения о сжатом файле можно получить при помощи команды *gzip -l*, например:

```
rutabaga% gzip -l garbage.txt.gz
compressed  uncompr. ratio uncompressed_name
   103115    312996  67.0% garbage.txt
```

Для того чтобы получить обратно оригинал из сжатой версии, мы используем *gunzip*, например:

```
gunzip garbage.txt.gz
```

После выполнения этой команды получаем:

```
rutabaga% gunzip garbage.txt.gz
rutabaga% ls -l garbage.txt
-rw-r--r--   1 mdw      hack       312996 Nov 17 21:44 garbage.txt
```

Полученный файл идентичен оригиналу. Обратите внимание, что после декомпрессии файла сжатая версия удаляется. Вместо *gunzip* можно воспользоваться *gzip -d* (например, если *gunzip* не установлена).

gzip записывает имя исходного файла в его сжатую версию. Поэтому если имя сжатого файла (вместе с расширением *.gz*) оказывается слишком длинным для выбранного типа файловой системы (например, если вы производите сжатие в файловой системе MS-DOS с именами формата 8.3), имя исходного файла может быть восстановлено *gunzip*, даже если при сжатии оно было усечено. Для декомпрессии файла с присвоением исходного имени используйте параметр *–N*. Чтобы убедиться в полезности этого параметра, рассмотрим последовательность команд:

```
rutabaga% gzip garbage.txt
rutabaga% mv garbage.txt.gz rubbish.txt.gz
```

Если теперь декомпрессировать *rubbish.txt.gz*, то в результате получим файл *rubbish.txt*, т. е. с тем же именем, что и сжатый файл. Однако, задав параметр *–N*, мы получим:

```
rutabaga% gunzip -N rubbish.txt.gz
rutabaga% ls -l garbage.txt
-rw-r--r--   1 mdw      hack       312996 Nov 17 21:44 garbage.txt
```

gzip и *gunzip* могут также осуществлять компрессию и декомпрессию данных со стандартного ввода и вывода. Если не сообщить *gzip* имен файлов, подлежащих компрессии, то программа пытается сжимать данные со стандартного ввода. Аналогично *gunzip* с параметром *-c* выводит декомпрессированные данные на стандартное устройство вывода. Можно, например, передать по каналу выходные данные некоторой команды на *gzip*, чтобы за один шаг сжать выходной поток и записать его в файл:

```
rutabaga% ls -laR $HOME | gzip > filelist.gz
```

Эта команда произведет вывод содержимого вашего исходного каталога с подкаталогами и запишет результат в сжатый файл *filelist.gz*. Содержимое файла можно вывести командой

```
rutabaga% gunzip -c filelist.gz | more
```

Эта команда декомпрессирует *filelist.gz* и передаст результат по каналу команде *more*. Если вы используете *gunzip -c*, на диске остается сжатый файл.

Команда *zcat* аналогична *gunzip -c*. Можете рассматривать ее как версию *cat* для сжатых файлов. В Linux есть даже версия команды *less* для сжатых файлов, которая называется *zless*.

При сжатии файлов можно использовать параметры *-1*, *-2*, . . ., *-9*, чтобы задать скорость и степень используемого сжатия. При этом *-1* (или *--fast*) задает самый быстрый метод, слабее сжимающий файлы, а *-9* (или *--best*) задает более медленный, но сильнее всего сжимающий файлы метод. По умолчанию значение метода сжатия равно *-6*. Эти параметры не влияют на использование команды *gunzip*, которая способна осуществить декомпрессию любого файла независимо от использованного метода сжатия.

gzip в мире Unix появилась относительно недавно. В большинстве Unix-систем при сжатии используются команды *compress* и *uncompress*, которые были в оригинальной Berkeley-версии Unix. *compress* и *uncompress* очень похожи на *gzip* и *gunzip* соответственно. *compress* сохраняет сжатые файлы как *filename.Z*, в отличие от *gzip*, сохраняющей сжатый файл как *filename.gz*, и использует несколько менее эффективный алгоритм сжатия.

Однако по ряду причин сообщество свободного программного обеспечения переходит на *gzip*. Во-первых, *gzip* работает лучше. Во-вторых, в результате спора по поводу авторских прав на алгоритм сжатия, используемый в *compress*, сторонним разработчикам может быть отказано в праве самостоятельной реализации алгоритма. Поэтому Фонд свободного программного обеспечения настаивал на переходе на *gzip*, что приветствовалось, по крайней мере, в сообществе Linux. *gzip* уже перенесена на некоторые архитектуры, за которыми последуют другие. Удачно, что *gunzip* может разархивировать файлы формата *.Z*, которые создает *compress*.

Появилась еще одна программа компрессии/декомпрессии, bzip2, перенявшая лидерство у *gzip*. *bzip2* – это новая программа, сжимающая в среднем на 10–20% сильнее, чем *gzip*, за счет большей продолжительности процесса сжатия. Нельзя использовать *bunzip2* для декомпрессии файлов, сжатых

gzip, и наоборот. Поскольку нельзя рассчитывать на то, что у всех установлена *bunzip2*, вам, возможно, следует ограничиться *gzip*, если вы собираетесь кому-либо посылать сжатые файлы. Однако стоит установить у себя *bzip2*, т. к. все большее число FTP-серверов начинает хранить пакеты, сжатые *bzip2*, с целью экономии дискового пространства и полосы пропускания. Узнать файлы, сжатые *bzip2*, можно по обычному для них расширению *.bz2*.

Вообще говоря, параметры командной строки для *bzip2* и *gzip* отличаются друг от друга, но те, которые описаны в данном разделе, одинаковы. Более подробные сведения можно получить на странице руководства для *bzip2*(1).

Подводя итоги, скажем, что для сжатия следует использовать *gzip*/*gunzip* или *bzip2*/*bunzip2*. Если вам встретится файл с расширением *.Z*, то он, скорее всего, был создан с помощью *compress*, и *gunzip* может выполнить его декомпрессию.

В старых версиях *gzip* для сжатых файлов использовалось расширение *.z* (нижний регистр), а не *.gz*. Из-за опасности путаницы с *.Z* оно было изменено. В любом случае *gunzip* поддерживает обратную совместимость с рядом расширений и типов файлов.

Использование tar

tar является универсальной утилитой архивирования, способной упаковать несколько файлов в один архивный, сохраняя при этом данные, необходимые для полного восстановления, такие как права доступа и владения. Название *tar* происходит от *tape archive*, поскольку первоначально эта утилита предназначалась для архивирования файлов в виде резервных копий на магнитных лентах. Однако, как мы увидим, использование *tar* вовсе не ограничивается изготовлением резервных копий на магнитной ленте.

Формат команды *tar* следующий:

```
tar functionoptions files...
```

Здесь `function` является буквой, указывающей выполняемую операцию, `options` является списком однобуквенных параметров этой функции, а `files` – списком упаковываемых или распаковываемых файлов в архиве. (Обратите внимание, что `function` не отделяется пробелом от параметров.)

Параметр `function` может принимать следующие значения:

c – создать новый архив;

x – извлечь файлы из архива;

t – перечислить содержание архива;

r – дописать файлы в конец архива;

u – заменить файлы в архиве более новыми;

d – сравнить архивированные файлы с файлами файловой системы.

Чаще всего используются функции *c*, *x* и *t*.

Обычно используются следующие значения *options*:

k – сохранить при разархивации существующие файлы, т. е. не заменять имеющиеся файлы извлекаемыми из архива;

f filename – задать имя архивного файла, который нужно прочесть или записать;

z – указать, что записываемые в архив файлы или имеющиеся в нем сжимаются с помощью *gzip*;

v – заставить *tar* показать имена файлов, помещаемых в архив или извлекаемых из него. Полезно посмотреть с помощью этой опции, что в действительности происходит (если только вы не пишете сценарии оболочки, конечно).

Есть и другие опции, о которых мы расскажем ниже.

Синтаксис *tar* может вначале показаться сложным, но на практике он очень прост. Допустим, есть каталог *mt*, содержащий следующие файлы:

```
rutabaga% ls -l mt
total 37
-rw-r--r--   1 root     root           24 Sep 21  1993 Makefile
-rw-r--r--   1 root     root          847 Sep 21  1993 README
-rwxr-xr-x   1 root     root         9220 Nov 16 19:03 mt
-rw-r--r--   1 root     root         2775 Aug  7  1993 mt.1
-rw-r--r--   1 root     root         6421 Aug  7  1993 mt.c
-rw-r--r--   1 root     root         3948 Nov 16 19:02 mt.o
-rw-r--r--   1 root     root        11204 Sep  5  1993 st_info.txt
```

Мы хотим упаковать содержимое этого каталога в один архивный *tar*-файл, для чего воспользуемся командой

```
tar cf mt.tar mt
```

В данном случае первым аргументом *tar* является функция *c* (от *create*), за которой следуют параметры. Здесь мы используем один параметр *f mt.tar*, указывающий, что результирующий архивный файл должен иметь имя *mt.tar*. Последним аргументом является имя (или имена) архивируемого файла. В данном случае мы задаем имя каталога, чтобы *tar* упаковал в архив все файлы из этого каталога.

Обратите внимание, что первым аргументом должна быть буква функции, за которой следует список ее параметров. Поэтому нет смысла ставить дефис (-) перед параметрами, как того требуют многие команды Unix. *tar* разрешает использовать дефис следующим образом:

```
tar -cf mt.tar mt
```

но в этом нет необходимости. В некоторых версиях *tar* первой буквой должна быть функция, то есть *c*, *t* или *x*. В других версиях порядок букв не является существенным.

Описанные здесь буквы функций следуют так называемому «старому стилю» параметров. Есть также более новый «стиль коротких параметров», в котором параметрам функций предшествует дефис, и «стиль длинных параметров», где используются длинные имена параметров с двумя дефисами перед нами. См. страницу Info для *tar*, чтобы найти интересующие вас подробности.

Не забудьте указать имя файла, если используются буквы *cf*, иначе будет переписан первый файл в списке упаковываемых файлов, поскольку он будет принят за имя архива!

Часто бывает полезно использовать *tar* с параметром *v*, что позволяет вывести перечень архивируемых файлов. Например:

```
rutabaga% tar cvf mt.tar mt
mt/
mt/st_info.txt
mt/README
mt/mt.1
mt/Makefile
mt/mt.c
mt/mt.o
mt/mt
```

Если задать *v* несколько раз, выводятся дополнительные сведения, например:

```
rutabaga% tar cvvf mt.tar mt
drwxr-xr-x root/root         0 Nov 16 19:03 1994 mt/
-rw-r--r-- root/root     11204 Sep  5 13:10 1993 mt/st_info.txt
-rw-r--r-- root/root       847 Sep 21 16:37 1993 mt/README
-rw-r--r-- root/root      2775 Aug  7 09:50 1993 mt/mt.1
-rw-r--r-- root/root        24 Sep 21 16:03 1993 mt/Makefile
-rw-r--r-- root/root      6421 Aug  7 09:50 1993 mt/mt.c
-rw-r--r-- root/root      3948 Nov 16 19:02 1994 mt/mt.o
-rwxr-xr-x root/root      9220 Nov 16 19:03 1994 mt/mt
```

Это очень полезно, т. к. позволяет убедиться в том, что *tar* делает то, что требуется.

В некоторых версиях *tar* буква *f* должна быть последней в списке параметров, поскольку за ней ожидается имя файла для чтения или записи. Если вообще не задать *f filename*, *tar* предполагает (в силу исторических причин), что нужно использовать устройство */dev/rmt0* (т. е. первый ленточный привод). В разделе «Создание резервных копий» главы 8 мы поговорим об использовании *tar* с ленточными устройствами для создания резервных копий.

Теперь мы можем передать файл *mt.tar* другим пользователям для распаковки его на своих машинах. Сделать это они могут командой

```
tar xvf mt.tar
```

В результате выполнения этой команды создается подкаталог *mt*, в который помещаются все исходные файлы с теми же правами доступа, которые были в исходной системе. Владельцем новых файлов будет пользователь, выпол-

нивший команду *tar xvf* (то есть вы), если только разархивацию не произвел root. В последнем случае сохраняется первоначальный владелец файла. Параметр *x* означает извлечение файлов, параметр *v* используется для перечисления всех извлекаемых файлов. Вывод будет таким:

```
courgette% tar xvf mt.tar
mt/
mt/st_info.txt
mt/README
mt/mt.1
mt/Makefile
mt/mt.c
mt/mt.o
mt/mt
```

Как можно видеть, *tar* сохраняет имя пути каждого файла относительно того расположения, где первоначально был создан архивный файл. Когда мы создавали архив командой *tar cf mt.tar mt*, единственным заданным именем входного файла было *mt* – имя содержащего файлы каталога. Поэтому *tar* записывает в архивный файл сам каталог и все находящиеся в нем файлы. Когда мы извлекаем файлы из архива, создается каталог *mt*, в который помещаются все файлы, что является процедурой, прямо противоположной созданию архива.

По умолчанию *tar* извлекает из архива все файлы относительно текущего каталога, в котором вы запускаете *tar*. Например, если бы вы собрались упаковать содержимое каталога */bin* командой

```
tar cvf bin.tar /bin
```

то получили бы предупреждение:

```
tar: Removing leading / from absolute path names in the archive.
```
(удаление ведущих / из абсолютных путей в архиве)

Это означает, что файлы будут сохранены в архиве в подкаталоге *bin*. При распаковке каталог *bin* создается в рабочем каталоге *tar*, а не как */bin* в системе, в которой производится распаковка. Это очень важно и имеет целью предотвращение катастрофических ошибок при распаковке. В противном случае распаковка файла, содержащего упакованный */bin*, привела бы к уничтожению вашего текущего каталога */bin*.[1] Если вам действительно нужно распаковать такой архив в */bin*, то это нужно делать, находясь в корневом каталоге /. Можно переопределить такое поведение при создании *tar*-архива с помощью параметра *P*, но делать так не рекомендуется.

Другим способом создания архива *mt.tar* был бы переход командой *cd* непосредственно в каталог *mt* и использование такой команды, как

```
tar cvf mt.tar *
```

[1] Некоторые старые реализации Unix (например, Sinix и Solaris) именно так и поступали.

При таком способе подкаталог *mt* не записывается в архивный файл, и при распаковке файлы помещаются в текущий рабочий каталог. Одна из тонкостей этикета работы с *tar* состоит в том, что всегда нужно упаковывать файлы при помощи *tar* таким образом, чтобы они содержали подкаталог, как мы делали это в первом примере с *tar cvf mt.tar mt*. Благодаря этому при распаковке архива создается также и каталог, в который помещаются файлы. Это позволяет избежать путаницы, которая могла бы возникнуть при помещении распакованных файлов непосредственно в рабочий каталог. Кроме того, пользователь освобождается от необходимости создания отдельного каталога для распаковываемых файлов. Конечно, есть масса ситуаций, когда вам это не потребуется. Но этикет таков.

При создании архивов можно указать утилите *tar* список подлежащих упаковке файлов и каталогов. В первом примере мы передали *tar* единственный каталог *mt*, но затем мы воспользовались символом *, который оболочка превращает в список всех имен в текущем каталоге.

Прежде чем распаковывать *tar*-архив, неплохо посмотреть на таблицу его содержимого и определить, как он был упакован. В результате вы можете решить, нужно ли создавать подкаталог для распаковки архива. Команда

```
tar tvf tarfile
```

выводит таблицу содержимого архива с именем *tarfile*. Обратите внимание, что при использовании функции *t* требуется только один параметр *v*, чтобы получить «длинную» распечатку перечня файлов, как показано в следующем примере:

```
courgette% tar tvf mt.tar
drwxr-xr-x root/root         0 Nov 16 19:03 1994 mt/
-rw-r--r-- root/root     11204 Sep  5 13:10 1993 mt/st_info.txt
-rw-r--r-- root/root       847 Sep 21 16:37 1993 mt/README
-rw-r--r-- root/root      2775 Aug  7 09:50 1993 mt/mt.1
-rw-r--r-- root/root        24 Sep 21 16:03 1993 mt/Makefile
-rw-r--r-- root/root      6421 Aug  7 09:50 1993 mt/mt.c
-rw-r--r-- root/root      3948 Nov 16 19:02 1994 mt/mt.o
-rwxr-xr-x root/root      9220 Nov 16 19:03 1994 mt/mt
```

Распаковка при этом не производится, мы видим только таблицу содержимого архива. Имена файлов говорят о том, что файл архива был создан из всех файлов подкаталога *mt*, и при распаковке архива будет создан подкаталог *mt*, в который будут помещены все файлы.

Можно извлекать из архива и отдельные файлы. Для этого служит команда

```
tar xvf tarfile files
```

Здесь *files* является списком извлекаемых из архива файлов. Если не указывать имен, *tar* распаковывает весь архив.

При указании отдельных файлов, которые нужно извлечь, следует задавать полный путь, как он указан в архиве. Например, если потребовалось извлечь из предыдущего архива только файл *mt.c*, это можно сделать командой

```
tar xvf mt.tar mt/mt.c
```

В результате создается подкаталог *mt*, в который помещается файл *mt.c*.

Утилита *tar* имеет много параметров помимо перечисленных здесь. Мы говорили лишь о функциях, которыми вы будете пользоваться чаще всего, однако GNU-версия *tar* (особенно она) имеет функции, делающие ее идеальной для создания резервных копий и тому подобных задач. Более подробные сведения можно получить из страниц руководства по *tar* и следующего раздела.

Использование tar совместно с gzip и bzip2

tar не осуществляет компрессии данных, сохраняемых в его архивах. Если вы создаете *tar*-архив из трех файлов по 200 Кбайт, то получите архив размером около 600 Кбайт. Очень часто *tar*-архивы сжимаются с помощью *gzip* (или более старых программ *компрессии*). Сжатый *tar*-архив можно создать, выполнив команды:

```
tar cvf tarfile files...
gzip -9 tarfile
```

Однако это весьма утомительно и требует наличия достаточного объема памяти для хранения несжатого архивного файла, прежде чем подвергнуть его обработке *gzip*.

Гораздо более ловкий способ выполнить ту же задачу – использовать функцию *tar*, позволяющую вывести архив на стандартное устройство вывода. Если задать архивный файл для чтения или записи как «-», то данные будут считываться со стандартного устройства ввода и выводиться на стандартное устройство вывода. Например, можно создать сжатый архивный файл с помощью команды

```
tar cvf - files... | gzip -9 > tarfile.tar.gz
```

В этом примере *tar* создает архив из файлов с перечисленными именами и выводит его на стандартное устройство вывода; затем *gzip* читает данные из стандартного устройства ввода, сжимает их и выводит на свое стандартное устройство вывода; наконец, мы перенаправляем сжатый архивный файл в *tarfile.tar.gz*.

Распаковать такой архивный файл можно с помощью команды

```
gunzip -9c tarfile.tar.gz | tar xvf -
```

gunzip декомпрессирует указанный архивный файл, выводит результат на стандартное устройство вывода, откуда он считывается утилитой *tar*, как со стандартного ввода, и распаковывается. Ну не прелесть ли Unix?

Конечно, обе эти команды печатать долго. К счастью, GNU-версия *tar* имеет параметр *z*, задание которого автоматически создает или распаковывает сжатые архивы. (Мы приберегли разговор об этом параметре до этого момента, чтобы вы могли по достоинству оценить его удобство.) Например, для создания сжатых архивных файлов и их распаковки можно использовать команды

```
tar cvzf tarfile.tar.gz files...
```

и

```
tar xvzf tarfile.tar.gz
```

Обратите внимание, что создаваемым таким образом файлам нужно присваивать расширение *.tar.gz* (или столь же часто используемое *.tgz*, которое удобно использовать в системах с ограничениями на имена файлов), чтобы был ясен их формат. Параметр *z* нормально работает и с другими функциями *tar*, такими как *t*.

Поддержка параметра *z* есть только в GNU-версии *tar*. При использовании *tar* в других Unix-системах для выполнения таких же задач вам может потребоваться использование более длинных команд. Почти все системы Linux используют GNU-версию *tar*.

Если вы хотите использовать *tar* вместе с *bzip2*, то о своем выборе программы сжатия вы можете сообщить следующим образом:

```
tar cvf tarfile.tar.bz2 --use-compress-program=bzip2 files...
```

или короче:

```
tar cvf tarfile.tar.bz2 --use=bzip2 files...
```

или еще короче:

```
tar cvjf tarfile.tar.bz2 files
```

Последняя версия работает только с последними версиями GNU *tar*, поддерживающими параметр *j*.

Имея это в виду, можно писать короткие сценарии оболочки или псевдонимы для создания архивных файлов и их распаковки. Если вы пользуетесь *bash*, то можно включить в файл *.bashrc* следующие функции:

```
tarc () { tar czvf $1.tar.gz $1 }
tarx () { tar xzvf $1 }
tart () { tar tzvf $1 }
```

Если определены такие функции, то создать сжатый архивный файл из отдельного каталога можно с использованием команды

```
tarc directory
```

Полученный архивный файл будет назван *directory.tar.gz*. (Следите, чтобы в конце имени каталога не было косой черты, иначе архив будет создан как *.tar.gz* в данном каталоге.) Чтобы вывести список файлов сжатого *tar*-архива, выполните команду

```
tart file.tar.gz
```

Для распаковки такого архива выполните команду

```
tarx file.tar.gz
```

В заключение отметим, что файлы, созданные с помощью *gzip* и/или *tar*, могут распаковываться хорошо известной утилитой *WinZip* в Windows. *WinZip* пока не поддерживает *bzip2*. И наоборот, файл в формате *.zip* можно распаковать на машине под Linux с помощью команды *unzip*.

Приемы работы с tar

Поскольку *tar* записывает в архив права владения и доступа к файлам, а также сохраняет полную структуру каталогов, символические и жесткие ссылки, с его помощью очень удобно копировать или перемещать целое дерево каталогов из одного места в другое в пределах данной системы и даже, как мы увидим позднее, с одной машины на другую. Используя описанный выше синтаксис с минусом, можно вывести *tar*-архив на стандартное устройство вывода, откуда он читается и распаковывается в любое место через стандартное устройство ввода.

Допустим, имеется каталог с двумя подкаталогами – *from-stuff* и *to-stuff*. Каталог *from-stuff* содержит целое дерево файлов, символических ссылок и т. д., которые трудно точно отобразить, используя команду *cp* с рекурсией. Для того чтобы скопировать все дерево из каталога *from-stuff* в каталог *to-stuff*, можно выполнить такие команды:

```
cd from-stuff
tar cf - . | (cd ../to-stuff; tar xvf -)
```

Просто и элегантно, не правда ли? Мы начинаем работать в каталоге *from-stuff* и создаем архив текущего каталога, который выводится на стандартное устройство вывода. Этот архив читается подоболочкой (содержащиеся в скобках команды); подоболочка совершает переход в целевой каталог *../to-stuff* (относительно *from-stuff*) и выполняет *tar xvf*, осуществляя чтение со стандартного устройства ввода. Никаких архивных файлов на диск не пишется; данные посылаются через канал от одного процесса *tar* другому. Второй процесс *tar* запускается с параметром *v*, благодаря чему выводятся имена всех распаковываемых файлов, что позволяет контролировать правильность работы команды.

С помощью такого приема можно перемещать целые деревья каталогов с одной машины на другую по сети. Для этого нужно лишь включить соответствующую команду *rsh* (или *ssh*) в подоболочку с правой стороны канала. Удаленная оболочка выполнит *tar*, чтобы считать архив со стандартного устройства ввода. (Фактически GNU *tar* имеет возможность автоматически читать или выводить *tar*-архивы на других машинах через сеть; подробности см. на странице руководства по *tar*(1).)

Обновление программного обеспечения

Linux находится в непрерывном развитии. Благодаря кооперативной природе проекта постоянно становится доступным новое программное обеспечение, и обновляются версии программ. В особенности это касается ядра Linux, над которым трудится несколько групп разработчиков. Когда идет процесс разработки, новые патчи для ядра нередко появляются ежедневно. Другие части системы могут развиваться не столь динамично, но действуют те же принципы.

Можно ли в такой ситуации рассчитывать, что у вас на машине всегда будут самые новые версии системного программного обеспечения? Если ответить коротко, то нельзя. Хотя есть люди, которым необходимо иметь, скажем, последнюю исправленную версию ядра на текущий день, по большей части нет необходимости утруждать себя столь частыми исправлениями. В этом разделе мы обсудим, зачем и в какой момент обновлять систему, и покажем, каким образом можно обновить некоторые важные компоненты системы.

Когда следует производить обновление? Обычно следует рассматривать вопрос об обновлении части системы, только если выявилась *необходимость* обновления. Например, если вы узнали о выпуске новой версии некоторого приложения, в которой исправлены важные ошибки (т. е. ошибки, действительно влияющие именно на вашу работу с приложением), то вам, возможно, следует обновить это приложение. Если в новой версии есть новые функции, которые вы считаете полезными для себя, или она обеспечивает резкий рост производительности по сравнению с текущей версией, тогда тоже есть смысл произвести обновление. Если ваша машина тем или иным способом подключена к Интернету, то причиной обновления может быть возможность заделать брешь в системе защиты, о которой вы недавно узнали. Однако делать обновление только для того, чтобы иметь последнюю версию какой-либо программы, не очень разумно.

Обновление иногда оказывается достаточно трудоемкой операцией. Например, вы можете пожелать обновить программу, для работы которой требуются новейшие версии компилятора, библиотек или другого программного обеспечения. Обновление этой программы может также потребовать обновления некоторых других компонентов системы, что займет много времени. С другой стороны, это может быть аргументом в пользу того, чтобы поддерживать на машине свежие версии программного обеспечения: если у вас уже стоят свежие версии компилятора и библиотек, обновить такую программу не составит труда.

Как узнать о выпуске новых версий программного обеспечения Linux? Лучше всего следить за объявлениями в телеконференции *comp.os.linux.announce* (см. раздел «Телеконференции Usenet» главы 1), где помещаются объявления о новых выпусках и другие важные сведения. При наличии доступа в Интернет можно загрузить программное обеспечение через FTP и установить его. Другим хорошим источником новостей о программном обеспечении Linux является сайт *http://www.freshmeat.net*.

При отсутствии доступа в Usenet или Интернет лучше всего следить за последними изменениями, приобретя подписку на CD-ROM. В результате раз в несколько месяцев вы будете получать на CD-ROM обновленную копию разных сайтов Linux FTP. Такую услугу предоставляет ряд поставщиков Linux, и она полезна, даже если у вас есть доступ в Интернет.

Тем самым мы переходим к следующему вопросу: какой метод обновления лучше? Некоторые считают, что когда появляется новая версия их любимого дистрибутива, проще всего полностью обновить систему, установив все с нуля. При этом не приходится беспокоиться о том, смогут ли различные вер-

сии программ работать совместно. При отсутствии доступа в Интернет это может быть действительно самым легким методом: если вы получаете новый CD-ROM только раз в два месяца, значительная часть вашего программного обеспечения может быть устаревшей.

Однако, по нашему мнению, переустановка отнюдь не является хорошим способом обновления. Большинство современных дистрибутивов Linux не предназначены для такого рода обновления, и полная переустановка может занять много времени. Кроме того, при таком обновлении вы обычно теряете все свои модификации и настройки системы, и вам придется делать резервные копии личных каталогов пользователей и других важных файлов, которые могут быть потеряны при переустановке. Многие новички выбирают такой путь, поскольку он проще всего. На практике различия между версиями могут быть не столь велики, и в полной переустановке системы обычно нет необходимости. Ее можно избежать, если немного разбираться в том, как производить обновление.

Ниже мы покажем, как можно обновить отдельные части системы. Мы покажем, как обновить системные библиотеки и компилятор, и расскажем об общем методе установки нового программного обеспечения. Затем мы расскажем, как собрать новое ядро.

Обновление библиотек

Большинство программ в Linux компилируется в расчете на использование библиотек совместного доступа. В этих библиотеках содержатся полезные функции, использующиеся во многих программах. Вместо того чтобы хранить экземпляры этих функций во всех обращающихся к ним программах, их собирают в файлы системных библиотек, которые все программы считывают во время выполнения. Это означает, что при выполнении программы считывается код самой программы, а затем все подпрограммы из библиотечных файлов совместного доступа. Благодаря этому сберегается значительный объем дискового пространства: на диске хранится только один экземпляр библиотечной функции.

В некоторых случаях вместо использования разделяемых библиотек программа компилируется так, чтобы в ней был собственный экземпляр библиотечных функций (обычно это нужно при отладке). О собранных таким образом программах говорят, что в них использована *статическая компоновка*, в отличие от программ, использующих библиотеки совместного доступа, в которых применена *динамическая компоновка*.

Таким образом, динамически компонуемые исполняемые модули требуют наличия на диске библиотек совместного доступа. Библиотеки совместного доступа реализуются так, чтобы компилируемые для их использования программы в целом не зависели от версий имеющихся библиотек. Это означает, что библиотеки можно обновлять, а все программы, собранные таким образом, чтобы использовать эти библиотеки, автоматически будут вызывать новые функции. (Существует исключение: если в библиотеках сделаны значительные изменения, старые программы не смогут с ними работать. Это видно, ко-

гда изменяется основной номер версии; более подробно мы расскажем об этом позже. В этом случае вам придется хранить и старые, и новые библиотеки. Все старые программы будут продолжать использовать старые библиотеки, а вновь скомпилированные программы будут использовать новые библиотеки.)

При сборке программы, использующей библиотеки совместного доступа, к ней добавляется участок кода, вызывающий при запуске программы динамический компоновщик *ld.so*. На *ld.so* возлагается задача найти требуемые для программы библиотеки совместного доступа и загрузить их функции в память. Кроме того, динамически компонуемые программы собираются с «заглушками» (stubs), которые просто занимают в выполняемых модулях место действительных функций из библиотек совместного доступа. Во время выполнения программы *ld.so* заменяет заглушки программным кодом, находящимся в библиотеках.

Для того чтобы получить список библиотек совместного доступа, от которых зависит данная исполняемая программа, можно воспользоваться командой *ldd*, например:

```
rutabaga% ldd /usr/bin/X11/xterm
        libXft.so.1 => /usr/X11R6/lib/libXft.so.1 (0x40032000)
        libXrender.so.1 => /usr/X11R6/lib/libXrender.so.1 (0x40088000)
        libXaw.so.7 => /usr/X11R6/lib/libXaw.so.7 (0x4008d000)
        libXmu.so.6 => /usr/X11R6/lib/libXmu.so.6 (0x400e4000)
        libXt.so.6 => /usr/X11R6/lib/libXt.so.6 (0x400fa000)
        libSM.so.6 => /usr/X11R6/lib/libSM.so.6 (0x40148000)
        libICE.so.6 => /usr/X11R6/lib/libICE.so.6 (0x40152000)
        libXpm.so.4 => /usr/X11R6/lib/libXpm.so.4 (0x4016a000)
        libXext.so.6 => /usr/X11R6/lib/libXext.so.6 (0x40179000)
        libX11.so.6 => /usr/X11R6/lib/libX11.so.6 (0x40188000)
        libncurses.so.5 => /lib/libncurses.so.5 (0x4026b000)
        libc.so.6 => /lib/libc.so.6 (0x402b5000)
        /lib/ld-linux.so.2 => /lib/ld-linux.so.2 (0x40000000)
```

Здесь мы видим, что программа *xterm* зависит от ряда библиотек совместного доступа, включая *libXaw*, *libXt*, *libX11* и *libc*. (Первые три, имена которых начинаются с *libX*, относятся к X Window System, а последняя является стандартной библиотекой C.) Видны также номера версий библиотек, для которых была скомпилирована программа (т. е. версий использованных функций-заглушек), и имена файлов, в которых находятся библиотеки. Это те файлы, которые *ld.so* должен будет найти при выполнении программы.

Для использования библиотеки совместного доступа версия заглушки в исполняемом файле должна быть совместима с версией библиотеки совместного доступа. Обычно библиотека является совместимой, если основные номера ее версии и версии заглушки совпадают. Основной номер версии – это та часть номера, которая находится сразу после *.so*. В данном случае *libX11* (самая главная библиотека, используемая X Window System) используется с основным номером версии 6. Файл библиотеки *libX11.so.6* (который обычно находится в */usr/X11R6/lib/*) вполне может быть просто символической ссылкой, например на *libX11.so.6.2*. Это означает, что основной номер вер-

сии библиотеки – 6, а дополнительный – 2. Поэтому, если программа компилировалась с версией заглушек 6.0, ее исполняемый модуль может использовать библиотеки версий 6.1, 6.2 и т. д. Если бы была выпущена новая версия библиотеки с основным номером версии 6 и дополнительным номером 3 (имя файла для нее было бы *libX11.so.6.3*), для ее использования достаточно было бы изменить символическую ссылку *libX11.so.6*, чтобы она указывала на новую версию. В результате исполняемый модуль *xterm* мог бы автоматически воспользоваться исправлениями ошибок или другими преимуществами, присутствующими в новой версии. В разделе «Кое-что еще о библиотеках» главы 13 мы описываем, как использовать библиотеки совместного доступа с вашими собственными программами.

В файле */etc/ld.so.conf* находится список каталогов, в которых *ld.so* ищет библиотечные файлы. Вот пример такого файла:

```
/usr/lib
/usr/local/lib
/usr/X11R6/lib
```

Независимо от содержания *ld.so.conf* компоновщик *ld.so* всегда осуществляет поиск в */lib* и */usr/lib*. Обычно не требуется изменять содержание этого файла, а в переменной окружения `LD_LIBRARY_PATH` к этому пути поиска можно добавить другие каталоги (например, если у вас есть собственные библиотеки совместного доступа, которые не должны использоваться во всей системе). Однако если вы все же добавите каталоги в */etc/ld.so.conf* либо обновите или добавите в систему дополнительные библиотеки, нужно будет выполнить команду *ldconfig*, которая заново создаст кэш библиотек совместного доступа в */etc/ld.so.cache* соответственно пути поиска для *ld.so*. Этот кэш используется *ld.so* для быстрого поиска библиотек во время выполнения вместо фактического просмотра указанных в пути каталогов. За дополнительными сведениями обратитесь к страницам руководства по *ld.so* и *ldconfig*.

Теперь, когда вам стало ясно, как используются библиотеки совместного доступа, перейдем к тому, как они обновляются. Чаще всего обновляются две библиотеки: *libc* (стандартная библиотека C) и *libm* (библиотека математических функций). Поскольку имена этих библиотек несколько отличаются от общей схемы, мы рассмотрим здесь другую библиотеку – *libncurses*, которая «эмулирует» графическую оконную систему в текстовой консоли.

Каждой библиотеке совместного доступа соответствуют два отдельных файла:

library.***a***

Это статическая версия библиотеки. При статической компоновке программы функции копируются из этого файла непосредственно в выполняемый модуль, который в результате содержит собственный экземпляр библиотечных функций.[1]

[1] В некоторых дистрибутивах статические версии библиотек помещаются в отдельный пакет и могут не устанавливаться по умолчанию. В таких случаях файлы *.a* вы обнаружите только после того, как установите их.

library.so.version

Это образ самой библиотеки совместного доступа. При динамической компоновке в исполняемый модуль копируются функции-заглушки из этого файла, что позволяет *ld.so* во время выполнения программы найти библиотеку. При выполнении программы *ld.so* копирует функции из библиотеки совместного доступа в память, чтобы программа могла их использовать. Если программа компонуется динамически, файл *library.a* не используется.

Для библиотеки *libncurses* могут существовать файлы *libncurses.a* и *libncurses.so.5.2*. Файлы *.a* обычно находятся в */usr/lib*, а файлы *.so* хранятся в */lib*. При компиляции программы для ее сборки используется либо файл *.a*, либо файл *.so*, которые по умолчанию ищутся в */lib* и */usr/lib* (а возможно, и в других местах). Если у вас есть собственные библиотеки, вы можете хранить их где угодно, управляя их поиском при компоновке через параметр *–L*, передаваемый компилятору. (См. подробнее в разделе «Кое-что еще о библиотеках» главы 13.)

Для большинства общесистемных библиотек образ *library.so.version* хранится в */lib*. Библиотеки совместного доступа могут находиться в любом из каталогов, в которых *ld.so* осуществляет поиск во время выполнения, в том числе */lib*, */usr/lib* и перечисленных в *ld.so.conf*. (См. подробнее на странице руководства по *ld.so*.)

Если вы посмотрите каталог */lib*, то обнаружите примерно такой список файлов:

```
lrwxrwxrwx    1 root     root           17 Jul 11 06:45 /lib/libncurses.so.5 \
    -> libncurses.so.5.2
-rwxr-xr-x    1 root     root       319472 Jul 11 06:45 /lib/libncurses.so.5.2
lrwxrwxrwx    1 root     root           13 Jul 11 06:45 libz.so.1 -> libz.so.1.1.3
-rwxr-xr-x    1 root     root        62606 Jul 11 06:45 libz.so.1.1.3
```

Здесь видны образы двух библиотек совместного доступа – *libncurses* и *libz*. Обратите внимание, что для каждого образа есть символическая ссылка на него в виде *library.so.major*, где *major* – основной номер версии библиотеки, а дополнительный номер опущен, поскольку *ld.so* ищет библиотеку только по основному номеру версии. Когда *ld.so* видит программу, скомпилированную с заглушками для *libncurses* версии 5.2, он ищет в своем пути поиска файл *libncurses.so.5*. В данном случае */lib/ libncurses.so.5* является символической ссылкой на */lib/libncurses.so.5.2* – версию фактически установленной библиотеки.

При обновлении библиотеки необходимо заменить соответствующие ей файлы *.a* и *.so.version*. Заменить файл *.a* просто: скопируйте новую версию поверх него. Однако при замене образов библиотек совместного доступа *.so.version* нужно проявлять осторожность: многие программы текстового режима в системе зависят от этих образов, поэтому их нельзя так просто удалить или переименовать. Иными словами, символическая ссылка *library.so.major* должна *всегда* указывать на действительный образ библиотеки. Для этого

сначала скопируйте новый файл образа в */lib*, а затем за один шаг измените командой *ln -sf* символическую ссылку так, чтобы она указывала на новый файл. Покажем это на примере.

Допустим, вы обновляете библиотеку *libncurses* с версии 5.2.18 на версию 5.4.47. У вас должны быть файлы *libncurses.a* и *libncurses.so.5.4.47*. Сначала скопируйте файл *.a* в соответствующее место, переписав прежнюю версию:

```
rutabaga# cp libncurses.a /usr/lib
```

Теперь скопируйте новый файл образа в */lib* (или туда, где он должен находиться):

```
rutabaga# cp libncurses.so.5.4 /lib
```

Теперь после выполнения команды *ls -l /lib/libncurses* вы должны увидеть примерно следующее:

```
lrwxrwxrwx    1 root     root           17 Dec 10  1999 /lib/libncurses.so.5 ->
libncurses.so.4.2
-rwxr-xr-x    1 root     root       319472 May 11  2001 /lib/libncurses.so.5.2
-rwxr-xr-x    1 root     root       321042 May 11  2001 /lib/libncurses.so.5.4
```

Измените символическую ссылку так, чтобы она указывала на новую библиотеку, командой

```
rutabaga# ln -sf /lib/libncurses.so.5.4 /lib/libncurses.so.5
```

В результате вы получите:

```
lrwxrwxrwx  1 root  root      14 Oct 23 13:25 libncurses.so.5 ->\
    /lib/libncurses.so.5.4
-rwxr-xr-x  1 root  root  623620 Oct 23 13:24 libncurses.so.5.2
-rwxr-xr-x  1 root  root  720310 Nov 16 11:02 libncurses.so.5.4
```

Теперь можно благополучно удалить старый файл образа – *libncurses.so.5.2*. Необходимо заменять символическую ссылку за один ход с помощью *ln -sf*, особенно при обновлении таких библиотек, как *libc*. Если бы вы сначала удалили ссылку, а затем попробовали создать ее снова командой *ln -s*, то по всей вероятности *ln* не смогла бы выполниться, поскольку символической ссылки уже нет, и *ld.so* не смог бы найти библиотеку *libc*. Если ссылка уничтожена, то почти все программы в системе не смогут выполняться. При обновлении образов библиотек совместного доступа нужно проявлять большую осторожность. Для *libncurses* ситуация не так критична, потому что у вас всегда остаются программы командной строки, с помощью которых можно убрать учиненный беспорядок, но если вы привыкли пользоваться программами, основанными на *libncurses*, такими как Midnight Commander, то также могут возникнуть неудобства.

При обновлении библиотеки или добавлении библиотеки к системе очень неплохо запустить утилиту *ldconfig*, чтобы заново создать библиотечный кэш, используемый *ld.so*. В некоторых случаях *ld.so* не распознает новую библиотеку, пока не выполнена команда *ldconfig*.

Остается еще один вопрос: где брать новые версии библиотек? Некоторые основные системные библиотеки (*libc*, *libm* и т. д.) можно загрузить из каталога */pub/Linux/GCC* на *ftp://ftp.ibiblio.org*. Там содержатся Linux-версии компилятора *gcc*, библиотек, include-файлы и прочие утилиты. Для каждого файла должен иметься файл *README* или *release*, в котором описано, что нужно делать и как его устанавливать. Другие библиотеки поддерживаются и архивируются отдельно. В любом случае для всех устанавливаемых библиотек должны иметься файлы *.so.version*, а возможно, и *.a*, а также комплект include-файлов для использования с компилятором.

Обновление компилятора

Другой важной частью системы, которую нужно поддерживать в обновленном состоянии, является компилятор C и связанные с ним утилиты. В их число входят *gcc* (собственно компилятор GNU C и C++), компоновщик, ассемблер, препроцессор C и различные include-файлы и библиотеки, используемые самим компилятором. Все они входят в состав дистрибутива Linux *gcc*. Обычно новая версия *gcc* выходит вместе с новыми версиями библиотеки *libc* и include-файлов, причем одно требуется для другого.

Текущую версию *gcc* для Linux можно найти в разных архивах FTP, в том числе */pub/Linux/GCC* на *ftp://ftp.ibiblio.org*. В комментариях к выпуску должно быть сказано, как действовать. Обычно обновление компилятора состоит просто в распаковке нескольких *tar*-файлов при регистрации суперпользователем и иногда в удалении некоторых файлов. Если у вас нет доступа в Интернет, свежую версию компилятора можно получить на CD-ROM с архивом FTP-сайта, о чем говорилось выше.

Выяснить, какая версия *gcc* установлена, можно, выполнив команду

```
gcc -v
```

Она выдает примерно следующее:

```
Reading specs from /usr/lib/gcc-lib/i486-suse-linux/2.95.3/specs
gcc version 2.95.3 20010315 (SuSE)
```

Учтите, что *gcc* является лишь оболочкой к действительному компилятору и средствам генерации кода, находящимся в каталоге

```
/usr/lib/gcc-lib/machine/version
```

gcc (обычно находится в */usr/bin*) можно использовать с разными версиями собственно компилятора, используя ключ *-V*. В разделе «Программирование с помощью gcc» главы 13 мы подробно опишем использование *gcc*.

Здесь мы хотим предупредить вас, чтобы вы не пытались пользоваться более новыми компиляторами, если нет точного представления о том, для чего вам это нужно. Новые компиляторы могут генерировать объектные файлы, не совместимые со старыми, что приводит к серьезным неприятностям. На момент написания книги *gcc* версии 2.95.3 считается стандартным компилятором для Linux, который должен подойти всем. Когда один поставщик дист-

рибутивов (Red Hat) начал поставлять вместо него более новую версию (которая даже не была официальным выпуском), пользователи столкнулись с массой проблем. Конечно, когда вы будете читать эту книгу, стандартной, возможно, будет считаться другая версия компилятора. Если вы ищете приключений, попробуйте более новые версии, но будьте готовы к необходимости серьезных настроек системы.

Общая процедура обновления программного обеспечения

Разумеется, периодически нужно обновлять и другие части системы. Как обсуждалось в предыдущем разделе, обычно легче и правильнее обновлять только те приложения, которые в этом действительно нуждаются. Например, если вы не используете Emacs, зачем стремиться установить самую свежую его версию? Возможно, не стоит также стараться установить самые последние версии часто используемых приложений. Если какая-то программа работает нормально, нет особой необходимости ее обновлять.

Современные Linux-системы предоставляют различные способы обновления программного обеспечения, некоторые из которых являются ручными (самыми гибкими, но и самыми сложными), а другие – вполне автоматизированными. Мы рассмотрим ниже три различных технологии: систему пакетов RPM, систему пакетов Debian и ручной способ.

Хотим подчеркнуть, что пользоваться пакетами и системами пакетов действительно удобно, и даже если вы опытный пользователь, стоит применять эту технику, чтобы сберечь время для других более интересных задач. Вот краткий список преимуществ пакетных систем:

- Все, что относится к программному пакету, находится в одном загружаемом файле.
- Программный пакет может быть удален целиком, не нарушая работоспособности других пакетов.
- Пакетные системы ведут учет зависимостей в базах данных, что позволяет автоматически отслеживать зависимости. Например, они могут сообщить о необходимости установки более новой версии некоторой библиотеки, необходимой для работы приложения, которое вы намерены установить (и отказаться удалить библиотечный пакет, если его библиотеки используются другими установленными пакетами).

Конечно, у систем пакетов есть и недостатки, часть которых мы отметим, когда будем рассказывать о системах пакетов RPM и Debian. Общая проблема состоит в том, что, начав пользоваться системой пакетов (что почти неизбежно, если использовать интерфейсы автоматизированной установки приложения), приходится уже все устанавливать с помощью пакетов, иначе невозможно следить за зависимостями. По той же причине не рекомендуется смешивать разные системы пакетов.

Использование RPM

RPM (Red Hat Package Manager – менеджер пакетов Red Hat) является средством автоматизации установки двоичных программных файлов, которое запоминает список необходимых файлов, благодаря чему вы можете быть уверены в правильной работе программного обеспечения. Несмотря на свое название RPM может использоваться не только с Red Hat, но и со многими другими дистрибутивами, включая SuSE и Caldera. Использование RPM значительно облегчает установку и удаление программного обеспечения.

Главная идея RPM состоит в ведении базы данных пакетов и принадлежащих им файлов. При установке нового пакета информация о нем записывается в базу данных. Затем, когда вы пытаетесь удалить пакет, для каждого входящего в него файла RPM проверяет, нет ли других пакетов, использующих его. Если они есть, то рассматриваемый файл не удаляется.

Кроме того, RPM отслеживает зависимости. Каждый пакет может зависеть от одного или нескольких других пакетов. При установке пакета RPM проверяет, установлены ли уже пакеты, от которых зависит данный пакет. Если не установлены, то RPM сообщает об этом и отказывается устанавливать пакет.

Зависимости используются и при удалении пакетов: когда вы хотите удалить пакет, от которого зависят какие-либо другие пакеты, RPM тоже сообщает об этом и отказывается выполнить задание.

За удобства, предоставляемые RPM, приходится платить. Во-первых, разработчику существенно сложнее сделать пакет RPM, чем просто упаковать все файлы в *tar*-архив. Во-вторых, из пакета RPM нельзя извлечь один файл: вам приходится устанавливать все или ничего.

Если у вас уже стоит система RPM, то устанавливать RPM-пакеты очень легко. Допустим, у вас есть RPM-пакет с названием *SuperFrob-4.i386.rpm* (RPM-пакеты всегда имеют расширение *.rpm*; *i386* указывает на то, что это двоичный пакет, скомпилированный для машин Intel). Его можно установить командой

```
tigger # rpm -i SuperFrob-4.i386.rpm
```

Вместо *–i* можно использовать длинную версию названия этого параметра; выбирайте, что вам предпочтительнее:

```
tigger # rpm --install SuperFrob-4.i386.rpm
```

При нормальной установке на экран ничего не выводится. Если вы хотите, чтобы RPM был более словоохотлив, можете дать такую команду:

```
tigger # rpm -ivh SuperFrob-4.i386.rpm
```

В результате будет выведено имя пакета и ряд контрольных сообщений, позволяющих следить за ходом установки.

Если пакету, который вы пытаетесь установить, нужен другой пакет, который еще не установлен, вы получите сообщение примерно следующего содержания:

```
tigger # rpm -i SuperFrob-4.i386.rpm
failed dependencies:
        frobnik-2 is needed by SuperFrob-4
```

Увидев такое сообщение, вам придется начать поиски пакета *frobnik-2* и установить сначала его. Разумеется, и этот пакет в свою очередь может зависеть от других пакетов.

Если вы хотите обновить уже установленный пакет, воспользуйтесь параметром *-U* или *--update* (что представляет собой сочетание параметра *-i* с некоторыми другими параметрами):

```
tigger # rpm -U SuperFrob-5.i386.rpm
```

Удаление пакета производится с помощью параметра *-e* или *--erase*. При этом вы указываете не имя файла пакета (которого у вас уже может не быть), а имя пакета и номер его версии:

```
tigger # rpm -e SuperFrob-5
```

Кроме уже описанных параметров, посредством которых изменяется состояние вашей системы, есть параметр *-q*, с помощью которого можно получить разнообразные данные обо всем, что записано в базе данных RPM и файлах пакетов. Вот несколько полезных применений параметра *-q*:

- Выяснить номер версии установленного пакета:

```
tigger# rpm -q SuperFrob
SuperFrob-5
```

- Получить список всех установленных пакетов:

```
tigger# rpm -qa
SuperFrob-5
OmniFrob-3
...
libc-5.4.47-1
```

- Выяснить, какому пакету принадлежит файл:

```
tigger# rpm -qf /usr/bin/dothefrob
SuperFrob-5
tigger# rpm -qf /home/kalle/.xinitrc
file /home/kalle/.xinitrc is not owned by any package
```

(не принадлежит ни одному пакету)

- Вывести сведения о заданном пакете:

```
tigger# rpm -qi rpm
Name        : rpm                          Relocations: (not relocateable)
Version     : 3.0.6                             Vendor: SuSE GmbH, Nuernberg,
Germany
Release     : 78                            Build Date: Fri 11 May 2001 05:18:18
PM CEST
Install date: Sun 15 Jul 2001 03:06:14 PM CEST      Build Host: hewitt.suse.de
Group       : System Environment/Base       Source RPM: rpm-3.0.6-78.src.rpm
Size        : 7624258                          License: GPL
```

```
Packager    : feedback@suse.de
Summary     : RPM Package Manager
Description :
RPM Package Manager is the main tool for managing software packages
of the SuSE Linux distribution.
rpm can be used to install and remove software packages; with rpm it's easy to
update packages.  rpm keep track of all these manipulations in a central
database.  This way it is possible to get an overview of all installed
packages; rpm also supports database queries.
```

(Описание: rpm (Red Hat Package Manager) является основным средством управления программными пакетами дистрибутива SuSE Linux. RPM можно использовать для установки и удаления программных пакетов; с его помощью легко обновлять пакеты. Все эти действия регистрируются в центральной базе данных. Благодаря этому можно получить обзор всех установленных пакетов. RPM поддерживает также запросы к базе данных.)

```
Authors:
--------
    Erik Troan <ewt&commat;redhat.com>
    Marc Ewing <marc&commat;redhat.com>

SuSE series: a
```

- Показать файлы, которые будут установлены для указанного пакета:

```
tigger# rpm -qpl SuperFrob-5.i386.rpm
/usr/bin/dothefrob
/usr/bin/frobhelper
/usr/doc/SuperFrob/Installation
/usr/doc/SuperFrob/README
/usr/man/man1/dothefrob.1
```

Мы показали только основные режимы работы, дополнением к которым служит большое число вспомогательных параметров. О них можно узнать на странице руководства для команды *rpm*(8).

Если вы столкнулись с необходимостью установки RPM-пакета на такой системе, как Slackware или Debian, которые не основываются на RPM, ситуация несколько сложнее.

Вы можете либо использовать довольно очевидную в применении команду *alien*, способную преобразовывать пакеты из одного формата в другой и имеющуюся в большинстве дистрибутивов, либо самостоятельно создать базу данных RPM.

В последнем случае вам сначала нужно получить саму программу *rpm*. Ее можно загрузить с *http://www.rpm.org*. Соберите ее и установите согласно имеющейся инструкции по установке. Если компилятор *gcc* у вас установлен, никаких проблем при этом не должно возникнуть.

Следующей задачей является инициализация базы данных RPM. Дистрибутивы, содержащие RPM, проводят инициализацию автоматически, но в других системах вам придется выполнить команду

```
tigger # rpm --initdb
```

Эта команда создаст ряд файлов в каталоге */var/lib/rpm*. Каталог */var/lib* должен уже существовать; если его нет, создайте его командой *mkdir*.

Теперь можно устанавливать пакеты RPM обычным образом, но поскольку вы устанавливали базовые части системы, такие как библиотека C, без помощи RPM, то будете получать ошибки типа:

```
tigger # rpm -i SuperFrob-4.i386.rpm
failed dependencies:
        libm.so.5 is needed by SuperFrob-4
        libdl.so.1 is needed by SuperFrob-4
        libc.so.5 is needed by SuperFrob-4
```

поскольку эти файлы не внесены в базу данных RPM. Конечно, в действительности эти файлы есть в вашей системе, иначе большинство программ просто не смогло бы работать. Для того чтобы заставить RPM работать, необходимо сообщить программе, что она не должна обращать внимания на зависимости. Это можно сделать, задав параметр командной строки *--nodeps*:

```
tigger # rpm -i --nodeps SuperFrob-4.i386.rpm
```

Теперь RPM установит пакет безо всяких жалоб. Конечно, он будет работать, только если установлены необходимые для него библиотеки. Один лишь факт задания *--nodeps* не спасает, когда «зависимая» библиотека или программное обеспечение не установлены на машине.

Представленных сведений достаточно для управления вашей системой с помощью RPM. Если вы хотите узнать больше, почитайте страницу руководства по команде *rpm* или обратитесь на сайт *http://www.rpm.org*.

Некоторые коммерческие фирмы предлагают платные службы по автоматизированному обновлению, основанные на RPM. Подписавшись на эти услуги, можно автоматически обновлять систему: такая служба определяет, какие имеются новые пакеты, и устанавливает их. Если вы являетесь пользователем SuSE, то SuSE предоставит вам такую службу бесплатно. Даже для дистрибутива Debian (его пакетная система описывается в следующем разделе) есть система автоматического обновления (описывается там же). Однако некоторые эксперты считают, что такое автоматизированное обновление создает угрозу информационной безопасности.

Использование dpkg и apt

После *rpm* самым популярным менеджером пакетов для Linux является *dpkg*, который управляет архивами *.deb*. Как подсказывает имя, формат *.deb* ведет происхождение от дистрибутива Debian, но им также пользуются Libranet, Xandros и другие поставщики. Подобно формату RPM, формат *.deb* ведет учет зависимостей и файлов, что помогает обеспечить согласованность системы.

Технические различия между двумя форматами весьма незначительны: несмотря на несовместимость форматов RPM и *.deb* (нельзя, например, непо-

средственно установить пакет Debian в систему Red Hat), можно воспользоваться утилитой *alien* и транслировать пакеты *.deb* для установки в других дистрибутивах (и обратно). Главное различие форматов в том, что пакеты *.deb* создаются с помощью инструментов, позволяющих обеспечить непротиворечивость их структуры и соответствие политике (главным образом, Debian Policy Manual, которую можно найти в пакете *debian-policy*), что позволяет разработчикам создавать пакеты высокого качества.

Тогда как *dpkg* представляет собой интерфейс низкого уровня к менеджеру пакетов Debian, большинство функций обычно осуществляется через комплект программ *apt* либо такие программы или интерфейсы, как *dselect*, *aptitude*, *gnome-apt*, *synaptic* и *kpackage*.

Установить пакет *.deb* в системе Debian очень просто. Например, если есть пакет с именем *superfrob_4-1_i386.deb*, его можно установить с помощью команды

```
tigger # dpkg -i superfrob_4-1_i386.deb
Selecting previously deselected package superfrob.
(Reading database ... 159540 files and directories currently installed.)
Unpacking superfrob (from superfrob_4-1_i386.deb) ...
Setting up superfrob (4-1) ...
```

Если для пакета *superfrob* отсутствует некоторый связанный с ним пакет, *dpkg* выводит предупредительное сообщение:

```
tigger # dpkg -i superfrob_4-1_i386.deb
Selecting previously deselected package superfrob.
(Reading database ... 159540 files and directories currently installed.)
Unpacking superfrob (from superfrob_4-1_i386.deb) ...
dpkg: dependency problems prevent configuration of superfrob:
 superfrob depends on frobnik (>> 2); however:
  Package frobnik is not installed.
dpkg: error processing superfrob (--install):
 dependency problems - leaving unconfigured
Errors were encountered while processing:
 superfrob
```

Этот вывод показывает, что для полной установки пакета потребуется *frobnik* Version 2 или более новый. (Файлы пакета установлены, но не будут работать, пока не будет также установлен *frobnik*.)

В отличие от RPM, *dpkg* не делает различия между установкой нового пакета и обновлением существующего: в обоих случаях используется параметр *-i* (или *--install*). Например, чтобы обновить *superfrob* с помощью нового загруженного пакета *superfrob_5-1_i386.deb*, нужно просто ввести:

```
tigger # dpkg -i superfrob_5-1_i386.deb
(Reading database ... 159546 files and directories currently installed.)
Preparing to replace superfrob 4-1 (using superfrob_5-1_i386.deb) ...
Unpacking replacement superfrob ...
Setting up superfrob (5-1) ...
```

Чтобы удалить пакет, можно воспользоваться параметрами *-r* (*--remove*) или *-P* (*--purge*). Параметр *--remove* удаляет большую часть пакета, но сохраняет файлы конфигурации, тогда как *--purge* удаляет также системные файлы конфигурации. Например, полностью удалить *superfrob* можно командой

```
tigger # dpkg -P superfrob
(Reading database ... 159547 files and directories currently installed.)
Removing superfrob ...
```

С помощью *dpkg* можно также выяснить, какие пакеты установлены в системе, если задать параметр *-l* (*--list*):

```
tigger $ dpkg -l
Desired=Unknown/Install/Remove/Purge/Hold
| Status=Not/Installed/Config-files/Unpacked/Failed-config/Half-installed
|/ Err?=(none)/Hold/Reinst-required/X=both-problems (Status,Err: uppercase=bad)
||/ Name           Version        Description
+++-==============-==============- ============================================
ii  a2ps           4.13b-15       GNU a2ps 'Anything to PostScript' converter
ii  aalib1         1.4p5-10       ascii art library
ii  abcde          2.0.3-1        A Better CD Encoder
...
ii  zlib1g-dev     1.1.3-19       compression library - development
```

Первые три строки объясняют значение первых трех колонок перед именем каждого пакета. Как правило, в них должно стоять *ii*, что соответствует корректной установке пакета. Если там что-то другое, следует выполнить команду *dpkg -audit*, которая объяснит, какая неполадка обнаружена в системе и как ее исправить.

Параметр *-l* можно использовать с именем пакета или шаблоном имени в стиле *glob*. Узнаем, например, какая версия *superfrob* установлена:

```
tigger $ dpkg -l superfrob
Desired=Unknown/Install/Remove/Purge/Hold
| Status=Not/Installed/Config-files/Unpacked/Failed-config/Half-installed
|/ Err?=(none)/Hold/Reinst-required/X=both-problems (Status,Err: uppercase=bad)
||/ Name           Version        Description
+++-==============-==============- ============================================
ii  superfrob      4-1            The superfrobulator
```

С помощью *dpkg* можно узнать, какому пакету принадлежит конкретный файл:

```
tigger $ dpkg --search /bin/false
shellutils: /bin/false
tigger $ dpkg --search /home/kalle/.xinitrc
dpkg: /home/kalle/.xinitrc not found.
```

Можно также показать информацию об установленном пакете или архиве *.deb*:

```
tigger $ dpkg --status dpkg
Package: dpkg
```

```
 Essential: yes
 Status: install ok installed
 Priority: required
 Section: base
 Installed-Size: 3156
 Origin: debian
 Maintainer: Dpkg Development <debian-dpkg@lists.debian.org>
 Bugs: debbugs://bugs.debian.org
 Version: 1.9.19
 Replaces: dpkg-doc-ja
 Pre-Depends: libc6 (>= 2.2.4-4), libncurses5 (>= 5.2.20020112a-1), libstdc++2.10-
 glibc2.2 (>= 1:2.95.4- 0.010810)
 Conflicts: sysvinit (<< 2.80)
 Conffiles:
  /etc/alternatives/README 69c4ba7f08363e998e0f2e244a04f881
  /etc/dpkg/dpkg.cfg 1db461ac9a1d4f4c8b47f5061078f5ee
  /etc/dpkg/dselect.cfg 190f7cf843556324495ef12759b752e3
  /etc/dpkg/origins/debian 24926c0576edec3e316fd9f6072b8118
 Description: Package maintenance system for Debian
  This package contains the programs which handle the installation and
  removal of packages on your system.
  .
  The primary interface for the dpkg suite is the 'dselect' program;
  a more low-level and less user-friendly interface is available in
  the form of the 'dpkg' command.
  .
  In order to unpack and build Debian source packages you will need to
  install the developers' package 'dpkg-dev' as well as this one.
 tigger $ dpkg --info reportbug_1.43_all.deb
  new debian package, version 2.0.
  size 66008 bytes: control archive= 1893 bytes.
       40 bytes,     2 lines      conffiles
     1000 bytes,    24 lines      control
      986 bytes,    15 lines      md5sums
     1014 bytes,    41 lines   *  postinst             #!/bin/sh
      147 bytes,     5 lines   *  postrm               #!/bin/sh
      416 bytes,    19 lines   *  prerm                #!/bin/sh
  Package: reportbug
  Version: 1.43
  Section: utils
  Priority: standard
  Architecture: all
  Depends: python
  Recommends: python-newt
  Suggests: postfix | mail-transport-agent, gnupg | pgp, python-ldap (>= 1.8-1)
  Conflicts: python (>> 2.3), python-newt (= 0.50.17-7.1)
  Installed-Size: 195
  Maintainer: Chris Lawrence <lawrencc@debian.org>
  Description: Reports bugs in the Debian distribution.
   reportbug is a tool designed to make the reporting of bugs in Debian
   and derived distributions relatively painless.  Its features include:
```

```
 .
  * Integration with the mutt, af, and mh/nmh mail readers.
  * Access to outstanding bug reports to make it easier to identify
    whether problems have already been reported.
  * Support for following-up on outstanding reports.
  * Optional PGP/GnuPG integration.
 .
 reportbug is designed to be used on systems with an installed mail
 transport agent, like exim or sendmail; however, you can edit the
 configuration file and send reports using any available mail server.
```

С помощью *dpkg* можно также вывести список файлов и каталогов, включенных в архив *.deb*:

```
tigger $ dpkg --contents superfrob_4-1_i386.deb
-rwxr-xr-x root/root     44951 2002-02-10 12:16:48 ./usr/bin/dothefrob
-rwxr-xr-x root/root     10262 2002-02-10 12:16:48 ./usr/bin/frobhelper
...
```

dpkg, как и *rpm*, обладает многими другими параметрами. Подробности см. на страницах руководства для *dpkg* и *dpkg-deb*.

Помимо *dpkg* дистрибутив Debian и другие, основанные на нем, предлагают комплект программ *apt*.[1] APT (сокращение от «Advanced Package Tool») разрабатывался как независимая от способа архивирования система, способная работать с несколькими форматами пакетов. Одной из важнейших особенностей APT является его способность автоматически разрешать зависимости. Например, если *superfrob* требует вторую или более позднюю версию *frobnik*, APT постарается найти *frobnik* в доступных ему источниках (включая CD-ROM, локальные зеркала и Интернет).

Самым полезным интерфейсом к APT служит команда *apt-get*. Она управляет списком доступных пакетов («кэшем пакетов») и может быть использована для разрешения зависимостей и установки пакетов. Типичный сеанс работы с ней начинается с обновления кэша APT:

```
tigger # apt-get update
Get:1 http://http.us.debian.org stable/main Packages [808kB]
Get:2 http://http.us.debian.org stable/main Release [88B]
Hit http://non-us.debian.org stable/non-US/main Packages
Hit http://non-us.debian.org stable/non-US/main Release
Get:3 http://security.debian.org stable/updates/main Packages [62.1kB]
Get:4 http://security.debian.org stable/updates/main Release [93B]
Fetched 870kB in 23s (37kB/s)
Reading Package Lists... Done
Building Dependency Tree... Done
```

Этот вывод показывает, что имеются обновления для стабильной версии дистрибутива, поэтому можно обновить пакеты, которые уже установлены в

[1] В некоторые дистрибутивы на базе RPM теперь включается APT, поскольку APT предназначен для работы с любым форматом пакетов.

системе. Чтобы осуществить такое обновление автоматически, можно воспользоваться параметром *upgrade*:

```
tigger # apt-get upgrade
The following packages have been kept back:
  gnumeric
17 packages upgraded, 0 newly installed, 0 to remove and 1 not upgraded.
Need to get 16.3MB of archives.  After unpacking 5kB will be freed.
Do you want to continue? [Y/n] y
Get:1 http://http.us.debian.org stable/main base-passwd 3.4.6 [17.2kB]
Get:2 http://security.debian.org stable/updates/main ssh 1:3.1.6p4-1 [600kB]
...
(Reading database ... 159546 files and directories currently installed.)
Preparing to replace ssh 1:3.0.3p2-6 (using .../ssh_1%3a3.1.6p4-1_i386.deb) ...
Unpacking replacement ssh ...
...
```

Как можно заметить, в отличие от большинства команд Linux, для команд APT действия задаются *без* черточек. *apt-get* использует некоторые параметры, но они служат только для модификации поведения основной заданной операции.[1]

Обратите внимание, что *gnumeric* не был обновлен автоматически, вероятно потому, что это потребовало бы установки дополнительных пакетов. Для того чтобы обновить его и разрешить зависимости, можно воспользоваться параметром *install* и указать имя одного или нескольких пакетов:[2]

```
tigger # apt-get install gnumeric
The following extra packages will be installed:
  libgal36 libglade3
The following NEW packages will be installed:
  libgal36
2 packages upgraded, 1 newly installed, 0 to remove and 0 not upgraded.
Need to get 8.3MB of archives.  After unpacking 503kB will be used.
Do you want to continue? [Y/n] y
...
```

Еще одна удобная функция APT – нахождение информации о пакетах в хранилище. С помощью команды *apt-cache* осуществляется поиск сведений о доступных для установки пакетах. Часто с помощью *apt-cache* ищут пакеты по ключевым словам в описании пакета, используя слова, предложения (заключенные в кавычки) или регулярные выражения. Например, если нужно найти пакет, способный воспроизводить музыкальные файлы в кодировке Ogg Vorbis, можно воспользоваться параметром *search*:

---

[1] Есть и другие команды Linux, действующие так же, например *cvs*.

[2] Обратите внимание, что *apt-get* не устанавливает пакеты непосредственно из архивов *.deb*. Для установки архива *.deb* на компакт-диске или непосредственно загруженного из Интернета нужно использовать *dpkg* с параметром *--install*. При работе с *dpkg* все зависимости придется разрешать самостоятельно.

```
tigger $ apt-cache search "ogg vorbis"
audacity - A fast, cross-platform audio editor
bitcollider-plugins - bitcollider plugins
cplay - A front-end for various audio players
gqmpeg - a GTK+ front end to mpg321/mpg123 and ogg123
libapache-mod-mp3 - turns Apache into a streaming audio server
libvorbis0 - The Vorbis General Audio Compression Codec
mp3blaster - Full-screen console mp3 and ogg vorbis player
mp3burn - burn audio CDs directly from MP3s or Ogg Vorbis files
oggtst - Read comments in ogg vorbis files
python-pyvorbis - A Python interface to the Ogg Vorbis library
vorbis-tools - Several Ogg Vorbis Tools
xmms - Versatile X audio player that looks like Winamp
xmms-dev - XMMS development static library and header files
mq3 - a mp3/ogg audio player written in Qt.
```

Если вас заинтересовал какой-либо из этих пакетов, можно узнать о нем поподробнее, воспользовавшись параметром *show* команды *apt-cache*:

```
tigger $ apt-cache show xmms
Package: xmms
Priority: optional
Section: sound
Installed-Size: 4035
Maintainer: Josip Rodin <jrodin@jagor.srce.hr>
...
Description: Versatile X audio player that looks like Winamp
 XMMS (formerly known as X11Amp) is an X/GTK+ based audio player
 for various audio formats.
 .
 It's able to read and play:
  * Audio MPEG layer 1, 2, and 3 (with mpg123 plug-in),
  * WAV, RAW, AU (with internal wav plug-in and MikMod plug-in),
  * MOD, XM, S3M, and other module formats (with MikMod plug-in),
  * CD Audio (with CDAudio plug-in), with CDDB support,
  * .cin files, id Software,
  * Ogg Vorbis files.
 It has eSound, OSS, and disk writer support for outputting sound.
 .
 It looks almost the same as famous Winamp, and includes those neat
 features like general purpose, visualization and effect plug-ins,
 several of which come bundled, then spectrum analyzer, oscilloscope,
 skins support, and of course, a playlist window.
```

Рассмотрение всех функций APT выходит за рамки этой главы. Ответы на возникшие вопросы ищите на странице руководства для *apt* (и тех, на которые она ссылается), а также в APT HOWTO (из пакета *apt-howto-en*).

В дополнение к утилитам командной строки разработан ряд удобных в работе текстовых и графических интерфейсов. К числу наиболее развитых из них принадлежит *kpackage*, входящий в состав графической среды KDE, но допускающий использование в других средах, например в GNOME. *kpackage*

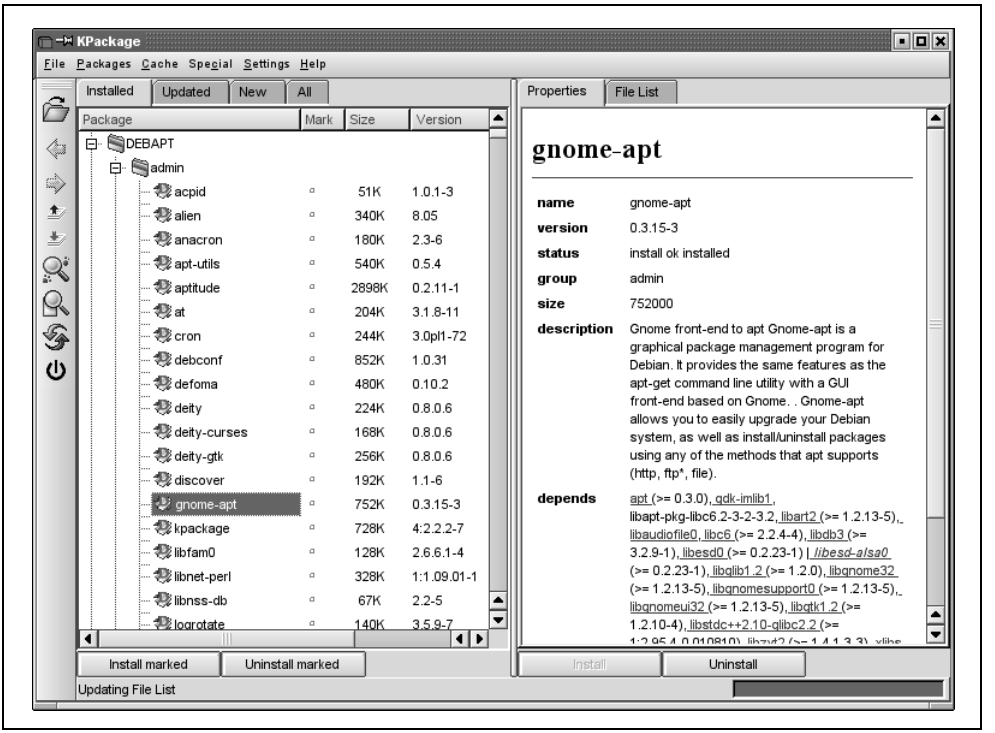


Рис. 7.1. *Менеджер пакетов kpackage*

можно запустить из командной строки или через меню System KDE. Рисунок 7.1 демонстрирует пример экрана, выводимого *kpackage*.

Главное окно *kpackage* показывает в левой своей части список всех доступных для системы пакетов, а в окне, находящемся в правой части, выводится информация о пакете, который выбран из списка в левой части. Начать установку или удаление пакета можно, выбрав пункт *Install* или *Uninstall* в меню *Packages* либо пометив пакет щелчком на поле *Mark* и нажав кнопку *Install marked* или *Uninstall marked*. Пакеты *.deb* можно устанавливать непосредственно, если щелкнуть по кнопке *Open* на инструментальной панели в левой части экрана и выбрать файл, или перетаскивая значки *.deb* из окна файлового менеджера KDE в окно *kpackage*. В *kpackage* также есть средства для поиска пакетов с определенными именами. Как и во всех приложениях KDE, в *kpackage* можно получить подсказку, нажав <F1> или выбрав пункт меню *Help*.

Обновление другого программного обеспечения

Для того чтобы обновить другие приложения, необходимо получить их последние версии. Обычно их можно получить в виде *tar*-архива, сжатого с помощью *gzip* или *compress*. Такой пакет может быть в нескольких формах.

Чаще всего, это *двоичные дистрибутивы*, в которых двоичные файлы и другие связанные с ними файлы заархивированы и готовы к распаковке на машине. А также *дистрибутивы исходного кода*, когда предоставляется исходный код программного обеспечения или его части, и чтобы скомпилировать его и установить на машине, необходимо ввести некоторые команды.

Библиотеки совместного доступа облегчают распространение программного обеспечения в двоичной форме. Если версии установленных у вас библиотек совместимы с заглушками, использованными при сборке программы, то все в порядке. Однако во многих случаях проще (и правильнее) распространять программу в виде исходного кода. Это позволяет не только изучать код и развивать его дальше, но и собрать приложение специально для своей системы, причем с собственными библиотеками. Во многих программах можно задать определенные параметры компиляции, например включить в собираемую программу только необходимые вам функции. Такого рода настройка невозможна, если вы получаете готовые двоичные файлы.

При установке двоичных файлов без исходного кода возникают также проблемы безопасности. Хотя в Unix-системах о вирусах почти не слышали,[1] не столь сложно создать «троянского коня» – программу, которая по видимости делает что-то полезное, а на самом деле наносит ущерб системе. Например, некто мог бы написать программу, имеющую такую «функцию», как уничтожение всех файлов в исходном каталоге пользователя, запустившего ее. Поскольку такая программа выполнялась бы с правами доступа запустившего ее пользователя, ничто не помешало бы ей произвести такое разрушительное действие. (Конечно, система защиты Unix не позволит повредить файлы других пользователей или какие-либо важные системные файлы, владельцем которых является `root`.)

Хотя наличие исходного кода не обязательно предотвращает такого рода ситуацию (не будете же вы читать исходный код каждой программы, которую компилируете на своей машине?), оно, по крайней мере, предоставляет возможность проверить, чем же в действительности занимается программа. Помимо того, если доступен исходный код, весьма вероятно, что найдутся люди, которые его изучат, поэтому пользоваться исходным кодом несколько безопаснее, но полагаться на это нельзя.

---

[1] В классическом случае «вирус» – это программа, прикрепляющаяся к «хозяину» и выполняемая при его запуске. В Unix-системах обычно нужны права суперпользователя, для того чтобы навредить, и если кто-то завладевает его привилегиями, то, вероятно, нет нужды утруждать себя созданием вирусов.

Конечно, можно снабжать пакеты цифровой подписью, чтобы проверять происхождение пакета и самостоятельно решать, заслуживает ли источник доверия.

Некоторые фирмы предлагают средства обнаружения вирусов для Linux. Однако обычно такие средства предназначены для почтовых серверов, работающих под Linux, и сканируют не локальные жесткие диски, а почту, хранящуюся на машине, перед доставкой клиентам, которые могут работать под Windows. Таким образом, эти сканеры ищут Windows-вирусы, а не Linux-вирусы.

В любом случае с дистрибутивами исходного кода или двоичных файлов обращаться очень просто. Если пакет имеет вид *tar*-архива, сначала определите вид архивации с помощью команды *tar t*. Если это двоичный дистрибутив, обычно его можно распаковать прямо в систему из корневого каталога / или каталога */usr*. При этом постарайтесь удалить прежние версии программы и сопровождающие ее файлы (которые не переписываются новым архивом). Если прежняя программа указана в пути поиска раньше, чем новая, у вас будет запускаться старая версия, пока вы ее не удалите.

С дистрибутивами исходного кода несколько сложнее. Прежде всего, нужно распаковать исходные файлы в отдельный каталог. В большинстве систем */usr/src* используется как раз для этой цели. Поскольку, как правило, не требуются привилегии суперпользователя для того, чтобы собрать пакет (такие права обычно нужны для установки программы после компиляции!), можно рекомендовать разрешить запись в */usr/src* всем пользователям, для чего надо выполнить команду

```
chmod 1777 /usr/src
```

После ее выполнения любой пользователь может создавать подкаталоги в */usr/src* и помещать в них файлы. Первая 1 в аргументе команды – это «липкий» (sticky) бит, предотвращающий удаление одним пользователем каталога, созданного другим.

Затем можно создать подкаталог в */usr/src* и распаковать в него tar-файл либо распаковать его прямо в */usr/src*, если в архиве указаны собственные подкаталоги.

Теперь, когда имеются исходные тексты, нужно прочесть все имеющиеся среди них файлы *README* и *INSTALL* или замечания по установке, содержащиеся среди исходного кода. Такие документы почти всегда присутствуют в архивах. Основная методика сборки большинства программ такова:

1. Проверьте *Makefile*. В нем содержатся команды для утилиты *make*, которая управляет процессом создания программ компилятором. Во многих случаях требуется внести в *Makefile* мелкие изменения, чтобы настроить его на вашу систему; обычно они очевидны. В замечаниях по установке будет сказано, есть ли в этом необходимость. Если вам нужны дополнительные сведения о *Makefile*, читайте раздел «Makefile» главы 13. Если в пакете нет *Makefile*, то, возможно, вам придется его создать. Как это сделать, сказано в пункте 3.
2. Возможно, нужно отредактировать другие файлы, используемые программой. В некоторых приложениях требуется отредактировать файл с именем *config.h*. О том, как это сделать, тоже должно быть рассказано в инструкции по установке.
3. Возможно, нужно запустить сценарий конфигурирования. Такой сценарий используют для определения возможностей машины, что необходимо для сборки более сложных приложений.

В частности, если в каталоге верхнего уровня исходного кода нет файла *Makefile*, но вместо него есть файлы *Makefile.in* и *configure*, то пакет создан с помощью системы Autoconf. В таком случае (а это происходит все чаще) нужно выполнить сценарий конфигурирования:

```
./configure
```

Точка и косая черта (./) используются для того, чтобы выполнить *configure* из текущего каталога, а не из какого-либо другого каталога из вашего пути поиска, в котором случайно может оказаться другая программа *configure*. В некоторых пакетах можно передавать *configure* параметры, которыми подключаются или отключаются отдельные функции пакета (узнать, каковы эти параметры, можно с помощью команды *configure --help*). Когда сценарий *configure* отработает, можно переходить к следующему шагу.

4. Запустите *make*. Обычно при этом выполняются соответствующие команды компиляции, указанные в *Makefile*. Часто требуется указать *make* «цель», например *make all* или *make install*. Это часто встречающиеся цели. Первая обычно не нужна, но таким образом можно собрать все целевые программы, перечисленные в *Makefile* (например, если в пакет входит несколько программ, но по умолчанию компилируется только одна). Вторая часто используется для установки в системе исполняемых модулей и сопровождающих файлов после компиляции. По этой причине *make install* обычно выполняется пользователем `root`.

При компиляции или установке в системе нового программного обеспечения могут возникнуть проблемы, особенно если эта программа не проверялась под Linux или зависит от других программ, которые у вас не установлены. В главе 13 мы подробно расскажем о компиляторе, *make* и сопутствующих инструментальных средствах.

Большинство пакетов помимо исходного кода и исполняемых модулей включают страницы руководства и прочие файлы. Установочный сценарий (если таковой имеется) поместит эти файлы в нужное место. Что касается страниц руководства, то вы обнаружите файлы с такими именами, как *foobar.1* или *foobar.man*. Обычно это исходные тексты *nroff*, отформатированные таким образом, чтобы их можно было читать командой *man*. Если файл страниц руководства имеет числовое расширение, например *.1*, скопируйте его в каталог */usr/man/man1*, где *1* – это то число, которое используется в расширении имени файла. (Оно соответствует номеру раздела руководства; чаще всего это 1.) Если файл имеет расширение типа *.man*, то обычно достаточно скопировать его в */usr/man/man1*, изменив расширение *.man* на *.1*.

Сборка нового ядра

Может показаться, что перекомпиляция ядра – занятие для хакеров, но на самом деле это умение необходимо любому системному администратору. Во-первых, заново компилировать ядро необходимо для того, чтобы избавиться от ненужных драйверов устройств. Благодаря этому сокращается объем па-

мяти, используемой ядром, о чем говорилось в разделе «Управление пространством для свопинга» главы 6. Ядро всегда присутствует в памяти, и занятую им память не могут использовать другие программы, когда им это требуется.

Кроме того, время от времени необходимо обновлять ядро, чтобы иметь более новую его версию. Как и в отношении другого программного обеспечения, когда становится известно об исправлении важных ошибок или добавлении новых функций в новую версию ядра, может потребоваться осуществить обновление, чтобы использовать их. Тем, кто активно разрабатывает код ядра, также требуется поддерживать ядро в текущем состоянии, если изменения в нем затрагивают тот код, над которым они работают. Иногда необходимо обновить ядро, чтобы иметь возможность использовать новые версии компилятора или библиотек. Некоторые приложения (например, X Window System) могут работать только с определенной версией ядра.

Выяснить версию работающего ядра можно с помощью команды *uname -a*. Она производит примерно такой вывод:

```
rutabaga% uname -a
Linux owl 2.4.19-64GB-SMP #2 SMP Fri Aug 9 21:46:03 CEST 2002 i686 unknown
```

Мы видим, что на машине используется ядро версии 2.4.19 (в конфигурации для нескольких процессоров [SMP] с максимальным размером оперативной памяти 64 Гбайт), скомпилированное 9 августа. Есть и другие данные, например имя хоста машины, сколько раз компилировалось ядро (два) и то, что это машина Pentium Pro или лучше (на что указывает `i686`). На странице руководства по *uname*(1) можно получить дополнительные сведения.

Ядро Linux – это чудовище со многими щупальцами. Над различными его частями работает много групп людей, и некоторые части кода представляют собой пестрое собрание идей, связанных с различными целями проектирования. Однако в целом код ядра ясный и единообразный, и для тех, кто захочет изучить его тайны, это не составит особого труда. Однако, поскольку ведется очень интенсивная разработка ядра, новые выпуски появляются очень часто, иногда ежедневно! Главная причина этого в том, что почти все драйверы устройств содержатся в ядре, и когда кто-либо обновляет драйвер, необходим выпуск новой версии. Когда сообщество Linux перейдет на загружаемые драйверы устройств, те, кто сопровождает эти драйверы, смогут выпускать их независимо от основного ядра, что уменьшит необходимость в таких частых обновлениях.

В настоящее время «официальную» версию ядра поддерживает Линус Торвальдс. Хотя лицензия GPL позволяет всем модифицировать и заново выпускать ядро под тем же копирайтом, поддержка Линусом официального ядра является удобной договоренностью, помогающей сохранять единообразную нумерацию версий и добиваться единого понимания того, о какой версии ядра идет речь. Чтобы исправить ошибку или добавить новую функцию, нужно лишь отправить исправление Линусу (или тому, кто отвечает за соответствующую версию ядра, потому что сам Линус всегда работает с са-

мой свежей версией ядра), и обычно он включает предлагаемые изменения, если они не конфликтуют со всем остальным.

Номера версий ядра имеют следующий формат:

```
major.minor.patchlevel
```

major является основным номером версии, который редко изменяется, *minor* является дополнительным номером версии, указывающим на текущий «штамм» версии ядра, а *patchlevel* является номером «патча» (заплатки) к текущей версии ядра. Примерами версий ядра являются 2.4.4 (патч уровня 4 ядра версии 2.4) и 2.5.1 (патч уровня 1 ядра версии 2.5).

По существующему соглашению четные номера версий (2.2, 2.4 и т. д.) являются «стабильными» версиями, патчи к которым содержат только исправления ошибок и не содержат никаких новых функций. Нечетные номера версий (2.3, 2.5 и т. д.) являются «разрабатываемыми» версиями, патчи к которым могут содержать любой новый код, который хотят добавить разработчики, и исправления к этому коду. Когда разрабатываемая версия созреет настолько, что становится достаточно стабильной для широкого использования, она переименовывается, получая очередной, более высокий четный дополнительный номер версии, и цикл разработки повторяется заново.

Например, работа над версиями ядра 2.2 и 2.3 шла одновременно. Патчи к 2.2 были исправлениями ошибок, предназначенными только для устранения проблем в существующем коде. Патчи к 2.3 включали в себя как исправления ошибок, так и много нового кода – новые драйверы устройств, новые функции и т. д. Когда ядро версии 2.3 стало достаточно стабильным, оно было переименовано в 2.4. Копия версии 2.4 была переименована в версию 2.5. Разработка была продолжена с версиями 2.4 и 2.5. Версия 2.4 стала новым «стабильным» ядром, а версия 2.5 стала разрабатываемым ядром для новых функций.[1]

Заметьте, что это соглашение по нумерации версий относится только к официальным выпускам ядра, осуществляемым Линусом, и только к версиям выше 1.0. До версии 1.0 (теперь это далекая история) существовала только одна «текущая» версия ядра, к которой применялись все патчи. Сообщество разработчиков ядра сочло, что наличие двух одновременных версий ядра позволяет тем, кто готов экспериментировать, использовать разрабатываемую версию, а тем, кому нужна надежная платформа, придерживаться стабильного ядра. Благодаря этому, если новый код существенно нарушает работу разрабатываемого ядра, то это не влияет на работу тех, у кого используется последнее стабильное ядро. По общему правилу следует использовать разрабатываемое ядро, если вы хотите быть на переднем крае, где применяются новые функции, и готовы использовать их на свой страх и риск.

---

[1] В действительности первые версии ядра серии 2.4 были не так стабильны, как предполагает номер, поэтому многие пользователи некоторое время продолжали работать на версии 2.2. К настоящему времени ядро 2.4 можно считать очень стабильным, если не считать версии, предшествующие 2.4.16.

Ознакомиться с историей развития существующих версий ядра можно на *http://www.kernel.org*.

В вашей системе исходный код ядра находится, скорее всего, в */usr/src/linux* (если только у вас не дистрибутив Debian, в котором исходный код ядра находится в каталоге */usr/src/kernel-source-versionsnumber*). Если вы собираетесь заново собрать ядро, используя лишь имеющийся у вас исходный код, нет необходимости получать новые файлы или патчи. Если вы хотите обновить ядро, установив его новейшую версию, вам следует воспользоваться указаниями из следующего раздела.

Получение исходного кода ядра

Официальная версия ядра выпускается в виде сжатого утилитой *gzip* архива *tar*, в котором содержатся исходный код и ряд патчей – по одному файлу для каждого уровня. Архив *tar* содержит исходный код версии без патчей. Например, есть *tar*-файл, содержащий исходный код ядра версии 2.4, в который не внесены заплатки. Каждый последующий уровень патчей выпускается как патч-файл (созданный с помощью программы *diff*), который можно применить, используя программу *patch*. В разделе «Применение патч-файлов» главы 14 мы подробно опишем, как использовать патчи.

Допустим, вы собираетесь обновить ядро до версии 2.4 и уровня патчей 4. Вам потребуется исходный код для версии 2.4 (файл может называться *v2.4.0.tar.gz*) и патчи для уровней с 1 по 4. Эти файлы будут называться *patch1*, *patch2* и т. д. (Вам потребуются *все* эти файлы патчей до того номера версии, который вы хотите в итоге установить. Обычно эти файлы весьма невелики и находятся на архивных сайтах в сжатом виде.) Все файлы можно найти в каталоге *kernel* архивных FTP-сайтов Linux. Например, на *ftp://ftp.kernel.org* каталог, в котором находятся исходный код и патчи для версии 2.4, называется */pub/linux/kernel/v2.4*. Исходный код находится в них в виде *tar*-архивов, сжатых *gzip* и *bzip2*.

Если у вас уже установлен какой-то уровень патчей ядра (например, версия 2.4 уровень патчей 2), и вы хотите обновить ядро до нового уровня патчей, вы можете просто применить патчи, следующие за версией, которая у вас уже есть, до того уровня, который вам требуется. Если вы обновляете ядро, например, с версии 2.4 уровень 2 до версии 2.4 уровень 4, вам понадобятся патч-файлы для 2.4.3 и 2.4.4.

Распаковка исходного кода

Прежде всего, нужно распаковать архив с исходным кодом из каталога */usr/src*. Для этого нужно выполнить такие команды:

```
rutabaga# cd /usr/src
rutabaga# mv linux linux.old
rutabaga# tar xzf v2.4.0.tar.gz
```

В результате ваше прежнее дерево каталогов с исходным кодом ядра будет сохранено как */usr/src/linux.old*, и будет создано дерево */usr/src/linux*, со-

держащее новый исходный код. Обратите внимание, что *tar*-файл с исходным кодом содержит подкаталог *linux*.

Исходный код текущей версии ядра нужно хранить в каталоге */usr/src/linux*, поскольку есть две символические ссылки – */usr/include/linux* и */usr/include/asm*, указывающие на дерево с исходным кодом текущего ядра, которые обеспечивают доступ к некоторым заголовочным файлам при компиляции программ. (Исходный код ядра всегда должен быть доступен, чтобы можно было использовать include-файлы при компиляции программ.) Если вы хотите сохранить несколько деревьев с исходным кодом ядра, следите за тем, чтобы */usr/src/linux* указывал на самое новое из них.

Применение патчей

Для применения патч-файлов используется программа *patch*. Допустим, имеются сжатые файлы от *patch1.gz* до *patch4.gz*. Они должны применяться из главного каталога исходного кода ядра. Это означает не то, что сами они должны там находиться, но что *patch* должна запускаться из этого каталога, например */usr/src/linux*. Для каждого патч-файла выполните команду

```
gunzip -c patchfile | patch -p1
```

из */usr/src*. Параметр *-p1* указывает *patch*, что она не должна отрезать какую-либо часть имени у файлов, находящихся в патч-файле, за исключением первой.

Все патчи должны применяться в порядке нумерации их уровня, что очень важно. Учтите, что использование символов-заместителей типа *patch** недопустимо, поскольку * использует порядок ASCII, а не числовой. (В противном случае за *patch1* будут следовать *patch10* и *patch11*, а не *patch2* и *patch3*.) Лучше всего вручную выполнить указанную команду последовательно для каждого патч-файла, тогда вы будете уверены в том, что делаете это в правильном порядке.

Проблем при применении патчей к дереву исходного кода возникать не должно, если делать это по порядку и не пытаться дважды применить один и тот же патч. При возникновении проблем посмотрите страницу руководства по *patch*. Если ничто другое не помогает, удалите новое дерево исходного кода и повторите все сначала из исходного *tar*-файла.

Чтобы еще раз убедиться в том, что все патчи применены успешно, выполните команды:

```
find /usr/src/linux -follow -name "*.rej" -print
find /usr/src/linux -follow -name "*#" -print
```

В результате будут выведены файлы, «отвергнутые» в процессе применения патчей. Если такие файлы существуют, в них будут содержаться те части патч-файлов, которые по какой-либо причине не смогли быть применены. Просмотрите их и при наличии каких-либо сомнений начните все сначала. Нельзя надеяться на то, что ядро правильно скомпилируется или будет работать, если процесс установки патчей прошел неудачно и что-либо было отвергнуто.

Есть удобный сценарий для применения патчей к ядру, который можно найти в *scripts/patch-kernel*. Но, как и в других подобных случаях, следует хорошо представлять себе смысл процедуры, осуществляемой с помощью средств автоматизации, в особенности когда она касается самой основы операционной системы – ядра.

Сборка ядра

Сборка ядра производится в шесть этапов, которые должны пройти без труда. На следующих страницах все эти этапы описываются подробно.

1. Установите нужные версии всех необходимых инструментов и утилит. Перечень требований находится в подкаталоге *Documentation/Changes* каталога исходного кода ядра.
2. Выполните *make config*, при этом вам будут заданы различные вопросы относительно того, какие драйверы вы собираетесь включить в ядро. Можно также воспользоваться более удобными вариантами *make menuconfig* или (только если у вас запущена X Window System) *make xconfig*.

   Если вы уже выполняли ранее компиляцию ядра, а потом применяли патчи к новой версии, можно запустить *make oldconfig*, чтобы использовать старый файл настройки, но получить приглашение для ввода новых параметров, отсутствовавших в прежнем ядре.
3. Выполните *make dep*, чтобы собрать сведения о зависимостях для каждого исходного файла и включить их в различные make-файлы.
4. Если раньше вы уже собирали ядро из этого дерева исходного кода, выполните *make clean*, чтобы удалить прежние объектные файлы и инициировать полную повторную сборку ядра.
5. Выполните *make zImage*, чтобы построить само ядро.
6. Сходите выпить чашечку кофе (или пару чашечек, в зависимости от скорости работы вашей машины и объема оперативной памяти).
7. Установите новый образ ядра на загрузочную дискету или с помощью LILO. Поместить ядро на загрузочную дискету можно с помощью *make bzDisk*.

Все эти команды нужно выполнять из */usr/src/linux*, за исключением шага 6, который можно выполнить в любом месте.

Среди исходного кода ядра находится файл *README*, который должен располагаться в вашей системе в */usr/src/linux/README*. Прочтите его. В нем содержатся самые последние сведения о компиляции ядра, которые могут быть более свежими, чем те, которые здесь изложены. Следуйте описанным там шагам, используя в качестве справочника описания, приведенные далее в этом разделе.

На первом этапе нужно выполнить *make config*. При этом запускается сценарий, задающий ряд вопросов относительно того, какие драйверы должны быть включены в ядро, на которые нужно отвечать «yes» или «no». Для каждого вопроса установлен ответ по умолчанию, но будьте внимательны: ответы

по умолчанию могут не соответствовать тому, что вам требуется. (Если имеется несколько вариантов ответа, вариант по умолчанию будет показан заглавной буквой, как в [Y/n].) Ваши ответы станут ответами по умолчанию, когда вы в следующий раз будете собирать ядро из этого дерева исходного кода.

Отвечайте на каждый вопрос, нажимая <Enter> при выборе ответа по умолчанию или вводя y или n, а затем нажимая <Enter>. Не все вопросы предполагают ответ типа yes/no, иногда вас могут попросить ввести число или какое-либо другое значение. Ряд вопросов конфигурирования допускает помимо y или n ответ m. Такой ответ позволяет скомпилировать соответствующую функцию ядра как загружаемый модуль ядра, а не встраивать ее непосредственно в образ ядра. Загружаемые модули, о которых говорится в следующем разделе «Загружаемые драйверы устройств», позволяют частям ядра (например, драйверам устройств) загружаться и выгружаться во время работы системы по мере необходимости. Если вы не уверены в том, какой ответ выбрать, введите в ответ на приглашение «?». Для большинства параметров будет выведено сообщение с некоторыми сведениями о конфигурируемом параметре.

Некоторые считают, что *make config* требует сейчас столько параметров, что работать с ней вручную стало слишком сложно, поскольку приходится тратить много усилий для выбора правильного ответа на задаваемые вопросы. По этой причине многие переходят на альтернативные варианты, описываемые ниже.

Альтернативой запуску *make config* служит *make xconfig*, которая компилирует и запускает программу конфигурирования ядра в системе X Window. Для использования этой возможности у вас должна быть запущена система X Window, кроме того, должны быть установлены соответствующие библиотеки X11 и Tcl/Tk и т. д. Вместо ответов на серию вопросов утилита конфигурирования, выполняемая в системе X, требует пометить флажками те параметры ядра, которые вы хотите включить. Система вспоминает прежние параметры конфигурации при каждом запуске *make config*, поэтому если вы добавляете к ядру или удаляете из него лишь несколько функций, нет необходимости вводить все остальные параметры.

Можно также применить *make menuconfig*, которая использует в текстовом режиме библиотеку *curses* и предоставляет аналогичную, основанную на меню возможность конфигурирования ядра, если у вас не установлена система X. Пользоваться *make menuconfig* и *make xconfig* значительно удобнее, чем *make config*, особенно потому, что можно вернуться к некоторому параметру и изменить его, прежде чем сохранить свою конфигурацию.

Ниже приведена часть сеанса работы с *make config*. При работе с *make menuconfig* или *make xconfig* вы столкнетесь с тем же набором параметров, только представленным в более дружественном пользователю виде (мы весьма рекомендуем по возможности пользоваться этими инструментами, поскольку в многочисленных параметрах конфигурации легко запутаться):

```
rm -f include asm
( cd include ; ln -sf asm-i386 asm)
/bin/sh scripts/Configure arch/i386/config.in
```

```
#
# Using defaults found in .config
#
*
* Code maturity level options
*
Prompt for development and/or incomplete code/drivers (CONFIG_EXPERIMENTAL) [Y/n/?]
*
* Loadable module support
*
Enable loadable module support (CONFIG_MODULES) [Y/n/?]
  Set version information on all module symbols (CONFIG_MODVERSIONS) [N/y/?]
  Kernel module loader (CONFIG_KMOD) [Y/n/?]
*
* Processor type and features
*
Processor family (386, 486, 586/K5/5x86/6x86/6x86MX, Pentium-Classic, ...
  defined CONFIG_MPENTIUMIII
Toshiba Laptop support (CONFIG_TOSHIBA) [N/y/m/?]
/dev/cpu/microcode - Intel IA32 CPU microcode support (CONFIG_MICROCODE) [M/n/y/?]
/dev/cpu/*/msr - Model-specific register support (CONFIG_X86_MSR) [M/n/y/?]
/dev/cpu/*/cpuid - CPU information support (CONFIG_X86_CPUID) [M/n/y/?]
High Memory Support (off, 4GB, 64GB) [4GB]
  defined CONFIG_HIGHMEM4G
Math emulation (CONFIG_MATH_EMULATION) [N/y/?]
MTRR (Memory Type Range Register) support (CONFIG_MTRR) [Y/n/?]
Symmetric multi-processing support (CONFIG_SMP) [Y/n/?]
*
* General setup
*
Networking support (CONFIG_NET) [Y/n/?]
...и т. д....
*** End of Linux kernel configuration.
*** Check the top-level Makefile for additional configuration.
*** Next, you may run 'make bzImage', 'make bzdisk', or 'make
install'.
```

Если вы сохранили информацию о том, какие аппаратные устройства установлены на вашем компьютере во время установки Linux, то ее может оказаться достаточно для того, чтобы ответить на вопросы относительно конфигурации, которые в большинстве своем должны быть просты. Если вам не знакома какая-то функция, значит, это специальная функция, которая вам не нужна. Дальше мы опишем вопросы, которые задаются при конфигурировании ядра версии 2.4.4. Если вы установили какие-либо дальнейшие патчи, могут появиться дополнительные вопросы. То же относится и к более новым версиям ядра. Учтите, что в приведенном ниже списке мы показали не все параметры конфигурации ядра, поскольку их просто слишком много и по большей части они очевидны. Мы выделили только те из них, для которых может потребоваться разъяснение. Если вы не уверены в выборе правильного ответа на какой-либо вопрос, лучше всего выбрать ответ по умол-

чанию. При возникновении сомнений можно также ввести «?» и прочесть сообщение подсказки.

Заметим, что не все драйверы устройств Linux реально встраиваются в ядро. Некоторые драйверы имеются только в виде загружаемых модулей, распространяемых отдельно от исходного кода ядра. (Как уже говорилось, некоторые драйверы можно либо встраивать в ядро, либо компилировать как модули.) Одним из известных драйверов, доступных только в виде модуля, является драйвер «флоппи-ленты» для ленточного механизма QIC-117, подключаемого к контроллеру гибких дисков.

Если вы не можете найти свое любимое устройство в списке, представленном *make config*, вполне возможно, что его драйвер существует в виде модуля или отдельного патча ядра. Поищите по сайтам FTP и архивным CD-ROM, если не видите того, что требуется. В разделе «Загружаемые драйверы устройств» далее в этой главе подробно описываются модули ядра.

`Prompt for development and/or incomplete code/drivers`

Ответьте «yes», если вы хотите использовать новые функции, которые пока считаются разработчиками недостаточно стабильными. Эта опция вам не нужна, если вы не хотите принять участие в тестировании новых функций.

`Processor family (386, 486, 586/K5/5x86/6x86/6x86MX, Pentium-Classic, Pentium-MMX, Pentium-Pro/Celeron/Pentium-II, Pentium-III/Celeron/Coppermine, Pentium-4, K6/K6-II/K6-III, Athlon/Duron/K7, Crusoe, Winchip-C6, Winchip-2, Winchip-2A/Winchip-3, CyrixIII/C3) [Pentium-III/Celeron/Coppermine]`

Здесь нужно указать тип установленного у вас центрального процессора. В результате ядро компилируется с оптимизацией, специально настроенной на вашу машину. Учтите, что если вы зададите более высокий тип процессора, чем тот, который у вас есть в действительности, ядро может оказаться неработоспособным. Кроме того, Pentium II MMX нужно задавать как 686, а не 586.

`Math emulation`

Ответьте «no», если у вас Pentium или лучший процессор. Ответьте «yes», если у вас нет арифметического сопроцессора. Это необходимо, чтобы ядро могло эмулировать наличие арифметического сопроцессора.

`Symmetric multi-processing support`

Включает поддержку ядром более одного процессора. Если у вас установлено несколько процессоров, ответьте «yes», в противном случае – «no».

`Enable loadable module support`

Включает поддержку динамически загружаемых дополнительных модулей. Вам наверняка нужно активировать ее.

`Set version information on all symbols for modules`

Специальный параметр, делающий возможным использование с одной версией ядра модулей, компилированных с другой версией ядра. С этим связан ряд проблем. Ответьте «no», если нет полной уверенности в том, что вы делаете.

`Kernel module loader`

При включении этого параметра ядро может автоматически загружать и выгружать динамически загружаемые модули по мере необходимости.

`Networking support`

Ответьте «yes», если требуется любая поддержка ядром работы в сети (в том числе TCP/IP, SLIP, PPP, NFS и т. д.).

`PCI support`

Включите эту опцию, если у вашей материнской платы есть шина PCI и в системе установлены устройства PCI. Для обнаружения и активации PCI-устройств используется PCI BIOS. Поддержка его ядром необходима для использования системой каких-либо PCI-устройств.

`System V IPC`

Ответ «yes» включает поддержку ядром функций взаимодействия между процессами (IPC) для System V, таких как *msgrcv* и *msgsnd*. Это требуется для некоторых программ, перенесенных с System V. Следует ответить «yes», если только у вас нет сильного отвращения к этим функциям.

`Sysctl support`

Эта опция указывает на то, что ядро должно позволять менять свои параметры на лету без перезагрузки. Полезно включить ее, если только вы не настолько ограничены в памяти, что не выдержите увеличения ядра на 8 Кбайт, вызываемого этим параметром.

`Parallel port support`

Включите эту опцию, если у вас есть параллельный порт и вам необходим доступ к нему из Linux. Linux может использовать параллельный порт не только для принтеров, но и для PLIP (сетевой протокол для параллельных линий), накопителей ZIP, сканеров и других устройств. В большинстве случаев для подключения устройства к параллельному порту нужен дополнительный драйвер.

`Normal floppy disk support`

Ответьте «yes», если только вы не желаете, чтобы гибкие диски не поддерживались (это поможет сберечь немного памяти в системах, где не нужна поддержка гибких дисков).

`Enhanced IDE/MFM/RLL disk/cdrom/tape/floppy support`

Отвечайте «yes», если вам нужна поддержка дисковых приводов MFM/RLL/IDE. После ответа «yes» вам будет предложено ввести типы устройств (жестких дисков, приводов CD-ROM, ленточных приводов и приводов гибких дисков), к которым вы хотите обращаться через драйвер IDE. Если устройств IDE у вас нет (только SCSI), можно смело отключить эту опцию.

`XT harddisk support`

Отвечайте «yes», только если у вас есть старый дисковый контроллер XT и вы планируете использовать его с Linux.

`Parallel port IDE device support`

Эта опция включает поддержку IDE-устройств, подключаемых к параллельному порту, таких как переносные приводы CD-ROM.

`Networking options`

Если ранее вы выбрали поддержку сети, вам будет задан ряд вопросов относительно того, какие сетевые возможности вы хотите видеть в ядре. Если только у вас нет особых требований к работе в сети (и тогда вы знаете правильные ответы на вопросы), достаточно ввести значения по умолчанию. Ряд вопросов понятен только посвященным (например, `IP: Disable Path MTU Discovery`), и почти всегда в качестве ответа на них следует выбирать значения по умолчанию.

`SCSI support`

Если у вас есть хотя бы один контроллер SCSI, отвечайте «yes». Вам будет задан ряд вопросов относительно конкретных SCSI-устройств, имеющихся в машине. Вам необходимо знать, какие аппаратные устройства у вас установлены. Все эти вопросы касаются конкретных типов чипов и плат SCSI-контроллеров, которые у вас установлены. Если вы не уверены в том, какой тип SCSI-контроллера есть у вас, посмотрите документацию по своим устройствам или загляните в документы Linux HOWTO.

Вам также будет задан вопрос о необходимости поддержки дисков, ленточных механизмов, приводов CD-ROM и других устройств SCSI. Задайте те параметры, которые соответствуют имеющимся у вас устройствам.

Если устройств SCSI у вас нет, нужно ответить «no», в результате чего размер ядра существенно уменьшится.

`Network device support`

Задается ряд вопросов о конкретных сетевых контроллерах, поддерживаемых Linux. Если вы собираетесь использовать карту Ethernet (или другой сетевой контроллер), включите те опции, которые соответствуют вашему устройству. Как и в случае устройств SCSI, следует обратиться к имеющимся для ваших устройств руководствам или к документам Linux HOWTO (например, Ethernet HOWTO), чтобы определить, какой драйвер подходит для вашего сетевого контроллера.

`Amateur Radio support`

Эта опция включает базовую поддержку сетевого взаимодействия через общедоступные радиочастоты. Если у вас есть соответствующее оборудование, включите эту опцию и прочтите AX25 и HAM HOWTO.

`ISDN subsystem`

Если у вас установлено оборудование ISDN, включите эту опцию и выберите соответствующий вашему устройству драйвер ISDN. Скорее всего, вам понадобится также выбрать опцию `Support synchronous PPP` (см. раздел «PPP поверх ISDN» главы 15).

`Old CD-ROM drivers`

Ряд вопросов относительно специфических драйверов CD-ROM, поддерживаемых ядром, таких как Sony CDU31A/33A, Mitsumi, SoundBlaster Pro и др. Если у вас есть контроллер CD-ROM типа SCSI или IDE (и вы уже выбрали его поддержку ранее), эти опции выбирать не нужно. У некоторых приводов CD-ROM есть специальные интерфейсные платы, и эти опции подключают драйверы для таких плат.

`Character devices`

Linux поддерживает несколько особых «символьных» устройств, таких как контроллеры последовательных и параллельных портов, ленточные приводы QIC-02 и мыши с особыми интерфейсами (не те, которые подключаются к последовательному порту, как мышь Microsoft). Сюда входит также поддержка джойстика и драйверы «Video for Linux», поддерживающие видеоаппаратуру и аппаратуру захвата кадров. Включите опции, соответствующие вашим устройствам.

`Filesystems`

Ряд вопросов по файловым системам, поддерживаемым ядром. Как было отмечено в разделе «Управление файловыми системами» главы 6, система поддерживает несколько типов файловых систем, и вы можете выбрать те, которые желаете включить в ядро. Практически всегда нужно включать поддержку второй расширенной файловой системы и файловой системы */proc*. Если вы хотите иметь непосредственный доступ из Linux к файлам MS-DOS, нужно включить файловую систему MS-DOS, а кроме того, файловую систему ISO 9660 для доступа к файлам на CD-ROM (чаще всего на CD-ROM используется эта система).

`Console drivers`

В этом разделе нужно выбрать хотя бы текстовую консоль VGA, иначе вы не сможете работать со своей системой с консоли.

`Sound card support`

Если вы ответите «yes», то последует ряд вопросов относительно вашей звуковой карты: какие драйверы вы хотите установить, IRQ и адреса, используемые звуковыми устройствами.

`Kernel hacking`

В этом разделе содержатся параметры, имеющие смысл, если вы собираетесь сами заняться разработкой ядра. Если не собираетесь, ответьте «no».

После выполнения *make config* или его эквивалента вам будет предложено отредактировать «make-файл верхнего уровня», под которым подразумевается */usr/src/linux/Makefile*. В большинстве случаев в этом нет необходимости. Если вы хотите изменить какие-либо параметры компиляции ядра, поменять корневое устройство, устанавливаемое по умолчанию, или режим SVGA, это можно сделать редактированием make-файла. Установить корневое устройство и режим SVGA всегда можно с помощью команды *rdev* после компиляции ядра, как было показано в разделе «Использование загрузочной дискеты» главы 5.

Если вы хотите, чтобы ядро было полностью перекомпилировано, то в этом месте нужно выполнить *make clean*. Эта команда удалит из дерева исходного кода все объектные файлы, оставшиеся от предыдущей компиляции. Если вы ни разу не собирали ядро из этого дерева, то вполне можете пропустить этот шаг, хотя он и не повредит. Если вы произвели в ядре небольшие изменения, то этот шаг можно пропустить, чтобы перекомпилировались только изменившиеся файлы. Запуск *make clean* всего лишь гарантирует, что все ядро будет перекомпилировано «с нуля», и если у вас есть какие-то сомнения, выполните для верности эту команду.

Теперь можно компилировать ядро. Это делается командой *make zImage*. Лучше всего собирать ядро, когда система не сильно загружена и много свободной памяти, которая может использоваться для компиляции. Если к машине подключены другие пользователи или вы сами запустили большие приложения, например X Window System или другой процесс компиляции, то сборка может идти очень медленно. Основным фактором является память. Если памяти мало и системе приходится осуществлять свопинг, компиляция происходит медленно, даже если процессор мощный.

Продолжительность компиляции ядра может быть самой различной – от нескольких минут до многих часов, в зависимости от того, какая у вас машина. Это не должно вызывать удивление, поскольку объем кода полного ядра довольно велик – более 10 Мбайт. На медленных машинах с 4 Мбайт памяти или менее для полной сборки ядра может потребоваться несколько часов. На более быстрых машинах с большим объемом памяти можно уложиться менее чем в полчаса.

Если во время компиляции появляются сообщения об ошибках или предупреждения, то не следует рассчитывать, что получившееся ядро будет нормально работать. Обычно сборка прекращается при возникновении ошибки. Ошибки могут быть связаны с неправильным применением патчей, проблемами на этапе *make config* или действительными ошибками в коде. В «покупном» ядре последнее случается редко, а вот если вы работаете с разрабатываемой версией или тестируете новые драйверы, такое может произойти. Если имеются сомнения, удалите все дерево исходного кода ядра и начните все сначала.

По завершении компиляции вы получите файл *bzImage* в каталоге */usr/src/linux/arch/i386/boot*. (Если вы собираете Linux на платформе, отличной от Intel x86, то образ ядра будет в другом соответствующем подкаталоге *arch*.) Название файла объясняется тем, что это исполняемый образ ядра, в котором используется сжатие с помощью алгоритма *bzip2*. При загрузке ядро само разархивирует себя в память. Не пытайтесь применять *bzip2* или *bunzip2* к *bzImage*! Благодаря такому сжатию ядро занимает значительно меньше дискового пространства и может поместиться на дискету. Раньше ядро поддерживало оба алгоритма сжатия – *gzip* и *bzip2*, причем первый создавал файл с именем *zImage*. Поскольку *bzImage* имеет большую степень сжатия, не следует пользоваться *gzip*, так как получающиеся файлы для ядра современных версий оказываются слишком велики, чтобы их можно было установить.

Если включить в ядро слишком много функций, то можно в конце компиляции получить ошибку `kernel too big` (слишком большое ядро). Это случается редко, поскольку одной машине требуется поддержка лишь ограниченного количества аппаратных средств, но все же может произойти. В такой ситуации остается только один выход: скомпилировать некоторые функции ядра в виде модулей (см. следующий раздел «Загружаемые драйверы устройств»).

Теперь нужно выполнить *rdev* с новым образом ядра, чтобы убедиться в правильной установке устройства корневой файловой системы, режима консоли SVGA и других параметров. Эта процедура описана в разделе «Использование загрузочной дискеты» главы 5.

Имея в руках новое ядро, можно настроить его начальную загрузку. Для этого нужно поместить образ ядра на дискету или сконфигурировать LILO для загрузки ядра с жесткого диска. Эти вопросы обсуждались в разделе «Начальная загрузка системы» главы 5. Для работы с новым ядром установите один из описанных способов его начальной загрузки и перезагрузите систему.

Предупредим, что всегда следует сохранять заведомо работоспособное ядро, которое можно загрузить. Либо обеспечьте возможность выбора сохраненного предшествующего ядра в LILO, либо проверяйте новое ядро, загружаясь сначала с дискеты. Это выручит вас в случае, если вы по ошибке не включите в новое ядро критически важный драйвер, сделав тем самым невозможной загрузку системы.

Загружаемые драйверы устройств

Обычно драйверы устройств включаются в состав ядра. Это делается по нескольким причинам. Во-первых, почти всем драйверам устройств требуется особый доступ к аппаратным устройствам, который обеспечивается, если драйвер является частью кода ядра. Такой доступ к устройствам нелегко получить пользовательской программе. Во-вторых, драйверы значительно легче реализуются при включении их в состав ядра: они получают полный доступ к структурам данных и другим находящимся в ядре функциям, к которым они могут свободно обращаться.

С таким конгломерированным ядром, содержащим в себе все драйверы, возникает ряд проблем. Во-первых, для избирательного включения драйверов устройств в ядро необходимо, чтобы администратор заново скомпилировал ядро, как было показано в предыдущем разделе. Во-вторых, при таком механизме разработчики драйверов могут неряшливо программировать: ничто не мешает программисту написать не вполне модулярный код, который, например, напрямую обращается к данным, принадлежащим другим частям ядра. Эта проблема осложняется кооперативным характером разработки ядра Linux, из-за чего не во всех частях кода соблюдается надлежащая сдержанность. Это осложняет сопровождение и отладку.

В попытке уйти от этой парадигмы в ядро Linux была включена поддержка загружаемых драйверов устройств – таких, которые с помощью нескольких команд можно поместить в память или удалить из нее во время работы системы. Такие драйверы продолжают оставаться частью ядра, но они компилируются отдельно и включаются только после загрузки. Загружаемые драйверы устройств, или *модули*, обычно загружаются в память с помощью команд, помещаемых в один из загрузочных *rc*-сценариев.

Модули обеспечивают более ясный интерфейс для написания драйверов. В известной мере они требуют модульности кода и следования определенным правилам программирования. (Это не мешает, впрочем, программисту нарушить соглашение и написать немодульный код. После того как модуль загружен, он может сеять беспорядок столь же свободно, как если бы компилировался непосредственно в ядре.) Благодаря использованию модулей становится легче отлаживать драйверы: можно выгрузить модуль, перекомпилировать его и заново загрузить, без необходимости перезагружать систему или перекомпилировать все ядро. Модули можно использовать не только для драйверов устройств, но и для других частей системы, например файловых систем.

Большинство драйверов устройств и многие другие функции ядра реализованы в Linux в виде модулей. Одним из таких модулей является драйвер флоппи-лент (драйвер *ftape*) для ленточных приводов, подключаемых к контроллеру гибких дисков, таких как Colorado Memory Jumbo моделей 120/250. Если вы собираетесь использовать этот драйвер в своей системе, неплохо разобраться, как компилировать, загружать и выгружать модули. Хотя ничто не мешает скомпилировать этот модуль статически в составе ядра, ленточное устройство используется слишком редко (обычно раз в день), чтобы позволить его драйверу постоянно занимать драгоценную память. Об этих устройствах и поддерживаемой аппаратуре см. Linux *ftape* HOWTO.

Прежде всего, вам понадобится пакет *modules*, в котором содержатся команды, используемые для загрузки и выгрузки модулей из ядра. На архивных сайтах FTP его обычно можно найти как *modules.tar.gz* в каталоге с исходным кодом ядра. В этом пакете содержится исходный код команд *insmod*, *modprobe*, *rmmod* и *lsmod*. Эти команды включаются в большинство дистрибутивов Linux и помещаются в *sbin*. Если эти команды у вас уже установлены, вероятно, загружать пакет *modules* не требуется. Однако если вы загрузите пакет и заново скомпилируете эти команды, то можете быть уверены в том, что у вас стоит их самая последняя версия.

Для того чтобы заново скомпилировать эти команды, распакуйте *modules.tar.gz* (скажем, в подкаталог */usr/src*). Следуйте инструкции по установке, которая там находится. Обычно все, что требуется, – это выполнить *make*, а затем *make install* (в качестве суперпользователя). После этого нужные команды будут установлены в */sbin* и будут готовы к использованию.

Модуль является просто отдельным объектным файлом, содержащим весь код драйвера. Модуль *ftape*, например, может называться *ftape.o*. Во многих системах сами модули хранятся в дереве каталогов в */lib/modules/kernelversion*,

где находятся каталоги, соответствующие различным типам модулей. Например, модули, скомпилированные для ядра 2.4.4, будут находиться в */lib/modules/2.4.4*. Возможно, какие-то модули уже установлены в вашей системе; проверьте соответствующий каталог.

Модули могут входить в состав исходного кода ядра или находиться вне его. В состав исходного кода ядра включаются наиболее часто используемые драйверы устройств, файловые системы и другие функции, поддерживаемые как часть официального исходного кода ядра. Использовать такие модули просто: во время выполнения *make config*, *make menuconfig* или *make xconfig* введите `m`, чтобы скомпилировать некоторую функцию в виде модуля. Это нужно сделать для всех функций, которые вы хотите скомпилировать как модули. Затем, после этапа *make bzImage*, выполните команды *make modules* и *make modules_install*. В результате модули будут скомпилированы и установлены в */lib/modules/kernelversion*. Неплохо после этого выполнить команду *depmod -a*, чтобы исправить зависимости между модулями (о цели этого будет рассказано ниже в этом разделе).

Новые модули, которые еще не интегрированы в официальный исходный код ядра или просто слишком экзотичны, чтобы в него включаться (например, драйвер какого-нибудь заказного устройства, отсутствующего в свободной продаже), могут поставляться как отдельные внешние модули. Распакуйте архив с таким модулем, откомпилируйте согласно инструкции, которая, надо надеяться, входит в архив, и скопируйте полученный файл в соответствующий подкаталог каталога */lib/modules/kernelversion*. Для некоторых модулей может поставляться сценарий установки, либо на последнем этапе нужно выполнить команду *make install*.

Получив скомпилированный модуль (из исходного кода ядра или внешнего источника), можно загрузить его с помощью команды

```
insmod module
```

где *module* — это имя объектного файла модуля. Например,

```
insmod /lib/modules/2.4.4/kernel/drivers/char/ftape/lowlevel/ftape.o
```

устанавливает драйвер *ftape*, если он находится в этом файле.

После установки модуль может вывести некоторые данные на консоль (а также в системные журналы), указывающие на то, что он инициализирован. Например, драйвер *ftape* может вывести следующее:

```
ftape v1.14 29/10/94 (c) 1993, 1994 Bas Laarhoven (bas@vimec.nl)
  QIC-117 driver for QIC-40 and QIC-80 tape drives
[000] kernel-interface.c (init_module) - installing QIC-117 ftape\
driver....
[001] kernel-interface.c (init_module) - 3 tape_buffers @ 001B8000.
[002]  calibr.c (time_inb) - inb() duration: 1436 nsec.
[003]  calibr.c (calibrate) - TC for `udelay()' = 2944 nsec (at 2049\
counts).
[004]  calibr.c (calibrate) - TC for `fdc_wait()' = 2857 nsec (at 2049\
counts).
```

Разумеется, точный вид сообщений зависит от модуля. Каждый модуль должен сопровождаться подробной документацией, описывающей, что он делает и как его отлаживать в случае возникновения проблем.

Вполне возможно, что *insmod* сообщит о невозможности загрузки модуля в ядро ввиду «symbols missing» (отсутствуют символы). Это означает, что модулю, который вы хотите загрузить, требуются некоторые функции ядра, которые не скомпилированы в ядре и отсутствуют в загружаемом модуле. Нужно постараться узнать, в каком модуле содержатся эти функции, и загрузить с помощью *insmod* сначала этот модуль, а потом другой. В конце концов, вам удастся добиться успеха, хотя и не без некоторого труда, и Linux не была бы верна себе, если бы не предоставила способа получше.

Для начала вам нужно иметь базу данных модулей в файле */lib/modules/kernelversion/modules.dep*. Создать ее можно, выполнив команду

```
depmod -a
```

Эта команда просматривает все имеющиеся у вас модули и регистрирует те случаи, когда им требуются другие модули. Когда такая база данных есть, вы можете вместо *insmod* воспользоваться командой *modprobe*, которая сверяется с базой данных модулей и при необходимости загружает те модули, которые должны быть загружены раньше требуемого. Например, в нашем файле *modules.dep* среди прочих содержится строка:

```
/lib/modules/2.4.4/kernel/drivers/isdn/hisax/hisax.o:        /lib/modules/2.4.4/
kernel/drivers/isdn/isdn.o
```

Она означает, что для загрузки модуля *hisax* (драйвера для ряда плат ISDN) сначала должен быть загружен модуль *isdn*. Теперь, если мы загружаем модуль *hisax* командой *modprobe* (этот пример несколько упрощен, поскольку модулю *hisax* требуются дополнительные параметры):

```
modprobe hisax
```

то *modprobe* проверит зависимости и загрузит модуль *isdn*. Если вы скомпилировали модуль для текущего ядра, то нужно сначала выполнить *depmod -a*, чтобы *modprobe* могла его найти.

Некоторым модулям требуются так называемые параметры модуля. Например, драйверу может понадобиться номер прерывания (IRQ). Эти параметры можно передать в виде *parametername=parametervalue* как для *insmod*, так и для *modprobe*. В следующем примере модулю *hisax* передается несколько параметров:

```
tigger # modprobe hisax type=3 protocol=2 io=0x280 irq=10
```

В документации к модулю должно быть сказано, какие параметры он поддерживает. Если вам лень читать документацию, можно воспользоваться умной утилитой *modinfo*, которая, помимо всего прочего, сообщает, какие параметры принимает модуль.

Следует предупредить пользователей дистрибутива Debian относительно модулей. Debian использует файл */etc/modules*, содержащий список всех модулей, которые необходимо загрузить во время начальной загрузки. Если модуль, который вам не нужен, продолжает загружаться, проверьте, не находится ли он в этом списке.

Список загруженных драйверов можно получить командой *lsmod*, например:

```
rutabaga% lsmod
Module:        #pages:                Used by
ftape             40
```

Выводится также объем используемой модулем памяти в страницах; в Linux размер страницы равен 4 Кбайт. Таким образом, драйвер *ftape* использует здесь 160 Кбайт памяти. Если от этого модуля зависят другие модули, они показываются в третьей колонке.

Модуль можно выгрузить из памяти командой *rmmod*. Например:

```
rmmod ftape
```

Аргументом *rmmod* является имя драйвера в том виде, как его выводит команда *lsmod*.

Если вы удовлетворены работой загруженных модулей, можно включить соответствующие команды *insmod* в один из *rc*-сценариев, выполняемых во время начальной загрузки системы. В некоторых дистрибутивах в одном из *rc*-сценариев может быть уже зарезервировано место для команд *insmod*.

Одной из особенностей поддержки модулей является необходимость перекомпиляции модулей при обновлении ядра до нового номера версии или уровня патчей. (Если вы перекомпилируете ядро с сохранением номера версии, этого делать не требуется.) Это делается для того, чтобы обеспечить совместимость модуля с используемой вами версией ядра. Если вы попытаетесь загрузить модуль с более новой или более старой версией ядра в сравнении с той, с которой он компилировался, *insmod* откажется загружать модуль. При компиляции модуля должно работать ядро той версии, с которой он должен исполняться. Поэтому при обновлении ядра сначала обновите ядро и перезагрузите машину с новым ядром, а затем откомпилируйте и загрузите модули. Есть вариант, позволяющий сохранять модули при переходе на другое ядро, но с ним связан ряд проблем, и мы не рекомендуем его использовать.

Автоматическая загрузка модулей

Автоматическая загрузка модулей – очень удобная функция ядра, которая реализована в компоненте ядра с именем *kmod*. С его помощью ядро может автоматически загружать требуемые драйверы устройств и другие модули без вмешательства системного администратора. Если в течение некоторого времени (60 секунд) модули не используются, они автоматически выгружаются.

Для того чтобы воспользоваться *kmod*, необходимо во время конфигурирования ядра включить его поддержку (`Kernel module loader`) в разделе `Loadable module support`.

Модули, для работы которых нужны другие модули, должны быть перечислены в */lib/modules/kernelversion/modules.dep*, а кроме того, должны иметься псевдонимы для основного и дополнительного номеров в */etc/conf.modules*. Дополнительные сведения можно получить в документации к пакету *modules*.

Если модуль загружен не вручную с помощью *insmod* или *modprobe*, а автоматически ядром, то в выводе *lsmod* к нему приписывается строка `autoclean`. Это говорит о том, что ядро удалит модуль, если он не будет использоваться в течение минуты.

Мы разобрали большой объем материала, и теперь у вас должны быть все инструменты, необходимые для компиляции и сопровождения своего собственного ядра.

8

Другие задачи администрирования

После чтения предыдущих трех глав у вас есть все навыки, необходимые для того, чтобы приступить к работе в системе. Но, в конечном счете, информация данной главы вам тоже понадобится. Есть ряд важных видов работ, которые должны войти у вас в привычку, в том числе создание резервных копий. Также может оказаться полезным умение организовать доступ к файлам и программам MS-DOS и Windows. И наконец, мы поможем вам справиться с происшествиями, которые, будем надеяться, никогда не произойдут, но иногда все же случаются, – паникой системы и нарушением целостности данных.

Создание резервных копий

Создание резервных копий системы является важным способом защиты от повреждения или потери данных при возникновении проблем с аппаратным обеспечением или совершением таких ошибок, как непреднамеренное уничтожение важных файлов. При работе с Linux вы, вероятно, сделаете многочисленные настройки системы, которые нельзя будет восстановить, просто переустановив систему с исходного носителя. Однако если исходные дискеты Linux или CD-ROM находятся под рукой, необходимости делать резервную копию системы целиком может и не возникнуть. Исходный инсталляционный носитель уже является превосходной резервной копией.

Работая в Linux, как и во всякой Unix-подобной системе, будучи зарегистрированным как `root`, можно совершить ошибку, в результате которой станут невозможными загрузка системы или регистрация в ней. Часто новички решают эту проблему, целиком переустанавливая систему из резервной копии или, что того хуже, с нуля. Однако в действительности в этом редко возникает необходимость. В разделе «Действия в аварийной ситуации» мы поговорим о том, что делать в таких случаях.

Если вы действительно потеряли данные, иногда их можно восстановить с помощью средств сопровождения системы, описанных в разделе «Проверка и восстановление файловых систем» главы 6. Однако в отличие от некото-

рых других операционных систем, обычно оказывается невозможным восстановить файл, удаленный командой *rm* или переписанный поверх опрометчиво выполненной командой *cp* или *mv* (например, копирование одного файла поверх другого уничтожает последний). В этих крайних случаях резервные копии являются ключом к решению возникших проблем.

Резервные копии обычно записываются на магнитные ленты, гибкие диски или CD-R(W). Никакой носитель не является надежным на все 100%, хотя в долгосрочном плане ленты и CD-R(W) более надежны, чем дискеты. Есть много программ, облегчающих изготовление резервных копий. В простейшем случае можно использовать сочетание *gzip* (или *bzip2*) и *tar* для сохранения файлов с жесткого диска на дискете или ленте. Это самый хороший способ, если резервные копии делаются достаточно редко, скажем, не чаще раза в месяц.

Если на вашей машине много пользователей или вы часто изменяете конфигурацию, то более разумно использовать инкрементальную схему резервирования. Такая схема предполагает, что полную резервную копию системы вы создаете примерно раз в месяц. Затем каждую неделю вы сохраняете только те файлы, которые изменились за последнюю неделю. Аналогично, каждый вечер вы можете сохранять лишь те файлы, которые изменились за последние сутки. Есть несколько программ, которые облегчают резервирование такого типа.

В основе инкрементального резервирования лежит мысль, что более эффективно осуществлять резервирование небольшими шагами. При этом используется меньше дискет, лент или компакт-дисков, а еженедельные и ежедневные резервные копии меньше по размеру и их легче делать. Благодаря такому методу у вас есть резервная копия не более чем однодневной давности. Если, скажем, вы случайно удалите всю систему целиком, то можно восстановить ее из резервной копии следующим образом:

1. Восстановите последнюю ежемесячную копию. Например, если вы затерли систему 17 июля, восстановите ее из полной резервной копии от 1 июля. Теперь ваша система находится в том состоянии, в каком она была в момент создания резервной копии 1 июля.
2. Восстановите все еженедельные резервные копии, сделанные в течение месяца. В нашем случае можно восстановить две еженедельные копии от 7 и 14 июля. Восстановление каждой еженедельной резервной копии обновляет все файлы, изменившиеся в течение соответствующей недели.
3. Восстановите все ежедневные копии за последнюю неделю. В нашем случае должны быть восстановлены ежедневные резервные копии за 15 и 16 июля. Теперь система находится в том состоянии, в каком она была при создании резервной копии 16 июля; потеряны файлы не более чем за один день.

В зависимости от размера системы для сохранения полной ежемесячной резервной копии может потребоваться 4 Гбайт пространства или более. Часто для этого нужно не больше пространства, чем имеется на одной современной

магнитной ленте, но достаточно много ZIP-дисков. Однако для еженедельных и ежедневных резервных копий требуется обычно значительно меньше места. Характер использования вашей системы может позволить вам создавать еженедельную копию вечером в воскресенье и не утруждать себя ежедневными копиями по выходным дням.

Важная характеристика, которой обычно должны обладать резервные копии, – это возможность выбирать для восстановления отдельные файлы. Благодаря такой возможности при случайном удалении файла или группы файлов они могут быть восстановлены без необходимости проводить полное восстановление системы. В зависимости от того, как вы создаете резервные копии, такая задача может выполняться очень просто или с мучительными сложностями.

В этом разделе мы намерены обсудить использование *tar*, *gzip* и некоторых других нужных программ для создания резервных копий на дискетах и лентах. Мы даже расскажем о том, как использовать дискеты и ленты. Эти средства позволяют делать резервные копии в основном «вручную». Этот процесс можно автоматизировать, написав сценарии оболочки, и даже осуществлять создание резервных копий по расписанию в ночное время с помощью демона *cron*. Все, что от вас требуется, – это вовремя вставлять ленты в механизм. Есть другие программные пакеты для создания резервных копий, восстановления из резервных копий отдельных файлов и т. д. с прекрасными управляемыми меню интерфейсами. Многие из них являются на самом деле удобными оболочками для *tar* и *gzip*. Вы сами должны решить, какой тип системы создания резервных копий вам больше подходит.

Простые резервные копии

Простейший способ создания резервной копии – это использование *tar* для архивирования всех файлов системы или только файлов, расположенных в избранных каталогах. Однако, прежде всего, следует решить, какие файлы сохранять. Следует ли сохранять резервную копию каждого файла системы? В этом редко есть необходимость, особенно если у вас есть исходные инсталляционные дискеты или CD-ROM. Если вы сделали в системе существенные изменения, но все остальное сохранилось таким, каким оно было на инсталляционном носителе, то можно обойтись архивированием только тех файлов, в которые вы внесли изменения. Хотя с течением времени уследить за такими изменениями становится все труднее.

Обычно производятся изменения в конфигурационных файлах, находящихся в каталоге */etc*. Есть и другие конфигурационные файлы, и не помешает заархивировать такие каталоги, как */usr/lib* и */etc/X11* (содержащий файлы конфигурации XFree86, как мы узнаем из раздела «Установка XFree86» главы 10).

Следует также сделать резервную копию исходного кода ядра (если вы его обновили или скомпилировали собственное ядро), который находится в каталоге */usr/src/linux*.

Развлекаясь с Linux, неплохо вести учет изменений, которые вы произвели в системе, чтобы обдуманно делать резервные копии. В параноидальном случае можно сделать резервную копию и всей системы; это обойдется вам не дороже, чем стоимость используемых при этом носителей.

Конечно, надо сделать резервные копии и исходных каталогов всех пользователей системы. Обычно они находятся в */home*. Если в вашей системе есть электронная почта (см. раздел «Электронная почта» главы 16), то может понадобиться сохранить файлы поступившей почты всех пользователей. Часто старые, но важные почтовые сообщения пользователи хранят в спулинге входящей почты, и эти файлы нетрудно случайно повредить из-за ошибки почтовой программы или по иной причине. Обычно эти файлы расположены в */var/spool/mail*. Конечно, это относится только к локальной почтовой системе, а не к тем, кто использует для доступа к почте POP3 или IMAP.

Создание резервной копии на магнитной ленте

Если вы уже решили, какие файлы и каталоги нужно резервировать, можно приступать к делу. Можно непосредственно использовать команду *tar* для создания резервных копий, как было показано в разделе «Использование tar» главы 7. Например, команда

```
tar cvf /dev/qft0 /usr/src /etc /home
```

архивирует все файлы из каталогов */usr/src*, */etc* и */home* на */dev/qft0*. */dev/qft0* является первым устройством «флоппи-ленты», т. е. ленточного привода, подключенного к контроллеру флоппи-дисков. Этим интерфейсом пользуются многие распространенные ленточные приводы для PC. Если у вас есть ленточные устройства SCSI, то они именуются */dev/st0*, */dev/st1* и т. д. в зависимости от номера устройства. Ленточные устройства с другими типами интерфейсов именуются иначе. Эти имена можно выяснить в документации к драйверу устройства, включенному в ядро.

Считать обратно архив с ленты можно командой

```
tar xvf /dev/qft0
```

Это ничем не отличается от работы с *tar*-файлом на диске, демонстрировавшейся в разделе «Утилиты архивирования и сжатия» главы 7.

При использовании ленточного привода лента рассматривается как поток, читать из которого или записывать в который можно лишь в одном направлении. После того как *tar* закончит работу, устройство магнитных лент будет закрыто, а лента перемотана обратно. На ленте не создается файловая система, также не нужно монтировать ее как файловую систему или пытаться обращаться к данным на ленте как к файлам. Воспринимайте ленточное устройство как один архивный «файл».

Магнитные ленты должны быть отформатированы, прежде чем ими можно будет воспользоваться. При форматировании на ленту записываются маркер начала ленты и сведения о сбойных блоках. Для форматирования лент QIC-80 (с которыми работают драйверы флоппи-лент) можно воспользоваться утили-

той *ftformat*, которая либо содержится в дистрибутиве, либо может быть загружена с *ftp://sunsite.unc.edu/pub/Linux/kernel/tapes* в составе пакета *ftape*.

Использование ленты для хранения лишь одного *tar*-архива может оказаться расточительным, если этот архив занимает незначительную часть емкости ленты. Чтобы разместить на ленте несколько архивов, нужно избегать перемотки ленты в начало после каждого ее использования, а также иметь возможность подводить ленту к очередному «маркеру файла» как при создании архива, так и при его распаковке.

Это можно осуществить при использовании устройств без перемотки ленты в начало; они называются */dev/nqft0*, */dev/nqft1* и т. д. для флоппи-лент и */dev/nst0*, */dev/nst1* и т. д. – для ленточных устройств SCSI. При использовании таких устройств для чтения или записи лента не перематывается в начало при закрытии устройства (т. е. по завершении работы *tar*). Затем можно снова воспользоваться *tar* для добавления на ленту еще одного архива. Два *tar*-файла на одной ленте никак не связаны между собой. Конечно, если вы затем перепишете первый *tar*-файл, то можете затереть при этом второй или оставить между первым и вторым файлом нежелательный промежуток, который будет восприниматься как мусор. Обычно нужно стремиться к тому, чтобы не переписывать отдельные файлы, если на ленте их несколько.

При использовании ленточного устройства, не перематывающего ленту в начало, можно записать на нее столько файлов, сколько позволит емкость ленты. Возвратить ленту в начало после работы можно командой *mt*. *mt* является командой общего назначения, выполняющей с ленточным устройством ряд операций.

Например, команда

```
mt /dev/nqft0 rewind
```

возвращает в начальное положение ленту на первом устройстве флоппи-лент. (В этом случае можно использовать и соответствующее устройство с обратной перемоткой ленты, при этом перемотка ленты будет осуществлена как побочный эффект закрытия устройства.)

Аналогично команда

```
mt /dev/nqft0 reten
```

снимает механические напряжения в ленте путем перемотки ее в конец, а затем обратной перемотки в начальное положение.

При чтении файлов с ленты, содержащей несколько файлов, нужно использовать устройство без перемотки ленты назад, а также команды *tar* и *mt* для позиционирования ленты на нужный файл.

Например, перейти к следующему файлу на ленте можно командой

```
mt /dev/nqft0 fsf 1
```

При этом на ленте пропускается один файл. Чтобы пропустить два файла, нужно выполнить команду

```
mt /dev/nqft0 fsf 2
```

Команду *mt* нужно использовать с соответствующим устройством, не перематывающим ленту назад. Обратите внимание, что эта команда не перемещает ленту к «файлу номер 2», а пропускает очередные два файла относительно текущего положения ленты. Если вы точно не знаете, в каком положении лента находится в данный момент, отмотайте ее назад командой *mt*. Можно пропускать файлы и в обратном направлении. Полный список параметров команды вы найдете на странице руководства по *mt*.

При чтении ленты, на которой находится несколько файлов, необходимо использовать *mt*. Прочесть с ленты два последовательных архивных файла, дважды выполняя команду *tar*, обычно не удается, поскольку *tar* не распознает маркер файла, размещенный на ленте между файлами. По завершении выполнения первой команды *tar* лента оказывается позиционированной на начало маркера файла. Если сразу попытаться выполнить вторую команду *tar*, будет получено сообщение об ошибке, поскольку *tar* попытается прочесть маркер файла. Для того чтобы перейти к следующему файлу, после чтения файла с ленты нужно просто выполнить команду

```
mt device fsf 1
```

Создание резервных копий на дискетах

Как мы видели в предыдущем разделе, команда

```
tar cvf /dev/fd0 /usr/src /etc /home
```

создает резервную копию каталогов */usr/src*, */etc* и */home* на */dev/ fd0* – первом устройстве гибких дисков. Прочесть резервную копию можно после этого командой

```
tar xvf /dev/fd0
```

Поскольку емкость дискет весьма ограничена, GNU *tar* позволяет создавать «многотомные» архивы. (Эта функция применима и к лентам, но гораздо полезнее в случае дискет.) Когда используется эта функция, *tar* предлагает вам установить очередной том после чтения или записи каждой дискеты. Чтобы воспользоваться этой функцией, нужно лишь передать *tar* параметр *M*:

```
tar cvMf /dev/fd0 /usr/src /etc /home
```

Снабдите ваши дискеты метками и следите за порядком их использования при восстановлении архива.

Предостережение, касающееся этой функции: она не поддерживает автоматическое сжатие, осуществляемое при использовании параметров *z* и *I*. Однако по ряду причин вы, возможно, и не захотите сжимать резервные копии, создаваемые *tar*; об этом мы поговорим позже. Тем не менее, можно создавать многотомные резервные копии с использованием *tar* и *gzip* совместно с программой, читающей или записывающей данные на серию дискет (или лент) и выдающей приглашение при необходимости установки очередного носителя. Одной из таких программ является *backflops*, имеющаяся в ряде дистрибутивов Linux и на архивных сайтах FTP. Можно самостоятельно осуществить такую операцию, создав сначала резервную копию в виде

дискового файла, а затем отправив этот архив по частям на дискеты с помощью команды *dd* или аналогичной ей. Если у вас достаточно смелости попробовать, вы сообразите, как это сделать.

Сжимать или не сжимать?

Есть веские аргументы как в пользу, так и против компрессии *tar*-архивов при создании резервных копий. Общая проблема состоит в том, что при всем удобстве использования как *tar*, так и утилиты компрессии *gzip* и *bzip2* не очень устойчивы в отношении возможных ошибок. Сжатие с помощью *gzip* или *bzip2* может значительно сократить объем носителей, необходимых для сохранения архива, однако сжатие целых *tar*-файлов при выводе на дискеты или ленты порождает опасность потерять всю копию при повреждении лишь одного блока архива в результате, скажем, ошибки на носителе (что для дискет и магнитных лент не такая уж редкость). Большинство алгоритмов сжатия, в том числе *gzip*, зависит от согласованности данных на протяжении больших участков. При повреждении каких-либо данных в сжатом архиве *gunzip* может оказаться вообще не в состоянии декомпрессировать файл, что сделает его полностью недоступным для *tar*.

Это значительно хуже, чем если бы *tar*-файл находился на ленте в несжатом виде. Хотя *tar* не обеспечивает большой защиты от повреждения данных в архиве, но если повреждения данных невелики, то обычно можно без особых трудностей извлечь большую часть упакованных файлов, по крайней мере, до того места, где произошло повреждение. Это не идеально, но лучше, чем полная потеря резервной копии.

Для изготовления резервных копий более эффективно использовать вместо *tar* специальные средства архивации. Есть выбор из нескольких вариантов. *cpio* является утилитой архивации, упаковывающей файлы сходным с *tar* образом. Однако благодаря использованию более простого способа хранения данных *cpio* корректно восстанавливает данные из поврежденных архивов. (Ошибки в сжатых файлах она все же обрабатывает не очень хорошо.)

Наилучшим решением может быть использование такой программы, как *afio*. *afio* поддерживает многотомные архивы и во многих отношениях сходна с *cpio*. Однако *afio* осуществляет сжатие и более надежна, поскольку каждый файл сжимается отдельно. Благодаря этому при повреждении архива ущерб может быть ограничен отдельными файлами вместо утраты всей резервной копии.

Эти программы должны иметься в вашем дистрибутиве Linux или их можно получить с любого находящегося в Интернете архива Linux. Ряд других утилит резервирования с разной степенью популярности и удобства разработан или перенесен на Linux. Если вас серьезно беспокоит проблема создания резервных копий, следует посмотреть эти утилиты.[1] В число программ такого рода входят свободно распространяемые *taper*, *tob* и *Amanda*; есть также

[1] Этот раздел был написан после того, как автор сделал резервную копию своей системы Linux впервые за почти четыре года работы!

коммерческие программы, такие как *ARKEIA* (бесплатна при использовании не более чем на двух компьютерах), *BRU* и *Arcserve*. Массу бесплатных утилит копирования можно найти на *http://velocom.linux.tucows.com/system/backup.html*.

Инкрементальное резервирование

Инкрементальное резервирование, описанное ранее в этой главе, является хорошим способом поддержания обновленных резервных копий. Например, можно каждую ночь делать резервные копии файлов, которые изменились за последние сутки, еженедельные копии файлов, изменившихся в течение недели, и ежемесячные копии всей системы.

Инкрементальные копии можно создавать с помощью упомянутых выше средств: *tar*, *gzip*, *cpio* и др. При создании инкрементальной копии нужно сначала создать список файлов, изменившихся в течение некоторого промежутка времени. Это легко сделать с помощью команды *find*.[1] Если пользоваться специальной программой для создания резервных копий, то, скорее всего, вам понадобится не *find*, а установка какого-то параметра, указывающего на желание сделать инкрементальную резервную копию.

Например, чтобы создать список файлов, измененных в течение последних 24 часов, можно использовать команду

```
find / -mtime -1 \! -type d -print > /tmp/filelist.daily
```

Первым аргументом *find* является каталог, с которого нужно начать поиск. В данном случае это корневой каталог /. Параметр *-mtime -1* указывает на то, что нужно найти все файлы, измененные за последние сутки.

Параметр *\! -type d* является сложным (и необязательным), но он удаляет из выдачи некоторые ненужные сведения. Он извещает *find* о необходимости исключить из результирующего списка файлов каталоги. Восклицательный знак (!) является оператором отрицания (который в данном случае означает «исключить файлы типа d»), но перед ним нужно поставить обратную косую черту, поскольку иначе оболочка интерпретирует его как специальный символ.

Параметр *-print* вызывает вывод имен файлов, удовлетворивших критерию поиска, на стандартное устройство вывода. Мы перенаправляем стандартный вывод в файл, который в последующем используем. Аналогично для поиска всех файлов, измененных за последнюю неделю, воспользуемся командой

```
find / -mtime -7 -print > /tmp/filelist.weekly
```

---

[1] Если вы еще не знакомы с *find*, постарайтесь поскорее это сделать. *find* предоставляет прекрасный способ найти среди разных каталогов файлы с определенными именами, правами доступа или временем модификации. *find* может даже выполнить заданную программу для каждого найденного файла. Короче, *find* – это ваш друг, и всем грамотным системным администраторам хорошо известно, как ею пользоваться.

Обратите внимание на то, что при таком использовании *find* эта команда осуществляет поиск во всех смонтированных файловых системах. Если, например, у вас смонтирован CD-ROM, *find* будет пытаться найти файлы и на нем (хотя вы, возможно, не собираетесь делать резервную копию CD-ROM). Параметр *-prune* можно использовать для исключения некоторых каталогов из числа обрабатываемых командой *find*. Можно также выполнить *find* несколько раз, указывая при этом первый аргумент, отличный от /. Подробности см. на страницах руководства по *find*(1).

Теперь у вас есть список файлов для включения в резервную копию. Раньше при использовании *tar* мы указывали архивируемые файлы в командной строке. Однако список файлов может быть слишком велик, чтобы уместиться в одной командной строке, которая обычно ограничена 2048 символами, да и сам список хранится в файле.

Можно воспользоваться параметром *-T*, чтобы указать *tar* на файл, содержащий список подлежащих архивированию файлов. Чтобы применить этот параметр, нужно использовать для *tar* альтернативный синтаксис, в котором все параметры задаются явно с дефисами. Например, для архивирования файлов, перечисленных в */tmp/filelist.daily*, на устройство */dev/qft0* выполните команду

```
tar -cv -T /tmp/filelist.daily -f /dev/qft0
```

Теперь можно написать коротенький сценарий, автоматически создающий список файлов и делающий их резервную копию с помощью *tar*. С помощью демона *cron* можно выполнять этот сценарий каждую ночь в определенное время; вам нужно будет позаботиться только о том, чтобы в приводе находилась магнитная лента. Аналогичные сценарии можно написать для создания еженедельных и ежемесячных копий. О *cron* рассказывается в следующем разделе.

Запуск задач по расписанию с помощью cron

Первоначальной задачей компьютера была автоматизация рутинных задач. Если вам нужно производить резервное копирование диска регулярно в час ночи, зачем каждый раз вручную вводить команды, особенно если для этого нужно вылезать из постели? Нужно сообщить компьютеру о действиях, которые необходимо произвести, и затем об этом забыть. В Unix-системах для автоматизации таких функций существует демон *cron*. Если сказать вкратце, то *cron* используется путем запуска команды *crontab* и ввода строк в специальном формате, понятном *cron*. В каждой строке указывается, какую команду нужно выполнить и когда.

Невидимо для вас *crontab* записывает ваши команды в файл с вашим именем пользователя, находящийся в каталоге */var/spool/cron/crontabs*. (Например, для пользователя `mdw` используемый *crontab* файл будет называться */var/spool/cron/crontabs/mdw*.) Демон с именем *crond* регулярно читает этот файл и в должное время выполняет команды. Один из *rc*-файлов системы за-

пускает *crond* при начальной загрузке системы. Фактически команды с именем *cron* нет, есть только утилита *crontab* и демон *crond*.

В некоторых системах запускать *cron* разрешается только суперпользователю. В любом случае взглянем на полезную команду, которую вам может понадобиться выполнить от имени `root`, и рассмотрим, как должна выглядеть запись для выполнения задания планировщиком. Допустим, вы хотите ежедневно удалять устаревшие файлы из каталога */tmp*, служащего для хранения временных файлов, создаваемых разнообразными утилитами.

Учтите, что *cron* никогда не выводит ничего на консоль. Все выходные сообщения и сообщения об ошибках отправляются электронной почтой пользователю, являющемуся владельцем соответствующего файла *crontab*. Можно переопределить такое поведение, задав строку `MAILTO=address` в файле *crontab* перед самими заданиями.

В большинстве систем содержимое */tmp* очищается при перезапуске системы, но если система перезагружается редко, может оказаться полезным использование *cron* для проверки устаревших файлов (скажем, таких, к которым не было доступа в течение последних трех дней). Ввести нужно следующую команду:

```
ls -l filename
```

Но как узнать, какое имя файла *filename* нужно задать? Нужно поместить эту команду внутрь команды *find*, которая просмотрит все файлы, содержащиеся в каталоге, и выполнит над каждым из них указанную вами операцию.

Мы уже пользовались командой *find* в разделе «Инкрементальные резервные копии». Сейчас мы зададим */tmp* в качестве каталога, в котором необходимо осуществлять поиск, и используем параметр *-atime* для поиска тех файлов, доступ к которым осуществлялся не позднее, чем три дня назад. Параметр *-exec* означает «выполнить с каждым найденным файлом следующую команду»:

```
find /tmp \! -type d -atime +3 -exec ls -l {} \;
```

Мы просим *find* выполнить команду *ls -l*, которая просто выведет сведения о файле. (Многие часто используют аналогичную запись *crontab* для удаления файлов, но при этом легко нарушить систему безопасности.) Странная строка `{}` означает лишь способ сообщить системе: «Сделать это с каждым найденным в соответствии с критерием отбора файлом». Символы `\;` обозначают конец параметра *-exec*.

Теперь у нас есть команда, отыскивающая в */tmp* устаревшие файлы. Но нам еще нужно указать, как часто ее нужно выполнять. В формате, используемом *crontab*, есть шесть полей:

```
minute   hour   day   month   dayofweek   command
```

Заполняются поля следующим образом:

1. Минуты (от 0 до 59).
2. Час (от 0 до 23).

3. День месяца (от 1 до 31).
4. Месяц (от 1 до 12 или по названиям: `jan`, `feb` и т. д.).
5. День недели (от 0 до 6, где 0 соответствует воскресенью, или по названиям: `mon`, `tue` и т. д.).
6. Команда (может состоять из нескольких слов).

На рис. 8.1 показана запись *cron* со всеми заполненными полями. Команда является сценарием, выполняемым в *sh* – оболочке Борна. Эта запись не очень реалистична: сценарий выполняется только при соблюдении всех условий в первых пяти полях. Это означает, что он будет выполняться в воскресенье, выпадающее на 15 января или 15 июля – не часто такое случается! Поэтому наш пример не очень полезен.

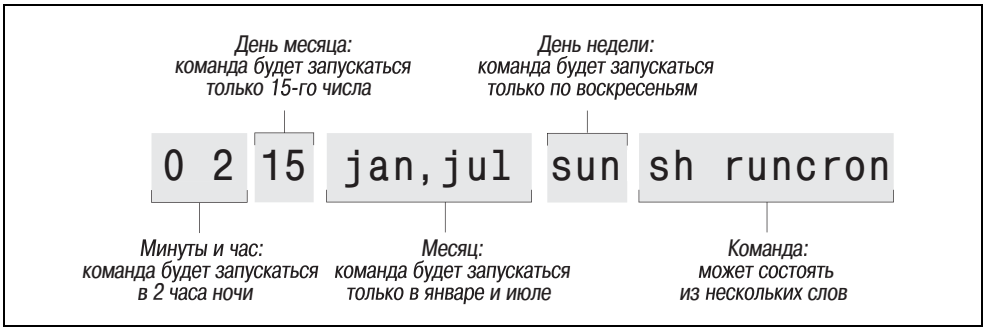


Рис. 8.1. *Пример записи cron*

Если вы хотите, чтобы команда выполнялась ежедневно в 1 час ночи, задайте 0 в поле `minute` и 1 в поле `hour`. В остальных трех полях должны стоять звездочки, что означает «каждый день каждого месяца в указанное время». Полная строка для *crontab* выглядит так:

```
0 1 * * * find /tmp -atime 3 -exec ls -l {} \;
```

Поскольку с полями, задающими время, можно делать много интересных вещей, еще немного поразвлекаемся с этой командой. Предположим, необходимо, чтобы команда выполнялась только по первым числам каждого месяца. Тогда нужно сохранить первые два поля, но в третье поле поместить 1:

```
0 1 1 * * find /tmp -atime 3 -exec ls -l {} \;
```

Для выполнения команды еженедельно по понедельникам верните звездочку в третье поле, но в пятом поле задайте 1 или `mon`:

```
0 1 * * mon find /tmp -atime 3 -exec ls -l {} \;
```

Можно достичь еще большей сложности, задавая в каждом поле несколько временных отметок. Например, в следующей строке запятая означает «выполнять 1 и 15 числа» каждого месяца:

```
0 1 1,15 * * find /tmp -atime 3 -exec ls -l {} \;
```

дефис означает «выполнять ежедневно с 1 по 15 число включительно»:

```
0 1 1-15 * * find /tmp -atime 3 -exec ls -l {} \;
```

а косая черта, за которой следует 5, означает «выполнять каждый пятый день», то есть 1-го, 6-го, 11-го и т. д.:

```
0 1 */5 * * find /tmp -atime 3 -exec ls -l {} \;
```

Теперь можно действительно поместить запись в файл *crontab*. Зарегистрируйтесь как root (такого рода вещи должен делать суперпользователь) и введите команду *crontab* с ключом *-e* (от «edit»):

```
rutabaga# crontab -e
```

По умолчанию эта команда начинает сеанс редактирования *vi*. Если вы предпочитаете Emacs, это можно указать до запуска *crontab*. Для Борн-совместимой оболочки введите команду

```
rutabaga# export VISUAL=emacs
```

Для оболочки C:

```
rutabaga# setenv VISUAL emacs
```

В некоторых версиях *crontab* можно вместо VISUAL использовать переменную окружения EDITOR. Введите одну-две строки, начинающиеся с решетки (#), в которых будут располагаться комментарии, объясняющие ваши действия, а затем введите свою запись в *crontab*:

```
# Вывести список файлов из /tmp, которые старше 3 дней.
# Запускается в 1:00 каждую ночь.
0 1 * * * find /tmp -atime 3 -exec ls -l {} \;
```

При выходе из *vi* команды будут сохранены. Посмотреть свою запись *crontab* можно, введя команду

```
rutabaga# crontab -l
```

Мы еще не говорили о важной стороне нашей записи *crontab*: а куда будет направляться вывод? По умолчанию *cron* отправляет вывод не на стандартное устройство вывода и стандартное устройство вывода ошибок, а пользователю в виде сообщения электронной почты. В данном случае почта будет отправлена пользователю root, но ее следует автоматически перенаправить вам как системному администратору. В */usr/lib/aliases* (или */etc/aliases* в дистрибутивах SuSE, Debian и RedHat) должна быть следующая строка:

```
root: имя-вашей-учетной-записи
```

Чуть ниже мы покажем, как сохранить вывод в файле, а не отправлять его почтой.

Приведем пример еще одной команды, часто используемой в файлах *crontab*. Она осуществляет создание на магнитной ленте резервной копии каталога. Предполагается, что перед выполнением команды в привод будет установлена лента. Сначала команда *mt* обеспечивает перемотку ленты в устройстве

/dev/qft0 к ее началу. Затем команда *tar* переносит на ленту все файлы из каталога */src*. Точка с запятой служит разделителем команд – это стандартный синтаксис оболочки:

```
# делать резервную копию каталога /src раз в два месяца.
0 2 1 */2 * mt -f /dev/qft0 rewind; tar cf /dev/qft0 /src
```

Первые два поля обеспечивают выполнение команды в 2:00, а третье поле указывает на первое число месяца. Четвертое поле указывает на выполнение раз в два месяца. Того же результата можно добиться более понятной командой, например:

```
0 2 1 jan,mar,may,jul,sep,nov * mt -f /dev/qft0 rewind; \
          tar cf /dev/qft0 /src
```

В разделе «Создание резервных копий» объясняется, как проводить создание резервных копий на регулярной основе.

В следующем примере каждые два дня запускается *mailq*, чтобы проверить, не застряло ли какое-нибудь письмо в очереди отправки почты, и послать письмо администратору с результатами проверки. Если почта в очереди есть, то в отчет включаются сведения об адресации и проблемах с доставкой, в противном случае сообщение пусто:

```
0 6 */2 * * mailq -v | \
          mail -s "Tested Mail Queue for Stuck Email" postmaster
```

Вероятно, не хотелось бы получать каждый день письмо, если все в порядке. В приведенных нами примерах команды не порождают никакого вывода, если только не происходит ошибка. Однако вы можете перенаправить стандартный вывод в */dev/null* или посылать вывод в журнал (обратите внимание на использование двух знаков >, чтобы избежать затирания предшествующего вывода):

```
0 1 * * * find /tmp -atime 3 -exec ls -l {} \; >> /home/mdw/log
```

В этой записи мы перенаправляем стандартный вывод, но позволяем отправлять стандартный вывод ошибок как почтовые сообщения. Это может быть очень удобно, поскольку при возникновении проблем мы получим сообщение об ошибке. Если вы не хотите получать почтовые сообщения ни при каких обстоятельствах, перенаправьте в файл как стандартное устройство вывода, так и стандартное устройство вывода ошибок:

```
0 1 * * * find /tmp -atime 3 -exec ls -l {} \; >> /home/mdw/log 2>&1
```

При сохранении выдачи в файле журнала вы сталкиваетесь с проблемой неуклонного роста размера файла. Можно создать еще одну запись *cron*, которая обеспечит удаление файла, скажем, раз в неделю.

В записях *crontab* можно использовать команды только оболочки Борна. Это означает невозможность использования удобных расширений, применяемых в *bash* и других современных оболочках, например псевдонимов или знака «~» для обозначения исходного каталога пользователя. Однако можно

использовать $HOME; *cron* распознает переменные окружения $USER, $HOME и $SHELL. Каждая команда выполняется из вашего исходного каталога, используемого в качестве рабочего.

Некоторые предпочитают задавать в записях *crontab* абсолютные пути команд, например */usr/bin/find* и */bin/rm*. Тем самым всегда выбирается правильная команда независимо от корректности задания пути поиска.

Если команда становится слишком длинной и сложной, чтобы разместиться в одной строке, создайте сценарий оболочки и запускайте его через *cron*. Проверьте, чтобы сценарий был исполняемым (выполните *chmod +x*) или запускайте его через оболочку:

```
0 1 * * * sh runcron
```

Системному администратору часто приходится создавать файлы *crontab* для фиктивных пользователей, таких как news или UUCP. Запуск всех утилит от имени root был бы излишним и потенциально опасным, поэтому создаются такие специальные пользователи.

Выбор пользователя влияет также на право владения: файл *crontab* для news должен запускать файлы, владельцем которых является news, и т. д. В целом, нужно следить, чтобы владельцем утилит был тот пользователь, от имени которого создается файл *crontab*.

Суперпользователь может редактировать файлы *crontab* других пользователей с помощью параметра *-u*:

```
tigger # crontab -u news -e
```

Это удобно, поскольку нельзя зарегистрироваться в системе как news, но может потребоваться отредактировать файл *crontab* для news.

Управление системными журналами

Утилита *syslogd* регистрирует в журналах различные виды активности системы, такие как отладочные сообщения *sendmail* и предупреждения, выводимые ядром. *syslogd* выполняется как демон и обычно запускается из какого-либо *rc*-файла во время загрузки.

Управление тем, куда *syslogd* записывает информацию, осуществляется с помощью файла */etc/syslog.conf*. Вот образец этого файла (обычно он имеет значительно более сложный вид):

```
*.info;*.notice    /var/log/messages
mail.debug         /var/log/maillog
*.warn             /var/log/syslog
kern.emerg         /dev/console
```

В первом поле каждой строки перечислены типы сообщений, которые нужно регистрировать, а во втором поле указано место, где они должны быть зарегистрированы. Первое поле имеет формат

```
facility.level [; facility.level ... ]
```

где *facility* является системным приложением или средством, генерирующим сообщение, а *level* является уровнем важности сообщения.

Например, *facility* (средство) может принимать значение mail (для почтового демона), kern (для ядра), user (для программ пользователей) или auth (для программ аутентификации, например *login* или *su*). Звездочка в этом поле обозначает все средства.

level (уровень) может принимать значения (по возрастанию уровня важности): debug, info, notice, warning, err, crit, alert или emerg.

В приведенном примере файла */etc/syslog.conf* мы видели, что все сообщения уровня важности info и notice (информация и извещения) регистрируются в */var/log/messages*, все сообщения уровня debug (отладка) от почтового демона регистрируются в */var/log/maillog*, а все warn (предупредительные) сообщения регистрируются в */var/log/syslog*. Кроме того, все emerg (чрезвычайные) предупреждения ядра посылаются на консоль (которой является рабочая виртуальная консоль или *xterm*, запущенный с параметром *-C*).

Регистрируемые *syslogd* сообщения обычно содержат дату, указание процесса или средства, доставившего сообщение, и само сообщение, причем все это в одной строке. Например, сообщение ядра об ошибке данных в файловой системе *ext2fs* может появиться в журнале в виде:

```
Dec  1 21:03:35 loomer kernel: EXT2-fs error (device 3/2):
  ext2_check_blocks_bit map: Wrong free blocks count in super block,
  stored = 27202, counted = 27853
```

Аналогично успешное выполнение команды *su* для регистрации с именем root может выглядеть в журнале так:

```
Dec 11 15:31:51 loomer su: mdw on /dev/ttyp3
```

Файлы журналов могут иметь большое значение при отслеживании возникших в системе проблем. Если журнал слишком разрастается в размере, его можно очистить командой *cat /dev/null>logfile*; сам файл для ведения журнала при этом сохраняется.

Возможно, в вашей системе уже организован запуск *syslogd*, и файл */etc/syslog.conf* настроен на правильную работу. Однако важно знать, где находятся ваши файлы системных журналов и какие программы в них представлены. Если приходится регистрировать много сообщений (например, отладочных сообщений ядра, которые могут быть очень многословными), то можно отредактировать *syslog.conf* и потребовать от *syslogd* заново прочесть файл конфигурации, выполнив команду

```
kill -HUP `cat /var/run/syslog.pid`
```

Обратите внимание на использование обратных кавычек при получении идентификатора процесса *syslogd*, содержащегося в */var/run/syslog.pid*.

Могут иметься и другие системные журналы. В их число входят:

/var/log/wtmp

Этот файл содержит двоичные данные о времени регистрации и продолжительности работы всех пользователей системы. Он используется командой *last* для вывода списка регистрировавшихся пользователей. Вывод *last* может выглядеть так:

```
mdw      tty3                Sun Dec 11 15:25   still logged in
mdw      tty3                Sun Dec 11 15:24 - 15:25  (00:00)
mdw      tty1                Sun Dec 11 11:46   still logged in
reboot   ~                   Sun Dec 11 06:46
```

В */var/log/wtmp* также добавляются записи о перезагрузке системы.

/var/run/utmp

Это еще один двоичный файл, содержащий сведения о пользователях, зарегистрированных в системе в настоящее время. Он используется такими командами, как *who*, *w* и *finger*, для вывода данных о подключенных пользователях. Например, команда *w* может вывести такие данные:

```
  3:58pm  up  4:12,  5 users,  load average: 0.01, 0.02, 0.00
User     tty       login@  idle   JCPU   PCPU  what
mdw      ttyp3    11:46am    14                -
mdw      ttyp2    11:46am            1         w
mdw      ttyp4    11:46am                      kermit
mdw      ttyp0    11:46am    14                bash
```

Мы видим время регистрации каждого пользователя (в данном случае один пользователь зарегистрировался несколько раз), а также выполняемую им в данное время команду. На странице руководства по *w* дается объяснение всех выводимых полей.

/var/log/lastlog

Этот файл аналогичен *wtmp*, но используется другими программами (такими как *finger*) для определения времени последней регистрации пользователя.

Учтите, что формат файлов *wtmp* и *utmp* в разных системах может быть различен. Программы могут быть откомпилированы в расчете на тот или иной формат. Поэтому команды, которые используют эти файлы, могут выводить неточную информацию, особенно если файлы повреждены программой, которая пишет в них информацию в неверном формате.

Журнальные файлы могут быстро разрастаться в объеме, и при недостаточном дисковом пространстве нужно принимать меры, чтобы разделы не слишком быстро заполнялись данными. Конечно, можно периодически удалять файлы журналов, но вам, возможно, не следует этого делать, поскольку в файлах журналов могут содержаться данные, представляющие ценность в критических ситуациях.

Один из выходов – периодическое копирование журналов в другой файл с последующим его сжатием. Файл журнала при этом очищается. Вот короткий сценарий, осуществляющий такую процедуру для журнального файла */var/log/messages*:

```
mv /var/log/messages /var/log/messages-backup
cp /dev/null /var/log/messages

CURDATE=`date +"%m%d%y"`

mv /var/log/messages-backup /var/log/messages-$CURDATE
gzip /var/log/messages-$CURDATE
```

Прежде всего, мы копируем журнальный файл под другим именем и усекаем исходный файл до нулевой длины путем копирования в него из устройства */dev/null*. Это делается для того, чтобы можно было без помех продолжить ведение журнала, пока выполняются следующие команды сценария. Затем мы создаем строку из текущей даты, которая будет использоваться в качестве суффикса имени файла, переименовываем сохраняемую копию и, наконец, сжимаем ее с помощью *gzip*.

Возможно, вы пожелаете выполнять этот сценарий через *cron*, но в том виде, как он здесь представлен, его нельзя выполнять чаще, чем раз в сутки, иначе сжатая резервная копия будет перезаписана, т. к. в имени файла отражена только текущая дата, но не время суток. Если вы хотите выполнять этот сценарий чаще, то следует добавить к имени другие цифры, чтобы копии различались между собой.

Можно сделать много других усовершенствований. Например, вы можете сначала проверить размер журнального файла и делать его сжатую копию только в случае превышения заданного порога.

Даже несмотря на такое усовершенствование, на вашем разделе, содержащем журнальные файлы, с течением времени не останется места. Эту проблему можно решить, установив максимальное количество (например, 10) сохраняемых сжатых файлов журналов. Когда число журнальных файлов достигает установленного вами предела, самый старый из них уничтожается, и поверх него записывается очередной копируемый. Такая схема называется ротацией журналов. В некоторых дистрибутивах есть сценарии типа *savelog* или *logrotate*, автоматически производящие такую операцию.

Завершая обсуждение этой темы, отметим, что в новейших дистрибутивах, например SuSE, Debian и Red Hat, уже есть встроенные сценарии управления файлами журналов для *cron*; они значительно сложнее представленного здесь маленького сценария.

Управление службами печати

Система печати в Linux довольно сложна по сравнению со службами печати, имеющимися в большинстве персональных компьютеров. Она позволяет одновременно печатать документы нескольким пользователям, при этом каждый пользователь может посылать на печать документы из одного или нескольких приложений, не дожидаясь окончания печати предшествующих документов. Система печати корректно обрабатывает файлы, подлежащие печати на различных типах печатающих устройств, подключенных к ком-

пьютеру разными способами. При сетевой печати можно создавать файлы на одной машине и выводить их на принтер, управляемый другой машиной.

Прежде чем вникать в механизм системы печати Linux, хотим обратить ваше внимание на сайт *www.linuxprinting.org*, содержащий весьма полную информацию о печати в Linux. Туда следует обратиться при возникновении проблем или вопросов, касающихся печати, на которые не сможет ответить данная глава.

Когда вы нажимаете кнопку Print в каком-либо приложении или вводите для печати документа такую команду, как *lpr*, весь процесс происходит без помех. Однако документ не отправляется прямо на принтер, который может быть занят в данный момент. Вместо этого документ сохраняется во временном файле в каталоге, который называется каталогом спулинга принтера. Это предполагает, что документы поочередно изымаются из этого каталога по мере освобождения принтера. У каждого принтера имеется свой каталог спулинга.

В начале своей работы Linux запускает демон печати (независимо выполняющийся процесс) с именем *lpd*. Этот процесс находится в ожидании появления файлов в каталогах спулинга. Когда процесс обнаруживает файл, то создает копию самого себя. Новый *lpd* берет на себя управление спулингом печати, в который был помещен файл, и помещает файл в очередь печати. Он не пошлет на принтер новый файл, пока не будет закончена печать последнего файла. Главный *lpd* запускает новые *lpd* для каждого каталога спулинга, в который направлен файл. Таким образом, одновременно может выполняться столько демонов, сколько есть активных каталогов спулинга, плюс один главный *lpd*. Каждый подчиненный *lpd* продолжает действовать, пока не очистится его каталог спулинга.

В процессе установки Linux порт принтера системы связывается с устройством, поименованным в каталоге */dev*. Вы должны связать имя этого устройства с удобными названиями принтеров, используемыми в командах печати. Это делается в файле свойств принтера, носящем имя */etc/printcap*.

Другая важная задача управления печатью – обеспечить наличие фильтров для *lpd* при форматировании документов для печати. Эти фильтры тоже указаны в файле */etc/printcap*, и на протяжении этого раздела мы будем много о них говорить.

Есть несколько пакетов поддержки печати для Linux. В большинстве дистрибутивов используется производный от BSD пакет с демоном печати *lpd*. Эти пакеты содержат набор утилит и страниц руководств для поддержки в Linux традиционной печати в стиле Unix. В системе печати BSD меньше административных и пользовательских средств, чем, скажем, в системе управления печатью Unix System V (в которой используется демон *lpsched* или *lprng*), но каждый пользователь может управлять файлами, посылаемыми им на печать. В этом разделе описываются установка и конфигурирование пакета поддержки печати BSD. (Различные утилиты печати описаны в разделе «Печать» главы 9.)

Существует новая система под названием Common Unix Printer System (CUPS), которая намеревается подчинить себе сферу печати если не в Unix, то в Linux. Однако в настоящее время очень немногие дистрибутивы устанавливают CUPS – система печати BSD по-прежнему применяется повсеместно. По этой причине мы сосредоточили свое внимание на прежней системе, но вкратце рассмотрим CUPS далее в этой главе.

В некоторых дистрибутивах Linux есть средство администрирования печати, облегчающее установку принтеров и управление ими благодаря графическому интерфейсу пользователя. Такие средства документированы их поставщиком. Они управляют печатью посредством тех же программ и файлов, которые мы намерены описать, но предоставляют при этом меньше возможностей. Вначале они могут очень облегчить вам жизнь, но не всегда все делают правильно. Если вы пожелаете изменить настройки, сделанные с помощью этих инструментов, или улучшить производительность, то вам все равно придется пройти через процедуры, описываемые в данном разделе.

Проверка оборудования для печати

Прежде чем устанавливать службы печати, убедитесь, что печатающие устройства находятся в активном режиме. Если у вас на машине установлена еще какая-нибудь операционная система, например Microsoft Windows, проверьте с ее помощью правильность подключения и работы печатающих устройств, прежде чем загружать Linux. Если вам удалось напечатать документ из другой операционной системы, то тем самым сразу устраняется один из главных источников переживаний и головной боли. Аналогичным образом, если вы собираетесь использовать сетевые службы печати, то предварительно нужно проверить подключение к сети и функционирование всех сетевых протоколов.

Здесь уместно сказать несколько слов о так называемых GDI-принтерах (или Windows-принтерах). GDI-принтеры – это «безмозглые» принтеры в полном смысле этого слова: из них удален «мозг», т. е. устройство внутренней обработки, формирующее страницу по переданным ему данным. Эту задачу выполняет драйвер принтера на компьютере. Сам принтер состоит лишь из механизма печати и небольшой программы, которая им управляет. Естественно, драйверы таких устройств есть обычно только в Microsoft Windows (графическая подсистема которой называется GDI, откуда и происходит название этих принтеров), поэтому надежд на то, что удастся заставить такой принтер работать с Linux, весьма мало.

Для того чтобы установить службы печати, необходимы привилегии суперпользователя. Суперпользователь является единственным пользователем, кроме демона печати *lpd*, обладающим правом непосредственного вывода на принтер путем направления выходных данных на соответствующее устройство вывода. Остальные пользователи не могут посылать данные непосредственно на принтер и должны использовать принтеры посредством утилит печати.

Прежде чем начать, можно воспользоваться привилегиями суперпользователя и проверить, что назначенные в системе файлы устройств действитель-

но связаны с физическими устройствами. Используя перенаправление, отправьте короткий тестовый текстовый файл непосредственно на принтер. Допустим, ваш принтер подключен к первому параллельному порту, при этом имя его устройства, вероятно, */dev/lp0* или */dev/lp1,* в зависимости от установки. Следующая команда выводит некоторый текст для проверки печатающего устройства, который мы перенаправим на принтер. (Если у вас старый принтер с поддержкой PostScript, то вместо этого вам, возможно, во избежание неприятностей понадобится послать небольшой файл в формате PostScript. Новые PostScript-принтеры обычно в состоянии самостоятельно произвести необходимую конверсию.)

```
lptest > /dev/lp1
```

Утилита *lptest* предназначена для того, чтобы проверить правильность работы ASCII-принтера или терминала. Она посылает заранее подготовленный файл, составленный из 96 символов ASCII в последовательности, создающей эффект ряби или спиральных полос. По умолчанию в Linux *lptest* выдает 16 000 символов по 79 символов в строке, для печати которых потребуется более одной страницы. Если запустить *lptest* без аргументов, она выполнит вывод на экран, и вы увидите то, что должно было быть напечатано на принтере. Команда *lptest* позволяет уменьшить количество выводимых колонок и строк. Например, чтобы вывести 6 строк по 35 символов, выполните команду

```
lptest 35 6
```

Должно получиться примерно следующее:

```
!"#$%&'()*+,-./0123456789:;<=>?@ABC
"#$%&'()*+,-./0123456789:;<=>?@ABCD
#$%&'()*+,-./0123456789:;<=>?@ABCDE
$%&'()*+,-./0123456789:;<=>?@ABCDEF
%&'()*+,-./0123456789:;<=>?@ABCDEFG
&'()*+,-./0123456789:;<=>?@ABCDEFGH
```

Это достаточно короткий вывод, чтобы можно было увидеть результат перенаправления вывода на принтер, не расходуя при этом много бумаги, и достаточно длинный, чтобы выявить многие очевидные проблемы при печати.

Конечно, для вывода файла на печать можно воспользоваться и командой *cat*. Чтобы послать тестовый файл в формате PostScript на принтер, поддерживающий PostScript, введите следующую команду:

```
cat testfile.ps > /dev/lp1
```

Если у вас принтер для последовательного порта, направьте вывод в последовательный порт, к которому он подключен. Для первого последовательного порта (COM1 в MS-DOS) попробуйте что-то вроде:

```
lptest > /dev/ttyS0
```

или:

```
lptest > /dev/ttyS0
```

Проверьте, что вы выбрали правильный порт, и не пытайтесь вывести файл на мышь последовательного порта. Если ваш принтер подключен ко второму последовательному порту, к нему нужно обращаться как к */dev/ttyS1* или */dev/ttys1*.

Если у вас установлено устройство постраничной печати с буферизацией неполной страницы, то после окончания печати вам может потребоваться перевести его в автономный режим и нажать кнопку подачи страницы, чтобы напечатать последнюю неполную страницу. Не забудьте после этого вернуть принтер в активный режим. (Для такого принтера в Linux можно организовать постоянную отправку на принтер символа подачи листа, если отредактировать */etc/printcap* или приписывать этот символ к концу файла через фильтр принтера. Далее мы обсудим эти возможности.)

Если ваш маленький тест привел к появлению текста «лесенкой» (текста, выглядящего, примерно как в следующем примере), продолжающегося за пределами страницы, значит, принтер не вставляет в конце каждой строки символ возврата каретки:

```
!"#$%&'()*+,-./0123456789:;<=>?@ABC
                                  "#$%&'()*+,-./0123456789:;<=>?@ABCD
                                                                     #$
```

Можно выяснить, почему это происходит. В текстовых файлах Unix каждая строка заканчивается символом новой строки (или подачи строки, код ASCII равен 10). В MS-DOS символы новой строки и возврата каретки используются в паре. Следовательно, ваш принтер настроен на конец строки в стиле MS-DOS с обоими символами в конце каждой строки. Чтобы печатать текстовый файл из Unix, нужно установить фильтр принтера так, чтобы он соответствующим образом обрабатывал конец строки Unix, либо изменить настройку принтера, чтобы он возвращал каретку при получении символа новой строки. Обычно это делается установкой dip-переключателей в соответствующее положение. Посмотрите инструкцию к вашему принтеру. (Будьте осторожны при изменении характеристик принтера, если вы используете несколько операционных систем.)

Проблем с «лесенкой» не будет, если ваш принтер использует язык описания страниц, такой как PostScript (повсеместно распространенный язык структуры страницы, разработанный Adobe), и вы всегда фильтруете обычный текст в этот формат перед тем, как его напечатать. Фильтрация описывается далее в этой главе.

Подготовка информационных ресурсов

Итак, вы настроили свой принтер и подключили его к компьютеру. Соберите вместе печатную документацию, относящуюся к выводу на принтер (по крайней мере, страницы руководства по утилитам печати и описанные здесь файлы, а также файл Printing HOWTO). Полезно также иметь на руках техническое описание вашего принтера. Сейчас оно часто не предлагается при покупке принтера, но обычно его можно загрузить с FTP-сайта или веб-сай-

та производителя. При получении этих данных посмотрите, нет ли там также документации (такой как список ошибок, выдаваемых принтером, с описанием их значений), которая поможет в устранении неполадок принтера и его обслуживании. Большинство изготовителей принтеров предлагает также техническое руководство по принтеру, которое можно приобрести отдельно. Оно может одновременно являться и руководством по техническому обслуживанию, но необязательно.

Например, на веб-сайте Hewlett Packard (*http://www.hp.com/cposupport/software.html*) можно получить таблицы технических характеристик принтеров, спецификации изделий, данные по конфигурированию портов, а также файлы PostScript, PCL и HP-GL для тестирования принтеров (и фильтров). Кроме того, можно получить описание управляющих последовательностей, которые посылаются на принтеры с целью изменения режима их работы, а также документы, в которых сказано, как интегрировать базирующиеся на *lpd* службы печати с пакетом HP JetAdmin (и с его помощью с принтерами в сетях Netware).

Прежде чем приступить, сделайте глубокий вдох и наберитесь терпения. Конфигурирование служб печати – это умение, для выработки которого требуется время. Если ваше оборудование достаточно стандартно и вам удалось быстро и эффективно установить и сконфигурировать службы печати с помощью какой-либо из новомодных утилит управления печатью, то можете себя поздравить! Но учтите, что вам, возможно, удастся осуществить более тонкую настройку установок для получения лучших характеристик работы с помощью описываемых нами ниже процедур или при использовании фильтров и утилит, написанных специально для полной поддержки всех функций вашей модели принтера. Если вы решите пересмотреть успешно проведенную установку принтера, обязательно сделайте заметки о проведенных изменениях, чтобы вернуться в исходное состояние, если сделанные изменения не приведут к желаемому результату.

Выбор программ печати

Для печати из Linux нужно установить систему печати BSD (или альтернативную ей). Она предоставляет базовые средства работы, но не поддерживает современные принтеры. Фактически, она была разработана для поддержки устройств построчной печати (отсюда и название «line printing daemon»), использовавшихся в вычислительных центрах в 60–70-х годах. Для поддержки современных принтеров есть мощные дополнительные пакеты, обеспечивающие возможности, считающиеся важными большинством пользователей. (Пакеты, о которых мы здесь говорим, есть на FTP-сайте *ftp://ftp.ibiblio.org* и его зеркалах.)

В этом разделе мы обсудим некоторые важные пакеты, поддерживающие современные службы печати. Мы предполагаем, что у вас на машине будут установлены, по крайней мере, форматтер *groff*, пакет форматирования страниц Ghostscript и пакеты фильтров *GNU Enscript*, описываемые в главе 9. В большинство дистрибутивов Linux включены эти и другие утилиты фор-

матирования и печати. Если в вашем дистрибутиве их нет, можете загрузить их с FTP-сайтов Linux или взять с CD-ROM другого дистрибутива.

Имеет значение источник получения пакетов форматирования и фильтрации. Например, если вы получите Ghostscript из европейского дистрибутива, то он, возможно, по умолчанию предполагает формат листа A4, а не размер 8,5×11 дюймов, как в дистрибутивах из США. В любом случае значение по умолчанию легко переопределить, передав фильтру параметр команды *lpr*. Можно также скомпилировать утилиты из исходного кода, предварительно внеся изменения.

Тенденция технологии печати состоит в отходе от вывода, ориентированного на символы, в сторону работы с языками описания страниц (PDL), обеспечивающими сложное управление графикой и шрифтами. Наиболее популярным среди PDL является язык PostScript, широко принятый в сообществах Unix и Интернет. Главной причиной такого признания является пакет Ghostscript – реализация PostScript, авторские права на которую имеет Aladdin Enterprises. Имеется также версия, распространяемая по общественной лицензии GNU через Фонд свободного программного обеспечения вместе с большой библиотекой шрифтов, которые можно использовать с любой версией Ghostscript, а также другими интерпретаторами PostScript. Ghostscript незаменим, если вы не ограничиваетесь символьной печатью, и он допускает возможность легкого расширения.

В Ghostscript реализованы почти все команды языка PostScript, а также утилиты просмотра, такие как Ghostview, позволяющие просматривать документы PostScript в X Window. Кроме того, доступны прекрасные фильтры, преобразующие вывод PostScript в другие языки печати, например PCL, разработанный Hewlett-Packard, а также в растровый вид, пригодный для вывода на струйные, матричные и лазерные принтеры. Пакет Ghostscript поддерживает шрифты PostScript Adobe type 1 и 3, а также предоставляет ряд утилит преобразования графических форматов и фильтрации. Он даже может создавать файлы в формате PDF, т. е. удовлетворяющие спецификации Adobe Portable Document Format.

Одного лишь Ghostscript может оказаться недостаточно, поскольку он не обеспечивает такого управления принтером, которое позволяет переключаться между режимами PostScript и текстовым. Хотя в Ghostscript есть фильтр, осуществляющий эту и другие возможности, мы опишем фильтр *nenscript*, который в большинстве случаев удовлетворяет критериям простоты, гибкости и надежности.

В типичной Linux-системе вначале для форматирования и создания выходных данных на PostScript используется *groff*, а затем Ghostscript обрабатывает данные для вывода на печать и экран.

Проверка утилит печати

Возможно, вы хотите установить пакет форматирования TEX. Даже если вы не станете устанавливать полный дистрибутив TEX, нужно установить хотя

бы утилиту *xdvi*, чтобы иметь возможность просматривать в X Window документы TEX и файлы Texinfo (если только вы не установили KDE, где есть более дружественный к пользователю инструмент *kdvi*). Если у вас аллергия на пакет Ghostscript или вам нужно использовать встроенные шрифты принтера для эффективной передачи данных и быстрой печати, то имеются другие фильтры, преобразующие выдачу DVI в формат PostScript (*dvips*) или PCL (*dvilj*).

Пакет Lout также достоин внимания как эффективное и компактное средство форматирования документов для вывода в PostScript. Он поддерживает Level 2 PostScript и соглашения Adobe по структуризации (Adobe Structuring Conventions), требует относительно немного памяти и поставляется с хорошей документацией, позволяющей быстро его освоить. Lout не создает промежуточного выходного формата; он преобразует входную разметку непосредственно в выходной PostScript.

Для поддержки графики и утилит X Window System вам, возможно, понадобятся дополнительные средства, которые могут иметься на вашем дистрибутиве. Собрание свежих версий самых популярных пакетов поддержки печати для Linux можно найти на *ftp://ftp.ibiblio.org* в архиве */pub/Linux/system/printing*. Пакеты *netpbm* и *pbmplus* поддерживают преобразования большого количества различных графических форматов файлов. (Такие форматы требуется преобразовывать в PostScript, прежде чем выводить на печать.) В пакете Ghostview есть средства просмотра файлов PostScript в среде X Window System, а также поддержки PostScript и PDF для других пакетов, например веб-броузера.

Особого упоминания заслуживает пакет ImageMagick, описываемый в главе 9. Он позволяет выводить в окне X большое количество графических форматов и осуществлять их взаимное преобразование. (При необходимости вывести изображение в формате PostScript он использует Ghostview и Ghostscript.) Большинство графических файлов, которые можно напечатать, можно также вывести на экран с помощью ImageMagick.

Пакет «волшебных» фильтров может уберечь вас от многих неприятностей при конфигурировании и поддержке различных выходных форматов документов. Мы коснемся пакета волшебных фильтров APSfilter, однако вам может больше понравиться пакет Magic-Filter. Оба пакета можно получить из архива *ftp://ftp.ibiblio.org*. (См. подробнее о волшебных фильтрах в разделе «Волшебные фильтры: APSfilter и другие» далее в этой главе.)

Если вам нужна поддержка факсимильных устройств, то утилиту *tiffg3* можно использовать с Ghostscript для выдачи факсимильных файлов в формате Group III. Для управления под Linux факс-модемом Class 1 или Class 2 можно воспользоваться пакетом *efax*, который поставляется в составе многих дистрибутивов, либо можно установить и сконфигурировать более мощные, но и более сложные пакеты FlexFax или HylaFax.

Существуют дополнительные средства, поддерживающие двустороннюю печать на лазерных принтерах, а также пакеты, преобразующие PostScript в менее распространенные языки управления принтерами, поддерживаемые,

например, устройствами Canon и IBM Proprinter. Есть пакет, поддерживающий печать китайского языка на лазерных принтерах и растровых устройствах. Большинство этих пакетов не влияет непосредственно на управление службами печати, поэтому подробно мы их здесь не описываем, но если вы собираетесь их использовать, самое время их установить.

Для блага ваших пользователей подготовьте все страницы руководства для устанавливаемых вами пакетов по завершении их установки. После этого выполните */sbin/mkwhatis* (*/usr/bin/mandb* в Debian), чтобы построить индексный файл страниц руководства, облегчающий поиск информации в электронных справочниках. Некоторые пакеты, например Ghostscript, предоставляют дополнительную документацию, которую можно распечатать или сделать доступной в системе. (В дистрибутивах Linux эти документы часто отсутствуют, но можно загрузить их по FTP с сайтов разработчиков. Например, архивы GNU Фонда свободного программного обеспечения доступны на анонимном FTP-сервере *ftp://GNU.ai.mit.edu*.)

Настройка файла printcap

Суть конфигурирования принтера состоит в создании нужных записей в файле свойств принтера */etc/printcap*. Простая запись свойств лазерного принтера HP Laserjet 4MP, подключенного к первому (двунаправленному) параллельному порту на шине ISA, может, например, выглядеть так:[1]

```
ljet|lp|ps|Postscript|600dpi 20MB memory|local|LPT1:\
     :lp=/dev/lp0:rw:\
     :sd=/var/spool/lpd/ljet4:mx#0:mc#0:pl#72:pw#85:\
     :lf=/var/log/lpd-errs:if=/usr/local/cap/ljet4:
```

Не пугайтесь. Прочтя последующие разделы, вы станете с легкостью разбираться в файлах свойств принтера.

В файле */etc/printcap* должны находиться данные обо всех принтерах, портах принтеров или их адресах, которые использует ваша система, будь они последовательные, параллельные, SCSI, USB или сетевые. Следите, чтобы в нем находили отражение все изменения в аппаратных устройствах. И, как всегда, не забывайте, что некоторые изменения в аппаратной конфигурации должны производиться только при отключенном питании.

Формат файла printcap

Формат файла printcap можно кратко описать следующими правилами:

- Строки комментариев начинаются с символа (#).

[1] В нескольких примерах данной главы мы используем ljet4. Учтите, что есть несколько вариантов модели HP Laserjet 4. Некоторые из них являются только принтерами PCL5, а другие используют PostScript. Если не знаете, что существуют разные модели, может оказаться весьма затруднительным отладить фильтр принтера, который, допустим, ожидает PostScript, в то время как Ghostscript посылает ему данные в формате PCL5.

- Каждая «строка» файла printcap определяет принтер. Строка, оканчивающаяся символом обратной косой черты (\), продолжается на следующей строке. Следите, чтобы за косой чертой в строке не оказалось символов табуляции или пробелов. Для удобочитаемости принято начинать строку продолжения с отступа. Несколько определений принтера могут относиться к одному и тому же физическому устройству, применяя одни и те же или разные фильтры.
- Поля в строке разделяются символом двоеточия (:); поля могут быть пустыми. Однако строка записи не может являться пустым полем.
- Не принято ставить двоеточие перед первым полем записи.
- В первом поле строки записи содержатся имена принтера, разделяемые вертикальной чертой (|). В приведенном примере записи поле имен следующее:

  ```
  ljet|lp|ps|Postscript|600dpi 20MB memory|local|LPT1
  ```

  Об именах принтера подробно рассказывается в следующем разделе. Вы должны создать подкаталоги */var/spool/lpd* с именами, совпадающими с первым идентификатором принтера во всех записях файла printcap. Однако реальный каталог спулинга печати назначается переменной `sd` в записи printcap. Если переменная `sd` не указывает на фактический каталог спулинга принтера, то файлы, посылаемые на определенный таким образом принтер, будут просто исчезать.
- В printcap должна быть запись хотя бы для принтера, назначаемого по умолчанию. Принтер с именем `lp` используется системой в качестве принтера по умолчанию. Не путайте `lp` – имя принтера по умолчанию, с `lp` – переменной локального принтера, описание которой следует ниже. Мы советуем использовать `lp` как псевдоним (одно из имен за символами |), а не главное имя принтера (которое идет первым в списке), чтобы можно было без труда менять принтер, используемый по умолчанию.
- Для всех локальных принтеров должна быть установлена переменная `lp`. В предыдущем примере эта переменная устанавливалась в такой части записи:

  ```
  lp=/dev/lp0
  ```

- Навязчивой, но иногда практичной рекомендацией некоторых администраторов является следование алфавитному порядку при расположении записей в файле printcap.

Имена принтеров

Принято начинать записи принтеров с короткого имени принтера, за которым следует, по крайней мере, одно более полное имя и еще одно более длинное описательное. Таким образом, `ljet` и `PostScript` являются именами принтера, для которого строка имен выглядит так:

```
ljet|lp|ps|Postscript|600dpi 20MB memory|local|LPT1:
```

Документы могут выводиться на любое имя принтера, указанное в записи */etc/printcap*.

Принтерам можно давать имена в соответствии с их моделью (HP, Epson), либо типом принтера (PS, PCL), либо особым режимом работы. Например, для принтера DeskJet 540 в файле printcap должно иметься два определения: одно для черно-белой печати, другое – для цветной. Для его поддержки подойдут, вероятно, фильтры для DeskJet 500 или 550C. Для простоты администрирования можно присваивать принтерам имена, являющиеся именами фильтров или параметров фильтров, используемых для конкретного устройства. Поэтому если у вас один принтер LaserJet 4 и фильтр ljet4 будет использоваться только с ним, то этому принтеру можно дать логическое имя `ljet4`. Аналогично матричному принтеру можно дать имя `72dpi` в строке определения принтера с низким разрешением и имя `144dpi` – для доступа с более высоким разрешением.

При использовании утилиты администрирования принтеров, включенной в ваш дистрибутив Linux, от вас может потребоваться следовать определенным правилам создания записей в printcap, чтобы эта программа могла работать. Например, если в дистрибутиве Red Hat вы пользуетесь менеджером принтеров, который работает в панели администратора, может потребоваться, чтобы первым именем в первой действующей записи файла printcap было `hp`. Из-за этого при необходимости изменить принтер, установленный по умолчанию, вы должны сделать запись нового принтера, устанавливаемого по умолчанию, первой в списке, удалить имя `hp` из записи прежнего принтера по умолчанию и сделать его первым именем нового принтера по умолчанию. Чтобы избежать путаницы в очередях спулинга, нужно отказаться от установки каталога */var/spool/lpd/lp* и создать новый каталог с фактическим именем каталога спулинга, соответствующим тому имени, по которому вы реально будете обращаться к принтеру. Таким образом, если вы хотите отправлять свои файлы печататься на принтер с именем `moa`, вам нужно создать каталог */var/spool/lpd/moa* с соответствующими правами доступа и указать этот каталог в качестве спулинга для этого принтера. Установка каталогов принтера описана в следующем разделе.

Другие переменные из файла printcap

В файле printcap можно определить ряд переменных. Большинство переменных служит для того, чтобы задать параметры страницы, файлы и каталоги, фильтры, параметры канала связи и возможности удаленного доступа. Если вам нужно редактировать файл printcap на новой системе, прочтите страницу руководства по printcap, чтобы использовать правильные имена переменных. Если переменные, заданные в записи printcap, не поняты системой, они передаются на обработку в фильтр.

Описанные здесь переменные, используемые в printcap, перечислены примерно в порядке их важности. Некоторые переменные являются булевыми и считаются установленными, если только они присутствуют. Другие переменные устанавливаются оператором присваивания (=) или оператором чи-

словой величины (#) и значением; переменная предшествует оператору, а строка или число следуют за ним. Примеры переменных, описанных в следующем списке, включены в искусственный образец файла */etc/printcap*, который приведен ниже. На странице руководства по printcap приведен более полный список переменных, распознаваемых *lpd*:

`sd`

Определяет используемый принтером каталог спулинга. Каталоги спулинга должны находиться в одном и том же дереве каталогов (на быстром жестком диске), которым обычно является */var/spool*. Файлы спулинга определяются даже для удаленных принтеров. Имя каталога должно совпадать с одним из имен, присвоенных обслуживаемому им принтеру.

`lp`

Назначает локальное устройство печати, обычно подключаемое через параллельный порт, последовательный порт или интерфейс SCSI. Переменной `lp` должно быть присвоено имя файла устройства в каталоге */dev*, который может являться ссылкой на физическое устройство. Переменной `lp` должно быть присвоено значение, если имеется локальный принтер. Этой переменной не должно присваиваться значение, если присвоено значение переменной `rp` (т. е. менеджер спулинга печати находится на другой машине).[1] Если `lp` ссылается на устройство последовательного порта, в переменной `br` должна быть указана скорость доступа.

`lf`

Задает имя файла журнала для регистрации сообщений об ошибках. Эта переменная должна быть установлена для всех принтеров, которые обычно используют один и тот же журнал ошибок. В записях об ошибках указывается имя принтера и может сообщаться о проблемах со средой печати пользователя, конфигурацией машины, используемым каналом связи и иногда самим устройством печати.

`rw`

Эта переменная должна быть задана, если принтер может посылать данные обратно машине через указанный файл устройства. Переменная `rw` сообщает *lpd*, что устройство должно быть открыто как для чтения, так и для записи. Это полезно, например, для последовательных или SCSI PostScript-принтеров, поскольку они могут возвращать *lpd* достаточно полезные сообщения, которые тот записывает в журнал регистрации ошибок.

---

[1] Особая ситуация возникает, если адресуемый принтер действительно является сетевым устройством, т. е. имеет собственный IP-адрес. В этом случае переменной `lp` присваивается имя фиктивного файла, используемого для временной блокировки, когда сетевой принтер занят. В документации к сетевому принтеру должна быть описана процедура установки служб печати и управления ими для обеспечения доступа к нему.

`mx`

Задает максимальный размер задания печати в спулинге. Значение, равное нулю, снимает ограничения на размер (по умолчанию `mx#0`), а все другие значения определяют максимальный размер файла в блоках. Как правило, не требуется устанавливать границу, однако при желании можно установить размер, несколько меньший предполагаемого минимального свободного пространства на диске.

`if`

Задает используемый входной фильтр. Если входной (`if`) или выходной (`of`) фильтр не задан, система использует по умолчанию фильтр */usr/sbin/lpf*. Для некоторых символьных MS-DOS-принтеров этого достаточно. Другие полезные фильтры предоставляются утилитами форматирования. Есть также несколько гибких пакетов с «волшебными фильтрами», которые определяют тип применяемого фильтра по содержанию передаваемых им данных (обычно правильно). См. следующий раздел «Фильтры печати».

`of`

Задает используемый «выходной» фильтр. Если вы задаете значение переменной `of` и не задаете значение переменной `if`, система использует фильтр, когда открывается устройство. Все стоящие в очереди задания посылаются на печать без повторного применения фильтра, пока очередь не опустеет (и *lpd* удалит файл блокировки из каталога спулинга). Обычно от этого мало пользы, но может применяться для таких целей, как отправка факсов в факс-модем при соединении по коммутируемой линии.

Если вы присваиваете значения обеим переменным, `if` и `of`, то фильтр для `if` нормально обрабатывает файл, но фильтр, заданный для `of`, печатает страницу-разделитель перед тем, как применяется входной фильтр. Особенно трудно эффективно использовать входные и выходные фильтры одновременно с одной и той же очередью печати.

`br`

Задает скорость передачи данных для последовательного порта. Это значение должно быть задано, если принтер подключен через последовательный порт. Знак `#` предшествует числовой величине, выражающей скорость передачи данных в битах в секунду (не совпадающей на самом деле со скоростью двоичной передачи, которая является эффективной скоростью передачи, в отличие от максимальной скорости). Заданная скорость не должна превышать ограничение аппаратного устройства. Например, если последовательный порт поддерживает скорость 57,6 Кбит/с, а принтер может обрабатывать 28,8 Кбит/с, назначенная скорость не должна превышать нижней границы (вероятно, `br#19200`). Поддерживаемые значения являются обычными кратными для последовательного порта: 300, 600, 1200, 2400, 4800, 9600 и т. д. Если вы присваиваете значение `br`, могут устанавливаться дополнительные условия передачи данных, но в типичной установке Linux они, по большей части, не нужны. Скорее всего, задачам вывода на печать удовлетворят значения, устанавливаемые по

умолчанию, однако если вы собираетесь печатать через последовательный порт, посмотрите описание переменных br, fc, fs, xc и xs на странице руководства по printcap.

pl

Задает высоту страницы как число строк с использованием символов шрифта, устанавливаемого по умолчанию для символьных устройств (и принтеров, способных работать в символьном режиме). Например, pl#66 соответствует 11-дюймовой странице при шести строках на дюйм. Эта величина позволяет оставить место для обрезания и учитывает ограничения некоторых других устройств, например струйных принтеров, которые не могут печатать по краям листа. Обычно используется вместе с переменной pw.

pw

Задает поддерживаемую ширину страницы в символах с использованием символов шрифта, устанавливаемого по умолчанию для символьных устройств. pw устанавливается аналогично переменной pl. Например, pw#85 соответствует 10 символам на дюйм при ширине страницы 8,5 дюймов.

px

Задает число пикселов по оси X при отправке файла растрового изображения на растровое устройство.

py

Задает число пикселов по оси Y при отправке файла растрового изображения на растровое устройство.

sh

Подавляет печать заголовков и страниц-разделителей. В большинстве случаев должна быть установлена.

rp

Определяет имя используемого удаленного принтера. Эту переменную нельзя задавать, если для того же самого принтера задана переменная lp. Управление принтером осуществляется на удаленной машине, указанной в переменной rm, которую в этом случае тоже необходимо задать. Обычно вместе с этими переменными задаются только переменные, определяющие спулинг и регистрацию ошибок. См. пример ниже.

rm

Задает удаленную машину, управляющую удаленным принтером, на который вы осуществляете печать. Присваиваемый rm идентификатор машины должен быть известен сетевым службам (например, он должен быть указан в */etc/hosts* или известен через NIS).

rs

Разрешает доступ к локальным принтерам только тем пользователям, у которых есть в системе учетная запись.

rg

Задает ограниченную группу тех, кто может использовать принтер. Например, чтобы зарезервировать этот принтер только за суперпользователем, введите rg=root.

sf

Подавляет посылку на принтер символа подачи листа в конце печатаемого файла, которая осуществляется по умолчанию.

ff

Устанавливает символ или строку, которые должны указывать устройству на необходимость подачи листа. По умолчанию это <Ctrl>+<L> (эквивалентно ff='\f'), что используется большинством устройств.

fo

Задает вывод на устройство символа подачи листа перед отправкой файла.

mc

Задает максимально допустимое число печатаемых экземпляров. Допустимы те же значения, что и для переменной mx; обычно устанавливается неограниченное количество экземпляров (mc#0), являющееся значением по умолчанию.

sc

Запрещает вывод нескольких экземпляров (эквивалентно mc#1).

В примере 8.1 приведен образец файла printcap, демонстрирующий использование многих из переменных, обсуждавшихся выше. В нем содержится запись для удаленного принтера. Вывод на принтер с именем *hp* (установленный принтером по умолчанию, поскольку это первая запись) посылает документы на spigot.berk.ora.com, где они попадают в очередь печати *lp*.

Пример 8.1. Пример файла /etc/printcap

```
# Fare well, sweet prints.
hp|bat|west|spigot|berkeley|TI MicroLaser Turbo:\
     :mx#0:rp=lp:\
     :lp=:sd=/var/spool/lpd:rm=spigot.berk.ora.com:\
     :lf=/var/log/lpd-errs:
# To the print room
kiwi|810|rint|Big Apple|Apple 810 via EtherTalk:\
     :lp=/var/spool/lpd/kiwi:sh:\
     :sd=/var/spool/lpd/kiwi:pl#72:pw#85:mx#0:\
     :lf=/var/log/lpd-errs:if=/usr/local/cap/kiwi:

# big bird--agapornis via shielded serial access
samoa|S|PostScript|secure|QMS 1725 by serial adapter:\
     :lp=/dev/tty01:br#38400:rw:xc#0:xs#0400040:sh:\
     :sd=/var/spool/lpd/samoa:pl#72:pw#85:mx#0:mc#0:\
     :lf=/var/log/lpd-errs:if=/usr/local/cap/samoa:
# agapornis via printer room subnet (standard access)
moa|ps|QMS 1725 via Ethernet:\
```

```
:lp=/var/spool/lpd/moa/moa:rm=agapornis:rp=samoa:\
:sd=/var/spool/lpd/moa:mx#0:sh:\
:lf=/var/log/lpd-errs:if=/usr/local/cap/samoa:
```

Конфигурирование Ghostscript

Ghostscript включается в стандартные пакеты Linux. Это важная утилита в среде X Window System, но она полезна, даже если вы не работаете в X. Ghostscript может выводить графику на стандартный дисплей VGA, даже если не запущен оконный менеджер. Она может также обрабатывать и создавать файлы в формате PostScript при отсутствии графического дисплея. Вы можете узнать, какие устройства позволяет распознавать и форматировать ваша конфигурация Ghostscript, войдя в программу в интерактивном режиме. Если ввести:

```
$ gs
```

то программа Ghostscript должна загрузиться в интерактивном режиме и ждать дальнейших команд:

```
GS>
```

Теперь вы можете спросить, какие устройства Ghostscript может распознавать в текущей конфигурации, и Ghostscript выведет их список:

```
GS> devicenames ==
[/lips3 /ljet3d /djet500 /cdj970 /st800 /iwlo /x11alpha /la75plus
/bjc800 /ljet2p /djet820c /cdj880 /lp8000 /appledmp /la70 /bj200 /stp
/DJ9xxVIP /cdj670 /eps9high /vgalib /dl2100 /lp2563 /pjxl300 /DJ8xx
/cdj500 /epson /x11mono /lex2050 /lj5gray /pjetxl /DJ6xx /cdjcolor
/tek4696 /x11gray2 /lex5700 /ljetplus /paintjet /hpijs /uniprint
/t4693d4 /x11cmyk4 /lxm5700m /ljet4 /djet500c /cdj1600 /stcolor
/iwlq /x11cmyk /ln03 /lbp8 /ljet3 /deskjet /cdj890 /lq850 /iwhi /x11
/la75 /bjc600 /laserjet /AP21xx /cdj850 /epsonc /sxlcrt /la50 /bj10e
/hpdj /DJ9xx /cdj550 /eps9mid /lvga256 /declj250 /cljet5 /pjxl /DJ6xxP
/cdjmono /ap3250 /x11gray4 /lex3200 /lj5mono /pj /DJ630 /cdeskjet
/t4693d8 /x11cmyk8 /lex7000 /lj4dith /dnj650c /chp2200 /hl7x0 /t4693d2
/x11cmyk2 /lj250 /jpeggray /png256 /mgrgray4 /tiffg4 /ppmraw /okiibm
/cif /pnggray /mgrmono /tiffg3 /pnmraw /oki182 /nullpage /cgm8
/pcxcmyk /bitrgb /faxg4 /pgnmraw /necp6 /pxlmono /bmp16m /pcx256
/psrgb /faxg3 /pgmraw /jetp3852 /pswrite /bmp16 /pcxgray /psmono
/dfaxhigh /pbmraw /ibmpro /pdfwrite /bmpmono /mgr8 /tifflzw /plan9bm
/xes /ccr /miff24 /png16m /mgrgray8 /tiff12nc /pkm /r4081 /jpeg /png16
/mgrgray2 /tiffg32d /ppm /oki4w /cgm24 /pngmono /bitcmyk /tiffcrle
/pnm /oce9050 /pxlcolor /cgmmono /pcx24b /bit /faxg32d /pgnm /m8510
/epswrite /bmp256 /pcx16 /psgray /dfaxlow /pgm /imagen /bbox /bmpamono
/pcxmono /tiffpack /inferno /pbm /cp50 /omni /sgirgb /mgr4 /tiff24nc
/pkmraw /sj48]
GS> quit
$
```

Если у вас не запущена система X, и Ghostscript не может инициализироваться при вызове, сообщая, что не может открыть дисплей X, это значит, что первым устройством, которое загружал Ghostscript, было устройство X Window System; Ghostscript использует первое устройство в качестве устройства по умолчанию. Обойти эту проблему можно, задав какое-нибудь другое устройство, которое установлено в Ghostscript, например `gs -sDEVICE=epson`. Можно защитить себя от этой проблемы на будущее, указав в глобальной переменной окружения `GS_DEVICE` какое-либо другое устройство системы, которое может открыть Ghostscript.

Если у вас такое необычное устройство вывода, что Ghostscript по умолчанию его не поддерживает, нужно перекомпилировать Ghostscript для поддержки этого устройства или пропустить ваши выходные файлы через фильтр, преобразующий их в формат, приемлемый для вашего устройства вывода. Ghostscript поставляется с make-файлами, и компиляция проста, если следовать имеющейся в дистрибутиве документации по Ghostscript.

Чем больше графических утилит, менеджеров окон X, игр и приложений вы используете, тем более вероятно, что придется переустановить Ghostscript, чтобы удовлетворить вашим требованиям. Прежде чем запускать make-файл, поступивший с вашим пакетом, прочтите документацию Ghostscript. (Для перекомпиляции необходимо, чтобы в системе был установлен *gcc*.)

В переменной окружения `GSDIR` можно указать путь к команде *ghostscript* и определить переменные `GS_LIB`, если требуется скомпилировать утилиты Ghostscript и добавить их в инсталляцию. Например:

```
export GSDIR=/usr/bin
export GS_LIB=/usr/lib/ghostscript:/usr/local/lib/fonts:/usr/X11R6/fonts
```

Определите переменную `GS_LIB_DEFAULTS`, прежде чем осуществлять новую компиляцию Ghostscript; см. страницу руководства по *gs*.

В пакете Ghostscript содержатся также некоторые PostScript-программы, обеспечивающие полезные функции поддержки печати, в том числе некоторые сложные возможности печати, о которых мы здесь не рассказываем. В частности, файл *gs_init.ps* определяет общий режим работы Ghostscript. Дополнительные сценарии (фильтры, сценарии оболочки и т. д.) можно найти в каталогах */usr/lib/ghostscript* или */usr/ local/lib/ghostscript*. Может оказаться полезным изучить утилиту *ps2epsi.ps*, которая преобразует PostScript в инкапсулированный PostScript, и утилиту *ps2ascii.ps*, преобразующую файлы PostScript в простой текст.

Фильтры печати

Каждый документ, прежде чем попасть на принтер, проходит через фильтр, определяемый переменной `if` в файле printcap. Фильтр принтера можно найти в дистрибутиве Linux, приобрести у поставщика принтера, отыскать в Интернете или даже создать самостоятельно, заново или смастерив из имеющихся фильтров и утилит оболочки.

Входной фильтр можно использовать и для того, чтобы ограничить доступ к принтеру, т. е. разрешить использовать принтер только отдельным пользователям или группе пользователей либо пользователям отдельной машины. Типичным фильтром, назначаемым `if`, является исполняемый сценарий оболочки, обрабатывающий текстовые файлы, однако им может быть любая программа, принимающая поток входных данных и обрабатывающая его для вывода на принтер.

Все чаще в коммерческих дистрибутивах Linux фильтр создается интерактивно. Хотя обычно можно сделать фильтр и получше, может оказаться полезным использовать его в качестве отправной точки. Например, следующий фильтр (с именем */var/spool/lpd/ljet4/filter*[1]) – это сценарий оболочки, созданный дистрибутивом Red Hat на одной из наших систем на основе данных, сообщенных в окне менеджера печати, и допущений по умолчанию. В файл */etc/printcap* были внесены изменения, указывающие на необходимость использования этого фильтра, который стал отлично работать в нашей системе:

```
#!/bin/sh

DEVICE=ljet4
RESOLUTION=600x600
PAPERSIZE=letter
SENDEOF=

nenscript -TUS -ZB -p- |
if [ "$DEVICE" = "PostScript" ]; then
        cat -
else
        gs -q -sDEVICE=$DEVICE \
                -r$RESOLUTION \
                -sPAPERSIZE=$PAPERSIZE \
                -dNOPAUSE \
                -dSAFER \
                -sOutputFile=- -
fi

if [ "$SENDEOF" != "" ]; then
        printf ""
fi

exit 0
```

В этом фильтре нет ничего экзотического. Сначала в нем устанавливаются некоторые переменные, которые оказываются впоследствии аргументами команды Ghostscript. Сценарий пропускает данные через *nenscript* и переда-

[1] Для удобства работы с программой управления принтером лучше при установке системы печати поместить фильтр в каталог спулинга принтера. Однако вы можете держать фильтры печати и фильтры преобразования графических форматов в одном и том же каталоге, как это принято в Unix, например */usr/sbin* или */var/spool/lpd/filters*. Конечно, тогда каждый создаваемый вами фильтр должен иметь уникальное имя.

ет файлы PostScript утилите *gs* пакета Ghostscript. (В данном случае автоматически созданный фильтр никогда не вызовет *gs*, поскольку `DEVICE` в операторе `if` никогда не принимает значение «PostScript».) Если ваш принтер не выталкивает последнюю страницу, можно заставить его делать это, присвоив значение переменной `SENDEOF` в начале файла. Например:

```
SENDEOF='\f'
```

вызовет посылку на принтер символа подачи листа по достижении конца файла.

Можно модифицировать такой сценарий, вставив фильтр, специально разработанный для принтера LaserJet 4, например `ljet4`, и организовать печать файлов DVI для TEX, фильтруя их через *dvips* и посылая затем на Ghostscript. Описание процедуры создания фильтров для начинающих есть в Linux Printing HOWTO [84].

Если вы используете символьный принтер, ожидающий возврата каретки в конце каждой строки, это вызовет проблему с Linux, которая, следуя стилю Unix, заканчивает строку одним символом новой строки без возврата каретки. Чтобы обеспечить на таких принтерах корректную обработку символов перевода строки, фильтр должен вставлять символ возврата каретки. Это можно сделать, организовав такую обработку в фильтре, или использовать фильтр, в котором эта функция уже реализована.

Некоторые изготовители принтеров поставляют фильтры и утилиты для своих устройств, особенно в тех случаях, когда обычные решения едва ли могут адекватно использовать возможности принтера. Например, Hewlett-Packard поставляет пакет JetAdmin и фильтры для использования с принтерами LaserJet, работающими в сетях TCP/IP.

В состав пакета администрирования печати BSD входит фильтр по умолчанию */usr/sbin/lpf*. Этот фильтр не документирован, и им, возможно, лучше не пользоваться, если только вы не решите получить его исходный код на C и, исследуя его, изучить возможности программы. (Полный исходный код BSD есть в пакете lpr-secure в каталоге printing на *ftp://ftp.ibiblio.org*.)

Большинство задач фильтрации печати, которые могут у вас возникнуть, уже давно решены, и существуют готовые фильтры, которые могут удовлетворить ваши потребности. Выполнив команду

```
apropos filter
```

вы, вероятно, найдете несколько фильтров печати, установленных на вашей машине.

Сменить фильтр легко. Вам нужно лишь изменить значение (`if`) в файле */etc/printcap*, чтобы указать фильтр, который вам требуется, а затем убить и заново запустить *lpd*, что можно сделать с помощью утилиты *lpc*. Введите (от имени `root`):

```
lpc restart all
```

Утилита *lpc* сообщит обо всех процессах *lpd*, которые она убьет, и перезапустит *lpd*. Если в спулинге печати есть файлы, ожидающие вывода на принтер, *lpd* также сообщит о запуске демона *lpd* для этого принтера. Утилита управления печатью *lpc* описана далее в этой главе в разделе «Управление сервисами печати с помощью *lpc*».

Прежде чем установить новый фильтр печати, изучите страницу руководства для него, а затем пропустите через него несколько тестовых файлов. Мы обнаружили, что фильтры не всегда действуют согласно «рекламе» на странице руководства. Часто документ является устаревшим или фильтр был скомпилирован с использованием параметров конфигурации, отличных от предполагаемых в документе. Нет другого выхода, кроме как проверить, делает ли фильтр все то, что вы от него хотите, прежде чем «принять его на вооружение». В следующем разделе обсуждаются два хороших пакета фильтрации: *nenscript*, более новая версия традиционного для Unix фильтра *enscript*, и APSfilter.

Фильтр nenscript

nenscript – типичный современный фильтр для Linux. Если он есть в вашем дистрибутиве Linux, то помещен в каталог */usr/bin/nenscript*. Этот фильтр может находиться и в */usr/local/bin*. *nenscript* управляет верхними и нижними колонтитулами, поворотом текста и т. д. Он преобразует простой текст ASCII (вызывая Ghostscript) в формат PostScript согласно Adobe Structuring Conventions. Вывод посылается на принтер, указанный в пользовательской переменной окружения `NENSCRIPT` (если она установлена) или пользовательской переменной окружения `PRINTER`. Если не установлена ни одна из этих переменных, *nenscript* использует тот же принтер, что и принтер по умолчанию для *lpr*.

При вызове *nenscript* с параметром *-Z* он должен передавать файлы PostScript, не внося в них изменений. *nenscript* просматривает входной файл, и если первые два символа входного файла суть `%!`, то *nenscript* подавляет форматирование. Ввиду примитивности такой проверки ее легко обойти. Если два первых символа отличны от `%!`, например, когда первым является символ подачи страницы, то файл не будет распознан как PostScript, даже если он таковым является. Это может легко произойти, если какая-либо фильтрация происходит до того, как файл попадает на *nenscript*. Конечно, файл может иметь первыми символами `%!` и не быть при этом в формате PostScript (или быть в формате PostScript, не удовлетворяющем соглашениям Adobe). В этом случае он будет неправильно обрабатываться при отправке на принтер PostScript. Для такого типа проверки есть более интеллектуальные фильтры, например Magic-Filter или APSfilter, однако *nenscript* может вполне удовлетворить ваши потребности, особенно если вы печатаете только на принтере PostScript.

Поэтому, если на вашей машине используется фильтр *nenscript* для принтера PostScript, то для всех оболочек пользователей можно установить по умолчанию в переменной окружения `NENSCRIPT` параметр *-Z*, чтобы пропускать без изменений получаемые фильтром файлы PostScript.

В пакете *nenscript* есть сценарий оболочки, который вызывает *nenscript*, конфигурируя его таким образом, что он ведет себя как другой традиционный фильтр Unix – *pstext*.

Для того чтобы использовать *nenscript* в качестве фильтра для печатаемых файлов, убедитесь, что он установлен в нужном каталоге, а затем задайте в настройках тех принтеров, с которыми вы хотите использовать *nenscript*, фильтры, вызывающие *nenscript*. Запись в файле printcap может указывать прямо на фильтр *nenscript*, если установить глобальную переменную окружения `NENSCRIPT`, управляющую его параметрами. Можно также создать простой фильтр, вызывающий *nenscript*, подобный показанному выше примеру конфигурационного файла.

Волшебные фильтры: APSfilter и другие

Наиболее универсальными являются так называемые «волшебные» (magic) фильтры. Волшебный фильтр изучает содержание переданного ему файла и фильтрует вывод на печать, основываясь на том, что ему стало известно о формате данных. Например, если он обнаруживает, что файл является DVI, направляемым на принтер PostScript, то применяет другой фильтр (возможно, *dvips*) для преобразования данных в PostScript для печати. Это очень удобно, но изредка фильтр может ошибаться. В этом случае пользователь может повторно передать файл с параметрами командной строки, указывающими, какую фильтрацию нужно применить, либо сделать предварительную обработку файла, пропустив его через нужный фильтр, а затем передать *lpr* для обычной обработки печати. Есть ряд хороших пакетов волшебных фильтров, в том числе APSfilter (который мы решили здесь описать), Magic-Filter и фильтр *gslp.ps*, поставляемый в полных пакетах Ghostscript.

К сожалению, в некоторых дистрибутивах Linux опущены дополнительные утилиты или документация Ghostscript, но вы всегда можете загрузить полный дистрибутив Ghostscript через FTP с архивного сайта GNU (*ftp://ftp.gnu.org/gnu/*) или одного из его зеркал. Пакеты APSfilter и Magic-Filter можно получить с сайта *ftp://ftp.ibiblio.org* из каталога */pub/Linux/ system/printing*. Главный FTP-сайт пакета Magic-Filter находится в архиве Linux на *ftp://tsx-11.mit.edu*. Фильтр Ghostscript *gslp.ps* написан на языке PostScript и может использоваться только с Ghostscript или другим Adobe-совместимым PostScript-интерпретатором.

Пакет APSfilter для Linux был перенесен из FreeBSD. По этой причине необходимо предпринять некоторые предосторожности, чтобы все правильно конфигурировалось при установке APSfilter. На машине Linux устанавливать пакет APSfilter, вероятно, лучше всего в */usr/ lib/apsfilter*. С *ftp://ftp.ibiblio.org* пакет поступает как архивный файл, сжатый *gzip*. Для распаковки пакета поместите его в каталог */usr/lib/apsfilter* и введите:

```
tar xvfz apsfilter*.tar.gz
```

Теперь пакет APSfilter распаковывается в подкаталоги *apsfilter*.

Перейдите в каталог *apsfilter*. Прежде чем выполнить команду *SETUP*, нужно убедиться, что все фильтры, которые могут понадобиться с APSfilter, установлены и сконфигурированы. В файле */usr/lib/apsfilter/FAQ* говорится о некоторых наиболее важных и полезных пакетах.

До начала установки прочтите документ *INSTALL*, чтобы предотвратить возникновение каких-либо неожиданностей. Затем выполните *./SETUP*. При этом производится проверка наличия и местонахождения графических утилит и других фильтров, которые APSfilter использует для преобразования файлов к виду, пригодному для печати на вашем принтере.

Сценарий *SETUP* позволит узнать, правильно ли установлен фильтр. Его можно запустить снова, если вам понадобится установить более одного принтера или более одного режима для единственного принтера. Например, при установке принтера Deskjet 540 вы, наверное, пожелаете использовать определение dj500 для черно-белого картриджа и определение dj550c для цветного картриджа CMYK. APSfilter использует очень длинные имена каталогов для спулинга. Если вам это не нравится, можете переименовать каталоги и изменить поля каталогов в соответствующей записи файла */etc/printcap*. Следите за тем, чтобы не укоротить имя используемого фильтра; путь поиска имеет большое значение. Мы не рекомендуем заниматься украшательством, пока вы не удовлетворитесь тем, как все работает.

Прежде чем опробовать свою новую установку, перезапустите демон печати:

```
/usr/sbin/lpc restart all
```

APSfilter устанавливает системные переменные для определений принтера в файле */etc/apsfilterrc*, который полезно посмотреть. Часто проблемы печати обусловлены установкой прав владения и доступа. Ниже, в разделе «Права доступа к файлам, каталогам и утилитам», мы покажем, как правильно их настроить. Затем прочтите файлы *FAQ* и *TROUBLESHOOTING* в каталоге */usr/lib/apsfilter*.

Если установленный пакет APSfilter не работает, всегда можно вернуться к предшествующей конфигурации, скопировав сохраненный для вас APSfilter файл */etc/printcap.orig* в */etc/printcap*.

APSfilter дает своим принтерам последовательные имена от `lp1` и выше. Не смущайтесь, они не имеют отношения к фактическим физическим устройствам, назначенным принтерам. Кроме того, вы можете поменять эти имена.

APSfilter позволяет ослабить ограничения с тем, чтобы обычные пользователи могли установить собственные файлы *.apsfilterr* в своих исходных каталогах. По умолчанию это не разрешается, что вносит некоторый элемент защиты.

Последнюю версию Magic-Filter (версию 1.2 на момент написания этой книги) очень легко устанавливать, в отличие от APSfilter. Однако при установке не рассчитывайте на помощь. Несмотря на наличие полезной страницы руководства там не так много сведений, помогающих установить альтернативную обработку, которую утилита Magic-Filter должна осуществлять для большинства печатающих устройств. В частности, если у вас имеется универсальный

принтер, работающий в нескольких режимах (PostScript, PCL5, текст и т. д.), установка и использование этого пакета может оказаться стоящим делом.

Элементы системы печати BSD: файлы, каталоги и утилиты

Система управления печатью требует создания каталогов, совпадающих с именами принтеров в файле printcap, чтобы осуществлять спулинг печати. Она также требует создания других файлов для управления собственно процессом печати. Вы должны установить для файлов и каталогов правильные права владения и доступа; правильные привилегии требуются и для самих утилит печати.

Установка каталогов принтеров

При установке Linux создает стандартный каталог спулинга. В идеале он находится на жестком диске с быстрым доступом. Базовый каталог спулинга (*/var/spool*) обычно используется для управления коммуникациями с использованием электронной почты, телеконференций и UUCP, а также для хранения печатаемых файлов. Мы рекомендуем придерживаться этой практики, стандартной для Linux. Некоторые ваши утилиты могут полагаться на то, что путь к спулингу печати – */usr/spool/lpd*. В таком случае вам придется внести исправления. Конечно, можно создать */usr/spool* и сделать его ссылкой на */var/spool*, но это хорошо, только если */usr* и */var* находятся на одном диске.

Вы должны создать собственные каталоги спулинга печати. Для каждого подкаталога принтера стандартным является каталог */var/spool/lpd*. Имя каждого подкаталога принтера должно использоваться в качестве имени принтера в первом поле соответствующей записи в файле */etc/printcap*. Например, */var/spool/lpd/moa* соответствует принтеру, в поле имени которого в записи файла printcap стоит `moa`. В свою очередь, запись в */etc/printcap* для этого принтера должна устанавливать значение переменной `sd` в каталог спулинга (например, `sd=/var/spool/lpd/moa`).

Не следует выбирать `lp` в качестве имени каталога спулинга, если только вы не уверены, что на вашей машине или в сети никогда не будет более одного принтера, поскольку `lp` является принтером, выбираемым по умолчанию. (Если ваш принтер по умолчанию находится где-то в сети, ваши файлы все равно сначала будут попадать в спулинг */var/spool/lpd/lp*, прежде чем ваш *lpd* перешлет их демону печати на другой машине.) Возможно, ваша утилита администрирования печати автоматически создает каталог для спулинга с именем `lp`, но вы всегда можете отредактировать файл printcap, указав в нем любой каталог по желанию.

Для того чтобы облегчить идентификацию, имя каталога спулинга должно быть первым в связанной с принтером записи в файле */etc/printcap*. В результате запись в printcap будет ассоциироваться с именами, которые утилиты *lpq* и *lpc* выводят в очереди печати.

Права доступа к файлам, каталогам и утилитам

Чаще всего проблемы с настройкой службы печати связаны с правами доступа к файлам и каталогам. В табл. 8.1 перечислены важные файлы, каталоги и утилиты, составляющие администрирование печати BSD в Linux. Место, куда установлен пакет, может различаться в зависимости от дистрибутива Linux. Рекомендуется установить для файлов и каталогов системы печати права владения и доступа, указанные в таблице. (В */etc/printcap* могут быть указаны дополнительные фильтры и нестандартные пути поиска.) Могут оказаться работоспособными и права, определенные по-другому, но если возникают проблемы с правами доступа, их можно исправить согласно этой таблице. Звездочка в первой колонке таблицы указывает на возможность существования нескольких файлов с именами разных принтеров. Разные дистрибутивы могут осуществлять настройку несколько по-разному. В некоторых дистрибутивах есть даже различные «режимы безопасности», выбираемые в зависимости от требований к безопасности и оказывающие воздействие на установку прав доступа.

Таблица 8.1. Файлы, каталоги и утилиты BSD для печати

Каталог или файл	Привилегии	Владелец/группа	Описание
/dev/ttys1	crwsr-----	root/lp	Обычное устройство печати последовательного порта
/dev/lp1	crws------	root/lp	Обычное устройство па раллельного порта (не двунаправленное)
/usr/bin/lpc	-rwsrwsr-x	root/lp	Управляет службами спулинга печати
/usr/bin/lpr	-rwsrwsr-x	root/lp	Получает файл для печати, назначает обработку данных и осуществляет спулинг
/usr/bin/lpq	-rwsrwsr-x	root/lp	Сообщает о файлах спулинга с данными о пользователях и очередности
/usr/bin/lprm	-rwsrwsr--	root/lp	Удаляет задания печати из спулинга
/usr/bin/tunelp	-rwsr-sr--	root/lp	Тестирует службы печати для лучшей настройки
/usr/bin/lptest	-rwxr-xr-x	root/root	Выводит файл ASCII для тестирования принтера и дисплея
/usr/sbin/lpd	-rwsr-s---	root/lp	Демон управления печатью, использующий данные printcap и данные, переданные *lpr*
/usr/sbin/lpf	-rwxr-xr-x	root/lp	Примитивный фильтр печати текста BSD
/usr/sbin/pac	-rwxr--r--	root/root	Утилита BSD, сообщающая об активности принтера и его использовании по ID пользователя

Таблица 8.1 (продолжение)

Каталог или файл	Привилегии	Владелец/группа	Описание
/var/spool/	drwxr-sr-x	root/daemon	Основной системный каталог для временных файлов
/var/spool/lpd	-rws--s--x	root/lp	Стандартный путь системы спулинга печати
*/var/spool/lpd/**	drwxr-sr-x	root/lp	Подкаталоги спулинга для всех определенных принтеров
/var/spool/lpd//filter*	-rwxr-xr-x	root/lp	Фильтры, создаваемые утилитой администрирования печати Red Hat для каждого спулинга печати
/var/spool/lpd/lpd.lock	-rw-rw----	root/lp	lock-файл управления очередью *lpd*
/var/spool/lpd//.seq*	-rw-rw----	lp/lp	Файл последовательности, используемый *lpd* для упорядочения файлов в спулинге
/var/spool/lpd//lock*	-rw-------	root/lp	Блокировка записи *lpd*, предотвращающая отправку очередного файла до готовности принтера
/var/spool/lpd//status*	-rw-------	lp/lp	Здесь *lpd* хранит последний отчет о статусе принтера
/var/log/lp-acct	-rw-------	root/root	Файл ведения учета, из которого *pac* извлекает данные и форматирует их для печати[a]
/var/log/lpd-errs	-rw-rw-r--	root/lp	Стандартный журнал BSD регистрации ошибок *lpd*

[a] Этот файл пуст, когда системный учет не установлен, если только вы не сконфигурировали Ghostscript для вывода в него отчетов и не дали всем право записи в этот файл.

Обычно утилиты управления печатью в Linux устанавливают владельцем файлов печати пользователя root и предоставляют доступ группе lp. Традиционно в дистрибутивах BSD владелец – пользователь root, и доступ предоставлен группе daemon. Вы можете использовать любую из этих групп, но если вы одновременно используете и группу lp, и группу daemon для разных утилит и файлов, то столкнетесь с проблемами. Будьте особенно внимательны в этом отношении при добавлении к службам утилит из других пакетов.

Допустим, вам (зарегистрированному как root) нужно создать каталог спулинга печати – */var/spool/lpd*. Вы выполняете команду

```
mkdir /var/spool/lpd
```

Если ваш каталог */var/spool* был создан с обычными правами доступа, то новый каталог *lpd* имеет права доступа `drwxrwxr-x`, т. е. каталог слишком открыт. Если вы введете команду

```
chmod 755 /var/spool/lpd
```

то права доступа изменятся на `drwxr-xr-x`. Это ближе к истине, но не то, что вам требуется. Вам нужно установить бит setuid, чтобы `lp` мог выполняться с привилегиями суперпользователя:

```
chmod +s /var/spool/lpd
```

В результате права становятся `drwsr-sr-x`, что вам и нужно. Однако группа должна быть `lp`, а не `root`, поэтому нужно сделать исправление:

```
chgrp lp /var/spool/lpd
```

Таким же образом создайте каталоги спулинга для каждого принтера как подкаталоги в */var/spool/lpd*, а затем с помощью команды *touch* создайте файл *.seq* в каждом каталоге печати:

```
touch .seq
```

Использование демона печати

Демон *lpd* обращается к файлу */etc/printcap* и затем посылает файлы на принтеры, направляя их на файл устройства в каталоге */dev*. По большей части принтеры на машинах Linux подключены к последовательным портам (обычно адресуемым через устройства с именами */dev/ttys0*, */dev/ttys1* и т. д. или */dev/ttyS0*, */dev/ttyS1* и т. д.) или параллельным портам (*/dev/lp0*, */dev/lp1* или */dev/lp2*, в зависимости от физических адресов, используемых портами). Назначение портов описывается в разделе «Устранение неполадок в системе печати». Частой ошибкой при конфигурировании служб печати является использование неверного порта.

Можно связать виртуальное устройство, например */dev/fax*, с фактически используемым устройством, создав символическую ссылку, например:

```
ln -s /dev/ttys1 /dev/fax
```

Благодаря этому пользователи могут создавать сценарии и фильтры, обращающиеся к */dev/fax*, что проще запомнить, чем */dev/ttysl*. При этом вы можете менять физическое устройство, например факс-модем, удаляя */dev/fax* и создавая его заново со ссылкой на новое устройство.

Демон печати BSD известен тем, что может становиться вялым или вообще умирать. Правда, теперь это происходит реже, чем несколько лет назад, но все же бывает. Если это происходит, убейте прежний демон и запустите новый. Однако если *lpd* не вполне надежен, для этого должна быть причина. Могут быть проблемы с пользовательским окружением или с параметрами командной строки, используемой с *lpr*, или с ошибками в фильтре, который посылает данные на принтер в непригодной для того форме. Однако вы вправе рассчитывать на «достаточно качественный» пакет печати. При возникновении проблем загляните в «Устранение неполадок в системе печати».

Ну что ж, проверим, работает ли ваша система печати. Можно с уверенностью утверждать, что после всех проведенных вами изменений *lpd* не знает, что происходит. Поэтому выполните команду *ps* и выясните ID процесса *lpd*. Затем введите

```
kill -9 processid
```

чтобы убить указанный процесс.[1] Теперь у вас не должно быть работающего демона печати. Для запуска его введите */usr/sbin/lpd*.

Теперь, следя за миганием светодиодов на передней панели вашего принтера (если таковые имеются), отправьте на печать файл (по-прежнему действуя с привилегиями суперпользователя):

```
lptest | lpr
```

На принтере, который определен по умолчанию в файле */etc/printcap*, должна начаться печать спиральной картинки символов ASCII. Если этого не происходит, то у вас проблемы с конфигурацией, не имеющие отношения к правам доступа.

Проявлял ли принтер какие-либо признаки жизни? Имеется ли у вашего принтера по умолчанию каталог для спулинга? Есть ли в этом каталоге файл *.seq*? Посмотрите, есть ли какие-нибудь сообщения в */var/log/lpd-errs*. С помощью команды *lpc* получите отчет о состоянии демона печати и спулинга печати.

Если кажется, что все правильно, убедитесь, что принтер использует тот порт, который вы ожидаете, послав файл непосредственно в порт, например:

```
# lptest > /dev/lp1
```

Или для проверки принтера последовательного порта:

```
# lptest > /dev/ttys1
```

и т. д. Если ничто не помогает, повторно изучите свой файл */etc/printcap*. Правильно ли составлена запись? Нет ли пробелов или знаков табуляции после символа продолжения строки (\) записи? Правильно ли указана очередь печати? Есть ли среди имен принтера имя `lp`? Совпадает ли первое имя в поле имен с именем используемого каталога спулинга?

Допустим, что вы прошли этот первый тест и получили в лотке принтера несколько чудесных полосатых страниц. Дальше следует задача для смелых. Сможете напечатать что-нибудь как простой пользователь? Зарегистрируйтесь (или выполните команду *su*), чтобы стать обычным пользователем системы. Попробуйте повторить тот же эксперимент. Если удалось, примите поздравления: у вас есть принтер! Если нет, существует проблема, вероятно, связанная с правами владения и доступа к файлам. Что делать, вы знаете. Снова станьте `root`, освежите в памяти страницы руководства по *chgrp*,

---

[1] Вы можете также использовать для этой цели *lpc*. Кроме того, если на рабочем столе суперпользователя есть средство администрирования печати, можете попробовать щелкнуть по кнопке *lpd*, чтобы убить и перезапустить демон печати.

chmod и *chown*, пройдитесь по списку файлов и каталогов, чтобы найти проблему и устранить ее. Повторяйте, пока Джо Юзер не начнет печатать.

Управление службами печати с помощью lpc

Утилита *lpc* позволяет управлять очередями печати и для выполнения большинства своих функций требует привилегий суперпользователя `root`. *lpc* сообщает обо всех очередях печати и обслуживающих их демонах *lpd*. Можно потребовать отчета по выбранному принтеру или пользователю системы печати. Для получения отчета по всем принтерам и пользователям введите:

```
$ lpc status
ibis:
       queuing is enabled
       printing is enabled
       no entries
       no daemon present
crow:
       queuing is enabled
       printing is enabled
       1 entry in spool area
       crow is ready and printing

ada:
       queuing is disabled
       printing is disabled
       no entries
       no daemon present
```

Ведение очередей может быть включено в *lpc* командой *enable* и выключено командой *disable*. Команда *disable* действует путем установки группе права исполнения на файл блокировки в каталоге спулинга печати.

Печать может быть включена в *lpc* командой *start* и отключена командой *stop*. Задания, хранящиеся в очереди печати, при остановке принтера сохраняются и остаются там до возобновления печати. Команда *stop* устанавливает файл блокировки в каталоге спулинга печати и убивает демон печати соответствующей очереди, однако она позволяет завершить задание печати, выполняемое в данный момент. Команда *abort* действует подобно *stop*, но она мгновенно прекращает любое задание печати. (Поскольку задание не завершено, *lpr* сохраняет его и запускает заново при перезапуске очереди.)

Команда *down* действует так, как если бы были выполнены обе команды – *disable* и *stop*, а команда *up* делает обратное, запуская команды *enable* и *start*.

Можно ограничить данные одним принтером:

```
$ lpc status crow
crow:
       queuing is enabled
       printing is enabled
       1 entry in spool area
       crow is ready and printing
```

Функция вывода отчета о состоянии полезна для всех пользователей, и *lpc* позволяет выполнять ее всем.

Настоящая работа для *lpc* обычно начинается, когда возникает необходимость разрешить кризис при печати. Иногда демон печати умирает, и задания печати резервируются. Иногда в принтере кончаются бумага или чернила или происходит сбой. Задания в спулинге печати должны быть приостановлены или перемещены в другой каталог спулинга, где они могут быть выполнены. У кого-то может возникнуть срочное задание печати, которое нужно переместить на самый верх очереди.

Команда *lpc* является классической командой Unix: неразговорчивая и непривлекательная. Если просто ввести команду *lpc*, то в ответ вы получите лишь приглашение

```
lpc>
```

Команда интерактивна и ждет инструкций. Вы можете получить подсказку, введя `help` или `?` в ответ на приглашение *lpc*. *lpc* отвечает и снова выдает приглашение. Например, введя знак вопроса, вы получите следующий результат:

```
#  lpc
lpc> ?
Commands may be abbreviated.  Commands are:
abort   enable  disable help    restart status  topq    ?
clean   exit    down    quit    start   stop    up
lpc>
```

Дополнительную подсказку можно получить, введя запрос об определенной команде. Например, чтобы больше узнать о возобновлении остановленной очереди печати, введите:

```
lpc> help restart
restart         kill (if possible) and restart a spooling daemon
lpc>
```

Подсказка *lpc* ничего не сообщает о дополнительных аргументах, которые можно задать в некоторых случаях. Некоторую помощь вам может оказать страница руководства. Большинство команд воспринимает в качестве дополнительного аргумента *all* или имя каталога спулинга.

Команда *lpc topq* воспринимает первый аргумент как имя каталога спулинга, а последующие – как номера заданий печати или идентификаторы пользователей. Аргументы используются для переупорядочения очереди печати. Например, для того чтобы переместить на вершину очереди печати задание 237 очереди печати `ada` и выстроить за ним все задания, которыми владеет `bckeller`, введите команду

```
lpc> topq ada 237 bckeller
```

Демон *lpd* запустит задание 237, как только закончится текущее задание, и поместит все файлы в очереди, владельцем которых является `bckeller`, перед остальными в спулинге печати. Если вы очень нетерпеливы, то, используя

команды *abort* и *clean*, можете прервать и удалить текущее задание печати, а затем командой *topq* поместить нужное вам задание в вершину очереди и выполнить *restart* для создания нового *lpd* и перезапуска очереди.

При выполнении команды *stop* для остановки спулинга печати (или всех спулингов) можно одновременно послать широковещательное сообщение всем пользователям системы, например:

```
lpc> stop ada "Printer Ada taken down to replace toner cartridge."
```

Если вы существенно перекраиваете спулинг печати – останавливаете очереди и перемещаете файлы – полезно использовать команду *lpc clean*. Это уменьшает опасность того, что какие-то потерянные задания заблокируют демон *lpd*:

```
lpc> clean
```

Затем получите новый отчет о состоянии печати и перезапустите или запустите все остановленные задания печати, прежде чем выйти из *lpc*. Есть различие между такими действиями, как прервать процесс (abort), остановить процесс (stop) и «убить» очередь (bring down). Если вы убьете очередь (например, *lpc down ada*), то обнаружите, что не можете заставить *lpd* снова обслуживать спулинг печати, пока не восстановите обслуживание командой *lpc up ada*. Аналогично после команды *stop* нужно выполнить *start* или *restart*.

После разрешения проблем со спулингом печати с помощью *lpc* можно продолжить работу. Получаемые в дальнейшем отчеты о состоянии быстро покажут вам, действительно ли проблемы решены.

Не стоит ждать, пока возникнут проблемы, чтобы ознакомиться с командами *lpc*, поскольку задания печати могут проходить через спулинг Linux очень быстро, особенно если у принтера достаточно памяти для буферизации посылаемых ему заданий. Изучите страницу руководства и поработайте с *lpc*, чтобы освоиться с возможностями, которые она предоставляет для управления спулингом печати и демонами *lpd*.

Имена подкоманд можно сокращать, но при этом не должна утрачиваться однозначность. Например, в следующей команде `h` заменяет *help*:

```
lpc> h topq
```

Для выхода из *lpc* введите команду

```
lpc> quit
```

или

```
lpc> exit
```

Оптимизация печати

Для повышения производительности можно сначала попытаться улучшить физическую настройку системы. Попытайтесь определить максимальные скорости передачи данных, которые выдерживают ваши принтеры. Не зада-

вайте скорость передачи выше той, что поддерживается устройством, если только ваш принтер не в состоянии передавать сигналы управления потоком данных демону печати. Это означает, что необходимо иметь двунаправленный канал связи (и принтер должен возвращать необходимые сигналы), либо вам придется ограничить скорости передачи, чтобы данные не терялись на пути к принтерам. Может потребоваться проведение экспериментов, чтобы выжать из принтеров максимальную производительность при ограниченной полосе пропускания.

Старые платы для последовательных и параллельных портов PC имеют более низкую пропускную способность, чем новые. В новых платах для последовательных портов более быстрые процессоры ввода/вывода. Более новые параллельные порты обычно работают быстрее и удовлетворяют стандарту EPP, поддерживая двунаправленные коммуникации, что позволяет *lpd* лучше управлять обменом данных с принтером. Можно получить значительный прирост производительности всего лишь за несколько долларов.

Если ваш принтер просто медленный и не может буферизовать задания печати, оптимизация скорости передачи данных даст вам не много, но если аппаратура позволяет, то использовать двунаправленную связь для управления потоком данных с помощью прерываний вместо опроса порта все равно может оказаться полезным, поскольку при этом уменьшается загрузка системы.

Различные виды оптимизации принтера можно испытать с помощью утилиты *tunelp*, но сначала внимательно прочтите соответствующую страницу руководства. Если попытка оптимизации окажется неудачной, может потребоваться выключить принтер и снова включить его, чтобы вернуть в рабочее состояние. Не забывайте также использовать *lpc* для перезапуска демона *lpd* после каждого изменения конфигурации. Сделайте резервную копию своих рабочих настроек, прежде чем экспериментировать с *tunelp*.

Для начала очень хорошо с помощью *tunelp* задать режим прекращения выполнения задания печати и извещения пользователя в случае ошибки принтера. (По умолчанию задание не прерывается.) Такая настройка сократит время тестирования. Для того чтобы при ошибке принтера вызывалась команда *abort*, введите от имени `root` команду

```
tunelp -aon
```

Если ваш принтер подключен к параллельному порту и этот порт поддерживает управление печатью через прерывания, можно использовать *tunelp* для ускорения доступа к принтеру:

```
tunelp /dev/lp1 -i7
```

Этот пример показывает, как включить управление печатью через прерывания для порта, управляемого через прерывание 7. Если после такого изменения печать оказывается невозможной, вернитесь к управлению через опрос вместо прерывания:

```
tunelp /dev/lp1 -r -i0
```

Если вы не знаете, какое прерывание использует устройство, вы можете сделать запрос командой *tunelp -q*, которая выведет значение IRQ.

Иногда можно несколько ускорить печать, сократив паузу, которую делает драйвер при невозможности послать на принтер символ в течение нескольких попыток. Например, для быстрого лазерного принтера вполне могут быть установлены более краткие паузы, и при этом не потребуются многократные попытки передачи. Для того чтобы использовать для передачи символа 10 попыток (по умолчанию устанавливается 250 попыток), а затем делать паузу в 0,01 секунды, введите команду

```
tunelp /dev/lp1 -c10 -t1
```

С параметром *-t* передается коэффициент кратности для 0,01 секунды. Пауза по умолчанию составляет 0,1 секунды.

Учтите, что если вы найдете оптимальную скорость передачи для простых текстовых файлов, то для графических файлов она может оказаться менее эффективной, поскольку обычно они обрабатываются с меньшей скоростью.

Завершив настройку системы печати, можно сбросить флаг прерывания работы принтера, чтобы процесс не прерывался при получении ошибки принтера:

```
tunelp -aoff
```

Утилита *tunelp* будет совершенствоваться в последующих версиях Linux. О ее возможностях, соответствующих вашей версии, можно узнать на странице руководства.

Устранение неполадок в системе печати

Если возникает проблема с печатью, нужно сначала получить отчет о статусе печати с помощью *lpc*. Демоны печати должны быть в порядке, и не должно быть сообщений об ошибках. Можно также проверить содержание файла */var/spool/lpd/printername/status* и посмотреть, нет ли в нем сообщений об ошибках, переданных принтером. Проверьте наличие в файле */var/log/lpd-errs* сообщений демона *lpd* об ошибках. Если вы используете Ghostscript и его функции отчетности включены, получите с помощью */sbin/pac* из журнала отчета Ghostscript, в котором может сообщаться об ошибках, генерированных Ghostscript. (Если система отчетов Linux не установлена, для хранения этих отчетов может использоваться */var/log/lp-acct*. Для этого нужно предоставить право записи в этот файл всем пользователям.)

Снова посмотрите на отчет *lpc* о статусе печати. Попадают ли файлы в спулинг печати? Покидают ли они спулинг? Присутствуют ли необходимые *lpd* файлы поддержки печати (*.seq*, *lock* и др.)? Если в отчете *lpc* указан принтер с именем « : », то в */etc/printcap* есть запись с неправильным форматом; последним символом на продолжаемой строке должна быть обратная косая черта, а не пробел или знак табуляции.

Иногда в */etc/printcap* указаны неверные настройки, из-за чего *lpd* неправильно маршрутизирует файл. Чтобы проверить это, подготовьте файл для печати, но не посылайте его в спулинг принтера, а сохраните на диске. Изу-

чите файл. Тот ли формат он имеет, который вами предполагался? Попробуйте сделать пару проверок исправности:

- Зарегистрировавшись как `root`, отправьте файл прямо в устройство печати (например, командой `cat filename.ps > /dev/lp1`). Печатается ли он? Если да, то корень проблемы – в конфигурации программного обеспечения, а не в аппаратных устройствах.
- Можете ли вы просмотреть свой файл в формате PostScript с помощью Ghostview? Если да, значит, формат файла правильный, но принтер или фильтр интерпретируют его неправильно.

Если вы выполняете проверку на текстовом файле, попробуйте подготовить его и вывести на экран через какую-нибудь утилиту, например *less*, и изучите результат. Фильтр собственной разработки тоже может неправильно маршрутизировать файл.

Иногда трудно разобраться в происхождении проблем с печатью. Например, ошибки конфигурации принтера могут скрываться (или создаваться) при переопределении значений по умолчанию. Для начала можно посмотреть, как работают с печатью отдельные пользователи, а затем пойти дальше. Отдельные пользователи могут устанавливать переменные окружения в своих стартовых файлах оболочек, указывая принтеры, которые собираются использовать по умолчанию, а также режим работы форматтеров и фильтров печати. Системные значения по умолчанию часто переопределяются переменными окружения, которые, в свою очередь, переопределяются параметрами, передаваемыми в командной строке *lpr*, или другими утилитами.

При ненормальном завершении задания печати может оказаться необходимым снять блокировку спулинга, чтобы *lpd* мог послать из этого спулинга на печать очередной файл (*/var/spool/lpd/printername/lock*). Демон *lpd* создает файл *lock* и изменяет его по завершении задания. Можно остановить демон печати с помощью *lpc* и очистить спулинг перед тем, как продолжить работу демона.

Некоторые проблемы возникают в процессе передачи данных. Принтер может пропускать символы или быть не в состоянии угнаться за потоком данных, особенно если принтер старый и медленный или кабель слишком длинный. Признаком проблем с передачей данных является положение, когда принтер нормально обрабатывает простой текст, но запинается при выводе графических файлов. При подозрении на проблемы этого рода попробуйте увеличить паузу, которую система делает перед попыткой повторной передачи данных, и замедлить цикл ожидания. Утилита *tunelp* позволяет легко это контролировать:

```
tunelp -t200 -w5
```

Эта команда сообщает *lpd* о необходимости сделать паузу в 2 секунды между повторными попытками. Параметр *-w* устанавливает число циклов счетчика ожидания, читаемых между стробовыми сигналами. Обычно параметр *-w* устанавливается в 0. Об утилите *tunelp* можно прочесть в разделе «Оптимизация принтера» выше в этой главе.

Если всегда оказывается, что *lpd* не запущен, возможно, он не запускается при начальной загрузке системы. В этом случае допишите строку */etc/lpd* в конец файла */etc/rc.d/rc.local*. Большинство дистрибутивов Linux во время установки по умолчанию устанавливает запуск *lpd*.

Некоторые проблемы могут возникнуть при установке другого пакета, создающего конфликт при попытке обратиться к тем же устройствам. Например, утилиты UUCP обращаются к последовательному порту, используя драйвер устройства */dev/ttyS**. Однако демон UUCP имеет больше привилегий, чем `lp`, и может (хотя и не должен) оставить устройство с такими правами доступа, что *lpd* не сможет осуществлять в него запись.

Пакет печати BSD в дистрибутивах Linux обычно устанавливается с правами группы `lp`. При традиционной установке системы администрирования печати BSD владельцем *lpd* назначается `daemon`, и права доступа к нему имеет группа `daemon` (специальной группы поддержки печати `lp` нет). Если вы предполагаете, что имеются тонкие проблемы, связанные с коллизиями доступа к устройствам между процессами, владельцами которых являются разные демоны, можете предоставить `daemon` все групповые права, относящиеся к утилитам печати, а также поменять права доступа к каталогам и файлам. В результате вы получите традиционную конфигурацию BSD. Лучше все же найти проблемные устройства и сделать их владельцем `lp`, поскольку UUCP сохранит возможность использования устройств, владельцем которых является `lp`. Учтите, что по адресу последовательного порта может обращаться ряд виртуальных устройств, связанных с реальным устройством. Владельца реального порта нужно корректно определить.

Иногда пользователю кажется, что его задание печати поступает «не на тот» принтер. Обычно это вызвано проблемой с переменными окружения. Тщательно проверьте свой файл */etc/printcap*, а также пользовательские переменные окружения. Например, пользовательская переменная `GS_DEVICE` может указывать на принтер, используемый Ghostscript по умолчанию. Если обработка в Ghostscript предшествует обработке, скажем, в *nenscript*, то назначение принтера в Ghostscript может быть передано *nenscript*, переопределяя устройства, заданные в переменных `NENSCRIPT` или `PRINTER`. Это может повлечь необычные последствия, когда переопределяется один из параметров, а остальные остаются неизменными, в результате чего фильтр может производить форматирование страницы для одного принтера, а файл будет послан на другой.

Старые принтеры PostScript могут просто игнорировать посылаемые на них ASCII-файлы. Если пользователь жалуется на исчезновение вывода, причина может быть в том, что файл не поступает на *nenscript* для преобразования в PostScript или (очень редко) *nenscript* ошибочно решает, что файл уже находится в формате PostScript.

Мультирежимный принтер, способный переключать режимы (например, между PCL и простым текстом), может, тем не менее, оказаться неспособным извлечь страницу и начать печать нового файла с новой страницы, когда в очереди печати находятся один за другим файлы одинакового типа.

В этом случае вы можете потребовать, чтобы фильтр добавлял символ подачи страницы в конец каждого документа (см. пример фильтра выше в разделе «Фильтры печати») ценой возможного пропуска лишних чистых страниц.

Адресация принтеров параллельного порта может иногда вызывать путаницу. В системах с шиной XT к первому параллельному порту обращаются как к */dev/lp0* по адресу `0x3bc`. В обычных системах с шиной ISA первым параллельным портом является */dev/lp1* по адресу `0x378`, который на системах XT является вторым параллельным портом (но по-прежнему устройством */dev/lp1*). Обычно вторым параллельным портом на шине ISA является */dev/lp2*, как и можно ожидать, по адресу `0x278`. Однако существуют необычные конфигурации, например системы с тремя параллельными портами (при правильной конфигурации они будут адресоваться как */dev/lp0*, */dev/lp1* и */dev/lp2*). Назначение IRQ тоже может быть необычным и может послужить источником проблем при попытке установить управление печатью через прерывания.

Если ничто не помогает, посмотрите, как осуществлялась первоначальная процедура установки. Убедитесь, что устройства действительно подключены и функционируют, путем загрузки другой операционной системы, если это возможно, проверив устройства при регистрации как `root` и т. д.

CUPS

CUPS, или *Common Unix Printer System* (стандартная система печати Unix), – новичок среди подсистем печати, который благодаря своей функциональности может целиком подчинить себе сферу печати в Linux. Эта система поддерживает многие функции современных принтеров, такие как загружаемые шрифты, цветная печать, броузинг сетевых принтеров, файлы описания принтеров (файлы PPD) и т. д. Некоторые дистрибутивы, например Caldera, уже устанавливают CUPS, но на момент написания книги ни один из дистрибутивов Red Hat, SuSE или Debian этого не делал (хотя этот пакет есть на их CD). Это не значит, что CUPS не работает на этих системах; установку CUPS всегда можно осуществить самостоятельно.

В настоящее время CUPS содержит драйверы принтеров только для устройств HP и Epson, но, поскольку с ними совместимо большинство имеющихся принтеров, проблем тут, как правило, не возникает. Разработчики CUPS продают также коммерческий пакет, в котором есть драйверы для большинства принтеров, даже весьма редких, а кроме того, есть ряд бесплатных драйверов от сторонних разработчиков.

Если вы хотите загрузить и установить на своей машине CUPS, то найдете программное обеспечение и массу документации к нему на *http://www.cups.org*.

Установка атрибутов терминала

Различные характеристики терминала (скажем, виртуальной консоли), например частоту повторения при нажатии клавиши клавиатуры, позиции табуляции и цвет текста, можно установить с помощью программы *setterm*.

Эту программу обычно используют для установки различных цветов в виртуальных консолях, чтобы по цвету текста можно было сразу определить, на какую консоль вы смотрите.

Например, чтобы установить для текущего терминала вывод белого текста на синем фоне, выполните команду

```
$ setterm -foreground white -background blue
```

Некоторые программы или действия вызывают восстановление атрибутов терминала в значения, присвоенные по умолчанию. Чтобы сохранить текущий набор атрибутов в качестве значений по умолчанию, выполните команду

```
$ setterm -store
```

setterm имеет много параметров (большинство из них вы, вероятно, никогда не будете использовать). Более подробные сведения можно получить на странице руководства *setterm*(1) или по команде *setterm –help*.

Если настройки вашего терминала окажутся сильно нарушенными (например, это может произойти, если просматривать содержимое двоичного файла с помощью *cat*), попробуйте вслепую ввести команду *setterm –reset*, которая должна установить приемлемые настройки терминала.

Действия в аварийных ситуациях

Зарегистрировавшись как `root`, легко сделать простую ошибку, в результате которой возникнут серьезные проблемы, например невозможность регистрации или утрата важных файлов. Это особенно часто случается с неопытными администраторами, начинающими исследовать свою систему. Почти все сисадмины-новички приобретают свои знания ценой суровых уроков, когда им приходися восстанавливать систему в чрезвычайной ситуации. В этом разделе мы дадим несколько советов, как действовать, когда неизбежное произойдет.

Необходимо знать меры профилактики, ослабляющие воздействие таких чрезвычайных ситуаций. Например, нужно делать резервные копии всех важных системных файлов, если не всей системы. Если ваш дистрибутив Linux находится на CD-ROM, то последний служит прекрасной страховой копией большинства файлов (если только в затруднительном положении у вас сохранился доступ к CD-ROM, о чем будет сказано ниже). Резервные копии имеют жизненно важное значение при восстановлении во многих случаях. Не дайте пропасть многим неделям тяжкого труда по конфигурированию системы.

Кроме того, ведите учет системной конфигурации, записывая содержимое таблицы разделов, размеры и типы разделов, файловые системы. Если вы каким-то образом разрушили таблицу разделов, иногда можно решить проблему, просто запустив *fdisk*. Однако это возможно только в том случае, если вы знаете, как выглядела ваша таблица разделов. (История из жизни: с одним из авторов такое однажды случилось в результате загрузки с чистой дискеты, когда содержание таблицы разделов записано *не* было. Нет нужды

объяснять, что над восстановлением таблицы разделов пришлось поломать голову.)

Конечно, для того чтобы эти приемы могли работать, необходимо иметь возможность в аварийной ситуации загрузить систему и получить доступ к файлам или восстановиться с резервных копий. Лучше всего сделать это, воспользовавшись «аварийной дискетой» или «корневой дискетой». На таких дискетах находится маленькая файловая система с основными элементами, необходимыми для запуска Linux с дискеты, т. е. самыми важными системными командами и файлами, а также программными средствами для устранения неполадок. Такая дискета используется путем загрузки ядра с другой дискеты (см. раздел «Использование загрузочной дискеты» главы 5) и передачи ядру указания использовать аварийную дискету в качестве корневой файловой системы.

В большинстве дистрибутивов Linux такая комбинация загрузочной/корневой дискет используется в качестве исходной для установки. Установочные дискеты обычно содержат небольшую систему Linux, которая может использоваться для установки программного обеспечения и выполнения основных операций обслуживания системы. Иногда ядро и корневая файловая система находятся на одной дискете, но в этом случае значительно сокращается количество файлов, которые можно записать на аварийную дискету. Польза от таких дискет при обслуживании системы зависит от того, есть ли на них средства (такие как *fsck*, *fdisk*, небольшой редактор типа *vi* и т. п.), необходимые для восстановления системы. В некоторых дистрибутивах процесс установки столь изощренный, что на инсталляционных дискетах остается мало места для чего-либо еще.

Во всяком случае вы можете создать такую корневую дискету сами. Способность сделать это с нуля зависит от глубокого понимания того, что требуется для загрузки и работы системы Linux и что можно сократить или выкинуть. Например, можно оставить в стороне такие запускаемые при старте системы программы, как *init*, *getty* и *login*, если вы знаете, как заставить ядро запустить оболочку на консоли, не прибегая к действительной процедуре начальной загрузки. (Один из способов – на файловой системе дискеты сделать */etc/init* символической ссылкой на */sbin/bash*.)

Не имея возможности раскрыть здесь все детали, отметим, что при создании аварийной дискеты прежде всего необходимо создать на дискете файловую систему с помощью *mkfs* (см. раздел «Создание файловых систем» главы 6). После этого дискета монтируется, и на нее помещаются все необходимые файлы, в том числе нужные элементы из */dev* (большинство из которых можно скопировать из каталога */dev* корневой файловой системы жесткого диска). Вам также понадобится загрузочная дискета, которая содержит просто ядро. С помощью *rdev* нужно установить в этом ядре корневое устройство в */dev/fd0*. Об этом рассказано в разделе «Использование загрузочной дискеты» главы 5. Вы должны также решить, нужно ли загружать файловую систему корневой дискеты на электронный диск, что тоже устанавливается с помощью *rdev*. Если у вас больше 4 Мбайт памяти, то это хорошее решение,

поскольку в результате привод гибких дисков можно высвободить для монтирования другой дискеты, содержащей дополнительные программы. Если у вас два привода гибких дисков, это можно сделать без использования электронного диска.

Если после всего сказанного создание аварийной дискеты кажется вам слишком сложным делом, можете попробовать использовать какие-нибудь сценарии, существующие для этой цели (например, *tomsrtbt* на *http://www.toms.net/rb/*). Каким бы способом вы ни создали аварийную дискету, опробуйте ее в действии *до того*, как несчастье произойдет!

Начать лучше всего с ваших инсталляционных дискет. Если на них нет всех необходимых вам средств, создайте файловую систему на отдельной дискете и поместите на нее недостающие программы. Если вы загружаете корневую систему с дискеты на электронный диск или располагаете вторым дисководом, вы сможете смонтировать эту дискету и получить доступ к своим инструментальным средствам.

Какие средства вам понадобятся? В следующих разделах мы расскажем о наиболее частых аварийных ситуациях и о том, как в этих ситуациях восстанавливать систему; там вы узнаете, какие программы требуются в различных ситуациях. Лучше всего, если в помещаемых на дискету программах использована статическая компоновка: это избавит от прблем, вызываемых недоступностью библиотек совместного доступа в аварийной ситуации.

Восстановление файловых систем

Как говорилось в разделе «Проверка и исправление файловых систем» главы 6, в некоторых случаях для восстановления поврежденных файловых систем можно использовать *fsck*. По большей части это касается относительно несерьезных проблем, которые исправляются при обычной перезагрузке системы и запуске *fsck* с жесткого диска. Однако обычно лучше проверять и исправлять не смонтированную файловую систему. В этом случае легче запустить *fsck* с аварийной дискеты.

Разницы между запуском *fsck* с дискеты и с жесткого диска нет: синтаксис совершенно одинаков и описан ранее в этой главе. Учтите, однако, что *fsck* обычно является интерфейсом для таких программ, как *fsck.ext2*. В некоторых системах нужно использовать *e2fsck* (для вторых расширенных файловых систем).

Повреждение файловой системы может быть таким, что ее не удастся смонтировать. Обычно это связано с повреждением *суперблока* файловой системы, в котором хранится информация о файловой системе в целом. При повреждении суперблока система вообще не сможет установить доступ к файловой системе, и любая попытка монтировать ее закончится отказом (возможно, с сообщением об ошибке «невозможно прочесть суперблок»).

Ввиду важности суперблока файловая система периодически сохраняет его резервные копии. Вторая файловая система разбивается на «группы блоков», в каждой из которых по умолчанию 8192 блоков. Резервные копии су-

перблока находятся по адресам со смещением (в блоках) 8193, 16 385 (т. е. 8192×2+1), 24 577 и т. д. При использовании файловой системы *ext2* можно проверить, что блоки файловой системы объединены в группы по 8192, выполнив команду

```
dumpe2fs device | more
```

(Разумеется, команда работает, если главный суперблок не поврежден.) Эта команда выводит много сведений о файловой системе, в числе которых вы увидите что-нибудь вроде:

```
Blocks per group:         8192
```

Если будет выведено другое значение, нужно использовать его для вычисления смещений к копиям суперблока, аналогичным показанным выше.

Если вы не можете смонтировать файловую систему из-за проблем с суперблоком, то, вероятно, *fsck* (или *e2fsck*) тоже откажется работать. Для восстановления файловой системы можно предложить *e2fsck* использовать одну из копий суперблока, для чего выполните команду

```
e2fsck -f -b offset device
```

где *offset* является смещением в блоках копии суперблока; обычно равно 8193. Ключ *-f* используется для принудительной проверки файловой системы. При использовании резервных копий суперблока файловая система может показаться «целой», и в этом случае проверка не требуется. Ключ *-f* переопределяет это поведение. Например, для исправления файловой системы на устройстве */dev/hda2* с поврежденным суперблоком можно выполнить команду

```
e2fsck -f -b 8193 /dev/hda2
```

Копии суперблока позволяют спасти положение. Приведенные команды можно выполнить с аварийной дискеты, что может помочь вам снова смонтировать ваши файловые системы.

В последнее время в большинстве дистрибутивов Linux появились так называемые *журнальные* файловые системы. Примерами таких систем служат *ext3*, файловая система Райзера и файловая система *ifs*. Эти системы менее подвержены повреждениям, поскольку ведут журнал всех выполненных изменений. Вполне возможно, что в таких системах вам никогда не понадобится прибегать к описываемым здесь приемам.

Доступ к поврежденным файлам

При загрузке с аварийной дискеты вам может потребоваться доступ к файлам на жестком диске. Для этого просто используйте команду *mount*, как описано в разделе «Монтирование файловых систем» главы 6, и монтируйте ваши файловые системы, например, в каталоге */mnt*. (Этот каталог должен существовать в корневой файловой системе на дискете.) Например, команда

```
mount -t ext2 /dev/hda2 /mnt
```

позволит получить доступ из каталога */mnt* к файлам второй расширенной файловой системы на устройстве */dev/hda2*. После этого вы можете получить непосредственный доступ к файлам на жестких дисках и даже выполнять с них программы. Например, если вы хотите запустить с жесткого диска *vi*, который обычно находится в каталоге */usr/ bin/vi*, следует выполнить команду

```
/mnt/usr/bin/vi filename
```

Можно даже поместить подкаталоги */mnt* в пути поиска, чтобы облегчить вызов программ.

Не забудьте размонтировать файловые системы на жестких дисках перед перезагрузкой системы. Если ваши аварийные дискеты не имеют функции корректного останова системы, для верности размонтируйте файловые системы явным образом командой *umount*.

При этом могут возникнуть две проблемы: потеря пароля `root` или искажение содержимого файла */etc/passwd*. В том и другом случае невозможно зарегистрироваться в системе или перейти к `root` по команде *su*. Для решения проблемы загрузитесь с аварийных дискет, смонтируйте корневую файловую систему в */mnt* и отредактируйте */mnt/etc/passwd*. (Неплохо хранить где-нибудь резервную копию этого файла на случай его непреднамеренного удаления.) Например, чтобы вообще удалить пароль суперпользователя, измените запись для `root` на следующую:

```
root::0:0:The root of all evil:/:/bin/bash
```

Теперь `root` не имеет пароля. Можно перезагрузить систему с жесткого диска и установить пароль командой *passwd*.

Если вас заботят проблемы безопасности системы, вы могли содрогнуться, прочтя предыдущее. Вы поняли правильно: тот, у кого есть физический доступ к вашей машине, может изменить пароль суперпользователя, загрузившись с дискеты. К счастью, есть способы защитить систему от возможной атаки. Более эффективны, конечно, физические средства: если компьютер заперт на замок, никто не доберется до него и не вставит загрузочную дискету. Существуют замки для привода гибких дисков, но учтите – чтобы от них была польза, такая защита должна быть и для привода CD-ROM. Если вы не хотите пользоваться физической защитой, можно воспользоваться паролем BIOS, если ваш компьютер это позволяет: отключите в настройках BIOS попытку загрузки с CD-ROM или дискеты (даже если во время начальной загрузки компакт-диск или дискета вставлены в привод) и защитите настройки BIOS паролем. Это не очень надежно, поскольку пароль BIOS можно сбросить аппаратными средствами, но все же защищает от возможного случайного вторжения. Конечно, существует возможность украсть компьютер целиком.

Другой часто возникающей проблемой является нарушение ссылок на системные библиотеки совместного доступа. Доступ к библиотекам в */lib* обычно осуществляется через символические ссылки, такие как */lib/libc.so.5*, ко-

торые указывают на фактические библиотеки */lib/libc.so.version*. Если ссылка удалена или не верна, многие команды системы не будут исполняться. Проблему можно решить, смонтировав файловые системы на жестких дисках и заново создав ссылки на библиотеки командами типа

```
cd /mnt/lib; ln -sf libc.so.5.4.47 libc.so.5
```

В результате ссылка *libc.so.5* указывает на *libc.so.5.4.47*. Помните, что в символических ссылках используются имена путей, заданные в командной строке *ln*. Поэтому команда

```
ln -sf /mnt/lib/libc.so.5.4.47 /mnt/lib/libc.so.5
```

сделает не то, что нужно: *libc.so.5* будет указывать на */mnt/lib/ libc.so.5.4.47*. При загрузке с жесткого диска */mnt/lib* будет недоступен, и библиотеку нельзя будет найти. Первая команда работает правильно, поскольку символическая ссылка указывает на файл в том же каталоге.

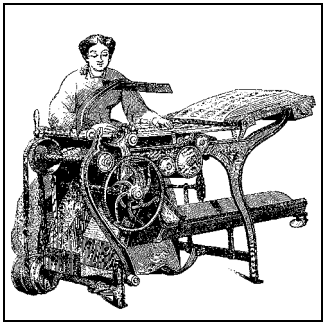
Восстановление файлов из резервных копий

Если вы удалили важные системные файлы, может оказаться необходимым загрузиться с аварийной дискеты и восстановить файлы из резервной копии. Поэтому необходимо иметь на аварийных дискетах средства, необходимые для восстановления с резервных копий. В число этих средств входят такие программы, как *tar* и *gzip,* а также драйверы, необходимые для доступа к устройствам, содержащим резервные копии. Например, если резервные копии сделаны на устройстве флоппи-лент, обеспечьте наличие на аварийной дискете модуля *ftape* и команды *insmod*. См. подробнее об этом в разделе «Загружаемые драйверы устройств» главы 7.

Все, что необходимо для восстановления резервных копий на файловые системы жестких дисков, – это смонтировать эти файловые системы, как описано ранее, и распаковать содержимое архивов поверх этих файловых систем (например, с помощью соответствующих команд *tar* и *gzip*; см. раздел «Создание резервных копий» ранее в этой главе). Не забывайте, что при восстановлении с резервных копий переписываются системные файлы; старайтесь сделать все корректно и не усугубить ситуацию. Большинство программ архивации позволяет извлекать из архива отдельные файлы.

Аналогично, если вы хотите использовать для восстановления файлов свой исходный CD-ROM, необходимо иметь в ядре на аварийной дискете драйверы, необходимые для доступа к приводу CD-ROM. Тогда можно смонтировать CD-ROM (не забудьте о *флагах монтирования -r -t iso9660*) и скопировать с него файлы.

Файловые системы на аварийных дисках должны также содержать некоторые важные системные файлы. Если какие-либо из них вы удалили из системы, то легко скопировать утраченный файл с аварийной дискеты на файловую систему жесткого диска.



9

Редакторы, средства обработки текста, графика и печать

В нескольких последующих главах вы познакомитесь с рядом популярных приложений для Linux. И начнем мы с редактирования текста, которое лежит в основе почти всякого действия в системе. (Чтобы создать файл более или менее значительного размера, будь то исходный код программы, файл конфигурации системы или почтовое сообщение, вам понадобится редактор.) Мы уделим внимание и родственной теме – расскажем о некоторых форматтерах текста, с помощью которых можно создавать документы с привлекательным внешним видом, а также об утилитах управления печатью.

Редактирование файлов с помощью vi

В этом разделе мы собираемся рассказать об использовании редактора *vi* (произносится «ви-ай»). *vi* был первым настоящим экранным редактором для Unix. Он прост, невелик по размеру и «вылизан». Если вы выполняете функции системного администратора, то знание *vi* необходимо: во многих аварийных ситуациях большие редакторы типа Emacs недоступны (например, при загрузке Linux с аварийного диска).

vi основан на тех же принципах, что и многие другие приложения Unix: каждая программа должна обеспечивать свою специфическую функцию и быть в состоянии взаимодействовать с другими программами. Например, в *vi* нет собственного средства проверки орфографии или заполнителя абзацев, но эти функции реализованы в других программах, которые легко запустить из *vi*. Поэтому несмотря на некоторую ограниченность *vi* может взаимодействовать с другими программами и обеспечить практически любые необходимые функции.

На первый взгляд, *vi* несколько сложен и неуклюж. Однако его однобуквенные команды покажутся быстрыми и мощными, когда вы их изучите. В следующем разделе мы опишем Emacs – более гибкий редактор (фактически интегрированную рабочую среду), который можно освоить быстрее. Но помните,

что умение работать с *vi* может оказаться решающим в ситуации, когда Emacs окажется недоступным, поэтому мы настоятельно советуем изучить основы *vi*, какими бы непривычными они ни показались. Следует добавить, что в настоящее время есть ряд клонов *vi*, с которыми значительно удобнее работать, чем с первоначальным *vi*. Среди них наиболее популярен *vim*. Весьма вероятно, что ваш дистрибутив устроен так, что при вызове *vi* вы фактически запускаете какой-то из этих новых редакторов. Мы будем говорить здесь только об основных функциях, которые применимы в любой версии *vi*. Освещение более новых версий можно найти в книге «Learning the vi Editor», написанной Линдой Лэм (Linda Lamb) и Арнольдом Роббинсом (Arnold Robbins) (O'Reilly).

Запуск vi

Давайте запустим *vi* и отредактируем файл. Синтаксис *vi* следующий:

```
vi filename
```

Например:

```
eggplant$ vi test
```

запустит *vi* для редактирования файла *test*. Ваш экран при этом должен выглядеть так:

```
~
~
~
~
~
~
~
~
~
"test" [New File]                                        0,0-1         All
```

Колонка символов ~ показывает, что вы в конце файла.

Вставка текста и перемещение по нему

При работе в *vi* вы всегда находитесь в одном из двух (или трех, в зависимости от того, как на это посмотреть) режимов, носящих названия: *командный режим* (*command mode*), *режим редактирования* (*edit mode*) и *режим ex* (*ex mode*) – *режим расширенных команд*.

После запуска *vi* вы попадаете в командный режим. Как мы вскоре увидим, в этом режиме можно использовать ряд команд (обычно однобуквенных) для модификации текста. Фактический ввод или модификация текста выполняются в режиме редактирования. Чтобы начать вставку текста, нажмите `i` (что переведет вас в режим редактирования) и введите текст:

```
Now is the time for all good men to come to the aid of the party.
~
~
~
~
~
~
~
~
~
-- INSERT --                                             1,66          All
```

Вставляя текст, вы можете вводить любое число строк (разумеется, нажимая после каждой строки <Enter>) и исправлять ошибки нажатием клавиши <Backspace>. Чтобы завершить редактирование и вернуться в командный режим, нажмите клавишу <Esc>.

В командном режиме можно пользоваться клавишами со стрелками для перемещения по документу. Допустимо также использование клавиш `h`, `j`, `k` и `l`, которые перемещают курсор влево, вниз, вверх и вправо соответственно.

Есть и другие способы вставки текста помимо команды `i`. Команда `a` («append») вставляет текст *после* текущей позиции курсора. Например, переместите курсор клавишей <стрелка влево> в положение между словами `good` и `men`:

```
Now is the time for all good men to come to the aid of the party.
~
~
~
~
~
~
~
~
~
                                                          1,29          All
```

Нажмите `a`, введите `wo` и вернитесь в командный режим, нажав <Esc>:

```
Now is the time for all good women to come to the aid of the party.
~
~
~
~
~
~
~
~
~
                                                          1,31          All
```

Чтобы открыть строку под текущей и начать ввод текста, используйте команду `o`. Нажмите `o` и введите еще одну-две строки:

```
Now is the time for all good women to come to the aid of the party.
Afterwards, we'll go out for pizza and beer.
~
~
~
~
~
~
~
~
-- INSERT --                                              2,45          All
```

Запомните, что вы всегда находитесь либо в командном режиме (в котором можно вводить такие команды, как `i`, `a` или `o`), либо в режиме редактирования (в котором вы вводите текст, а затем нажимаете <Esc> для возврата в командный режим). Если вы забыли, в каком режиме находитесь, то нажмите <Esc>. Это выведет вас из режима редактирования, если вы в нем находитесь, и не окажет никакого действия, если вы были в командном режиме.

Удаление текста и отмена изменений

В командном режиме нажатие x удаляет символ под курсором. Если вы нажмете x пять раз, то получите следующее:

```
Now is the time for all good women to come to the aid of the party.
Afterwards, we'll go out for pizza and
~
~
~
~
~
~
~
~
                                                          2,39          All
```

Теперь нажмите a и введите какой-нибудь текст, а затем нажмите <Esc>:

```
Now is the time for all good women to come to the aid of the party.
Afterwards, we'll go out for pizza and Diet Coke.
~
~
~
~
~
~
~
~
-- INSERT --                                               2,50          All
```

С помощью команды dd (т. е. нажимая d дважды) можно удалять целые строки. Если курсор находится на второй строке, команда *dd* произведет следующий результат:

```
Now is the time for all good women to come to the aid of the party.
~
~
~
~
~
~
~
~
~
                                                          1,1           All
```

Удаленный текст можно вставить заново командой p («put»). Если сейчас нажать p, то удаленная строка будет возвращена в буфер сразу за текущей. Если же ввести P (в верхнем регистре), то текст будет вставлен перед текущей строкой. По умолчанию p и P вставляют текст из буфера отмены, но можно копировать и вставлять текст из других буферов, как мы покажем позже.

Команда u отменяет последнее произведенное изменение (в данном случае нажатие u после dd эквивалентно p). Если вы вставили большой кусок текста по команде i, то, нажав u сразу после возврата в командный режим, вы отмените вставку.

Для удаления слова под курсором используйте команду dw. Поместите курсор на слово Diet и введите dw:

```
Now is the time for all good women to come to the aid of the party.
Afterwards, we'll go out for pizza and coke.
~
~
~
~
~
~
~
~
                                                          2,40          All
```

Изменение текста

Заменить текст можно командой R, которая переписывает текст, начиная с положения курсора. Поместите курсор на первую букву в pizza, нажмите R и введите:

```
Now is the time for all good women to come to the aid of the party.
Afterwards, we'll go out for burgers and fries.
~
~
~
~
~
~
~
~
-- REPLACE --                                              2,48          All
```

Команда r заменяет один символ под курсором. r не переводит вас в режим редактирования, поэтому нет необходимости нажимать <Esc> для возврата в командный режим.

Команда ~ изменяет регистр буквы под курсором на противоположный. Если вы поместите курсор на o в Now в предыдущем примере и несколько раз нажмете ~, то получите следующее:

```
NOW IS THE TIME FOR ALL GOOD WOMEN TO COME TO THE AID OF THE PARTY.
Afterwards, we'll go out for burgers and fries.
~
~
~
~
~
~
~
~
                                                          1,67          All
```

Другая полезная команда cw изменяет слова, позволяя ввести новое слово и после нажатия <Esc> удалить остатки прежнего слова. Если новый текст длиннее, чем модифицируемый, пространство автоматически расширяется.

Команды перемещения

Вы уже знаете, как использовать клавиши со стрелками для перемещения по документу. Помимо этого команда w перемещает курсор к началу следующего слова; команда b перемещает его к началу текущего слова. Команда 0 (ноль) перемещает курсор к началу текущей строки, а команда $ – к концу строки.

При редактировании больших файлов требуется перемещаться сразу на целый экран вперед или назад. Нажатие <Ctrl>+<F> перемещает курсор на один экран вперед, а <Ctrl>+<B> – на один экран назад.

Для перемещения в конец файла нажмите G. Можно также переместиться на произвольную строку: команда 10G переместит курсор на 10 строку файла. Переместиться в начало файла можно командой 1G.

Ввод «/», а затем шаблона и <Enter> вызывает переход на первое соответствие шаблона и текста, следующего за курсором. Например, поместив курсор на первую строку текста в нашем примере и введя /burg, мы переместим курсор на начало слова «burgers.» Если вместо / вводить ?, то поиск осуществляется в обратном направлении.

Шаблон, следующий за командой / или ?, является на самом деле *регулярным выражением*. Регулярные выражения являются мощным способом задания шаблонов для операций поиска и замены и используются во многих утилитах Unix. Дополнительные сведения о регулярных выражениях можно найти ниже в разделе «Регулярные выражения». С помощью регулярного выражения можно, например, найти очередную заглавную букву, выполнив команду

```
/[A-Z]
```

Если то, что вы ищете, не является статической строкой, для задания искомого текста можно использовать регулярные выражения.

Команды перемещения можно соединять с другими командами, такими как удаление. Например, команда d$ удалит текст от курсора до конца строки; dG удалит весь текст от курсора до конца файла.

Сохранение файлов и выход из vi

Большинство команд *vi*, работающих с файлами, вызывается из режима *ex*. В режим *ex* вы попадаете, нажав клавишу <:> в командном режиме. В результате курсор перемещается на последнюю строку экрана и позволяет вводить различные расширенные команды.

Например, для записи редактируемого файла нужно ввести :w. Ввод : влечет переход в режим *ex*, а нажатие w и затем <Enter> завершают команду. Команда :wq записывает файл и осуществляет выход из *vi*. Команда ZZ (в командном режиме, без «:») эквивалентна :wq, но проверяет сначала, изменился ли файл, и только тогда его записывает.

Для того чтобы выйти из *vi* без сохранения изменений, используется команда :q!. Просто команда :q позволяет завершить *vi*, но только если изменения в файле сохранены. Знак ! в команде :q! означает серьезность вашего намерения выйти из *vi*.

Переход к редактированию другого файла

Для того чтобы начать редактирование другого файла, используйте команду :e. Например, чтобы прекратить редактирование *test* и начать вместо него редактировать файл *foo*, выполните команду, показанную в нижней строке следующего примера:

```
NOW IS THE TIME FOR ALL GOOD WOMEN TO COME TO THE AID OF THE PARTY.
Afterwards, we'll go out for burgers and fries.
~
~
~
~
~
~
~
~
:e foo
```

Если вы попытаетесь выполнить `:e`, не сохранив сначала файл, то получите сообщение об ошибке:

```
No write since last change (:edit! overrides)
```

В таком случае можно выполнить `:w`, чтобы сохранить исходный файл, а затем выполнить `:e`, либо можно использовать команду `:e! foo`, которая заставляет *vi* редактировать новый файл без сохранения изменений в текущем файле. Этим удобно воспользоваться, когда при редактировании файла вы почувствуете, что запортили его. Тогда выполнение команды `:e!` без указания имени файла приведет к отмене всех изменений и редактированию текущего файла с начала.

Вставка других файлов

С помощью команды `:r` можно включить в буфер *vi* содержимое другого файла. Например, команда

```
:r foo.txt
```

вставляет содержимое файла *foo.txt* после текущей строки.

Выполнение команд оболочки

Команда `:!` позволяет ввести имя команды, выполняемой внутри *vi*. Например, команда

```
:! ls -F
```

выполняет команду `ls` и выводит результат на экран.

Команда `:r !` аналогична `:!`, но помещает данные, выводимые командой на стандартное устройство вывода, в буфер. Команда

```
:r! ls -F
```

производит следующее действие:

```
NOW IS THE TIME FOR ALL GOOD WOMEN TO COME TO THE AID OF THE PARTY.
Afterwards, we'll go out for burgers and fries.
letters/
misc/
papers/
test
~
~
~
~
4 more lines                                          6,1           All
```

Если нужно выполнить несколько команд оболочки, то зачастую проще использовать клавиши прерывания (обычно <Ctrl>+<Z>), при условии ис-

пользования оболочки, поддерживающей управление заданиями, например *tcsh* или *bash*.

Глобальный поиск и замена

В *vi* значительно больше функций, чем здесь описано; большинство этих функций реализовано путем сочетания простых функций, которые мы уже рассмотрели. Вот еще пара приемов, которые применяются многими пользователями *vi*. Команда

```
:[x,y]s/pattern/replacement/flags
```

ищет шаблон `pattern` в буфере между строками `x` и `y` и заменяет вхождения `pattern` текстом `replacement`. `pattern` является регулярным выражением; `replacement` является текстовым литералом, но может содержать некоторые специальные символы, указывающие на элементы исходного шаблона. Следующая команда заменяет первое вхождение `weeble` на `wobble` в строках с 1 по 10 включительно:

```
:1,10s/weeble/wobble
```

Вместо того чтобы указывать номера строк, можно использовать символ `%`, который является ссылкой на весь файл. Вместо `x` и `y` можно использовать и другие специальные символы. `$` указывает на последнюю строку файла. Если `x` или `y` не указаны, это означает ссылку на текущую строку.

В число используемых флагов *flags* входят `g` для замены всех вхождений шаблона в каждой строке и `c` для запроса подтверждения каждой замены. В большинстве случаев вам понадобится флаг `g`, если только вашим намерением не является замена первого вхождения шаблона в каждой строке.

Для ссылки на строки можно также использовать *метки* (*marks*). Метки являются однобуквенными именами, присваиваемыми положениям курсора в документе. Помещение курсора в некоторую позицию в файле и ввод `ma` устанавливает метку `a` в эту позицию. (Метки можно именовать любыми буквами a–z или A–Z.) Курсор можно переместить прямо на метку `a` командой `` `a `` (с обратным апострофом). Использование обычного апострофа (как в `'a`) приводит к перемещению курсора в начало строки, в которой находится метка `a`.

Метки позволяют «запоминать» положения курсора для обозначения области текста. Например, если вам нужно осуществить поиск и замену в участке текста, переместите курсор в начало текста, установите метку, переместите курсор в конец текста и выполните команду

```
:'a,.s/weeble/wobble
```

где `'a` указывает на строку, содержащую метку `a`, а «`.`» ссылается на текущую строку.

Перемещение текста и использование регистров

Копировать и перемещать текст можно, удаляя его (с помощью команд `d` или `dd`), а затем вставляя командой `P`, описанной выше. Например, если вы хоти-

те удалить 10 строк, начиная с той, в которой расположен курсор, а затем вставить их в какое-либо другое место, выполните команду `10dd` (чтобы удалить 10 строк), переместите курсор в то место, где должен находиться текст, и введите `p`. Таким же образом можно и копировать текст: ввод `10dd`, а затем `P` (в том же положении курсора) удаляет текст и сразу же его восстанавливает. После этого вы можете вставлять текст в любое место, перемещая курсор и всякий раз нажимая `p`.

Сходной с `dd` является команда `yy`, которая копирует («yanks») текст, не удаляя его. Для вставки скопированного текста используется команда `p`, так же как с `dd`. Но учтите, что каждая операция копирования удаляет из «буфера» ранее скопированный текст.

Команды удаления и копирования в буфер можно применять и к более общим областям, нежели строки. Напомним, что команда `d` удаляет текст с помощью команды перемещения; например, `d$` удаляет текст от текущего положения курсора до конца строки. Аналогично `y$` копирует текст от курсора до конца строки.

Допустим, вы хотите скопировать в буфер (или удалить) участок текста. Это можно сделать с помощью меток. Переместите курсор в начало копируемого текста и поставьте метку, например командой `ma`. Переведите курсор в конец копируемого текста и выполните команду `y`a`. В результате будет скопирован текст от позиции курсора до метки `a`. (Вспомните, что команда `` `a `` перемещает курсор на метку `a`.) Использование `d` вместо `y` удаляет текст от курсора до метки.

Удобнее всего вырезать, копировать и вставлять участки текста в *vi*, используя *регистры*. Регистр является именованным временным местом хранения текста, который вы хотите копировать в различные места, вырезать и вставлять в документе и т. д.

Имена регистров состоят из одной буквы в диапазоне a–z или A–Z. Команда `"` (знак кавычки) задает обращение к регистру, затем следует имя регистра, например `"a` для регистра `a`. Большие и маленькие буквы относятся к одним и тем же регистрам, но при использовании маленьких букв прежнее содержимое регистра затирается, а при использовании больших букв происходит дописывание к регистру.

Например, если переместить курсор к первой строке нашего примера:

```
NOW IS THE TIME FOR ALL GOOD WOMEN TO COME TO THE AID OF THE PARTY.
Afterwards, we'll go out for burgers and fries.
~
~
~
~
~
~
~
~
"test" 2L, 116C                                          1,1           All
```

и выполнить команду `"ayy`, то текущая строка скопируется в регистр `a`. Затем можно переместить курсор во вторую строку и выполнить команду `"ap`, чтобы вставить текст из регистра `a` после текущей строки:

```
NOW IS THE TIME FOR ALL GOOD WOMEN TO COME TO THE AID OF THE PARTY.
Afterwards, we'll go out for burgers and fries.
NOW IS THE TIME FOR ALL GOOD WOMEN TO COME TO THE AID OF THE PARTY.
~
~
~
~
~
~
~
"test" 2L, 116C                                          3,1           All
```

Сходным образом команда `"ay`a` копирует текст от курсора до метки `a` в регистр `a`. Обратите внимание, что между именами меток и регистров никакой связи нет!

Использование регистров позволяет копировать текст между файлами. Нужно просто скопировать текст в регистр, выполнить команду `:e` для редактирования нового файла и вставить текст из регистра.

Расширение vi

Есть много способов расширить *vi*. Большинство представленных нами команд можно обобщить на произвольные области текста. Как мы уже видели, такие команды, как `d` и `y`, действуют от позиции курсора до позиции, определяемой командой перемещения, например `$` или `G`. (`dG` удаляет текст от курсора до конца файла.) Многие другие команды обрабатывают текст посредством команды перемещения таким же способом. Используя метки, можно обрабатывать любую область текста.

Как уже отмечалось, *vi* является лишь текстовым редактором, и в нем нет средств проверки орфографии, компиляции программ и других подобных функций. Однако *vi* запускает другие программы, которые могут служить расширением редактора. Команда

```
:x,y!command
```

выполняет указанную команду, для которой строки с `x` по `y` служат стандартным вводом, и заменяет строки стандартным выводом команды. Как и в команде `s` (поиск и замена), для задания номеров строк можно использовать специальные символы, такие как `%` и `$`.

Допустим, к примеру, вы хотите поставить символ цитирования (`>`) в начале всех строк некоторой области текста. Можно добиться этого, написав короткий сценарий для оболочки или сценарий на языке Perl (см. раздел «Языки программирования и утилиты» главы 1), который считывает входные строки и выводит те же строки, предваряемые символом цитирования. (Можно также использовать команду *sed* – способов много.) Затем можно отправлять строки текста на этот фильтр, который будет заменять их цитированным текстом в *vi*. Если сценарий назвать *quote*, то нужно выполнить команду

```
:`a,.!quote
```

которая процитирует область текста между положением курсора и меткой `a`.

Ознакомьтесь с различными имеющимися командами режима *ex*. Команда `:set` позволяет устанавливать различные параметры. Например, `:set ai` включает автоматическое создание отступов в тексте (`:set noai` выключает его).

Можно задать команды *ex* (такие как `:set`), которые должны выполняться при запуске *vi*, поместив их в файл *.exrc* в вашем исходном каталоге. (Имя этого файла можно изменить через переменную окружения `EXINIT`.) Например, файл *.exrc* может содержать строку

```
set ai
```

для включения автоматического создания отступов. Ставить в этом файле «:» перед командами *ex* не требуется.

Есть ряд хороших руководств и справочников по *vi*, как электронных, так и печатных. Дополнительные сведения можно поискать в «Learning the vi Editor». При наличии доступа в Интернет можно найти ряд справочных и учебных документов, а также интересные приемы работы в *vi* в архивах *comp.editors*. Архивы находятся в *ftp://alf.uib.no:/pub/vi*, зеркально отображенным на *cs.uwp.edu* и в других местах. Редактору *vim* посвящен сайт *http://www.vim.org*.

Редактор Emacs

Текстовые редакторы являются одними из самых важных приложений в Unix. Они используются так часто, что многие проводят в редакторе больше времени, чем в какой-либо другой программе Unix. То же справедливо для Linux.

Выбор редактора может стать вопросом выбора религии. Есть много редакторов, но сообщество Unix разделилось на две основные группы: лагерь приверженцев Emacs и лагерь *vi*. По причине не вполне интуитивного интерфейса *vi* многие (как новички, так и опытные пользователи) предпочитают Emacs. Однако те, кто давно работает в *vi* (и печатает одним пальцем), используют его эффективнее, чем более сложный редактор, каким является Emacs.

На одном краю спектра редакторов находится *vi*, а на другом – Emacs. Они значительно различаются по архитектуре и философии. Emacs, отчасти, является плодом ума Ричарда Столмена (Richard Stallman), основателя Фонда свободного программного обеспечения и автора значительной доли программного обеспечения GNU.

Emacs – очень крупная система, в которой больше функций, чем в любом другом отдельном современном приложении Unix (некоторые даже называют его не редактором, а «интегрированной средой»). В нем есть собственный интерпретатор языка LISP, который можно использовать для создания расширений редактора. (Многие внутренние функции Emacs написаны на Emacs LISP.) В Emacs есть расширения для чего угодно – от компиляции и отладки программ до работы с электронной почтой и поддержки X Window System и т. д. В Emacs есть также собственный электронный учебник и документация. Популярным введением в редактор Emacs служит книга Дебры Камерон (Debra Cameron), Билла Розенблатта (Bill Rosenblatt) и Эрика Рэймонда (Eric Raymond) «Learning GNU Emacs», O'Reilly.

В большинстве дистрибутивов Linux есть две версии Emacs. Первоначальной версией является GNU Emacs. Она разрабатывается до сих пор, но разработка, похоже, замедлилась. XEmacs больше по размеру, но значительно более

дружествен к пользователю и лучше интегрирован с X Window System (несмотря на название его можно запускать в командной строке). Если вы не очень стеснены в памяти и обладаете достаточно быстрым компьютером, то мы бы посоветовали пользоваться XEmacs. Еще одним достоинством XEmacs является то, что многие полезные пакеты, которые в случае GNU Emacs нужно загружать и устанавливать отдельно, уже входят в поставку XEmacs. Мы не будем говорить о различиях между ними, а остановимся на том, что присуще обоим редакторам. Когда мы будем упоминать в этом разделе Emacs, будет иметься в виду любая из версий.

Запуск Emacs

Запуск GNU Emacs осуществляется просто:

```
$ emacs options
```

Аналогично, XEmacs вызывается следующим образом:

```
$ xemacs options
```

По большей части, параметры вам не потребуются. Можно задать в командной строке имена файлов, но проще открыть их после запуска программы.

На жаргоне Emacs `C-x` означает <Ctrl>+<X>, а `M-p` эквивалентно <Alt>+<P>. Нетрудно догадаться, что `C-M-p` означает <Ctrl>+<Alt>+<P>.

Учитывая эти соглашения, нажмите `C-x`, а затем `C-f`, чтобы прочесть файл или создать новый. В результате нажатия клавиш в нижней части экрана будет выведено приглашение, показывающее текущий рабочий каталог. Теперь можно создать буфер, который в итоге станет новым файлом; назовем его *wibble.txt*. В результате мы видим следующее:

Строка режима внизу указывает имя файла и тип буфера, в котором вы находитесь (в данном случае `Fundamental`). Emacs поддерживает много режимов редактирования. `Fundamental` является режимом по умолчанию для простых текстовых файлов, но есть режимы для редактирования исходного кода C и TEX, изменения каталогов и т. д. Как мы скоро увидим, для каждого режима существуют свои привязки клавиш и команды. Обычно Emacs определяет тип буфера, основываясь на расширении имени файла.

Справа от типа буфера вы видите слово `All`, означающее, что у вас перед глазами находится весь файл (который пуст). Обычно вы видите процентную величину, показывающую ваше положение относительно начала файла.

Если вы запускаете Emacs в X Window System, то для редактора создается новое окно с меню вверху, полосами прокрутки и прочими удобствами. В разделе «Emacs и другие редакторы» главы 11 мы обсудим особенности работы Emacs в среде X.

Простые команды редактирования

Основные операции редактирования выполняются в Emacs проще, чем в *vi*. Клавиши со стрелками должны перемещать курсор по буферу. Если этого не происходит (когда Emacs не настроен для вашего терминала), используйте клавиши `C-p` (предыдущая строка), `C-n` (следующая строка), `C-f` (вперед на символ) и `C-b` (назад на символ).

Если вам неудобно использовать клавишу <Alt>, нажмите <Esc>, а затем `p`. Нажать и отпустить <Esc> эквивалентно удержанию клавиши <Alt> в нажатом положении.

В этом месте нужно сделать первое отступление в нашем путешествии по Emacs. Можно переназначить буквально каждую команду и клавишу в Emacs. В конфигурации Emacs по умолчанию `C-p` вызывает внутреннюю функцию *previous-line*, которая перемещает курсор (называемый также «point», точка) к предыдущей строке. Однако нетрудно привязать к этим функциям другие клавиши или написать новые функции и привязать клавиши к ним и т. д. Если не сказано иное, то клавиши, о которых мы рассказываем, действуют в конфигурации Emacs по умолчанию. Ниже мы покажем, как переназначать клавиши для использования соответственно личным предпочтениям.

Вернемся к редактированию. Клавиши со стрелками или их эквиваленты перемещают курсор по текущему буферу. Начинайте печатать текст, и он будет вставляться в текущую позицию курсора. Нажатие клавиши <Backspace> или <Delete> должно удалять текст в позиции курсора. Если этого не происходит, то исправить положение можно, опираясь на сведения в разделе «Настройка Emacs» далее в этой главе. Теперь введите текст:

```
The rain falls on the main plains in Spain.

-1:** wibble.txt       (Text)--L1--All----------------------------------------
```

Комбинации клавиш `C-a` и `C-e` перемещают курсор соответственно к началу и концу текущей строки. `C-v` перемещает на страницу вперед, а `M-v` перемещает на страницу назад. Есть много других базовых команд редактирования, о которых можно узнать в электронной документации по Emacs (о ней мы вскоре скажем).

Выйти из Emacs можно с помощью команды `C-x C-c`. Это первая из встретившихся нам расширенных команд. Во многих командах Emacs требуется нажатие нескольких клавиш. Комбинация `C-x` служит «префиксом» для других клавиш. В данном случае нажатие `C-x` с последующим нажатием `C-c` приводит к окончанию работы Emacs, который предварительно спрашивает, действительно ли вы хотите выйти, не сохранив сделанные в буфере изменения.

Клавиши `C-x C-s` сохраняют текущий файл, а нажатие `C-x C-f` вызывает «поиск» нового файла для редактирования. Например, введя `C-x C-f`, можно получить приглашение

```
Find file: /home/loomer/mdw/
```

в котором отображается текущий каталог. В ответ введите имя файла, который нужно открыть. Нажатие клавиши <Tab> вызывает расширение имени файла, аналогичное используемому в *bash* и *tcsh*. Например, после ввода

```
Find file: /home/loomer/mdw/.bash
```

и нажатия клавиши <Tab> открывается другой буфер, в котором показаны все возможные завершения, например:

```
The rain falls on the main plains in Spain.

-1:** wibble.txt        (Text)--L1--All----------------------------------
Click mouse-2 on a completion to select it.
In this buffer, type RET to select the completion near point.

Possible completions are:
.bash_profile                    .bashrc

-1:-- *Completions*      (Completion List)--L1--All-------------------------
Find file: /home/loomer/mdw/.bash
```

После введения полного имени файла буфер `*Completions*` исчезает и отображается новый редактируемый файл. Это пример того, как Emacs использует временные буферы для вывода информации.

Emacs позволяет использовать при редактировании несколько буферов, в каждом из которых может содержаться свой файл. При загрузке файла командой `C-x C-f` для его редактирования создается новый буфер, при этом первоначальный буфер не удаляется.

Перейти в другой буфер можно командой `C-x b`, которая запрашивает имя буфера (обычно совпадающее с именем файла в буфере). Например, нажатие `C-x b` выводит приглашение

```
Switch to buffer: (default wibble.txt)
```

Буфером по умолчанию является тот, в котором вы были до этого. Нажмите <Enter> для перехода в буфер по умолчанию или введите имя другого буфера. Команда C-x C-b выводит список буферов (в собственном буфере), например:

```
MR Buffer           Size  Mode         File
-- ------           ----  ----         ----
 . % .bashrc        5507  Shell-script /home/loomer/mdw/.bashrc
   wibble.txt         43  Text         /home/loomer/mdw/wibble.txt
*% *Buffer List*     211  Buffer Menu
   *Completions*     159  Completion List
   *scratch*         182  Text
*  *Messages*       1303  Fundamental

-1:%*  *Buffer List*     (Buffer Menu)--L3--All-----------------------------
```

Появившееся меню буфера разделяет экран на два окна, переключаться между которыми можно командой C-x o. Одновременно можно открыть и более двух окон. Для того чтобы видеть одновременно только одно окно, перейдите в нужное окно и нажмите C-x 1. В результате остальные окна будут скрыты, но переключаться в них можно с помощью уже описанной команды C-x b. Нажатие C-x k приводит к удалению буфера из памяти Emacs.

Учебное руководство и оперативная подсказка

Если Emacs выглядит довольно сложным, то это из-за того, что он является очень гибкой системой. Прежде чем двигаться дальше, полезно познакомиться со встроенной оперативной подсказкой и учебным руководством по Emacs. Эта документация также доступна в виде книги «GNU Emacs Manual» Ричарда Столмана (Richard M. Stallman), GNU Press.

Команда C-h выводит в последней строке экрана список параметров подсказки. Повторное нажатие C-h выводит их описания. В частности, команда C-h t перемещает вас в учебное руководство по Emacs, которое написано ясно и расскажет о системе больше, чем мы сможем осветить в этой книге.

Изучив руководство по Emacs, вам следует освоить систему Info, в которой находится остальная документация по Emacs. Программа чтения Info вызывается командой C-h i. Гипотетическая страница Info может выглядеть так:

```
File: intercal.info,  Node: Top,  Next: Instructions,  Up: (dir)

This file documents the Intercal interpreter for Linux.

* Menu:

* Instructions::        How to read this manual.
* Overview::            Preliminary information.
* Examples::            Example Intercal programs and bugs.
* Concept Index::       Index of concepts.
```

Как вы видите, помимо текста имеется меню для перехода к другим «узлам». Нажав m и введя имя узла из меню, вы сможете прочесть текст соответствующего узла. Можно последовательно читать узлы, нажимая клавишу пробела, которая переводит к следующему узлу документа (указанному в информационной строке вверху буфера). В данном случае очередным узлом является первый узел меню — Instructions.

У каждого узла есть ссылка на родительский узел (Up), которым в данном случае является (dir), соответствующий каталогу страниц Info. Нажав u, вы переместитесь в родительский узел. Кроме того, у каждого узла есть ссылка на предыдущий узел, если он существует (в данном случае его нет). Команда p перемещает к предыдущему узлу. Команда l возвращает к последнему посещенному узлу.

Находясь в программе чтения Info, можно нажать ? для вывода списка команд и h для вывода краткого руководства по использованию системы. Поскольку вы запустили Info внутри Emacs, то можете использовать также и команды Emacs (например, C-x b для перехода в другой буфер).

Если система Info представляется вам загадочной и устаревшей, то учтите, что она разрабатывалась для работы на любых системах, включая те, в которых отсутствуют графические возможности и большие вычислительные мощности.

В Emacs есть и другая оперативная подсказка. Нажав C-h C-h, вы получите список опций подсказки. Одной из них является C-h k, после выбора которой можно нажать клавишу и получить документацию о связанной с ней функции.

Удаление, копирование и перемещение текста

Перемещать и дублировать блоки текста в Emacs можно различными способами. В этих методах используется объект *метка* (*mark*), представляющий собой сохраненное в памяти положение курсора. Метки можно устанавливать с помощью различных команд. Участок текста между текущим положением курсора (*point*) и меткой называется областью (*region*).

Метку можно установить клавишами C-@ (или C-Space в большинстве систем). Перемещение курсора в некоторое положение и нажатие C-@ устанавливает метку в этом положении. Теперь можно переместить курсор в другое место документа, и область будет определена как текст между меткой и текущим положением курсора.

Действие многих команд Emacs распространяется на область. Наиболее важные из них относятся к удалению и копированию текста. Команда C-w удаляет текущую область и помещает ее в *кольцо удаления* (*kill ring*). Кольцо удаления является списком удаленных текстовых блоков. Текст можно затем вставить (*yank*) в другое место командой C-y. (Семантика термина *yank* различна в *vi* и Emacs. В *vi* «yank» означает добавление участка текста к регистру отмены удаления без изъятия его из редактируемого текста, в то время как в Emacs «yank» означает вставку (paste) текста.) Используя кольцо уда-

ления, можно вставлять не только последний удаленный блок, но и предшествующие удаленные блоки.

Введите, например, в буфер Emacs следующий текст:

```
The rain falls on the main plains in Spain.
Here is a line that we wish to move.
She sells Bourne shells by the sea shore.

-1:**  wibble.txt        (Text)--L4--All----------------------------------
```

Теперь переместите курсор в начало второй строки («Here is a line...») и поставьте метку командой `C-@`. Переместитесь к концу строки (команда `C-e`) и удалите область командой `C-w`. Теперь буфер должен выглядеть так:

```
The rain falls on the main plains in Spain.

She sells Bourne shells by the sea shore.

-1:**  wibble.txt        (Text)--L2--All----------------------------------
```

Для того чтобы вставить только что удаленный текст, переместите курсор в конец буфера и нажмите `C-y`. Строка должна быть вставлена на новое место:

```
The rain falls on the main plains in Spain.

She sells Bourne shells by the sea shore.
Here is a line that we wish to move.

-1:**  wibble.txt        (Text)--L4--All----------------------------------
```

Повторное нажатие C-y приводит к многократной вставке текста.

Копирование текста можно производить аналогичным образом. Использование M-w вместо C-w приводит к копированию области в кольцо удаления без уничтожения ее в тексте. (Напомним, что M-w означает, что прежде чем нажать w, нужно нажать и удерживать клавишу <Alt> или нажать <Esc>.)

Текст, удаляемый с помощью других команд, например C-k, также помещается в кольцо удаления. Это значит, что для перемещения участка текста не обязательно устанавливать метку и использовать C-w, можно использовать любую команду удаления.

Для вставки ранее удаленных участков текста (сохраненных в кольце удаления) выполните команду M-y после вставки командой C-y. Команда M-y заменяет вставленный текст предыдущим блоком из кольца удаления. При повторном нажатии M-y происходит циклический перебор содержимого кольца удаления. Эта функция полезна при перемещении или копировании нескольких блоков текста.

В Emacs есть также более общий механизм *регистров*, аналогичный имеющемуся в *vi*. Помимо прочего, эту функцию можно использовать для сохранения текста, который вы собираетесь вставить позже. Имя регистра состоит из одного символа. В следующем примере используется имя a:

1. Установите метку в начало сохраняемого текста, нажав одновременно клавиши <Ctrl> и <пробел> (или, если это не работает, нажмите C-@).
2. Переместите курсор в конец сохраняемой области.
3. Нажмите C-x x, а затем введите имя регистра (в данном случае a).
4. Если вы хотите вставить текст в каком-либо месте, нажмите C-x g, а затем введите имя регистра a.

Поиск и замена

Чаще всего поиск строки осуществляется в Emacs командой C-s. Она инициирует так называемый *инкрементный поиск*. Вы начинаете вводить символы, которые хотите найти, и при вводе каждого символа Emacs осуществляет поиск в прямом направлении строки, состоящей из всех введенных вами символов. При ошибке нажмите клавишу <Delete> и продолжайте ввод правильных символов. Если строка не найдена, Emacs подает звуковой сигнал. Если вхождение найдено, но вы хотите продолжить поиск для другой строки, снова нажмите C-s.

Таким же образом можно осуществлять поиск и в обратном направлении, используя команду C-r. Есть несколько других типов поиска, в том числе поиск регулярных выражений, который можно инициировать, нажав M-C-s. Он позволяет отыскивать выражения типа jo.*n, которое соответствует, например, именам John, Joan и Johann (по умолчанию при поиске не различаются верхний и нижний регистры).

Для замены строки введите команду M-%. Вам будет показана та строка, которая находится в буфере в данный момент, и предложено ввести строку, кото-

рой ее нужно заменить. Emacs показывает все места в буфере, в которых обнаруживается заменяемая строка, запрашивая подтверждение на ее замену. Нажмите <пробел> для замены строки, клавишу <Delete> – для пропуска и точку – для прекращения поиска.

Если вы хотите заменить все вхождения строки, следующие за текущим положением в буфере, без запроса подтверждений, введите `M-x replace-string`. (Комбинация `M-x` позволяет ввести имя функции Emacs и выполнить ее, не используя привязку к клавишам. Многие функции Emacs доступны только через `M-x`, если вы сами не привяжете их к клавишам.) Замену регулярного выражения можно выполнить командой `M-x replace-regexp`.

Макросы

Название Emacs частично происходит от «macros». Макросы являются простой, но мощной возможностью, которая превращает работу в Emacs в удовольствие. Если вы собираетесь часто осуществлять какие-либо повторяющиеся операции, нажмите `C-x (`, выполните операцию и затем нажмите `C-x )`. Две команды `C-x` с открывающей и закрывающей скобками запоминают все ваши нажатия клавиш. Повторно выполнить команды можно, введя `C-x e`.

Приведем простой пример, который можно применить к любому текстовому файлу. В этом примере первое слово каждой строки записывается с прописной буквы.

1. Нажмите `C-x (`, чтобы начать запись макроса.
2. Нажмите `C-a`, чтобы переместить курсор в начало текущей строки. При каждом выполнении макроса нужно быть уверенным, что курсор находится в нужном месте. Нажимая `C-a`, вы можете быть уверены, что макрос начнет выполняться в начале строки, т. е. там, где нужно.
3. Нажмите `M-c`, чтобы сделать прописной первую букву первого слова.
4. Снова нажмите `C-a`, чтобы вернуться к началу строки, и `C-n` или <стрелка вниз>, чтобы перейти к началу следующей строки. Тем самым вы гарантируете, что в следующий раз макрос начнет выполнение с нужного места.
5. Нажмите `C-x )`, чтобы закончить запись макроса.
6. Нажмите несколько раз `C-x e`, чтобы последующие строки начинались с прописной буквы. Либо несколько раз нажмите `C-u`, а затем `C-x e`. Повторное нажатие `C-u` является префиксом, вызывающим многократное повторение следующей команды. Если вы дойдете до конца документа, а макрос все еще будет выполняться, ничего страшного не произойдет: Emacs подаст звуковой сигнал и прекратит выполнение макроса.

Выполнение команд и программирование в Emacs

Emacs предоставляет интерфейсы ко многим программам, которые можно запускать в буфере Emacs. Например, есть режимы работы Emacs для чтения и отправки электронной почты, чтения телеконференций, компиляции программ и взаимодействия с оболочкой. С некоторыми из этих функций мы познакомимся в этом разделе.

Для отправки из Emacs электронной почты нажмите `C-x m`. В результате будет открыт буфер, который позволит составить и отправить почтовое сообщение:

```
To:
Subject:
--text follows this line--

-1:--  *mail*          (Mail)--L1--All----------------------------------
```

Введите в этот буфер ваше сообщение и отправьте его командой `C-c C-s`. Можно вставлять текст из других буферов, расширять интерфейс с помощью собственных функций Emacs LISP и т. д.

Более того, в Emacs есть режим RMAIL, позволяющий читать электронную почту прямо в редакторе, но мы не станем о нем рассказывать, поскольку большинство пользователей предпочитает отдельные почтовые программы. (Обычно эти программы предлагают выбрать Emacs в качестве редактора почтовых сообщений.)

С почтовым интерфейсом RMAIL сходен интерфейс GNUS – программы чтения телеконференций, базирующейся на Emacs и запускаемой командой `M-x gnus`. После запуска (и некоторой обработки файла *.newsrc*) появляется список телеконференций вместе со счетчиком непрочтенных статей для каждой из них:

```
9: comp.os.linux.networking
5: comp.os.linux.setup
3: comp.os.linux.x

--:--  Gnus: *Group* {nntp:news.sonic.net}      (Grou
```

GNUS служит примером мощи использования интерфейсов Emacs к другим инструментам. В вашем распоряжении оказываются возможности навигации, поиска и создания макросов, предоставляемые Emacs, наряду со специфическими комбинациями клавиш, соответствующими используемому вами инструменту.

Используя клавиши со стрелками, можно выбрать телеконференцию для чтения. Нажмите клавишу <пробел>, чтобы начать чтение статей выбранной телеконференции. Будет выведено два буфера, в одном из которых отображается список статей, а в другом выводится текущая статья. Команды `n`

и p позволяют переместиться к следующей и предыдущей статьям, команды f и F служат для отправки ответа на текущую статью в телеконференцию (не включая или включая текст оригинала соответственно), а r и R служат для ответа на статью по электронной почте. В GNUS есть много других команд, список которых можно получить командой C-h m. Если вы привыкли пользоваться такой программой чтения телеконференций, как *rn*, то GNUS не покажется вам непривычной.

В Emacs есть ряд режимов для редактирования файлов различных типов. Например, есть режим C для редактирования исходного кода на C и режим TEX для редактирования (сюрприз!) исходного кода TEX. В каждом из этих режимов есть функции, облегчающие редактирование файлов соответствующего типа.

Например, в режиме C можно выполнить команду M-x compile, которая по умолчанию запускает *make -k* в рабочем каталоге и перенаправляет ошибки в другой буфер. Например, в буфере компиляции может быть такое содержание:

```
cd /home/loomer/mdw/pgmseq/
make -k
gcc -O -O2 -I. -I../include -c stream_load.c -o stream_load.o
stream_load.c:217: syntax error before `struct'
stream_load.c:217: parse error before `struct'
```

Можно переместить курсор на строку, содержащую сообщение об ошибке, нажать C-c C-c, и курсор переместится на соответствующую строку в соответствующем буфере с исходным кодом (при необходимости откроется буфер с соответствующим файлом исходного кода). Можно редактировать и компилировать программы, не выходя из Emacs.

Emacs обеспечивает также полный интерфейс к отладчику *gdb*, который описан в разделе «Использование Emacs с gdb» главы 14.

Обычно Emacs выбирает для буфера режим, основываясь на расширении имени файла. Например, редактирование файла с расширением *.c* автоматически устанавливает для буфера режим C.

Режим Shell является одним из наиболее популярных расширений Emacs. Он позволяет взаимодействовать с командным процессором в буфере Emacs, используя команду M-x shell. Редактирование командных строк, удаление и вставку можно осуществлять стандартными командами Emacs. Одиночные команды оболочки можно выполнять из Emacs, используя M-!. Если вместо этой команды использовать M-|, то заданной команде в качестве стандартного ввода передается по каналу содержимое текущей области. Это общий интерфейс запуска подпрограмм из Emacs.

Настройка Emacs

Электронной документации Emacs должно быть достаточно для того, чтобы больше узнать об этой системе и освоиться с работой в ней. Однако иногда трудно найти самые полезные советы, чтобы начать работать. Ниже мы

представим краткую сводку некоторых настроек, которые облегчают жизнь многим пользователям Emacs.

Личные настройки Emacs находятся в файле *.emacs*, расположенном в домашнем каталоге пользователя. Этот файл должен содержать код, написанный на Emacs LISP, который выполняет или определяет функции, настраивающие вашу среду Emacs. (Если вы никогда не писали на LISP, не беспокойтесь: большинство настроек, в которых он используется, очень просты.)

Чаще всего пользователи изменяют привязку клавиш. Например, если вы используете Emacs для редактирования документов SGML, вы можете привязать клавиши `C-c s` для переключения в режим SGML. Поместите в свой файл *.emacs* следующие строки:

```
; C-c и s переведут буфер в режим SGML.
(global-set-key «\C-cs» 'sgml-mode)
```

Комментарии в Emacs LISP начинаются с символа «точка с запятой». После комментария следует команда глобальной установки клавиш (*global-set-key*). Теперь вам не понадобится вводить длинную последовательность `M-x sgml-mode`, чтобы начать редактирование `SGML`. Нужно лишь нажать клавиши `C-c s`. Они сработают в любом месте Emacs независимо от режима текущего буфера, поскольку установлены как глобальные. (Конечно, Emacs может распознать файл `SGML` или `XML` по суффиксу и автоматически перевести его в режим `SGML`.)

Другая настройка, которая вам может понравиться, делает текстовый режим режимом по умолчанию и включает дополнительный режим «auto-fill», который заставляет текст, слишком длинный для одной строки, автоматически переноситься на следующую строку:

```
; Make text mode the default, with auto-fill
(setq default-major-mode 'text-mode)
(add-hook 'text-mode-hook 'turn-on-auto-fill)
```

Возможно, не всегда желательно глобальное назначение клавиш. При работе в режимах TEX, C и других вы обнаружите, что есть полезные вещи, которые нужны только в одном режиме. Сейчас мы определим простую LISP-функцию, вставляющую некоторые символы в исходный код на C, а затем для удобства привяжем эту функцию к клавише:

```
(defun start-if-block()
  (interactive)
  (insert «if () {\n}\n»)
  (backward-char 6)
)
```

Мы начали с того, что определили функцию как «интерактивную», чтобы ее мог вызывать пользователь, иначе ее можно было бы использовать только посредством других функций. Затем с помощью функции *insert* мы помещаем в буфер C следующие символы:

```
if () {
}
```

Строки в Emacs могут содержать стандартные управляющие символы C. Здесь мы использовали \n для перехода к новой строке.

Теперь у нас есть шаблон для блока if. Для пущей красоты наша функция перемещает курсор на шесть символов назад, помещая его внутрь скобок, чтобы сразу можно было начать ввод выражения.

Нашей задачей было облегчить ввод этих символов, поэтому привяжем функцию к клавише:

```
(define-key c-mode-map «\C-ci» 'start-if-block)
```

Функция *define-key* привязывает клавишу к функции. Задав c-mode-map, мы определили, что клавиша действует только в режиме C. Существуют также привязки клавиш tex-mode-map для режима TEX, lisp-mode-map, который вас заинтересует, если вы намерены серьезно заняться своим файлом *.emacs* и т. д.

Если вы хотите писать собственные функции на Emacs LISP, то следует прочесть страницы Info для *elisp*, которые должны иметься в вашей системе. Есть две хорошие книги о том, как писать функции на Emacs LISP: «An Introduction to Programming in Emacs Lisp» Роберта Чассела (Robert J. Chassel), GNU Press и «Writing GNU Emacs Extensions» Боба Гликстайна (Bob Glickstein), O'Reilly.

Теперь рассмотрим важную настройку, которая может вам понадобиться. На многих терминалах клавиша <Backspace> посылает символ C-h, который является клавишей подсказки в Emacs. Чтобы исправить это, следует изменить внутреннюю таблицу, используемую Emacs для интерпретации клавиш:

```
(keyboard-translate ?\C-h ?\C-?)
```

Довольно таинственный код. \C-h узнаваем как клавиша <Ctrl>, нажатая вместе с h, которая производит тот же код ASCII (8), что и клавиша <Backspace>. \C-? представляет клавишу <Delete> (код ASCII 127). Не путайте этот вопросительный знак с вопросительными знаками перед обратной косой чертой. ?\C-h означает «код ASCII, соответствующий \C-h». С тем же результатом можно непосредственно задать 8.

Теперь удаление будет происходить и при нажатии <Backspace>, и при нажатии C-h. Вы потеряли клавишу вызова подсказки. Поэтому другой полезной настройкой была бы привязка к C-h другой клавиши. Можно использовать C-\, которая редко для чего-либо используется. Нужно удваивать обратную косую черту, задавая ее в качестве клавиши:

```
(keyboard-translate ?\C-\\ ?\C-h)
```

В системе X Window есть способ изменить код, посылаемый клавишей <Backspace>, с помощью команды *xmodmap,* но мы оставим это для самостоятельных исследований. Это не вполне переносимое решение (поэтому мы не можем показать вам гарантированно работающий пример), и на ваш взгляд оно может оказаться слишком сильнодействующим (при этом изменяется значение клавиши <Backspace> в оболочке *xterm* и во всех других местах).

Вы можете применять и другие привязки клавиш. Например, вы захотите использовать клавиши C-f и C-b для прокрутки страницы вперед и назад, как в *vi*. Для этого в файл *.emacs* можно включить следующие строки:

```
(global-set-key «\C-f» 'scroll-up)
(global-set-key «\C-b» 'scroll-down)
```

Мы снова должны предупредить: старайтесь не переопределять клавиши, которые имеют другое важное применение. (Один из способов узнать, что делает клавиша в текущем режиме – выполнить команду C-h k. Следует также учесть, что клавиши могут быть определены и в других режимах.) В частности, вы потеряете доступ ко многим функциям, если измените привязку префиксных клавиш, начинающих команду, таких как C-x и C-c.

Если вы действительно хотите расширить некоторый режим множеством новых команд, можно создать собственные префиксные клавиши. Например, можно использовать:

```
(global-unset-key «\C-d»)
(global-set-key «\C-d\C-f» 'my-function)
```

Прежде всего, нужно «отвязать» клавишу C-d (которая удаляет символ под курсором), чтобы использовать ее в качестве префикса для других клавиш. Теперь нажатие C-d C-f будет выполнять *my-function*.

Возможно, вы захотите использовать для редактирования «обычных» файлов другие режимы, помимо Fundamental и Text. Например, режим Indented Text осуществляет автоматический отступ строк текста по отношению к предыдущей строке (как в функции *vi* :set ai). Для того чтобы этот режим включался по умолчанию, введите такие строки:

```
; Default mode for editing text
(setq default-major-mode 'indented-text-mode)
```

Следует также переопределить клавишу <Enter>, чтобы новая строка текста начиналась с отступа:

```
(define-key indented-text-mode-map «\C-m» 'newline-and-indent)
```

Emacs также поддерживает «дополнительные» режимы, которые используются вместе с основными. Например, режим Overwrite является дополнительным и обеспечивает замену текста в буфере вместо вставки. Чтобы привязать клавишу C-r к переключению режима вставки/замены, используйте команду

```
; Toggle overwrite mode.
(global-set-key «\C-r» 'overwrite-mode)
```

Другим дополнительным режимом является Autofill, который автоматически переносит строки при вводе. Это означает, что вместо нажатия <Enter> в конце каждой строки можно продолжать ввод текста, а Emacs будет автоматически разбивать текст на строки. Для включения режима Autofill используйте команды:

```
(setq text-mode-hook 'turn-on-auto-fill)
(setq fill-column 72)
```

Эти команды включают режим `Autofill` при входе в режим `Text` (с помощью функции *text-mode-hook*). Они также устанавливают длину строки в 72 символа.

Регулярные выражения

Даже небольшое число приемов, использующих регулярные выражения, может значительно повысить эффективность поиска в тексте и выполнения групповых замен. Долгое время регулярные выражения ассоциировались только с утилитами и языками в Unix, но теперь они появляются и в других средах, например в Microsoft .NET, но только Unix дает возможность пользоваться ими в самых разнообразных местах, например в текстовых редакторах и команде *grep*, где их могут применять обычные пользователи.

Допустим, вы просматриваете файл, содержащий сообщения электронной почты. Вы подписались на целый ряд почтовых списков рассылки с такими названиями, как `gyro-news` и `gyro-talk`, поэтому ищете строки Subject, в которых есть `gyro-`. Можно воспользоваться текстовым редактором или командой *grep* для поиска выражения

```
^Subject:.*gyro-
```

Это означает: «искать строки, начинающиеся с Subject:, после чего следует любое количество любых символов, а затем `gyro-`». Регулярное выражение состоит из нескольких частей, одни из которых воспроизводят обычный текст, а другие передают общие понятия, такие как «начало строки». На рис. 9.1 показано, что означают эти части и как они действуют вместе.

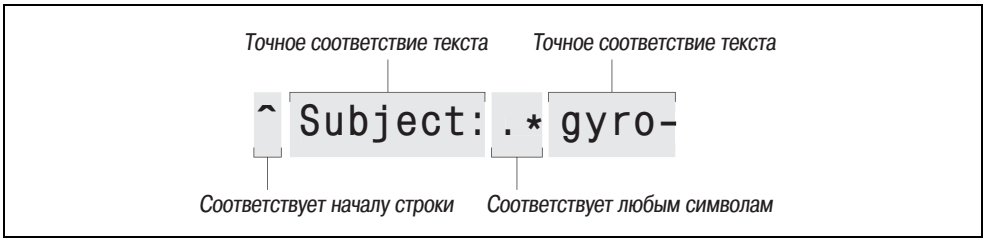


Рис. 9.1. *Простое регулярное выражение*

Чтобы дать представление о том, насколько мощными и сложными могут быть регулярные выражения, разовьем этот пример для осуществления более узкого поиска. Допустим, нам известно, что почтовые списки по гироскопам рассылают сообщения, строка Subject которых начинается с названия почтового списка, заключенного в квадратные скобки, например `Subject: [gyro-news]` или `Subject: [gyro-talk]`. Мы можем вести поиск именно таких строк:

```
^Subject: *\[gyro-[a-z]*\]
```

На рис. 9.2 показано, что означают части, из которых состоит это регулярное выражение. Мы отметим здесь несколько интересных моментов.

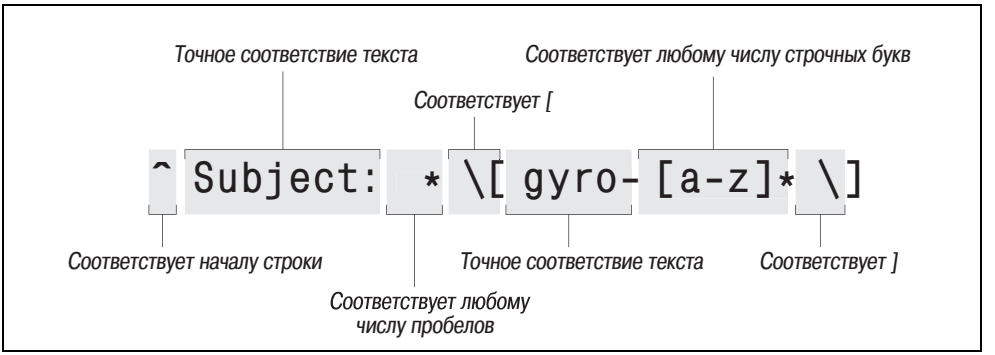


Рис. 9.2. *Регулярное выражение с большим количеством частей*

В регулярных выражениях скобки, крышки и звездочки представляют собой специальные символы. Квадратные скобки используются для обозначения целых классов символов, которые нужно искать. Например, `[a-z]` представляет «любую букву на нижнем регистре». Нам не нужно, чтобы скобка перед `gyro` выражала этот особый смысл, поэтому мы помещаем перед ней обратный слэш; это называется «преобразованием в escape-последовательность», или «обходом» (*escaping*). (Иными словами, мы позволяем квадратной скобке избежать рассмотрения в качестве метасимвола регулярного выражения.)

Первая звездочка в нашем выражении следует за пробелом, что означает «соответствует любому количеству смежных пробелов». Вторая звездочка следует за классом `[a-z]`, поэтому она применяется ко всей этой конструкции. Сама по себе конструкция `[a-z]` соответствует одной и только одной букве нижнего регистра. Вместе со звездочкой `[a-z]*` означает «соответствие любому количеству букв нижнего регистра, идущих подряд».

Чтобы научиться изощренно применять регулярные выражения, нужны недели. Рекомендуем прочесть книгу Джеффри Фридла (Jeffrey Friedl) «Mastering Regular Expressions», O'Reilly.[1]

Обработка текста и обработка документов

Зачем нужны анахронично выглядящие текстовые процессоры, описываемые в этом разделе, когда весь мир пользуется процессорами WYSIWYG, и даже для Linux есть несколько хороших? В действительности обработка текста (особенно в формате XML) – это то, что нас ждет в будущем. Пользователи будут стремиться к интерфейсам WYSIWYG, но на базе простого стандартного текстового формата, который сделает их документы переносимыми, предоставив при этом неограниченные возможности автоматизированной обработки документов.

[1] Джеффри Фридл «Регулярные выражения», 2-е издание, СПб: Питер, 2003.

Поскольку описываемые здесь средства являются программами с открытым исходным кодом и широко доступны, их форматами можно пользоваться без зазрения совести и обоснованно предполагать, что у читателей ваших документов есть доступ к форматтерам. Для работы с этими форматами и выполнения сложной обработки, такой, например, как создание библиографии в TEX, вы также можете применять широкий спектр инструментальных средств, разрабатывавшихся годами. Наконец, разработаны фильтры, хотя и не всегда совершенные, для преобразования документов в этих форматах в другие популярные форматы, в том числе коммерческие, и обратно. Поэтому вы не оказываетесь полностью замкнутыми на эти форматы, хотя для точного преобразования может потребоваться некоторая ручная работа.

В первой главе мы кратко остановились на различных системах обработки текста, существующих в Linux, и их отличиях от систем обработки слов, с которыми вы можете быть знакомы. В то время как большинство процессоров слов дают пользователю возможность вводить текст в среде WYSIWYG, системы обработки текста предлагают пользователю вводить исходный текст с использованием языка форматирования текста, который можно модифицировать в любом текстовом редакторе. (Фактически Emacs поддерживает специальные режимы редактирования для использования различных языков форматирования текста.) Затем исходный текст преобразуется в документ, пригодный для печати (или просмотра) с помощью собственно текстового процессора. Наконец, выходные данные обрабатываются и посылаются в файл или приложение, предназначенное для просмотра, либо передаются демону печати, который помещает их в очередь печати локального или удаленного устройства.

В этом разделе мы прежде всего поговорим о процессорах WYSIMWYG (what-you-see-is-maybe-what-you-get), которые доминируют в Windows и Macintosh. Затем мы обсудим некоторые наиболее популярные системы обработки текста для Linux: TEX, SGML, *groff* и Texinfo.

Процессоры слов

Одним из наглядных признаков того, что Linux вышла из младенческого возраста, является наличие в ней сегодня многочисленных популярных систем обработки слов типа WYSIWYG. Одно время даже ходили слухи, что Microsoft собирается перенести свой офисный пакет на Linux, но это вряд ли произойдет. В пакете Microsoft больше нет особой необходимости, поскольку имеются другие достаточно хорошие процессоры слов.

Одним из возможных вариантов является использование Anyware Office производства VistaSource, Inc. Anyware Office является коммерческим офисным пакетом, цена которого для Linux не высока. В него входят не только процессор слов, но и электронная таблица, графический редактор, почтовая программа и другие менее крупные программные средства. В некоторых отношениях Anyware Office работает иначе, чем процессоры слов Microsoft Word или WordPerfect, но если привыкнуть к нему, он становится полезным и удобным. Особенно следует отметить имеющуюся в нем поддержку

импорта и экспорта документов FrameMaker. Однако разработка этого пакета, похоже, замерла, и будущее его выглядит неопределенным.

Sun Microsystems предлагает свой офисный пакет StarOffice для ряда платформ, включая Linux. Некоторое время назад он был переведен в категорию open source и теперь развивается по двум направлениям: коммерческую версию StarOffice можно приобрести у Sun, а бесплатную версию OpenOffice можно загрузить с *http://www.openoffice.org*. Обе версии должны быть функционально примерно одинаковы, но в OpenOffice может не быть всех украшений в виде шаблонов, библиотек графики и пр., которые есть в StarOffice. OpenOffice распространяется по лицензии GPL, поэтому если вы мечтаете почитать 3 миллиона строк исходного кода офисного пакета, можете получить такую возможность.

Размер OpenOffice огромен, поэтому если у вас медленное соединение с Интернетом, проверьте сначала, нет ли этого пакета в вашем дистрибутиве.

Также существует KWord, процессор слов из офисного пакета KOffice для графической среды KDE. В последнее время KWord достиг значительного прогресса. В нем реализована интересная идея фреймов, которая делает его пригодным даже для выполнения некоторых задач настольной издательской системы. Но можно написать с его помощью и письмо бабушке. Дополнительные сведения о нем вы найдете на *http://www.kde.org*.

Наконец, хорошей и компактной альтернативой может явиться AbiWord, информация о котором есть на *http://www.abiwird.org*.

Все эти программы имеют одну общую характеристику, которую многие считают главным требованием для офисных приложений в Linux: они могут успешно импортировать документы Microsoft Word. Будучи неофитом Linux, вы можете решить, что не станете принимать документы, посылаемые вам в патентованных форматах, однако иногда они могут поступать от вашего начальника, и вы не сможете отказаться читать их лишь потому, что работаете в Linux. В таком случае приятно узнать, что есть решения для Linux.

Еще одну альтернативу предоставляет пакет LyX (имеющийся также в варианте KLyX с более современным интерфейсом пользователя). Он обеспечивает приемлемый X Window-интерфейс пользователя WYSIWYG, работающий с оконными менеджерами из стандартных дистрибутивов Linux и использующий при форматировании текста для печати пакеты LaTeX и TeX. Если вы можете смириться с ограничениями форматирования, налагаемыми этим пакетом (а большинство из нас может), то LyX/KLyX будет для вас отличным решением. LyX/KlyX не может отображать некоторые мощные функции форматирования, обеспечиваемые TeX, поэтому если вы усиленно используете TeX, то это решение не для вас. В большинство дистрибутивов Linux LyX/KlyX не входит. Для работы с ним вам придется загрузить его из архива Linux.

TEX и LATEX

TEX – это профессиональная система обработки текста для всех типов документов, статей и книг, особенно тех, которые содержат много математических символов. Отчасти это язык «низкого уровня», поскольку он описывает, как система должна расположить текст на странице, какими должны быть промежутки и т. д. TEX напрямую не связан с такими элементами текста высокого уровня, как главы, разделы, сноски и т. д. (то есть тем, что нас, писателей, заботит больше всего). Поэтому TEX известен как функциональный язык форматирования текста (относящийся к реальному физическому расположению текста на странице), а не логический (относящийся к логическим элементам, таким как главы и разделы).

TEX был разработан Дональдом Е. Кнутом, одним из крупнейших в мире специалистов по программированию. Одним из мотивов, побудивших его разработать TEX, было стремление разработать систему набора текста, достаточно мощную для обслуживания потребностей форматирования математических выражений в написанной им серии книг по компьютерным наукам. Кнуту потребовалось восемь лет, чтобы закончить разработку TEX, и большинство согласно с тем, что результат стоит ожиданий.

Конечно, TEX обладает большими возможностями расширения и допускает создание макросов, позволяющих авторам сосредоточиться преимущественно на логическом, а не физическом формате документа. И действительно, разработано несколько таких макропакетов, наиболее популярным из которых является LATEX – набор расширений TEX, созданный Лесли Лэмпортом (Leslie Lamport). Команды LATEX касаются, по большей части, логической структуры, но поскольку LATEX является просто набором макросов поверх TEX, он позволяет использовать и собственные команды TEX. LATEX значительно упрощает использование TEX, скрывая от пользователя значительную часть функций низкого уровня.

Для создания хорошо структурированных документов с помощью TEX нужно либо использовать готовый пакет макросов, такой как LATEX, либо разработать свой собственный (или использовать комбинацию обоих). В книге «The TEX book» (Addison Wesley) Кнут представил собственный набор макросов, использовавшихся им при создании своей книги. Как и можно было ожидать, в их число входят команды для начала новых глав, разделов и т. п. Эти команды имеют сходство с соответствующими командами LATEX. В этом разделе мы сосредоточимся на использовании LATEX, который обеспечивает поддержку многих типов документов: технических статей, руководств, книг, писем и т. д. Как и чистый TEX, LATEX допускает расширения.

Как это работает

Если вы никогда раньше не работали с системой форматирования текста, то нужно ознакомиться с рядом понятий. Как мы уже говорили, система обработки текста начинает с исходного текста, создаваемого с помощью обычного текстового редактора, такого как Emacs. Исходный текст создается на языке форматирования текста, что означает ввод как текста, который должен поя-

виться в документе, так и команд, сообщающих текстовому процессору, как он должен отформатировать этот текст. В первой главе мы привели простой пример того, как выглядит язык LaTeX и какого рода вывод он производит.

Без дальнейших предисловий перейдем к практике. Напишем простой документ и отформатируем его от начала и до конца. Для иллюстрации мы покажем, как с помощью LaTeX написать короткое деловое письмо. Возьмите свой любимый редактор и введите следующий текст (конечно, без номеров строк). Сохраните его в файле *letter.tex*:

```
1  \documentclass{letter}
2  \address{755 Chmod Way \\ Apt 0x7F \\
3           Pipeline, N.M. 09915}
4  \signature{Boomer Petway}
5
6  \begin{document}
7  \begin{letter}{O'Reilly and Associates, Inc. \\
8                 1005 Gravenstein Highway North \\
9                 Sebastopol, C.A. 95472}
10
11 \opening{Dear Mr. O'Reilly,}
12
13 I would like to comment on the \LaTeX\ example as presented in
14 Chapter~9 of {\em Running Linux}. Although it was a valiant effort,
15 I find that the example falls somewhat short of what
16 one might expect in a discussion of text-formatting systems.
17 In a future edition of the book, I suggest that you replace
18 the example with one that is more instructive.
19
20 \closing{Thank you,}
21
22 \end{letter}
23 \end{document}
```

Это полный документ LaTeX для делового письма, которое мы хотим послать. Как видите, в нем содержится фактический текст письма и ряд команд (с использованием обратных косых черт и фигурных скобок). Разберем его последовательно.

В первой строке содержится команда `documentclass`, указывающая тип создаваемого документа (в данном случае `letter` – письмо). В LaTeX команды начинаются с обратной косой черты, за которой следует действительное имя команды, в данном случае `documentclass`. За командой следуют аргументы, заключаемые в фигурные скобки. В LaTeX поддерживается несколько типов документов, в том числе `article` (статья), `report` (отчет) и `book` (книга), а кроме того, вы можете определить собственный тип. Задание типа в документе TeX определяет глобальные макросы, которые будут использоваться в документе, такие как команды `address` и `signature`, используемые в строках 2–4. Как можно догадаться, команды `address` и `signature` указывают в письме ваш собственный адрес и имя. Двойная обратная черта (\\) в адресе выполняет перевод строки в результирующем тексте адреса.

Несколько слов о том, как LaTeX обрабатывает входные данные. Как и в большинстве систем форматирования текста, пробелы, переводы строк и другие подобные элементы не передаются буквально в окончательный вывод, поэтому можно более или менее произвольно расставлять концы строк: при форматировании абзацев LaTeX снова соберет строки вместе. Есть, правда, исключения: пустые строки во входных данных начинают новые абзацы, и существуют команды, заставляющие LaTeX воспринимать исходный текст буквально.

В строке 6 команда `\begin{document}` используется для обозначения начала самого документа. Все, заключенное между `\begin{document}` и `\end{document}` в строке 22, считается частью форматируемого текста; все, что находится перед `\begin{document}`, называется *преамбулой* и задает параметры форматирования перед фактическим телом документа.

Само письмо начинается со строк 7–9 командой `\begin{letter}`. Это необходимо потому, что в одном исходном файле может быть много писем, и для каждого требуется `\begin{letter}`. Эта команда принимает в качестве аргумента адрес предполагаемого получателя. Как и в команде `address`, двойная обратная косая черта обозначает переход на новую строку.

Строка 11 использует команду `opening` для открытия письма. За ней в строках 12–18 располагается фактическое тело письма. Как ни просто оно выглядит, но и в нем заключены некоторые особенности. В строке 13 команда LaTeX генерирует логотип LaTeX. Заметьте, что обратная косая черта присутствует как перед командой LaTeX, так и после нее. Замыкающая косая черта принуждает вставить пробел после слова «LaTeX». Это связано с тем, что TeX игнорирует пробелы после вызовов команд, поэтому за командой должны следовать обратная косая черта и пробел. (В противном случае «LaTeX//» будет выглядеть, как «LaTeXis fun».)

В строке 14 есть две детали, заслуживающие комментария. Прежде всего, это тильда (~), присутствующая между `Chapter` и `9`. Этот символ вызывает появление пробела между двумя словами, но предотвращает разрыв строки между ними в окончательном тексте (не допуская, таким образом, появления слова «Chapter» в конце строки и «9» – в начале следующей строки). Тильду нужно использовать только для обозначения пробела между двумя словами, которые должны располагаться на одной строке, как в `Chapter~9` и `Mr.~Jones`. (Оглядываясь назад, мы могли бы использовать тильду в командах `\begin{letter}` и `opening`, но маловероятно, что TeX разобьет строку внутри адреса или обращения.)

Второе, на что нужно обратить внимание в строке 14, это использование `\em` для создания *выделенного текста* в выдаче. LaTeX поддерживает и другие шрифты, в том числе **boldface** (`\bf`) и `typewriter` (`\tt`).

В строке 20 письмо закрывается командой `closing`. Помимо этого она выполняет следующее действие: по окончании вывода приписывается подпись, определенная в строке 4. В строках 21–22 для завершения среды письма и документа, начатых в строках 6 и 7, используются команды `\end{letter}` и `\end{document}`.

Вы можете заметить, что команды в исходном тексте LaTeX совершенно не касаются таких функций, как установка полей, межстрочного интервала или других практических вопросов форматирования текста. Всем этим занимаются макросы LaTeX, работающие поверх ядра TeX. LaTeX устанавливает для этих параметров разумные значения по умолчанию. Если вам потребуется изменить какие-либо из этих параметров форматирования, можно использовать для этого другие команды LaTeX (или команды TeX низкого уровня).

Конечно, нельзя разобраться во всех сложностях использования LaTeX на таком ограниченном примере, но он должен дать представление о том, как выглядит живой документ LaTeX. Теперь давайте отформатируем документ, чтобы вывести его на печать.

Форматирование и печать

Можете не верить, но команда, используемая для превращения исходных текстов LaTeX в нечто печатаемое, называется *latex*. После редактирования и сохранения предыдущего примера в файле *letter.tex* вы можете выполнить команду

```
eggplant$ latex letter
This is TeX, Version 3.14159 (Web2C 7.3.1)
(letter.tex
LaTeX2e <2000/06/01>
Babel <v3.7h> and hyphenation patterns for american, french, german, ngerman, no
hyphenation, loaded.
(/usr/share/texmf/tex/latex/base/letter.cls
Document Class: letter 1999/04/29 v1.2z Standard LaTeX document class
(/usr/share/texmf/tex/latex/base/size10.clo))
No file letter.aux.
[1] (letter.aux) )
Output written on letter.dvi (1 page, 1128 bytes).
Transcript written on letter.log.
eggplant$
```

latex предполагает, что исходные файлы имеют расширение *.tex*. В данном случае LaTeX обработал исходный текст *letter.tex* и сохранил результат в файле *letter.dvi*. Это «независимый от устройства» файл, который может быть выведен на ряд принтеров. Существуют различные утилиты преобразования файлов *.dvi* в PostScript, HP LaserJet и другие форматы, что мы вскоре покажем.

Прежде чем печатать письмо, вы, возможно, пожелаете его просмотреть, чтобы убедиться, что все выглядит нормально. Если вы работаете в системе X Window, то для предварительного просмотра файлов *.dvi* на экране можно использовать команду *xdvi*. Если у вас также стоит KDE, то там есть *kdvi*, которая более дружественна, чем *dvi*. Так как же все-таки напечатать письмо? Прежде всего, необходимо конвертировать *.dvi* в формат, с которым может работать ваш принтер. Драйверы DVI существуют для многих типов принтеров. Почти всегда имена соответствующих программ начинаются с букв *dvi*, например *dvips*, *dvilj* и т. д. Если в вашей системе нет необходимого

драйвера, нужно найти его в архивах в Интернете. Подробности см. в FAQ для *comp.text.tex*.

Если вы счастливый обладатель принтера PostScript, то можете преобразовать *.dvi* в PostScript с помощью *dvips*:

```
eggplant$ dvips -o letter.ps letter.dvi
```

Затем можно вывести PostScript на печать с помощью *lpr*. Или сделать это за один шаг:

```
eggplant$ dvips letter.dvi | lpr
```

Есть также драйверы для конкретных принтеров, например *dvilj* для HP LaserJet, но в большинстве своем они считаются устаревшими. Воспользуйтесь лучше *dvips* и при необходимости Ghostscript (см. ниже).

Если вы не можете найти драйвер DVI для своего принтера, то, возможно, вам удастся с помощью Ghostscript преобразовать PostScript (полученный с помощью *dvips*) в какой-нибудь другой формат, который можно напечатать. Хотя некоторые шрифты Ghostscript далеко не оптимальны, он все же позволяет использовать шрифты Adobe (которые можно получить для Windows и использовать с Ghostscript под Linux). В Ghostscript есть также режим просмотра на SVGA, который можно использовать, если вы не работаете в X. Если вам удастся отформатировать и напечатать образец письма, оно должно выглядеть примерно так, как на рис. 9.3.

755 Chmod Way
Apt 0x7F
Pipeline, N.M. 09915

June 5, 1996

O'Reilly and Associates, Inc.
103 Morris Street Suite A
Sebastopol, C.A. 95472

Dear Mr. O'Reilly,

I would like to comment on the LaTeX example as presented in Chapter 9 of *Running Linux*. Although it was a valiant effort, I find that the example falls somewhat short of what one might expect in a discussion of text formatting systems. In a future edition of the book, I suggest that you replace the example with one that is more instructive.

Thank you,

Boomer Petway

Рис. 9.3. *Пример вывода файла*

Наконец, нужно также отметить, что с помощью TEX можно создавать файлы PDF, используя для этого драйвер *dvipdf* или специальную программу *pdftex*.

Что еще прочесть

Если LATEX подходит вам для обработки документов и вам удастся напечатать хотя бы этот начальный пример, мы советуем обратиться к книге Лесли Лэмпорта (Leslie Lamport) «LATEX User's Guide and Reference Manual» (Addison Wesley), в которой описано все, что нужно знать о LATEX для форматирования писем, статей, книг и прочего. Если вы интересуетесь программированием или хотите больше узнать о том, как работает TEX (что может оказаться очень ценным), то книга Дональда Кнута «The TEX book» (Addison Wesley) является исчерпывающим руководством по системе.

comp.text.tex является телеконференцией по этим системам, хотя данные, которые вы там обнаружите, предполагают, что у вас есть доступ к той или иной документации по TEX и LATEX, например к отмеченным выше руководствам.

SGML, XML и Docbook

SGML и XML делают еще один шаг вперед по сравнению с языками разметки, которые мы обсуждали до сих пор. Они налагают на текст структуру, которая показывает отношение каждого элемента к содержащим его элементам. Это делает возможным преобразование текста в ряд выходных форматов, включая PostScript и PDF (Adobe Portable Document Format).

SGML сам по себе дает лишь базу для определения структуры документа. Так называемое описание типа документа (Document Type Definition, или DTD) или схема устанавливают затем, какого рода разметку можно использовать в документе.

SGML не так широко известен, как два его потомка – HTML и XML, вокруг которых возник даже излишний ажиотаж. По существу, HTML является реализацией SGML с фиксированным набором тегов, полезной для форматирования веб-страниц. XML является общим решением, как и SGML, но лишен некоторых сложных функций последнего. Как SGML, так и XML дают пользователям возможность определить любой набор тегов. Конкретно теги и их связи указываются в DTD или схеме (в XML это необязательно).

Для каждого DTD или схемы, которые нужно использовать, необходимо иметь средства обработки, переводящие файл SGML или XML в нужный выходной формат. Исторически большинство бесплатных систем делало это с помощью системы под названием DCSSL. Сейчас для преобразования XML в другие форматы гораздо популярнее XSLT. Но это совершенно не должно беспокоить вас, если вы не стремитесь изменить внешний вид результата.

В области документации по компьютерам чаще всего используется DTD с названием DocBook. Помимо всего прочего, с помощью DocBook написана как большая часть бесплатной документации по Linux, так и эта книга. В число пользователей DocBook входит огромное число компаний и известных орга-

низаций, включая Sun Microsystems, Microsoft, IBM, Hewlett Packard, Boeing и Госдепартамент США.

Чтобы привести пример того, как может выглядеть текст DocBook, покажем фрагмент статьи для компьютерного журнала:

```
<!DOCTYPE Article  PUBLIC "-//OASIS//DTD DocBook V4.1.2//EN">
<article>
  <artheader>
    <title>Looping the Froz with Foobar</title>
    <author>
      <firstname>Helmer B.</firstname>
      <surname>Technerd</surname>
      <affiliation>
        <orgname>Linux Hackers, Inc.</orgname>
      </affiliation>
    </author>
  </artheader>
  <abstract>
    <para>This article describes a technique that you can employ to
loop the Froz with the Foobar software package.</para>
  </abstract>
  <sect1>
    <title>Motivation</title>
    <para>Blah, blah, blah, ...
    </para>
  </sect1>
</article>
```

В первой строке указано используемое DTD и корневой элемент. В данном случае мы создаем статью с использованием DocBook DTD. Остальная часть кода содержит саму статью. Если вы знакомы с HTML (языком разметки для WWW), текст должен показаться довольно знакомым. Теги используются для логической разметки текста. Все об HTML см. в книге Чака Муссиано (Chuck Musciano) и Билла Кеннеди (Bill Kennedy) «HTML & XHTML: The Definitive Guide», O'Reilly.[1]

Полное описание DocBook DTD выходит за рамки этой книги, но если вы заинтересовались, прочтите «DocBook: The Definitive Guide» Нормана Уолша (Norman Walsh) и Леонарда Мюльнера (Leonard Muellner), O'Reilly.

Когда вы напишете свою статью, документацию или книгу, то, конечно, захотите преобразовать их в формат, пригодный для печати или показа на экране. Для того чтобы сделать это, вам потребуется полностью установить SGML, что, к сожалению, сделать не так просто. В действительности это такая большая задача, что описать ее здесь невозможно. Но не все потеряно: ряд дистрибутивов, включая Red Hat, SuSE и Debian, поставляется с очень хорошими готовыми настройками SGML; нужно просто установить соответ-

[1] Чак Муссиано, Билл Кеннеди «HTML & XHTML. Подробное руководство», 4-е издание. – Пер. с англ. – СПб: Символ-Плюс, 2002.

ствующие пакеты SGML, которые в них входят. Если у вас есть работоспособная система SGML, то можно преобразовать приведенный выше текст в HTML (как один из вариантов) с помощью такой команды:

```
owl$ db2html myarticle.sgml
input file was called  -- output will be in myarticle
TMPDIR is db2html.GUBPgh
working on ../myarticle.sgml
about to copy cascading stylesheet and admon graphics to temp dir
about to rename temporary directory to "myarticle"
```

В файле *myarticle/t1.html* будет находиться сгенерированный HTML. Если вместо этого требуется сгенерировать PDF, нужно выполнить такую команду:

```
owl$ db2pdf myarticle.sgml
tex output file name is myarticle.tex
tex file name is myarticle.tex
pdf file name is myarticle.pdf
This is pdfTeX, Version 3.14159-13d (Web2C 7.3.1)
(myarticle.tex[/var/lib/texmf/pdftex/config/pdftex.cfg]
JadeTeX 2001/07/19: 3.11
(/usr/share/texmf/tex/latex/psnfss/t1ptm.fd)
Elements will be labelled
Jade begin document sequence at 20
No file myarticle.aux.
(/usr/share/texmf/tex/latex/cyrillic/t2acmr.fd)
(/usr/share/texmf/tex/latex/base/ts1cmr.fd)
(/usr/share/texmf/tex/latex/hyperref/nameref.sty)
(/usr/share/texmf/tex/latex/psnfss/t1phv.fd) [1.0.31[/var/lib/texmf/dvips/confi
g/pdftex.map]] (myarticle.aux) )<8r.enc>
Output written on myarticle.pdf (1 page, 4446 bytes).
Transcript written on myarticle.log.
```

Как видите, эта команда основывается на TEX, или, если говорить точнее, на специальной версии его под названием Jade, которая конкретно нацелена на документы, создаваемые DSSSL.

SGML – это целый новый мир инструментов и технологий. Хорошей отправной точкой для вдохновения и его освоения может быть веб-сайт проекта Linux Documentation Project, который, как отмечалось ранее, использует SGML/DocBook для всей своей документации. Вы обнаружите Linux Documentation Project на *http://www.tlpd.org*.

groff

Параллельно TEX и независимо от него развивались *troff* и *nroff* – две другие значительные системы обработки текста, разработанные в Bell Labs для первоначальной реализации Unix (фактически разработка Unix была частично ускорена необходимостью поддержки такой системы обработки текста). Первая версия этого текстового процессора называлась *roff* (от «runoff»). Затем появились *nroff* и *troff*, которые генерировали вывод для конкретных наборных машин, использовавшихся в то время. *nroff* был написан для

принтеров с фиксированным шагом, например матричных принтеров, а *troff* – для устройств с пропорциональным шагом, например, первоначально для наборных машин. Более поздние версии *nroff* и *troff* повсеместно стали стандартным текстовым процессором на системах Unix. *groff* является реализацией GNU *nroff* и *troff*, используемой в системах Linux. Он включает в себя несколько расширенных функций и драйверы устройств для ряда печатающих устройств.

С помощью *groff* можно создавать документы, статьи и книги весьма схожим с TEX образом. Однако в *groff* (и в первоначальном *nroff*) есть одна неотъемлемая функция, отсутствующая в TEX и его разновидностях: возможность выдавать результат в простом ASCII. В то время как TEX великолепен для создания документов для печати, *groff* может производить простой ASCII-текст, который можно просматривать на экране или печатать даже на простейших принтерах. Если вам нужно создавать документы, которые требуется и просматривать, и печатать, то, может быть, стоит обратить внимание на *groff* (хотя есть и другие альтернативы, в том числе программа Texinfo, о которой мы расскажем ниже).

Достоинство *groff* состоит и в значительно меньшем по сравнению с TEX размере: он требует меньше вспомогательных файлов и исполняемых модулей, чем минимальный дистрибутив TEX.

Одним из специальных применений *groff* является форматирование страниц руководства Unix. Если вы программируете в Unix, то, в конце концов, вам понадобится писать и оформлять те или иные страницы руководства. В этом разделе мы познакомим вас с *groff* на примере написания короткой страницы руководства.

Как и TEX, *groff* использует особый язык форматирования для описания того, как должен быть отформатирован текст. Этот язык несколько менее понятен, чем используемый в TEX, но и более лаконичен. Кроме того, есть несколько пакетов макросов для *groff*, выполняемых поверх базового форматтера *groff*. Эти пакеты макросов настроены на конкретный тип документа. Например, макросы `mgs` служат идеальным выбором при написании статей, а макросы `man` используются для создания страниц руководства.

Создание страницы руководства

С помощью *groff* очень просто создавать страницы руководства. Для того чтобы ваша страница выглядела подобно другим страницам руководства, при написании исходного кода нужно соблюдать несколько правил, которые иллюстрируются следующим примером. Мы напишем страницу руководства для гипотетической команды *coffee*, которая может управлять сетевой кофеваркой.

Введите в вашем текстовом редакторе следующий текст и сохраните его в файле *coffee.man*:

```
1  .TH COFFEE 1 «23 March 94»
2  .SH NAME
```

```
3  coffee \- Управление удаленной кофеваркой
4  .SH SYNOPSIS
5  \fBcoffee\fP [ -h | -b ] [ -t \fItype\fP ] \fIamount\fP
6  .SH DESCRIPTION
7  \fIcoffee\fP ставит в очередь запрос к удаленной кофеварке в устройстве
8  \fB/dev/cf0\fR. Обязательный аргумент \fIamount\fP задает
9  число чашек, обычно в диапазоне от 0 до 15, на кофеварках,
10 работающих в стандарте ISO.
11 .SS Options
12 .TP
13 \fB-h\fP
14 Варить горячий кофе. По умолчанию - холодный.
15 .TP
16 \fB-b\fP
17 Сжечь кофе. Особенно полезно при выполнении \fIcoffee\fP по поручению
18 вашего начальника.
19 .TP
20 \fB-t \fItype\fR
21 Указать сорт кофе для варки, \fItype\fP может иметь значение
22 \fBколумбийский\fP, \fBобычный\fP, \fBбез кофеина\fP.
23 .SH FILES
24 .TP
25 \fI/dev/cf0\fR
26 Устройство удаленной кофеварки
27 .SH «SEE ALSO»
28 молоко (5), сахар (5)
29 .SH BUGS
30 Может потребовать вмешательства оператора при израсходовании запаса кофе.
```

Пусть вас не пугает непонятность этого текста. Вам станет легче, когда вы узнаете, что последовательности символов `\fB`, `\fI` и `\fR` используются для изменения стиля шрифта на полужирный, курсив и roman соответственно. `\fP` сбрасывает шрифт в предыдущий установленный.

Другие запросы *groff* появляются в строках, начинающихся с точки (.). В строке 1 запрос `.TH` устанавливает, что заголовком страницы руководства является `COFFEE`, а разделом руководства является 1. (Раздел 1 используется для команд пользователя, раздел 2 – для системных вызовов и т. д.) Запрос `.TH` устанавливает также дату последнего пересмотра страницы.

Запрос `.SH` в строке 2 начинает раздел, озаглавленный `NAME`. Заметьте, что почти все страницы руководства по Unix используют следующую последовательность разделов: `NAME`, `SYNOPSIS`, `DESCRIPTION`, `FILES`, `SEE ALSO`, `NOTES`, `AUTHOR` и `BUGS`. При необходимости включаются дополнительные разделы. Это просто соглашение, используемое при составлении страницы, никак не требуемое программным обеспечением.

В строке 3 даются имя команды и, после дефиса (`\-`), ее краткое описание. Этот формат должен использоваться в разделе `NAME`, чтобы страницу руководства можно было ввести в базу данных *whatis*, используемую командами *man -k* и *apropos*.

В строках 4–5 дан краткий обзор синтаксиса команды *coffee*. Обратите внимание, что для обозначения параметров командной строки используется курсив `\fI. . .\fP`, а необязательные параметры заключены в квадратные скобки.

В строках 6–10 дано краткое описание команды. Курсив служит обычно для обозначения команд, имен файлов и параметров пользователя. В строке 11 запрос `.SS` начинает подраздел с именем `Options`. Затем в строках 11–22 следует список параметров, представленный в виде маркированного списка. Каждый элемент списка помечен запросом `.TP`. Строка *после* `.TP` является тегом элемента, за которым следует непосредственно его текст. Например, исходный код строк 12–14:

```
.TP
\fB-h\fP
Варить горячий кофе. По умолчанию - холодный.
```

в итоге будет выглядеть как:

```
-h      Варить горячий кофе. По умолчанию - холодный.
```

Каждый параметр командной строки должен документироваться в таком виде.

Строки 23–26 образуют раздел `FILES` страницы руководства, в котором описаны все файлы, которые могут быть использованы командой при работе. Для этого раздела также используется маркированный список с запросами `.TP`.

В строках 27–28 представлен раздел `SEE ALSO`, в котором даны перекрестные ссылки на другие страницы руководства, заслуживающие внимания. Обратите внимание, что строка «`SEE ALSO`» после запроса `.SH` в строке 27 заключена в кавычки. Это необходимо, поскольку `.SH` использует в качестве заголовка раздела первый аргумент, отделяемый пробелом. Поэтому все заглавия разделов, состоящие из нескольких слов, следует заключать в кавычки. Наконец, в строках 29–30 представлен раздел `BUGS`.

Форматирование и установка страницы руководства

Чтобы отформатировать эту страницу руководства и просмотреть ее на экране, выполните команду

```
eggplant$ groff -Tascii -man coffee.man | more
```

Параметр *-Tascii* требует, чтобы *groff* создал вывод в простом ASCII; параметр *-man* требует, чтобы *groff* использовал набор макросов для страниц руководства. В случае успеха страница руководства будет выведена в следующем виде:

```
COFFEE(1)                                          COFFEE(1)

NAME
       coffee - Управление удаленной кофеваркой

SYNOPSIS
       coffee [ -h | -b ] [ -t type ] amount
```

```
DESCRIPTION
       coffee ставит в очередь запрос к удаленной кофеварке в устройстве
       /dev/cf0. Обязательный аргумент amount задает число чашек, обычно
       в диапазоне от 0 до 15 на кофеварках, работающих в стандарте ISO.

   Options
       -h     Варить горячий кофе. По умолчанию - холодный.

       -b     Сжечь кофе. Особенно полезно при варке кофе
              по поручению вашего начальника.

       -t type
              Указать сорт кофе для варки, type может иметь
              значение колумбийский, обычный, без кофеина.

FILES
       /dev/cf0
              Устройство удаленной кофеварки

SEE ALSO
       milk(5), sugar (5)

BUGS
    Может потребовать вмешательства оператора при израсходовании запаса кофе.
```

Как уже говорилось, *groff* может производить другие типы вывода. Использование параметра *-Tps* вместо *-Tascii* приводит к выводу результата в формате PostScript, который можно сохранить в файле, просмотреть с помощью Ghostview или вывести на принтер PostScript. Параметр *-Tdvi* порождает вывод в независимом от устройства формате *.dvi*, аналогичном производимому TEX.

Если вы хотите, чтобы страницу руководства могли видеть на вашей машине другие пользователи, нужно установить исходный текст для *groff* в каталоге, указанном в пользовательской переменной окружения MANPATH. Стандартные страницы руководства находятся в каталоге */usr/share/man*, хотя в некоторых системах используются также */usr/man* и */usr/local/man*. Поэтому текст страниц руководства раздела 1 должен находиться в */usr/man/man1*. Команда

```
eggplant$ cp coffee.man /usr/man/man1/coffee.1
```

устанавливает эту страницу руководства для всеобщего использования в */usr/man* (обратите внимание на использование расширения *.1* вместо *.man*). Если после этого выполнить *man coffee*, то страница руководства будет автоматически переформатирована и текст для просмотра записан в */usr/man/cat1/coffee.1.gz*.

Если вам не разрешено копировать исходный текст страниц руководства прямо в */usr/man*, можно создать собственное дерево каталогов для страниц руководства и добавить его к своей переменной MANPATH. (См. раздел «Страницы руководства» главы 4.)

Texinfo

Texinfo является системой форматирования текста, используемой в проекте GNU для создания из одного и того же файла исходного текста электронной документации в виде гипертекстовых страниц Info и печатных материалов с помощью TEX. Из исходного текста Texinfo пользователи могут преобразовывать документацию в любой из форматов: Info, HTML, DVI, PostScript, PDF или простой текст.

Документация по Texinfo составлена целиком через собственные страницы Info, которые можно читать в Emacs (используя команду `C-h i`) или с помощью отдельной программы чтения Info, такой как *info*. Если в вашей системе установлены страницы GNU Info, то в них содержится полная документация по Texinfo. Подобно тому как *groff* используется для составления страниц руководства, Texinfo используется для составления документа Info.

Написание исходного кода Texinfo

В этом разделе мы покажем простой исходный файл Texinfo (иногда отрывки из него) и опишем, что делает каждый такой отрывок.

Наш исходный файл Texinfo будет называться *vacuum.texi*. Как обычно, исходный текст можно создать с помощью простого текстового редактора:

```
\input texinfo @c -*-texinfo-*-
@c %**start of header
@setfilename vacuum.info
@settitle The Empty Info File
@setchapternewpage odd
@c %**end of header
```

Это заголовок исходного текста Texinfo. Первая строка является командой TEX, используемой для ввода макросов Texinfo при создании печатной документации. Команды Texinfo начинаются с символа `@`. Команда `@c` служит началом комментария; комментарий `-*-texinfo-*-` является тегом, сообщающим Emacs, что это исходный текст Texinfo, благодаря чему Emacs может установить правильный основной режим. (Основные режимы обсуждались выше в разделе «Настройка Emacs».)

Комментарии `@c %**start of header` и `@c %**end of header` служат для выделения заголовка Texinfo. Это требуется, если вы хотите отформатировать лишь часть файла Texinfo. Команда `@setfilename` указывает имя получаемого в результате файла Info, `@settitle` устанавливает заглавие документа, а `@setchapternewpage odd` сообщает Texinfo о необходимости начинать новые главы на нечетных страницах. Это стандартные процедуры, которые должны использоваться для всех файлов Texinfo.

Следующий раздел исходного файла задает титульный лист, используемый при форматировании документа с помощью TEX. Смысл этих команд должен быть очевиден:

```
@titlepage
@title Vacuum
@subtitle The Empty Info File
@author by Tab U. Larasa
@end titlepage
```

Теперь перейдем к телу исходного файла Texinfo. Файл Info делится на узлы, которые являются чем-то вроде страниц документа. Каждый узел имеет ссылки на следующий, предыдущий и родительский узлы, а также может быть связан с другими узлами перекрестными ссылками. Каждый узел можно представлять себе как главу или раздел документа, с расположенным внизу меню для перехода к другим узлам. Например, узел уровня главы имеет меню, перечисляющее разделы этой главы. Каждый узел раздела указывает на узел уровня глав, являющийся его родителем. Каждый раздел указывает также на предыдущий и последующий разделы, если они существуют. Это несколько запутанно, но станет ясным, когда вы увидите такую организацию в действии.

Каждому узлу дается короткое имя. Самый верхний узел называется `Top`. Создание узла начинается с команды `@node`, принимающей в качестве аргументов имя узла, имя следующего узла, имя предыдущего узла и имя родительского узла. Как было отмечено ранее, следующий и предыдущий узлы должны находиться на том же уровне иерархии. Родительский узел находится над текущим в дереве узлов (например, родителем раздела 2.1 является глава 2). Пример иерархии узлов приведен на рис. 9.4.

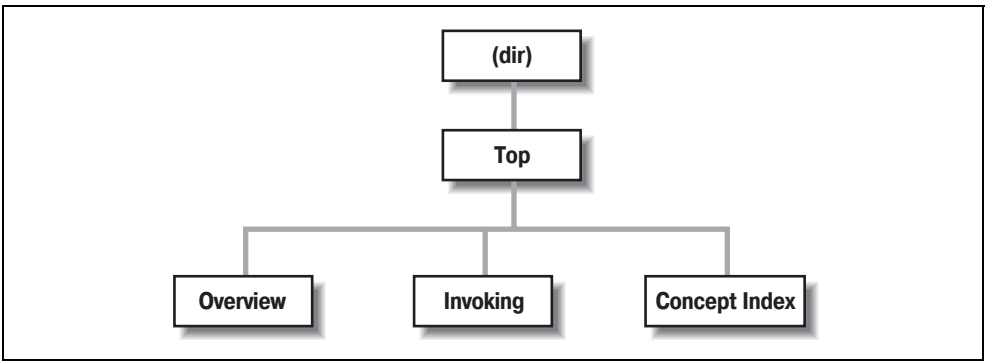


Рис. 9.4. *Иерархия узлов в Texinfo*

Вот исходный код узла `Top`:

```
@c    Node, Next, Previous, Up
@node Top ,     ,          , (dir)
@ifinfo
Данный Info-файл является хорошим приближением к вакууму.
В нем не описывается абсолютно ничего.
@end ifinfo
```

```
@menu
* Overview::              Overview of Vacuum
* Invoking::              How to use Vacuum
* Concept Index::         Index of concepts
@end menu
```

Команде @node предшествует комментарий, напоминающий о порядке аргументов, передаваемых @node. В данном случае у Top нет предыдущего и последующего узлов, поэтому эти аргументы оставлены пустыми. Родительским узлом для Top является (dir), который указывает на общесистемный каталог страниц Info. Предполагается, что ваш файл Info будет связан с системным деревом страниц Info, поэтому требуется, чтобы узел Top имел обратную ссылку на общий каталог.

За командой @node следует обзор всего документа, заключенный в пару команд @ifinfo...@end ifinfo. Они используются для того, чтобы фактический текст узла Top появлялся только в файле Info, но не в генерируемом TEX печатном документе.

Команды @menu...@end menu размечают меню узла. Каждый пункт меню состоит из имени узла, за которым следует его краткое описание. В данном случае меню указывает на узлы Overview, Invoking и Concept Index, исходный текст которых расположен дальше в файле. Эти три узла являются тремя «главами» нашего документа.

Итак, продолжим. Рассмотрим узел Overview, являющийся первой «главой»:

```
@c    Node,     Next,    Previous, Up
@node Overview, Invoking,        , Top
@chapter Overview of @code{vacuum}

@cindex Nothingness
@cindex Overview
@cindex Vacuum cleaners

A @code{vacuum} is a space entirely devoid of all matter. That means no
air, no empty beer cans, no dust, no nothing. Vacuums are usually found
in outer space. A vacuum cleaner is a device used to clean a vacuum.
See @xref{Invoking} for information on running @code{vacuum}.
```

Следующим узлом для Overview является Invoking — узел второй «главы», который должен находиться в меню после Overview. В своих документах Texinfo вы можете использовать почти любые структуры, однако часто удобно организовать их так, чтобы узлы напоминали главы, разделы, подразделы и т. д. Решение остается за вами.

Команда @chapter начинает главу и имеет действие только при форматировании исходного текста в TEX. Аналогично команды @section и @subsection начинают разделы и подразделы в результирующем документе TEX. Имена глав, а также разделов и подразделов могут быть более описательными, чем короткие имена, используемые для самих узлов.

Обратите внимание на использование команды `@code...` в названии главы. Это один из способов указать текст, который должен быть каким-то образом выделен. `@code` следует использовать для имен команд и исходного кода программ. Текст в команде `@code...` выводится моноширинным шрифтом в TEX и заключается в одинарные кавычки (например, `'this'`) в файле Info.

Затем следуют три команды `@cindex`, создающие элементы указателя в конце документа. После этого появляется фактический текст узла. И снова `@code` выделяет название «команды» `vacuum`.

Команда `@xref` порождает перекрестную ссылку на другой узел, к которому читатель может перейти командой `f` в программе чтения Info. `@xref` может производить перекрестные ссылки и на другие документы Texinfo. Полное изложение см. в документации по Texinfo.

Следующим узлом является `Invoking`:

```
@node Invoking, Concept Index, Overview, Top
@chapter Running @code{vacuum}

@cindex Running @code{vacuum}
@code{vacuum} is executed as follows:

@example
vacuum @var{options} @dots{}
@end example
```

Здесь команды `@example...@end example` выделяют пример. Внутри примера `@var` обозначает метапеременную, замещающую строку, вводимую пользователем (в данном случае это параметры, передаваемые пользователем команде *vacuum*). Команда `@dots{}` порождает многоточие. Пример будет выглядеть в отформатированном при помощи TEX документе так:

```
vacuum options …
```

а в файле Info так:

```
vacuum OPTIONS ...
```

Такие команды, как `@code` и `@var`, обеспечивают выделение, которое может осуществляться по-разному в TEX и Info.

Далее в узле `Invoking` мы видим:

```
@cindex Options
@cindex Arguments
The following options are supported:

@cindex Getting help
@table @samp
@item -help
Print a summary of options.

@item -version
Print the version number for @code{vacuum}.

@cindex Empty vacuums
```

```
@item -empty
Produce a particularly empty vacuum. This is the default.
@end table
```

Здесь описана таблица параметров, которые предположительно поддерживает *vacuum*. Команда `@table @samp` открывает таблицу из двух колонок (которая в итоге будет выглядеть, скорее, как маркированный список), каждый элемент которой будет выделен с использованием команды `@samp`. `@samp` аналогична `@code` и `@var`, за исключением того, что предназначена для обозначения ввода литералов, таких как параметры командной строки.

Обычный документ Texinfo содержит узлы для примеров, сведений по предоставлению отчетов по ошибкам и многого другого, но для краткости мы закончим этот пример завершающим узлом — `Concept Index`. Это указатель понятий, встречающихся в документе, автоматически создающийся командой `@printindex`:

```
@node Concept Index, , Invoking, Top
@unnumbered Concept Index

@printindex cp
```

Команда `@printindex cp` сообщает форматтеру о необходимости вставить в это место указатель понятий. Есть и иные типы указателей, такие как указатель функций, указатель команд и др. Все они создаются с помощью разновидностей команд `@cindex` и `@printindex`.

Последние три строки нашего исходного текста Texinfo следующие:

```
@shortcontents
@contents
@bye
```

Здесь форматтеру передаются команды для создания краткого оглавления (`@shortcontents`), полной таблицы содержания (`@contents`) и окончания форматирования (`@bye`). `@shortcontents` создает краткое оглавление, в котором перечислены только главы и приложения. Фактически только в случае длинных руководств в дополнение к `@contents` требуется `@shortcontents`.

Форматирование Texinfo

Для того чтобы создать Info-файл из исходного текста Texinfo, используется команда *makeinfo*. (Эта команда вместе с другими программами для обработки Texinfo включается в дистрибутив программного обеспечения Texinfo, который иногда идет в связке с Emacs.) Команда

```
eggplant$ makeinfo vacuum.texi
```

создает *vacuum.info* из *vacuum.texi*. *makeinfo* использует имя выходного файла, указанное в команде `@setfilename` в исходном коде. Его можно переопределить параметром *-o*.

Если результирующий файл оказывается большим, *makeinfo* разбивает его на ряд файлов с именами *vacuum.info-1*, *vacuum.info-2* и т. д., причем *vacu-*

um.info становится файлом верхнего уровня, указывающим на подчиненные файлы. Если все файлы *vacuum.info* находятся в одном каталоге, программа чтения Info сможет их найти.

Для создания Info из исходного текста Texinfo можно использовать также команды Emacs `M-x makeinfo-region` и `M-x makeinfo-buffer`.

Теперь файл Info можно просматривать в Emacs, используя команду `C-h i`. В режиме Emacs Info необходимо будет использовать команду `g` и указать полный путь к файлу Info, например:

```
Goto node: (/home/loomer/mdw/info/vacuum.info)Top
```

Это связано с тем, что Emacs обычно ищет файлы Info только в собственном каталоге Info (которым может быть */usr/local/emacs/info*).

Альтернативой является использование *info* – независимой от Emacs программы чтения Info-файлов. Команда

```
eggplant$ info -f vacuum.info
```

запускает *info* на чтение вашего нового Info-файла.

Если вы хотите установить новую страницу Info для использования всеми пользователями системы, нужно добавить ссылку на нее в файл *dir* в каталоге Emacs *info*. В документации по Texinfo подробно описывается, как это сделать.

Для того чтобы создать из исходного текста печатный документ, нужно установить TEX в вашей системе. Программное обеспечение Texinfo поступает с файлом макросов для TEX – *texinfo.tex*, содержащим все макросы, необходимые для форматирования Texinfo с помощью TEX. При правильной установке *texinfo.tex* должен оказаться в каталоге TEX *inputs*, где TEX сможет его обнаружить. Если нет, то скопируйте *texinfo.tex* в каталог, в котором находятся файлы Texinfo.

Сначала обработайте файл Texinfo с помощью TEX:

```
eggplant$ tex vacuum.texi
```

В результате в вашем каталоге образуется множество файлов, некоторые из которых связаны с TEX, а другие используются при создании указателей. Команда *texindex* (входящая в состав пакета Texinfo) переформатирует указатель в формат, который может использоваться TEX. Поэтому следующей должна быть выполнена команда

```
eggplant$ texindex vacuum.??
```

Использование `??` влечет обработку командой *texindex* всех файлов в каталоге, имеющих двухбуквенное расширение: это файлы, которые создаются Texinfo для генерации указателя.

Наконец, необходимо переформатировать файл Texinfo с помощью TEX, в результате чего удаляются перекрестные ссылки и вставляется указатель:

```
eggplant$ tex vacuum.texi
```

После выполнения этой команды вы должны получить файл *vacuum.dvi*, который можно просматривать с помощью *xdvi* или преобразовать в пригодный для печати вид. Ранее в этой главе в разделе «TEX и LATEX» обсуждалось, как печатать файлы *.dvi*.

Как обычно, осталось много нерассказанного об этой системе. Texinfo имеет полный набор собственных страниц Info, которые должны читаться вашей программой просмотра Info. Вы теперь также в состоянии отформатировать исходный текст документации по Texinfo с помощью TEX. Исходный текст документации по Texinfo – файлы *.texi* – должны быть в дистрибутиве исходного кода Texinfo.

Графика

Компьютерная графика зачаровывает многих. С помощью компьютеров создаются фотореалистичные изображения сюрреалистических сцен или фрактально генерируемые изображения горных хребтов, озер и долин. Изображения поворачиваются, шлифуются, им придается подержанный вид либо с ними производятся другие манипуляции.

Linux не приходится стыдливо отходить в сторонку, когда дело касается графики. Здесь можно делать практически то же самое, что и в других вычислительных средах, а в некоторых областях, например при одновременной обработке нескольких графических файлов, она даже лидирует. Описываемая в следующей главе система X Window создает очень хорошую основу для растровой графики. Теперь есть также поддержка 3D-графики для аппаратуры, соответствующей стандарту OpenGL.

Однако в Linux работа с графикой несколько отличается от привычной по другим операционным системам: здесь тоже действует принятая в Unix модель небольших, взаимодействующих между собой программ. Эту философию может отчетливо продемонстрировать графический пакет *ImageMagick*, который мы здесь опишем. *ImageMagick* является набором средств обработки графических файлов, запускаемых из командной строки или сценариев оболочки. Представьте себе, что у вас есть две тысячи файлов одного формата, и вы хотите сократить число используемых в них цветов до 256, изменить наклон и преобразовать в другой формат. В Linux для этого требуется несколько строчек в сценарии оболочки. Теперь представьте себе такую работу в Windows. Вам нужно щелкнуть по меню File, затем по пункту Open, выбрать файл, выбрать операцию и задать ее параметры в диалоговом окне, щелкнуть по кнопке OK, выбрать в меню пункт сохранения, выбрать имя файла, а затем щелкнуть по кнопке OK. А теперь повторите это для оставшихся 1999 файлов. Вам знакомо слово RSI?[1]

Графику можно не только рисовать, но и программировать. Существуют такие средства, как трассировщик лучей (ray-tracer) POVRAY, представлен-

[1] Repetitive Strain Injury – заболевание, возникающее при интенсивной работе на клавиатуре. Более известный запястный синдром является одним из видов RSI.

ный в этой главе, позволяющие задать подлежащий генерации графический объект с помощью набора команд, часто с помощью полноценного языка программирования графики. Хотя это занятие может оказаться более трудным, чем рисование мышкой, но зато после овладения им вы получаете несравнимо большую гибкость.

Не всем хочется работать с графикой из командной строки. Привлекательность работы с компьютерной графикой отчасти состоит в получении немедленной реакции при работе с графическими программами. GIMP, о котором мы расскажем ниже, представляет собой восхитительный пакет для интерактивной работы с графикой, превосходящий по своим возможностям значительную часть коммерческого программного обеспечения, существующего на других платформах.

Для Linux есть много других программ, включая 3D-симуляторы и видеопроигрыватели. Если порыться в архивах Linux, то можно обнаружить массу добротного программного обеспечения. Большинство этих программ требует наличия X Window System, об установке и настройке которой будет рассказано в последующих главах.

ImageMagick

ImageMagick является пакетом программ преобразования и манипуляции графическими файлами из командной строки. Он также хорошо приспособлен для пакетных преобразований, т. е. одновременного преобразования большого числа файлов. Кроме того, в состав ImageMagick входит PerlMagick, позволяющий вызывать утилиты ImageMagick при программировании на Perl. Он входит в состав большинства дистрибутивов Linux. Если ваш дистрибутив его не содержит, можно загрузить его с *ftp://ftp.x.org/contrib/applications/ImageMagick*.

ImageMagick состоит из библиотеки и ряда утилит:

display

display служит графическим интерфейсом пользователя ко многим функциям ImageMagick. Эта утилита очень полезна для предварительного экспериментирования с возможностями функции, прежде чем применять некоторое действие к сотням изображений в пакетном режиме. *display* может загружать файлы в различных графических форматах и отображать их в X Window System. При запуске *display* можно передать в командной строке одно или несколько имен файлов либо загрузить файл с помощью имеющегося меню. Если вы запустили *display*, но не видите меню, щелкните по изображению левой кнопкой мыши. У *display* очень много ключей командной строки. Подробности см. на странице руководства.

import

import является небольшой программой, позволяющей захватывать одно окно или весь экран в файл, другими словами, она делает снимки экрана. Например, для того чтобы сделать снимок всего экрана и сохранить его в

файле *myscreen.tiff* в формате графических файлов XPM, нужно выполнить команду

```
tigger$ import -window root myscreen.xpm
```

В начале захвата экрана раздается первый звуковой сигнал, в конце – второй, после чего данные сохраняются в указанном вами файле.

Если вы хотите захватить содержимое только одного окна, то проще всего сделать это, запустив *import* без параметра *-window*:

```
tigger$ import mywindow.xpm
```

В этом случае курсор превращается в крест, который подсказывает, что нужно щелкнуть по какому-нибудь окну. Содержимое этого окна захватывается и отправляется в заданный файл.

Как и все программы пакета ImageMagick, *import* имеет много параметров командной строки; см. страницу руководства *image*(1).

montage

montage является очень полезной маленькой программой с функциями, которые редко встречаются в других пакетах. Она берет несколько изображений и складывает их вместе в одно большое изображение. Многочисленные параметры определяют то, как именно складываются вместе изображения.

Для того чтобы взять все изображения в формате JPEG из рабочего каталога и создать из них изображение *all.jpg*, можно выполнить команду

```
tigger$ montage *.jpg all.jpg
```

По умолчанию под каждым изображением будет помещена метка с именем файла. Метки не ставятся, если выполнить команду

```
tigger$ montage +frame *.jpg all.jpg
```

convert

В некотором смысле *convert* является сердцевиной пакета ImageMagick. Она может преобразовывать огромное количество графических форматов. Допустим, вы хотите перенести GUI-программу из Windows в Linux. У вас есть большое число иконок для панели инструментов в формате Windows BMP, которые вы хотите преобразовать в XPM. Это можно сделать в сценарии:

```
for i in *.bmp
do
convert $i xpm:`basename $i .bmp`.xpm
done
```

convert переберет все изображения и преобразует их. Если вы хотите осуществить другие операции с изображениями, добавьте необходимые ключи, например *-despeckle* для удаления пятнышек с изображений.

mogrify

mogrify аналогична *convert*, но переписывает исходные изображения и предназначена, скорее, для применения фильтров, чем преобразования форматов файлов (хотя с помощью ключа *-format* можно легко изменить формат).

identify

identify выводит информацию об изображениях, переданных через командную строку. Например:

```
tigger$ identify tux.gif
tux.gif 257x303+0+0 DirectClass 10968b GIF 1s
```

Команда сообщает, помимо всего прочего, что *tux.gif* является файлом GIF шириной 257 пикселов и высотой 303 пиксела. Если этой информации вам не достаточно, используйте ключ *-verbose* (многословный)!

combine

Как указывает имя, команда *combine* собирает в одно несколько изображений. Можно, например, наложить изображения, полученные при разделении цветов, или поместить логотип поверх большего изображения.

xtp

Подобно *ftp*, программа *xtp* загружает файлы с удаленного сайта или выгружает их на него. Однако в отличие от *ftp*, она не требует интерактивных действий (чтобы добиться этого от *ftp*, вам нужно отредактировать конфигурационный файл). *xtp* можно успешно сочетать с другими утилитами ImageMagick, чтобы автоматически управлять изображениями на удаленных серверах.

Подводя итог, скажем, что программы пакета ImageMagick служат очень многогранным средством обработки графических файлов. Однако требуется некоторое время, чтобы ознакомиться со всеми параметрами командной строки и узнать, какие возможности доступны.

GIMP

GIMP (акроним от GNU Image Manipulation Program, или General Image Manipulation Program) специализируется на обработке изображений, как и описанный выше пакет ImageMagick. Но в то время как сила ImageMagick заключена в пакетной обработке, GIMP осуществляет все через свой графический интерфейс. GIMP часто называют одним из наиболее впечатляющих и удачных пакетов для Linux (и других версий Unix). Те, кто профессионально работает с графикой, говорят, что хотя GIMP может показаться не вполне удобным в работе, по функциональным возможностям он не уступает своему коммерческому конкуренту для Windows и Macintosh – пакету Adobe Photoshop. Некоторые называют его «бесплатной альтернативой Photoshop».

GIMP черпает свою мощь из бесчисленных подключаемых модулей, доступных для него. Поэтому для того чтобы использовать новый фильтр для обра-

ботки изображений или импорта графического формата, достаточно установить подключаемый модуль и перезапустить GIMP.

Кроме того, в GIMP используется разумный алгоритм, позволяющий загружать файлы произвольного размера: GIMP хранит в памяти только ту часть, которая в данный момент видима или обрабатывается. А если этих функций не достаточно, то в GIMP есть собственный язык сценариев – *script-fu*, но можно писать для него сценарии и на Perl.

GIMP поставляется вместе с большинством дистрибутивов Linux и может быть запущен просто вводом команды *gimp* в оболочке после запуска X. Если в вашем дистрибутиве нет GIMP, можно загрузить его исходный код с *www.gimp.org*. Если вы собираетесь серьезно работать в GIMP, вам следует заглянуть на этот сайт в любом случае, поскольку на нем есть также документация и примеры графических изображений, демонстрирующие возможности GIMP. На рис. 9.5 приведен снимок экрана, показывающий панель инструментов, диалоговые окна и рабочую область программы.

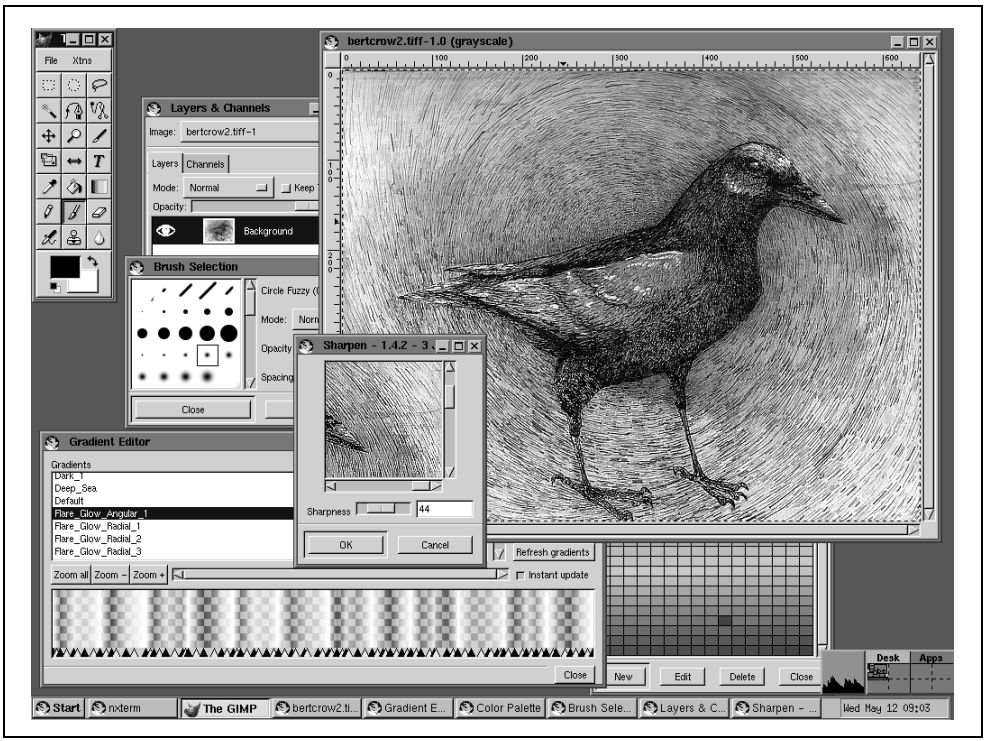


Рис. 9.5. *Gimp*

POVRAY

В то время как ImageMagick и GIMP служат в основном для обработки существующих изображений, POVRAY (первые три буквы происходят от «Persis-

tence Of Vision» – инертность зрительного восприятия) является программой для автоматического создания новых изображений. Это так называемый трассировщик лучей, программа, рассчитывающая сцену на основании некоторых данных о том, какие объекты должны присутствовать, каковы их физические и оптические свойства, откуда падает свет и где находится зритель. Все это описывается для POVRAY на его собственном языке программирования.

Уже это краткое описание должно показать, что пакетом POVRAY за один день не овладеешь. Если эта программа вас заинтересовала, можете загрузить ее с *http://www.povray.org* – лучшего источника для различных ресурсов POVRAY, включая файлы примеров, документацию и прочее. Поскольку невозможно дать здесь даже минимальный курс обучения POVRAY, мы воздержимся от этой тщетной задачи.

Если POVRAY вас заинтересовал, но отпугивает необходимость описывать свои сцены с помощью языка программирования, следует иметь в виду, что с трассировщиками лучей такого типа могут использоваться некоторые программы графического моделирования, поэтому необязательно создавать 3D-модели исключительно вручную. Одной из таких программ является Blender, о которой можно узнать подробнее на *http://www.blender3d.com*.

Настройка и работа со звуком в Linux

В этом разделе рассказывается о конфигурировании звуковых карт и других темах, касающихся поддержки звука в Linux.

Исторически звук был одним из самых сложных аспектов Linux, не получавшим должного внимания в ее дистрибутивах, возможно, потому, что первоначально Linux рассматривалась большинством как операционная система для серверов. На рабочих станциях пользователи стали воспринимать поддержку мультимедиа как нечто само собой разумеющееся. При наличии весьма скромных знаний заставить звуковую карту работать не так сложно, и на самом деле Linux по ряду причин хорошо подходит для звуковых и других мультимедийных приложений.

В начале этого раздела мы приведем беглый обзор понятий и терминологии цифрового аудио. Те, кому эта технология знакома, могут данный раздел пропустить. Если вам не очень интересно, как все это работает, или материал покажется непонятным, можете не отчаиваться: звук на вашей машине будет, даже если вы не знаете разницы между файлами MP3 и WAV.

Затем мы особо рассмотрим, как организована поддержка звука в Linux, какая аппаратура поддерживается, какие есть драйверы устройств и какие подходы к конфигурированию поддержки звука используются в разных дистрибутивах Linux.

Затем мы рассмотрим процедуру конфигурирования поддержки звука, компиляции или нахождения необходимых драйверов ядра, а также тестирования и отладки звуковых устройств. Мы дадим некоторые советы по устранению неполадок и укажем на часто встречающиеся ошибки.

Когда есть поддержка звука, желательно запустить какие-нибудь мультимедийные приложения. Мы кратко рассмотрим типы программ, имеющихся для работы со звуком под Linux.

Наконец, мы завершим этот раздел, приведя некоторые ссылки на дополнительную информацию о поддержке звука в Linux, которая поможет вам более глубоко изучить эту тему.

Небольшой совет: между дистрибутивами Linux есть незначительные различия. Кроме того, ядро Linux и приложения претерпевают постоянные изменения и усовершенствования. Мы постарались сделать информацию этой главы применимой ко всем системам Linux и указать области, в которых возможны различия, однако за деталями следует обращаться к документации по дистрибутиву и коллегам-пользователям.

Краткий обзор цифровой обработки звука

В данном разделе мы очень кратко рассмотрим некоторые понятия, относящиеся к цифровому звуку и звуковым картам.

Звук возникает при распространении волн переменного давления через среду, которой обычно является воздух. Это явление имеет *аналоговую* природу, что означает непрерывное изменение давления воздуха в некотором диапазоне значений.

Современные компьютеры являются цифровыми, т. е. они действуют с дискретными величинами, по существу, с двоичными единицами и нулями, обрабатываемыми центральным процессором компьютера. Для того чтобы компьютер мог обрабатывать звук, он должен преобразовывать аналоговую информацию звука в цифровой вид.

Аппаратное устройство, которое называется аналогово-цифровым преобразователем, преобразует аналоговые сигналы, такие как непрерывно изменяющиеся электрические сигналы микрофона, в цифровой формат, чтобы обрабатывать его компьютером. Аналогично, цифро-аналоговый преобразователь преобразует цифровые значения в аналоговый вид, в котором они могут быть отправлены в аналоговое выходное устройство, например громкоговоритель. Звуковые карты обычно содержат несколько аналогово-цифровых и цифро-аналоговых преобразователей.

Процедура преобразования аналоговых сигналов в цифровую форму заключается в дискретизации, или взятии замеров через равные промежутки времени и сохранении этих значений в виде чисел. Процедура аналогово-цифрового преобразования не является совершенной, что приводит к некоторым потерям или искажениям информации. Два важных фактора, влияющих на точность представления аналоговых сигналов в цифровой форме, – это точность и частота квантования.

Точность представления – это диапазон чисел, представляющих значения отсчетов сигнала, обычно выражаемый количеством двоичных разрядов. Например, 8-разрядные отсчеты преобразуют величины аналогового звукового сигнала в одно из 2^8, или 256, дискретных значений. 16-разрядное

квантование представляет звуковой сигнал с помощью 2^{16}, или 65 535, различных значений. Чем больше разрядов используется, тем точнее представляется сигнал и меньше ошибка квантования, возникающая при представлении сигнала дискретными значениями. Обратной стороной увеличения разрядности являются повышенная потребность в памяти и сложность аппаратуры, ведущая к ее удорожанию.

Частота дискретизации определяет интервал времени между последовательными измерениями аналогового сигнала. Правильно выражать ее количеством отсчетов в секунду, хотя иногда неформально и менее строго выражают ее в герцах. Чем ниже частота дискретизации, тем больше теряется информации об исходном аналоговом сигнале, и, наоборот, при увеличении частоты дискретизации сигнал представляется точнее. Теорема о частоте дискретизации утверждает, что для точного представления аналогового сигнала его отсчеты должны сниматься с частотой, как минимум вдвое большей, чем самая высокая частота, присутствующая в исходном сигнале.

Человеческое ухо в идеальном случае различает звуки в диапазоне примерно от 20 до 20 000 Гц, поэтому для точного представления звука, слышимого человеком, нужна частота выборки, вдвое большая 20 000 Гц. В технологии записи CD используется частота 44 100 отсчетов в секунду, что согласуется с этим простым расчетом. Спектр человеческой речи в основном укладывается в 4000 Гц. В системах цифровой телефонии обычно используется частота 8000 отсчетов в секунду, что отлично подходит для передачи речи. Увеличение частоты дискретизации ведет к росту потребности в памяти и усложнению аппаратуры.

При записи звука в цифровом виде возникают также проблемы количества каналов и формата кодирования отсчетов. Для стереозвучания нужны два канала. В некоторых аудиосистемах применяются четыре или более каналов.

Сами отсчеты сигнала могут кодироваться в различных форматах. Мы уже упоминали самые распространенные 8- и 16-разрядные форматы. Для заданной разрядности отсчеты могут кодироваться с помощью чисел со знаком или без знака, а когда отсчет занимает больше одного байта, необходимо указать принятый порядок следования байтов. Эти проблемы важны для передачи цифрового звука между программами или компьютерами, чтобы гарантировать использование ими одинакового формата. Форматы файлов, например WAV, стандартизируют представление звуковой информации, позволяя перемещать ее между различными компьютерами и операционными системами.

Часто нужно соединять звуки вместе или изменять их громкость. Эта процедура *микширования* может осуществляться как в аналоговом виде (например, регулятором громкости), так и в цифровом виде компьютером. Теоретически микшировать вместе два оцифрованных сигнала можно, просто складывая отсчеты, а изменять громкость можно, умножая значения отсчетов на некоторую константу.

До сих пор мы говорили о том, как записывать звук в виде цифровых отсчетов. Часто используются и другие технологии. *ЧМ-синтез* является старой

технологией создания звука с помощью аппаратуры, которая действует с сигналами различной формы, например с волнами синусоидальной или треугольной формы. Соответствующая аппаратура весьма проста и часто использовалась в звуковых картах первого поколения для синтеза музыки. Многие звуковые карты до сих пор поддерживают ЧМ-синтез для достижения обратной совместимости. В некоторых более новых картах используется технология синтеза на основе волновых таблиц, в которой ЧМ-синтез усовершенствован благодаря генерации звука с помощью цифровых образцов, хранящихся в самой звуковой карте.

MIDI служит сокращением от Musical Instrument Digital Interface (цифровой интерфейс музыкальных инструментов). Это стандартный протокол связи между электронными музыкальными инструментами. В число типичных устройств MIDI входят музыкальные клавиатуры, синтезаторы и ударные установки. MIDI оперирует не с хранящимися образцами звука, а с такими событиями, как нажатия клавиш на музыкальной клавиатуре. События MIDI можно записывать в MIDI-файл, благодаря чему музыкальное произведение представляется в очень компактном виде. MIDI наиболее популярен среди профессиональных музыкантов, хотя и многие бытовые звуковые карты поддерживают интерфейс шины MIDI.

Мы уже говорили о звуковых компакт-дисках, в которых используется 16-разрядное квантование уровня и частота дискретизации 44 100 отсчетов в секунду при двух каналах (стерео). Для одного часа звучания CD нужно хранить более чем 600 Мбайт данных. Для того чтобы уменьшить объем данных для хранения звука, были разработаны различные схемы компрессии. Один из подходов заключается в том, чтобы просто сжимать данные с помощью тех же алгоритмов, которые используются для компьютерных данных. Однако если учесть особенности человеческого слуха, можно добиться более эффективного сжатия звуковых данных путем удаления тех составляющих звука, которые ухо не слышит. Это называется сжатием с потерями, поскольку при такой процедуре происходит потеря информации, однако при правильной реализации объем данных сокращается очень значительно, а качество звука страдает незаметно. Такой подход использует алгоритм сжатия звука MPEG-1 уровня 3 (MP3), который может обеспечивать коэффициент сжатия 10:1 относительно первоначального цифрового звука. Другим алгоритмом с потерями, позволяющим добиться аналогичных результатов, является Ogg Vorbis. Он популярен среди многих пользователей Linux, поскольку позволяет избежать проблем с патентными правами, связанных с кодированием MP3. Существуют алгоритмы компрессии, оптимизированные для человеческой речи, например кодировка GSM, используемая в некоторых цифровых телефонных системах. Алгоритмы, с помощью которых осуществляется кодирование и декодирование звука, иногда называются *кодеками*.

Для приложений, в которых нужно передавать звук через Интернет, иногда широковещательно для множества пользователей, звуковые файлы не пригодны. Системы, способные передавать аудио- или иную мультимедийную информацию и воспроизводить ее в реальном времени, обозначают термином *потоковое мультимедиа* (*streaming media*).

Теперь, когда мы обсудили концепции, лежащие в основе цифрового звука, посмотрим на используемую для обработки звука аппаратуру. Звуковые карты развивались аналогично другим картам периферийных устройств для PC. Карты первого поколения были рассчитаны на шину ISA и в большинстве своем обеспечивали совместимость с серией SoundBlaster производства Creative Labs. Появившийся стандарт ISA Plug and Play (PnP) был принят многими звуковыми картами, что упростило настройку, устранив необходимость в аппаратных перемычках. В современных звуковых картах обычно используется шина PCI, будь это отдельные карты или поддержка звука, интегрированная на материнской плате, но доступ к которой все равно осуществляется через шину PCI. Сейчас появился ряд звуковых устройств USB, наиболее популярными из которых являются акустические системы, управляемые через шину USB.

Некоторые звуковые карты теперь поддерживают развитые функции, такие как круговое стереозвучание через шесть каналов и цифровой ввод/вывод для подключения к системам домашнего кинотеатра. Обсуждение таких устройств выходит за рамки нашей книги, и соответствующих звуковых карт мы касаться не будем. Много полезной информации о 3D-звуке можно найти на *http://www.3dsoundsurge.com*. Информацию о библиотеке для 3D-звука OpenAL 3D можно найти на *http://www.openal.org/home*.

Звук в Linux

Разобравшись с понятиями и терминологией цифрового звука в целом, мы можем взглянуть на особенности поддержки звука в Linux.

Программным компонентом самого низкого уровня, который взаимодействует непосредственно со звуковой аппаратурой, является ядро. На начальной стадии разработки Linux (т. е. до выхода версии ядра 1.0) Ханну Саволайнен (Hannu Savolainen) реализовал драйверы поддержки звука на уровне ядра для ряда популярных звуковых карт. Другие разработчики также внесли свой вклад, добавив в этот код поддержку новых функций и дополнительных карт. Эти драйверы, входящие в стандартную версию ядра, иногда называют OSS/Free, бесплатной версией Open Sound System.

Позднее Ханну стал работать в 4Front Technologies, компании, занимающейся разработкой коммерческих драйверов поддержки звука для Linux и ряда других Unix-совместимых операционных систем. Эти улучшенные драйверы распространяются на коммерческой основе под названием OSS/4Front.

В 1998 был организован проект Advanced Linux Sound Architecture, или ALSA, задачей которого стало создание новых звуковых драйверов для Linux и решение проблемы с отсутствием активной поддержки звуковых драйверов OSS. С учетом оценки прошлых ошибок и требований новой технологии звуковых карт возникла потребность в новых разработках.

Некоторые изготовители звуковых карт также создали драйверы для своих звуковых карт под Linux, в частности, это относится к серии Creative Labs Sound Blaster Live!.

В результате теперь предоставляется четыре разных комплекта звуковых драйверов для ядра. Это создает трудности при выборе необходимого звукового драйвера. В табл. 9.1 приведены преимущества и недостатки различных драйверов, что может помочь при принятии решения. Нужно также учесть, что в конкретном дистрибутиве Linux драйвер будет, скорее всего, один, и для выбора какого-либо другого потребуются дополнительные усилия.

Таблица 9.1. Сравнение драйверов звуковых карт

Драйвер	Преимущества	Недостатки
OSS/Free	Бесплатный Есть исходный код Входит в стандартное ядро Поддерживает большинство звуковых карт	Поддерживаются не все звуковые карты Большинство звуковых карт не определяется автоматически Не поддерживает некоторые новейшие карты Нет централизованного сопровождения
OSS/4Front	Поддерживает много звуковых карт Автоматически определяет большинство карт Есть коммерческая поддержка Совместим с OSS	Требуется оплата Исходный код закрыт
ALSA	Бесплатен Есть исходный код Поддерживает много звуковых карт Активно разрабатывается/поддерживается Ясная архитектура	Поддерживаются не все звуковые карты Не входит в стандартное ядро Не полностью совместим с OSS
Коммерческие	Могут поддерживать карты, для которых нет других драйверов	Исходный код может быть закрыт Поддержка может отсутствовать

Помимо драйверов, перечисленных в табл. 9.1, есть патчи ядра, решающие проблемы с некоторыми звуковыми картами.

Большинство звуковых карт поддерживается в Linux тем или иным драйвером. Отсутствие поддержки наиболее вероятно для очень новых карт, для которых еще не разработаны драйверы, и для некоторых высококачественных профессиональных карт, редко используемых обычными потребителями. Достаточно актуальный список поддерживаемых карт можно найти в

свежем документе Linux Sound HOWTO, но часто лучше всего заняться поиском в Интернете и поэкспериментировать с драйверами, которые могут подойти к вашей аппаратуре.

Многие приложения для работы со звуком непосредственно обращаются к звуковым драйверам ядра, но это служит источником проблемы: к звуковым устройствам ядра может одновременно обращаться только одно приложение. В графической настольной среде пользователь может пожелать одновременно воспроизводить файл MP3, сопровождать звуками операции оконного менеджера, получать звуковой сигнал при поступлении новой почты и т. д. Для этого требуется совместный доступ к звуковым устройствам со стороны различных приложений. Чтобы решить эту проблему, в современные графические среды Linux включается сервер звука, который берет на себя монопольное управление звуковыми устройствами и принимает запросы от приложений на воспроизведение звуков, микшируя их вместе. Возможно также перенаправление звука на другой компьютер подобно тому, как X Window System позволяет отображать результаты работы программы на компьютере, отличном от того, на котором она выполняется. Настольная среда KDE использует сервер звука *artsd*, а в GNOME такую задачу решает *esd*. Поскольку серверы звука являются относительным нововведением, не все приложения, работающие со звуком, пока их поддерживают.

В этом разделе мы не будем освещать вопросы разработки программного обеспечения, но для тех, кто намеревается разрабатывать мультимедийные приложения, есть ряд инструментальных средств, предоставляющих более простую поддержку звука, чем API ядра низкого уровня. Библиотека поддержки звука есть в ALSA; существует также много других инструментальных средств для звука, например SDL (предназначенный в основном для игр) и OpenAL (для 3D-звука). Если вы пишете мультимедийные программы, следует изучить эти библиотеки, чтобы не изобретать заново то, что уже сделано другими.

Установка и конфигурирование

В этом разделе мы расскажем о том, как установить и сконфигурировать звуковую карту под Linux.

Трудоемкость этой процедуры различна в зависимости от конкретного дистрибутива Linux. По мере роста зрелости Linux некоторые дистрибутивы стали предоставлять возможность автоматического определения и настройки звуковых карт. Времена ручной установки перемычек на карте и устранения аппаратных конфликтов уходят в прошлое в связи со стандартным использованием шины PCI. Если вам повезло и в вашем дистрибутиве Linux звуковая карта правильно определилась и работает, материал этого раздела не имеет особой важности для вас, поскольку все необходимое выполнено автоматически.

Некоторые дистрибутивы Linux предоставляют также утилиту для настройки звука, такую как *sndconfig*, которая пытается обнаружить и настроить звуковую карту, что обычно требует некоторого вмешательства пользователя. Необходимо свериться с системной документацией и, запустив утилиту

конфигурирования звуковых устройств, если таковая есть, проверить, работает ли она.

Если у вас более старая карта ISA или ISA PnP либо ваша карта неправильно определилась, потребуется прибегнуть к ручной процедуре, которую мы здесь описываем. Последующие инструкции исходят из предположения, что вы используете драйверы OSS/Free. При работе с ALSA процедура аналогична, но если вы пользуетесь коммерческими драйверами (OSS/4Front или драйвером, предоставленным изготовителем устройства), следует обратиться к документации, поставляемой с драйвером, поскольку процесс может иметь существенные отличия.

В приведенной здесь информации предполагается также, что Linux установлена на машине с архитектурой x86. Поддержка звука существует и на других архитектурах, но не все драйверы поддерживаются, и могут существовать некоторые отличия в именах устройств и т. д.

Сбор информации об установленной аппаратуре

Предполагается, что звуковая карта уже установлена на машине. Если это не так, нужно сначала ее установить. Если вы проверили работу звуковой карты с другой операционной системой, установленной на вашей машине, это будет свидетельствовать о том, что проблемы, если они возникнут в Linux, обусловлены программным обеспечением на том или ином уровне.

Нужно определить тип установленной у вас карты, включая изготовителя и модель. Определите тип шины – ISA, ISA PnP или PCI. Если на карте есть перемычки, следует записать их положение. Если вам известно, какие ресурсы использует карта в данный момент (IRQ, адрес I/O, канал DMA), зафиксируйте и эту информацию.

Если всю эту информацию получить не удалось, не отчаивайтесь. Можно обойтись и без нее, но впоследствии потребуется провести некоторое расследование. Например, на лэптопах или системах с интегрированной на материнской плате звуковой аппаратурой вам не удастся взглянуть отдельно на звуковую карту.

Конфигурирование ISA Plug and Play (необязательно)

Современные звуковые карты с шиной PCI не требуют конфигурирования. Старые звуковые карты с шиной ISA конфигурировались с помощью перемычек. Карты ISA PnP конфигурируются в Linux с помощью утилит ISA Plug and Play. Если вы не уверены, относится ли ваша звуковая карта к типу ISA PnP, попробуйте выполнить команду *pnpdump* и посмотреть, нет ли в ее выводе чего-либо, похожего на звуковую карту. Для типичной звуковой карты в выводе должны быть строки типа:

```
# Card 1: (serial identifier ba 10 03 be 24 25 00 8c 0e)
# Vendor Id CTL0025, Serial Number 379791851, checksum 0xBA.
# Version 1.0, Vendor version 1.0
# ANSI string -->Creative SB16 PnP<--
```

Общая процедура конфигурирования устройств ISA PnP следующая:

1. Сохранить файл */etc/isapnp.conf*, если он существует.
2. Сгенерировать файл конфигурации с помощью команды *pnpdump >/etc/isapnp.conf*.
3. Отредактировать файл, раскомментировав строки с нужными настройками устройства.
4. Выполнить команду *isapnp* для конфигурирования карт «Plug and Play» (обычно при запуске системы).

Большинство современных дистрибутивов Linux осуществляет инициализацию карт ISA PnP. У вас уже может быть подходящий файл */etc/isapnp.conf*, либо он потребует частичного редактирования.

Дополнительные подробности о конфигурировании карт ISA PnP можно найти на страницах руководства для *isapnp*, *pnpdump* и *isapnp.conf*, а также в ISA Plug and Play HOWTO из проекта Linux Documentation.

Конфигурирование ядра (необязательно)

Возможно, вам потребуется скомпилировать новое ядро. Если нужные вам модули звукового драйвера ядра отсутствуют в том ядре, с которым вы работаете в настоящее время, вам придется выполнить эту операцию. Если вы предпочитаете включать драйверы непосредственно в ядро, а не пользоваться загружаемыми модулями, вам также потребуется новое ядро.

Чаще всего складывается ситуация, когда вы работаете с ядром, полученным во время установки системы, а все звуковые драйверы включаются в виде загружаемых модулей, и тогда в этом шаге нет необходимости.

О том, как скомпилировать новое ядро, см. раздел «Сборка нового ядра» главы 7.

Конфигурирование модулей ядра

В большинстве случаев звуковые драйверы ядра имеют вид загружаемых модулей, которые могут динамически загружаться и выгружаться ядром. Необходимо обеспечить загрузку правильных драйверов. Это осуществляется с помощью файла конфигурации, например */etc/conf.modules*. Типичная запись для звуковой карты может выглядеть в нем так:

```
alias sound sb
alias midi opl3
options opl3 io=0x388
options sb io=0x220 irq=5 dma=1 dma16=5 mpu_io=0x330
```

Необходимо указать звуковой драйвер, который должен использоваться, и правильные значения адреса ввода/вывода, IRQ и каналов DMA, записанные вами ранее. Последние настройки требуются только для карт ISA и ISA PnP, поскольку карты PCI могут определять их автоматически. В предыдущем примере, приведенном для 16-разрядной карты SoundBlaster, нам при-

шлось указать в первой строке драйвер sb и задать параметры драйвера в последней строке.

В некоторых системах настройки содержатся в файле */etc/modules.conf* и/или нескольких файлах в каталоге */etc/modutils*, поэтому за деталями настройки модулей следует обратиться к документации по дистрибутиву Linux. В системах Debian для этой цели можно воспользоваться утилитой `modconf`.

Обычно на практике сложность заключается лишь в том, чтобы определить, какой драйвер использовать. Результаты работы *pnpdump* для карт ISA PnP и *lspci* для карт PCI могут помочь установить, какого типа карта у вас установлена. Затем можно поискать его в документации, имеющейся в Sound HOWTO или в исходном коде ядра, который на Linux-системах обычно находится в каталоге */usr/src/linux/Documentation/sound*.

Пусть, например, для некоторого лэптопа в результатах работы *lspci* сообщается о следующем звуковом устройстве:

```
00:05.0 Multimedia audio controller: Cirrus Logic CS 4614/22/24 [CrystalClear
SoundFusion Audio Accelerator] (rev 01)
```

Для этой системы подходящим звуковым драйвером будет «cs46xx». Возможно, потребуется провести некоторые эксперименты, и для верности можно попробовать загрузить различные модули ядра и посмотреть, не обнаруживают ли они звуковую карту.

Тестирование установок

Первое, что нужно сделать для проверки установки, это убедиться в том, что модуль ядра загружен. Это можно сделать с помощью команды *lsmod*: она должна показать, что соответствующий модуль загружен среди других:

```
% /sbin/lsmod
Module                  Size  Used by
parport_pc             21256   1 (autoclean)
lp                      6080   0 (autoclean)
parport                24512   1 (autoclean) [parport_pc lp]
3c574_cs                8324   1
serial                 43520   0 (autoclean)
cs46xx                 54472   4
soundcore               3492   3 [cs46xx]
ac97_codec              9568   0 [cs46xx]
rtc                     5528   0 (autoclean)
```

Здесь представляют интерес драйверы *cs46xx*, *soundcore* и *ac97_codec*. Если драйвер обнаружил карту, ядро должно также записать в журнал сообщение, которое можно извлечь с помощью команды *dmesg*. Она может выводить много результатов, поэтому можно направить их команде постраничного вывода, такой как *less*:

```
PCI: Found IRQ 11 for device 00:05.0
PCI: Sharing IRQ 11 with 00:02.0
PCI: Sharing IRQ 11 with 01:00.0
```

```
Crystal 4280/46xx + AC97 Audio, version 1.28.32, 19:55:54 Dec 29 2001
cs46xx: Card found at 0xf4100000 and 0xf4000000, IRQ 11
cs46xx: Thinkpad 600X/A20/T20 (1014:0153) at 0xf4100000/0xf4000000, IRQ 11
ac97_codec: AC97 Audio codec, id: 0x4352:0x5914 (Cirrus Logic CS4297A rev B)
```

Для карт ISA информация находится в файле устройства */dev/sndstat*. Для карт PCI это не так. Типичный вывод может выглядеть так:

```
% cat /dev/sndstat
OSS/Free:3.8s2++-971130
Load type: Driver loaded as a module
Kernel: Linux curly 2.2.16 #4 Sat Aug 26 19:04:06 PDT 2000 i686
Config options: 0

Installed drivers:

Card config:

Audio devices:
0: Sound Blaster 16 (4.13) (DUPLEX)

Synth devices:
0: Yamaha OPL3

MIDI devices:
0: Sound Blaster 16

Timers:
0: System clock

Mixers:
0: Sound Blaster
```

Если вывод выглядит нормально, можно протестировать звуковую карту. Вначале можно запустить программу-микшер и убедиться в том, что устройство микшера обнаруживается, уровни сигналов можно изменять и при этом не возникают ошибки. Нужно узнать, какие программы микширования есть в системе. Часто встречаются такие программы, как *aumix*, *xmix* и *kmix*. Установите для всех уровней сигнала подходящие значения.

Теперь попробуйте воспроизвести какой-нибудь звуковой файл, например WAV, с помощью программы-плеера и убедитесь в том, что слышите его. В графической среде, такой как KDE или GNOME, подходящий плеер должен быть, в противном случае поищите утилиту командной строки типа *play*.

Если воспроизведение действует, можно проверить запись. Подключите микрофон ко входу mic звуковой карты и запустите программу записи, такую как *rec* или *vrec*. Проверьте, можно ли записать звук в файл WAV и воспроизвести его. Проконтролируйте в настройках микшера выбор правильного устройства ввода и установите приемлемые уровни сигнала.

Можно также проверить правильность воспроизведения файлов MIDI. Одним программам MIDI-плееров требуется звуковая карта с ЧМ-синтезатором, другим – нет. Распространены такие MIDI-плееры, как *playmidi*, *kmid* и *kmidi*. Тестирование устройств на шине MIDI выходит за рамки данной книги.

Хорошим информационным сайтом по MIDI и MIDI-устройствам служит *http://midistudio.com*. Официальные спецификации MIDI можно взять у MIDI Manufacturers Association на веб-сайте *http://www.midi.org*.

Устранение неполадок и часто возникающие проблемы

В данном разделе перечисляются некоторые часто возникающие проблемы и способы их решения.

Модули ядра не загружены

Это может быть вызвано ошибками в файлах конфигурации модулей или тем, что не запущен загрузчик модулей ядра (*kerneld* или *kmod*). Проверьте наличие загружаемого модуля в нужном каталоге (обычно вида */lib/ modules/2.4.17/kernel/drivers/sound*).

Звуковая карта не обнаруживается

Возможно, используется не тот модуль ядра, который нужен, или неверно заданы значения адреса I/O, IRQ или канала DMA.

Тайм-аут IRQ/DMA или конфликты устройств

Указаны неверные значения адреса I/O, IRQ и DMA, либо существует конфликт с другой картой, использующей те же настройки.

После перезагрузки звук отсутствует

Если звук был, но исчез после перезагрузки системы, это может указывать на проблему с файлами конфигурации модулей. Это может произойти и тогда, когда системные сценарии *init* не настроены на инициализацию карт PnP или загрузку модулей.

Если драйверы загружены, то, возможно, в микшере установлена слишком низкая громкость.

Звук воспроизводится только для root

Это может указывать на неправильную установку прав доступа к файлам устройств. Во многих системах доступ к звуковым устройствам разрешен только пользователям, входящим в группу «audio». Добавьте в эту группу пользователей или измените права доступа к звуковым устройствам с помощью *chmod*.

Звука не слышно, но сообщений об ошибках нет

Если программы звуковоспроизведения по виду работают, но звука не слышно, это может быть связано с настройками микшера или надежностью подключения звуковых колонок.

Невозможно записать звук

Это может указывать на проблему с настройками микшера. Необходимо установить уровни громкости и входное устройство. Возможно также, что у вас плохой микрофон или вы включили его в неверное гнездо на звуковой карте.

Ошибка: устройство занято

Либо у вас конфликт устройств, либо другое приложение использует звуковые устройства. Это может быть связано с работой программы-сервера звука, такой как *esd* или *artsd*.

Отсутствие звука при воспроизведении звукового компакт-диска

Для воспроизведения звуковых компакт-дисков необходимо соединить кабелем привод CD-ROM и звуковую карту. Проверьте, выбрали ли вы вход с CD в программе-микшере. Попробуйте включить наушники в гнездо на передней панели привода CD-ROM; если слышен звук, проблема не в самом приводе. Если в наушниках нет звука, значит, проблема связана с приводом или программой воспроизведения CD.

Невозможно воспроизведение MIDI-файлов

Некоторые приложения MIDI работают только с картами на основе ЧМ-синтезатора, которого может не быть на вашей карте (либо он не поддерживается драйвером звуковой карты). Другие приложения MIDI используют стандартное аудиоустройство.

Мультимедийные приложения Linux

Когда поддержка звука организована, желательно запустить какие-нибудь приложения для работы со звуком. Таких приложений очень много, и их трудно было бы все перечислить, поэтому мы просто опишем некоторые общие категории имеющихся программ. Вы можете сами поискать приложения по ссылкам, которые здесь будут приведены. Мы также более подробно рассмотрим один из наиболее популярных сегодня видов приложений – программы воспроизведения MP3-файлов.

- Программы-микшеры для установки коэффициентов усиления сигнала при записи и воспроизведении
- Проигрыватели файлов таких форматов, как WAV, MP3 и MIDI
- CD-проигрыватели для воспроизведения звуковых компакт-дисков
- Средства для записи звука в файлы
- Инструменты для создания акустических эффектов и цифровой обработки сигналов
- Программные средства распознавания и синтеза речи
- Игры, реалистичность которых усиливается звуковым сопровождением
- Графические настольные среды KDE и GNOME, поддерживающие мультимедиа

MP3-проигрыватели

MP3 (MPEG-1 Layer 3) – это один из самых распространенных форматов файлов для цифрового звука, и существует много приложений для воспроизведения файлов MP3 под Linux. В графической среде типа KDE или GNOME MP3-проигрыватель, скорее всего, должен уже иметься. Его и рекомендует-

ся использовать, поскольку он должен нормально работать со звуковым сервером, используемым этими средами.

Вот некоторые характеристики, на которых следует основывать выбор MP3-проигрывателя:

- Поддержка различных звуковых драйверов (например, OSS и ALSA) или звуковых серверов (KDE и GNOME).
- Привлекательный интерфейс пользователя. Для многих MP3-проигрывателей можно менять «скины», что означает возможность загрузки и установки альтернативных интерфейсов пользователя.
- Поддержка списков воспроизведения, позволяющая определять и хранить списки любимых аудиофрагментов.
- Различные звуковые эффекты, такие как графический эквалайзер, расширение стерео, реверберация, удаление голоса и визуальные эффекты для представления звука в графическом виде.
- Поддержка других форматов файлов, таких как CD, WAV и видео.

Xmms – один из популярных MP3-проигрывателей с интерфейсом пользователя по умолчанию, аналогичным Winamp. Его можно загрузить с *http://www.xmms.org*.

Если вы хотите создавать собственные файлы MP3, вам потребуется программа-кодировщик. Существуют также программы, позволяющие захватывать отдельные записи со звуковых компакт-дисков.

Хотя есть инструменты `open source`, позволяющие выполнять кодировку MP3, законность этих средств спорна ввиду определенных патентных ограничений. Ogg Vorbis – это альтернативный формат файла и кодировщик, который считается свободным от проблем патентования. Для работы с ним ваша программа-плеер должна поддерживать файлы Ogg Vorbis, поскольку они не совместимы непосредственно с MP3. Однако многие MP3-проигрыватели, в том числе Xmms, уже поддерживают Ogg Vorbis; в других случаях существуют прямые эквиваленты, например *ogg123* для *mpg123*.

Установка MP3-проигрывателя обычно требует, чтобы был установлен соответствующий пакет (в формате RPM или *deb*, в зависимости от вашего дистрибутива). Можно также скомпилировать его из исходного кода. Проигрыватель MP3 должен установить типы MIME для связи с MP3 и другими поддерживаемыми типами файлов, чтобы запускать его из приложений так же, как файловые менеджеры, веб-броузеры и почтовые клиенты.

Ссылки

Ниже перечислены некоторые информационные ресурсы по работе со звуком в Linux:

- Linux Sound HOWTO из проекта Linux Documentation Project на *http://www.tlpd.org*
- Веб-сайт проекта ALSA на *http://www.alsa-project.org*

- Веб-сайт 4Front Technologies на *http://www.opensound.com*
- Веб-сайт Sound and MIDI Software for Linux на *http://sound.condorow.net*
- Книга «Linux Multimedia Guide» издательства O'Reilly
- Книга «Linux Music and Sound» издательства No Starch Press

Работе со звуком под Linux посвящен ряд почтовых списков рассылки. См. в Sound HOWTO подробности относительно того, как подписаться на эти списки.

Печать

Документ печатается в Linux командой *lpr*. Необязательно всегда вызывать эту команду непосредственно; можно нажимать кнопку Print в каком-нибудь глянцевом графическом интерфейсе, поддерживающем drag-and-drop. Но в конечном итоге печать будет обрабатываться *lpr* и другими утилитами управления печатью, о которых мы сейчас расскажем.

Если вы хотите распечатать листинг программы, можно ввести команду

```
lpr myprogram.c
```

Другие команды также могут отправлять данные на *lpr* по каналу. *lpr* начинает процесс печати, временно записывая данные в каталог, называемый спулингом печати. Другие части системы управления печатью, о настройке которых мы рассказали в разделе «Управление службами печати» главы 8, в нужном порядке изымают файлы из очереди печати, готовят файлы для печати и управляют потоком данных, отправляемых на принтер.

Для каждого принтера в системе существует, по крайней мере, один спулинг печати.[1] По умолчанию *lpr* ищет принтер с именем `lp`. Однако если нужно задать принтер с другим именем, используйте параметр *-P*.

Например, чтобы отправить документ на принтер с именем `nene`, введите команду

```
lpr -Pnene myprogram.c
```

Если вы забыли имя принтера, посмотрите имена каталогов спулинга в каталоге */var/spool/lpd* или имена, присвоенные каждому принтеру в записях файла */etc/printcap*. Если вы пользуетесь системой *lprng* (как это делается во многих современных дистрибутивах), то можете увидеть все установленные принтеры с помощью команды *lpq -a*.

Обратите внимание, что пользователь не видит, подключен ли его принтер непосредственно к компьютеру или находится где-то в сети: все, что вам можно и нужно знать, это имя очереди принтера. Если очередь печати связана с принтером на другой машине, то печатаемый файл сначала помещается

[1] Принтер, который используется в различных режимах, например для печати факсов и писем, может иметь отдельные спулинги для каждой из задач.

в очередь на вашей машине, затем пересылается в соответствующую область спулинга на машине, к которой подключен принтер, и наконец, выводится на печать. В разделе «Управление службами печати» главы 8 подробнее рассказывается об организации очередей печати.

Если вы хотите использовать для большинства задач некоторый конкретный принтер, можно указать его в переменной окружения PRINTER. В предположении, что используется оболочка *bash*, можно сделать nene личным принтером по умолчанию, вставив в файл *.bashrc* команду

```
export PRINTER=nene
```

Параметр *-P* команды *lpr* переопределяет значение, указанное в переменной PRINTER.

Теперь вы знаете, как напечатать файл, но можете столкнуться с проблемой, когда ваш файл не печатается сразу, как вы того ожидаете. Можно выяснить статус файлов в очереди печати с помощью команды *lpq*. Чтобы выяснить статус файлов, отправленных на принтер, установленный по умолчанию (переменная PRINTER участвует в работе всех обсуждаемых в этом разделе команд), введите команду

```
$ lpq
nene is ready and printing
Rank   Owner      Job  Files                                 Total Size
active lovelace   020  (standard input)                    776708 bytes
1st    parcifal   024  (standard input)                   2297842 bytes
1st    lark       023  (standard input)                     10411 bytes
```

Вы видите, что принтер работает, но перед вами в очереди находятся большие задания (если вы пользователь *lark*). Если вам некогда ждать, можно удалить задание из очереди печати. Для удаления задания используйте его номер, сообщенный программой *lpq*:

```
$ lprm 23
023 dequeued
023 dequeued
```

В результате из спулинга печати выбрасывается файл с номером задания 023, а также связанный с ним файл, содержащий команды форматирования.

Можно сузить отчет *lpq*, запросив сведения только по конкретному номеру задачи (что редко используется), принтеру или идентификатору пользователя. Например, чтобы получить отчет по файлам спулинга, посланным на принтер ada, следует ввести:

```
$ lpq ada
ada is ready and printing
Rank   Owner      Job  Files                                 Total Size
active lovelace   788  standard input                        16713 bytes
1st    lark       796  standard input                        70750 bytes
```

Если вы являетесь пользователем root, то можете убить все задания в очереди командой

```
lprm -
```

Если вы не являетесь пользователем root, эта команда убьет только ваши собственные задания. Это ограничение действует и в том случае, если вы укажете принтер:

```
lprm ada
```

Если вы являетесь пользователем root, очередь принтера ada будет очищена. Если вы являетесь обычным пользователем, из указанного спулинга печати будут удалены только те файлы, владельцем которых вы являетесь. Утилита *lprm* сообщает о задачах, которые она убила.

Пользователь root может убить все задания печати, созданные каким-либо пользователем, введя команду

```
lprm имя_пользователя
```

Команда *lprm* без аргументов удаляет активные задания печати, владельцем которых вы являетесь. Она эквивалентна команде

```
lprm ваше_имя_пользователя
```

Если вы хотите узнать, работает ли принтер, выполните команду *lpc*

```
/usr/sbin/lpc status ada
```

Подробности см. в разделе «Управление службами печати с помощью lpc» главы 8. Утилита *lpc* обычно устанавливается в каталог */sbin* или */usr/sbin*.

Теперь приведем еще несколько примеров распространенных задач печати и используемых фильтров.

Чтобы быстро напечатать страницу руководства по printcap, введите:

```
man printcap | col -b | lpr
```

Команда *man* находит, форматирует и выводит страницу руководства по printcap в формате ASCII с усилением, выражающемся в двойной печати с подчеркиванием символов (вместо курсива) с целью выделения. Выходные данные передаются команде *col* – текстовому фильтру Unix. Параметр *-b* задает необходимость удаления команд «backspace», встроенных в страницу руководства, в результате чего получаются простые текстовые строки с сохранением расположения форматированной страницы руководства. Вывод команды *col* передается *lpr*, которая отправляет текст в каталог спулинга.

Предположим, вы хотите напечатать страницу руководства с усилением, включающим выделение и прочее. Тогда можно использовать такую команду:

```
groff -man -Tps /usr/man/man5/printcap.5 | lpr
```

Команда *groff* применяет макрос man к указанному файлу, создавая вывод в формате PostScript (что задается параметром *-Tps*). Вывод передается *lpr*, которая отправляет его в спулинг, а *lpd* применяет команды обработки печати, установленные по умолчанию в файле */etc/ printcap*.

Другой полезной утилитой для печати чисто текстовых файлов является команда *pr*, которая форматирует их несколькими способами.

В большинстве дистрибутивов Linux используются утилиты печати в стиле BSD, разработанные для дистрибутива Unix Berkeley Software. Если у вас есть утилиты с именами *lp, lpstat, cancel* и *lpadmin,* то на вашей машине Linux стоит система печати, основанная на System V. Вам нужно почитать страницы руководства и, возможно, другую документацию для этого пакета. Есть и другие системы печати, которые могут быть установлены в вашей системе, например пакет PLP, но здесь мы рассказываем только об обычных, основанных на BSD, утилитах управления печати для Linux. Мы также опишем, как использовать другие часто устанавливаемые в Linux утилиты поддержки печати, например фильтры, специальным образом готовящие документы для вывода на принтеры.

О названиях систем печати

Система управления печатью в стиле BSD по традиции называется «lp», что связано с устройствами построчной печати (line printers), которые были стандартным устройством вывода в эпоху разработки пакета.[a] В сообществе Linux пакет управления печатью BSD чаще называют «пакетом lpr». Конечно, команда *lpr* является лишь одним из имеющихся в пакете средств. Наконец, есть еще пакет CUPS (Common Unix Printing System), более новый и развитый, чем другие пакеты управления печатью, который, возможно, вытеснит предыдущие в течение ближайших пары лет. Дополнительные сведения о CUPS см. в разделе «Управление службами печати» главы 8.

[a] Не дайте ввести себя в заблуждение, услышав от кого-либо термин *lp*, что является путаницей имен. В пакете управления печатью BSD нет утилиты *lp*, а вот в разработанном позднее пакете печати System V она есть. Однако возможно, что ваш пакет BSD позволяет использовать для печати команду *lp*. Некоторые системы используют сценарий оболочки (фильтр) с именем *lp* для преобразования параметров команды *lp* в параметры *lpr*, которой и передают их. Это сделано исключительно для удобства пользователей, знакомых с командой *lp* в System V.

Что происходит с файлом после постановки в очередь

Обычно после того, как вы передали документ или файл, который нужно напечатать, в систему печати, можно забыть об этом и ждать, когда из принтера появится отпечатанная страница. Но если что-то не так, и страница не появляется либо вас просто мучает (как нас) любопытство, можно выяснить, что происходит за кулисами в период между вводом команды *lpr* и появлением страницы на лотке принтера. При желании данный раздел можно пропустить, вернувшись к нему позднее.

Только суперпользователь обладает возможностью непосредственного доступа к принтерам без использования системы печати. (Кстати, пользоваться этой возможностью неблагоразумно.) Linux не предоставляет пользовате-

лям системы возможности непосредственного обращения к различным устройствам, поскольку в результате этого могут возникать разрушительные конфликты. Вместо этого утилиты вызывают фоновые системные процессы, которые планируют вашу печать наряду с другими задачами, преобразуют исходные данные файлов для печати на конкретных принтерах с использованием соответствующих принтеру языков и протоколов, устанавливают разрешение печати и формат страницы, добавляют (или учитывают, что не нужно добавлять) верхние и нижние колонтитулы и номера страниц. Linux конфигурируется при начальной загрузке для работы со своими физическими устройствами, в том числе настраиваются порты и протоколы для ведения печати.

Система управления печати контролируется *lpd* – демоном построчной печати, у которого есть необходимые права для доступа к принтерам вместо пользователя. После того как система печати установлена, *lpd* запускается при каждой инициализации (начальной загрузке) Linux. Управляющие данные, необходимые *lpd* для управления печатью файлов, содержатся в файле */etc/printcap*.

Рассмотрим, чем фактически занимается система управления печатью, когда вы вводите команду печати. Когда вызывается *lpr*, она сначала проверяет, установлена ли переменная окружения `PRINTER`. Если да, то *lpr* загружает хранящиеся в ней параметры в качестве инструкций по обработке печатаемого файла. Затем *lpr* применяет параметры, переданные в командной строке или программой, которая производила предшествующую обработку файла. Наконец, *lpr* посылает файл в каталог спулинга используемого принтера вместе с другим временным файлом, в котором хранятся все спецификации обработки для применения демоном *lpd*. После этого *lpr* извещает *lpd*, что в каталог спулинга помещен файл для печати. Если *lpr* получает непонятный ей параметр, то передает его фильтру печати, о котором мы расскажем немного позже.

Когда *lpd* обнаруживает в спулинге задание, она читает спецификации обработки (в которых сказано, как пропустить печатаемый файл через фильтры и направить вывод на устройство) и завершает задание печати, удаляя печатаемый файл и файл обработки после того, как принтер получил весь обработанный файл печати.

Все системы печати Unix пропускают файл по крайней мере через один фильтр, чтобы подготовить его к выводу. Соответствующие фильтры для удовлетворения обычных потребностей печати устанавливаются администратором и указываются в файле printcap. Примером может послужить фильтр, преобразующий простой текст в PostScript, что необходимо, если ваш принтер понимает только PostScript. Задания печати попадают в систему управления печатью через команду *lpr* (непосредственно в командной строке или передаются косвенно какой-либо другой командой). Команда *lpr* осуществляет спулинг печатаемого файла в нужном каталоге спулинга и записывает относящиеся к нему инструкции обработки в ассоциированный файл в том же каталоге.

Когда приходит время печатать файл, демон *lpd* считывает специальные указания по обработке печатаемого файла, которые могут переопределять инструкции, установленные в файле */etc/printcap*. Передавая файл на тот фильтр, который нужен для используемого принтера, *lpd* передает ему и те инструкции, которые он не выполнил. После этого фильтр производит собственную обработку согласно полученным инструкциям или обращается к другим фильтрам, чтобы произвести обработку, которую сам он не смог выполнить. После этого *lpd* передает файл на принтер. Этот скрытый фильтр осуществляет самое действенное управление форматированием при печати.

Если у вас есть особые требования к печати, то нужно знать, какие параметры должны быть переданы фильтру, управляющему форматированием печати. Стандартные параметры описаны на странице руководства по *lpr*. Дополнительные параметры фильтра документируются в используемом пакете, содержащем фильтр. Ниже мы расскажем о паре фильтров, которые, вероятно, есть в вашей системе. Едва ли мы сможем описать, как использовать все фильтры и пакеты фильтров, имеющиеся для работы в Linux. Следует прочесть страницы руководства для имеющихся в вашей системе фильтров, чтобы узнать, как получить требуемый вам вывод.

Правильно сконфигурированный файл printcap использует имя принтера для определения того, какие фильтры должны быть применены. Например, если ваш принтер может автоматически форматировать и печатать документ HTML (World Wide Web), вы можете предложить *lpr* осуществить печать на принтер `html`:

```
lpr -Phtml ~/homepage.html
```

Физически это может быть тот же принтер, который вы используете, например, для печати документов PostScript. Но, указывая принтер с именем `html`, вы заставляете *lpd* использовать другое определение в printcap, в результате чего файл преобразуется фильтром HTML в графический формат, понятный принтеру.

Поскольку содержимое файла */etc/printcap* определяется системным администратором, вы не можете просто заменить один процесс фильтрации другим. Вы можете в значительной мере управлять средствами набора, параметрами фильтрации и выбором принтера через командную строку и переменные окружения. При необходимости можно фильтровать файлы перед тем, как передать их системе управления печатью. Например, получив сообщение электронной почты с вложенным графическим файлом, который вам не удается вывести на экран или напечатать, можно сохранить файл на диске и использовать имеющиеся в Linux многочисленные утилиты преобразования графических файлов для преобразования (фильтрации) его в печатаемый формат, прежде чем передать *lpr*.

Многое зависит от того, насколько умен фильтр. Если параметр передан *lpr*, а затем фильтру, но они не смогли его интерпретировать, то, в конце концов, он может попасть на принтер – перед печатаемым документом или после него. Возможно, это не окажет никакого эффекта. В некоторых случаях таким

способом намеренно передается команда настройки принтера. Чаще переданный на принтер параметр вызывает печать лишних страниц, возможно, со случайными символами. Обычно это безвредно. Если вы заметите такое поведение, то проверьте, не сами ли вы создали проблему установкой переменных окружения, неправильным форматом команд или неподдерживаемыми параметрами. Если вина не ваша, то администратор должен найти источник проблемы, чтобы сберечь бумагу и время.

Некоторые фильтры автоматически используют переменную окружения `PRINTER`, если она вами установлена. Следует знать, что в системе печати System V эквивалентной переменной окружения является `LPDEST`, и некоторые фильтры печати могут ожидать или принимать эту переменную. Может оказаться, что вы работаете в смешанной сети с учетными записями на разных системах, одни из которых используют управление печатью BSD, а другие – System V. Если вы из тех, кто носит ремень вместе с подтяжками, можете установить в файле инициализации оболочки как `LPDEST`, так и `PRINTER`.

Проблемы с использованием фильтра печати могут воздействовать и на других пользователей в многопользовательской системе. Обо всех трудностях сообщайте администратору печати.

nenscript и enscript

Утилита *nenscript*, которую часто называют теперь *enscript*, является гибким фильтром, обеспечивающим хорошо форматированный вывод для принтеров PostScript даже из текстовых файлов ASCII. Она не является базовой утилитой Linux, но входит в состав ряда дистрибутивов Linux и может быть загружена с обычных FTP-сайтов Linux. Хотя можно вызывать *nenscript* для отправки подготовленного файла на принтер, обычно она указывается в файле */etc/printcap* в качестве проходного фильтра, принимающего текст с устройства стандартного ввода и отправляющего его на стандартное устройство вывода.

Предположим, что при распечатке программы на C вы хотите печатать номера строк, а также выводить текст на бумагу с зелеными полосками (не того формата, который вы используете при печати на вашем модном принтере PostScript графики, загружаемой из Интернета). Вам нужно обработать текст программы и вставить номера в начале строк. Это можно сделать, обработав файл таким фильтром, как утилита *nenscript* (если вы ее установили). Произведя свою обработку, *nenscript* передаст файл *lpr* для спулинга и печати на вашем надежном принтере с непрерывной подачей бумаги (здесь он назван `dino`):

```
nenscript -B -L66 -N -Pdino myprogram.c
```

Фильтр *nenscript* нумерует строки проходящего через него файла при задании параметра *-N*. Параметр *-B* подавляет печать обычной заголовочной информации на каждой странице, а параметр *-L66* задает вывод 66 строк на странице. Параметр *-Pdino* просто передается фильтром *nenscript* утилите *lpr*, которая интерпретирует его и направляет вывод в каталог спулинга `dino` для печати.

При вызове из командной строки *nenscript* автоматически передаст вывод *lpr*, если не задать стандартное устройство вывода с помощью параметра *-p*. Нет необходимости явно переназначать или пересылать вывод по каналу на *lpr*.[1]

Допустим, вы собрались сегодня распечатать много программных листингов. Для удобства можно установить переменную окружения для *nenscript*, чтобы каждый раз особым образом обрабатывать и печатать ваши листинги:

```
export NENSCRIPT=" -B -L66 -N -Pdino"
```

Теперь для печати в нужном формате требуется лишь ввести:

```
nenscript myprogram.c
```

nenscript может посылать вывод в файл, что часто полезно для создания файлов PostScript на системах Linux, фактически не подключенных к принтеру PostScript. Например, для преобразования текстового файла в формат PostScript для печати в две колонки на листах формата A4 шрифтом Courier кеглем 6 пунктов нужно ввести команду

```
nenscript -2 -fCourier6 -TA4 -pdocument.ps document.txt
```

Параметр *-2* переопределяет печать по умолчанию в одну колонку, а параметр *-fCourier6* переопределяет Courier в 7 пунктов, установленный по умолчанию для вывода в две колонки. (Для печати в одну колонку по умолчанию установлен Courier10; *nenscript* всегда использует шрифт Courier при преобразовании простого текста в PostScript.) Аргумент *-TA4* задает европейский стандарт бумаги A4. Если компилировать *nenscript* с установленной переменной окружения `US_VERSION`, то формат бумаги по умолчанию будет 8,5×11 дюймов (если вы не используете переменную `US_VERSION`, то можете задать этот размер параметром *-TUS*). Параметр *-p* указывает, что выходные данные должны быть сохранены в файле *document.ps*, а имя файла без параметра указывает *nenscript* на входной файл. Если входной файл не указан, *nenscript* использует в качестве входного файла стандартное устройство ввода.

Другой пример. Для печати страницы руководства по *nenscript* как базового текста на принтере PostScript введите:

```
man nenscript | col -b | nenscript
```

Команда *man* извлекает страницу руководства и форматирует ее для вывода текста. Команда *col -b* удаляет инструкции backspace, используемые для выделения и подчеркивания, оставляя простой текст, передаваемый по каналу на фильтр *nenscript*. Это преобразует обычный текст в простой PostScript с

---

[1] Утилита *nenscript* может быть фильтром по умолчанию, указанным в файле printcap для вашего принтера. Не будет вреда, если файл еще раз пройдет через фильтр, если тот правильно сконструирован. Фильтр передает дальше инструкцию обработки, только если не может ее осуществить сам. Выполнив инструкцию, фильтр отбрасывает параметр обработки. Не стоит беспокоиться, что в результате номера строк будут напечатаны дважды.

«художественной печатью», которая включает верхние и нижние колонтитулы, нумерацию страниц и т. п. Наконец, файл передается *lpr*, которая посылает его в спулинг. Файл еще раз проходит через фильтр, заданный в файле printcap, который может быть «фиктивным» и просто передавать текст дальше, но может что-то еще осуществлять, например посылать в конец печатаемого файла символ подачи листа.

Если задать *nenscript* параметр *-Z*, то он пытается определить, не имеет ли проходящий через него файл формат PostScript, и в таком случае пересылает его дальше без изменений.

Если PostScript-файл передается *nenscript* и принимается им за текстовый файл (возможно, из-за того, что не был задан параметр *-Z*), *nenscript* инкапсулирует его и передаст на печать. Это может привести к тому, что будет распечатан код PostScript. В этом случае даже маленький файл PostScript израсходует уйму бумаги.

Предположим, что демон *lpd* уже применяет *nenscript* для обработки посылаемых на принтер файлов. Файл должен обрабатываться правильно, но дополнительная фильтрация может привести к повторному проходу через *nenscript* и инкапсуляции исходного кода PostScript. Более безопасно установить переменную окружения `NENSCRIPT`, указав в ней параметры обработки для *nenscript* по умолчанию. Когда фильтр *nenscript* применяется к печатаемому файлу через */etc/printcap*, используются параметры, установленные в переменной окружения `NENSCRIPT`, но они переопределяются параметрами, явно передаваемыми в командной строке *lpr* или другой утилитой.

Обратите внимание, что принтер, используемый по умолчанию, можно задать в переменной `PRINTER` или как аргумент *-P*, хранимый в `NENSCRIPT`. Если указать используемый принтер в `NENSCRIPT`, то он будет использоваться всякий раз, когда `NENSCRIPT` фильтрует один из ваших файлов. Мы рекомендуем устанавливать принтер в переменной `PRINTER`, а не как аргумент *-P* в `NENSCRIPT`, благодаря чему можно менять принтер и получать правильную фильтрацию.

10

Установка X Window System

Теперь мы подошли к X Window System – одному из самых мощных и значительных программных пакетов для Linux. Если вы когда-либо пользовались системой X на Unix, вам повезло: работа с X под Linux практически не отличается от Unix-систем. А если вам не приходилось пользоваться ею ранее, не стоит бояться: спасение у вас под рукой.

Сложно описать X Window System в двух словах. X – это полнооконный графический интерфейс для Unix-систем. Он предоставляет огромное количество параметров как для программиста, так и для пользователя. В частности, есть как минимум полдюжины *менеджеров окон* для X, каждый из которых предлагает свой интерфейс для управления окнами. Настроив атрибуты менеджера окон, вы можете полностью контролировать положение окон на экране, их цвет и рамки, используемые для оформления, и т. д.

Даже если ранее вам не приходилось слышать о X Window System, то такие аббревиатуры, как KDE и GNOME, вам, наверное, встречались. Это так называемые графические настольные среды, обеспечивающие пользователю Linux удобное окружение при повседневной работе. Более подробно о них будет рассказано в следующей главе.

Система X первоначально была разработана в рамках проекта Athena в Массачусетском технологическом институте (MIT) и Digital Equipment Corporation. Текущая версия X – это версия 11 в редакции 6 (X11R6), которая впервые была представлена в апреле 1994 года. После этого релиза X стала стандартом de facto для графических сред Unix-систем.

Несмотря на коммерческое использование, X Window System продолжает распространяться по свободной лицензии от X Consortium. Поэтому полная реализация X свободно доступна для систем Linux. В Linux наиболее часто используется XFree86 – реализация X, первоначально созданная для систем Unix на базе i386. Сейчас эта версия поддерживает не только системы Intel, но и платформы Alpha AXP, MicroSPARC, PowerPC и другие архитектуры. Планируется поддержка других архитектур. XFree86 основана на X386-1.2, которая была частью официальных кодов X11R5, но больше не поддержива-

ется, и потому считается устаревшей. Текущие версии имеют очень мало общего со своими предшественниками. Была добавлена поддержка множества графических карт и операционных систем (включая Linux), при этом XFree86 реализует последнюю версию X11R6.3.

Следует отметить, что существуют коммерческие серверы X Window System для Linux, у которых могут быть преимущества над XFree86, например в виде дополнительной поддержки некоторых видеокарт. Тем не менее большинство обходится XFree86, с которой и следует начинать.

Дистрибутивы Linux обычно автоматически устанавливают X. Если вам повезет, эта глава вам вообще не потребуется. Но не всем пользователям так везет: иногда дистрибутив не распознает некоторое графическое оборудование; записывает файлы не туда, куда требуется, из-за чего сервер X не может запуститься; или возникают другие проблемы. Одним из достоинств этой книги является то, что настройка X обсуждается в ней настолько глубоко, что вы сможете запустить систему независимо от действий дистрибутива. Вам может и не потребоваться чтение этой главы, но если потребуется, вы оцените все, что здесь написано.

Еще один совет: если вы собираетесь использовать свою машину под Linux только как сервер, нет необходимости ставить на ней систему X (если только вы не хотите пользоваться графическими средствами администрирования). X требует системных ресурсов, а если у вашей машины нет монитора, то установка X – пустая трата времени и ресурсов.

В этой главе мы расскажем, как установить и настроить X Window System, а в следующей главе поговорим об ее использовании.

Основы X

X основана на модели клиент-сервер, в которой *X-сервер* – это программа, работающая на вашей системе и обрабатывающая все обращения к графическому оборудованию. *X-клиент* – это прикладная программа, которая общается с сервером, посылая запросы типа «нарисовать линию» или «обратить внимание на ввод с клавиатуры». X-сервер стремится обслужить эти запросы, рисуя на экране линию или посылая пользовательский ввод (с клавиатуры, мыши или чего-либо еще) приложению-клиенту. Примерами X-клиентов могут служить ставший знаменитым графический редактор *GIMP* и многие программы, входящие в упомянутые графические среды KDE и GNOME, например почтовая программа *kemail* из KDE.

Важно отметить, что X – это графическая система, ориентированная на работу в сети, т. е. клиенты X могут работать локально (на той же системе, где работает X-сервер) или удаленно (на системе, находящейся в сети TCP/IP). X-сервер прослушивает и локальные, и удаленные сетевые сокеты, ожидая поступление запросов от клиентов. Очевидно, что это весьма мощная возможность. Если есть подключение к сети TCP/IP, можно запустить приложение X на удаленной машине в сети и направить результаты его работы на монитор вашего локального X-сервера.

Другими преимуществами X являются безопасность (если пользователь пожелает), модульное разделение функций и поддержка множества различных архитектур. Все это делает X Window System технически более совершенной, чем все другие оконные системы.

X Window System делает различие между поведением приложения и *управлением окнами*. Клиенты, работающие под X, отображаются на экране в одном или нескольких окнах. Однако управление окнами (позиция на экране, изменение размеров и т. д.) и их оформление (вид оконных рамок) не контролируются X-сервером. Этим занимается другой клиент X, называемый *менеджером окон*, который работает одновременно с другими клиентами X. Выбор оконного менеджера в некоторой степени определяет, как будет выглядеть система X в целом. Большинство менеджеров окон обладает крайней гибкостью и множеством настроек. Путем редактирования файлов конфигурации пользователь может выбрать внешний вид оформления окон, поведение фокуса, значение кнопок мыши в зависимости от того, где находится указатель – над рабочим столом или окном приложения – и многое другое. Современные системы позволяют выполнять эти настройки через графический интерфейс.

Для того чтобы полностью понять концепцию менеджеров окон, нужно усвоить, что менеджер окон не влияет на то, что клиентское приложение выводит в свое окно. Он отвечает только за оформление окна, т. е. за рамки и кнопки, позволяющие закрывать окна, передвигать их и изменять размеры.

На любом X-сервере может быть только один менеджер окон. Теоретически можно обойтись и без оконного менеджера, но тогда вы не сможете перемещать окна по экрану, поместить окно на передний план, минимизировать, разворачивать или изменять размеры окон, если сами программы не предоставят такую функциональность.

Коснемся еще раз настольных графических сред. Графическая среда типа KDE или GNOME – это набор приложений и утилит со стандартизированным внешним видом и многими другими общими свойствами – например, меню всех приложений могут быть организованы по одному и тому же принципу. Графическим средам в X всегда нужен менеджер окон, как говорилось выше. Некоторые графические среды предоставляют собственный менеджер окон (например, *kwin* в среде KDE), в других собственного менеджера окон нет. Решение о том, какой менеджер окон должен быть установлен, принимает пользователь.

Требования к оборудованию

В этом разделе указаны чипсеты для видеоадаптеров, поддерживаемые в XFree86 версии 4.2.0, вышедшей в январе 2002 года. В документации к вашему видеоадаптеру должен быть указан используемый чипсет. Если вы хотите приобрести новую видеокарту или новый компьютер, который поставляется с видеоадаптером, попросите поставщика уточнить изготовителя, модель и набор микросхем. Это может потребовать от продавца вызова технической поддержки; обычно продавцы с удовольствием делают это. Многие

продавцы оборудования для PC заявят, что видеокарта является «стандартной SVGA-картой» и «должна работать» в вашей операционной системе. Объясните им, что ваше программное обеспечение (Linux и XFree86!) не поддерживает все типы видеочипсетов, и вам требуется подробная информация.

Хорошим источником для определения поддержки вашей видеокарты X-сервером является сайт *http://www.xfree86.org/cardlist.html*.

Вы можете также определить тип чипсета, вызвав команду *SuperProbe*, входящую в состав XFree86. Более подробно это будет описано ниже.

Поддерживаются следующие типы SVGA-чипсетов (с графическим ускорителем и без него):

3Dfx

Voodoo Banshee, Voodoo Graphics, Voodoo2, Voodoo3, Voodoo4, Voodoo5

3Dlabs

Permedia series, GLINT series

Alliance

AT24, AT25, AT3D

ARK Logic

ARK1000PV, ARK2000PV, ARK2000MT

ATI

Большинство чипсетов Mach32, Mach64 и Rage (GX, CX, CT, ET, VT, VT3, GT, RageII+DVD), RagePro (GB, GD, GI, FP, GQ), VT4, Rage IIC (GV, GW, GZ), Rage LT Pro (LD, LB, LI, LP), Rage LT, Rage XL or XC (GL, GM, GN, GO, GR, GS), Rage Mobility (LM, LN, LR, LS), чипсеты VGAWonder (18800, 18800-1, 28800-2, 28800-4, 28800-5, 28800-6), Rage 128

Chips and Technologies

65520, 65525, 65530, 65535, 65540, 65545, 65546, 65548, 65550, 65554, 65555, 68554, 69000, 64200, 64300

Cirrus Logic

Alpine (5430, 5434, 5436, 5446, 5480, 7548), Laguna (5462, 5464, 5465)

Compaq/Digital

DEC 21030 TGA 8-plane, 24-plane, 24-plane 3D chips

Cyrix

Cyrix MediaGX

IBM

Standard IBM VGA

Integrated Micro Solutions (IMS)

IMS Twin Turbo 128, Twin Turbo 3D

Intel

i740, i810, i815, i830

Matrox

MGA2064W (Millenium I), MGA1064SG (Mystique), MGA2164W (Millenium II) PCI and AGP, G100, G200, G400, G450, G550

NeoMagic

NM2070, NM2090, NM2093, NM2097, NM2160, NM2200, NM2230, NM2360, NM2380

NVIDIA

Riva 128, 128ZX, TNT, TNT2 (Ultra, Vanta, M64), GeForce (DDR, 256), Quadro, GeForce2 (GTS, Ultra, MX), GeForce3, Quadro2

Number Nine

Imagine 128, Ticket 2 Ride, Revolution 3D, Revolution IV

Rendition/Micro

Verite 1000, 2100, 2200

S3

964 (версии 0 и 1), 968, Trio32, Trio64, Trio64, Trio64V+, Trio64UV+, Aurora64V+, Trio64V2 и PLATO/PX (поддерживаются только модели, использующие IBM RGB 524, Texas Instruments 3025 или внутренний чип TrioDAC RAMDAC); ViRGE, ViRGE/VX, ViRGE/DX, ViRGE/GX, ViRGE/GX2, ViRGE/MX, ViRGE/MX+, Trio3D, Trio3D/2X; Savage3D, Savage3D/MV, Savage4, Savage2000, SuperSavage

SGI

Indy Newport (XL) cards

Silicon Integrated Systems (SIS)

300, 530, 540, 620, 630, 6326

Silicon Motion, Inc.

Lynx, LynxE, Lynx3D, LynxEM, LynxEM+, Lynx3DM

Sun Microsystems

BW2, CG3, CG6, CG14, FFB, LEO, TCX буферы кадров

Trident Microsystems

TVGA8900B, TVGA8900C, TVGA8900CL, TVGA9000, TVGA9000i, TVGA9100B, TVGA9200CXr, TVGA8900D, TGUI9440AGi, TGUI9660, TGUI9680, ProVidia 9682, ProVidia 9685, 3DImage975, 3DImage985, Blade3D, Cyber9320, Cyber9382, Cyber9385, Cyber9388, Cyber9397, Cyber9397/DVD, Cyber9520, Cyber9525/DVD, CyberBlade/Ai1, CyberBlade/i7, CyberBlade/i1, CyberBlade/DSTN/Ai1, CyberBlade/DSTN/i7, CyberBlade/DSTN/i1, CyberBlade/ e4, CyberBladeXP, BladeXP

Tseng Labs

ET4000AX, ET4000/W32, ET4000/W32i, ET4000/W32p, ET6000, ET6100

Следует отметить, что проект XFree86 недавно перешел на совершенно новую архитектуру драйверов, которая отличается значительно большей гибкостью, чем прежняя, и позволит более оперативно осуществлять поддержку новой

видеоаппаратуры. Однако пока не все драйверы, существовавшие в прежних версиях, перенесены на новую архитектуру. Поэтому изрядная часть графических чипсетов (преимущественно старых) поддерживается прежней версией (3.3.6), но не поддерживается текущей. Если это коснулось вашей графической аппаратуры (список драйверов есть на *http://xfree86.org*), то может потребоваться установить старую версию программного обеспечения.

Видеокарты с перечисленными микросхемами обычно поддерживаются для всех типов шин, включая PCI и AGP.

Все эти чипсеты поддерживаются в режиме 256 цветов, некоторые поддерживаются в моно- и 16-цветном режиме, а некоторые поддерживаются и при большей глубине цвета.

Со временем этот список будет, несомненно, расширяться. Полный список поддерживаемых видеочипсетов должен находиться в замечаниях к текущей версии XFree86. Кроме того, всегда смотрите файл *README* для вашего конкретного чипсета.

Помимо этих наборов микросхем в ядро, начиная с серии 2.2, включена также поддержка устройства буфера кадров через драйвер *fbdev*. Если ваша видеокарта поддерживается любым обычным драйвером X-сервера, для достижения большей производительности следует использовать его, если нет, то возможно, что запустить X все же удастся с помощью буфера кадров. Для некоторого оборудования даже устройство буфера кадров обеспечивает поддержку графического ускорения.

Одной из проблем, с которой столкнулись разработчики XFree86, стало использование производителями видеокарт нестандартных механизмов определения частот синхронизации для управления картой. Некоторые производители либо не описывали способов программирования карты, либо требовали подписания дополнительного соглашения о нераспространении полученной информации. Очевидно, что это ограничило бы свободное распространение XFree86, чего, конечно, команда разработчиков XFree86 допускать не хочет.

Предлагаемая минимальная конфигурация компьютера для установки XFree86 под Linux – это компьютер серии 486 с 32 Мбайт оперативной памяти и видеокартой с одним из вышеперечисленных наборов микросхем. Прежде чем делать решающий шаг и приобретать дорогостоящее оборудование, следует просмотреть документацию XFree86 и убедиться, что желаемая карта поддерживается. Сравнительные тесты для различных видеокарт под XFree86 периодически появляются в телеконференциях Usenet *comp.windows.x.i386unix* и *comp.os.linux.misc*.

К слову, личная Linux-система одного из авторов (Калле) работает на платформе AMD K6-2 с 128 Мбайт оперативной памяти и видеокартой PCI Permedia II с 8 Мбайт DRAM. Такая конфигурация уже значительно превосходит по скорости работы видеоподсистемы многие рабочие станции. XFree86 на системе Linux с акселератором SVGA выдаст большую производительность, чем можно встретить на коммерческих рабочих станциях Unix (которые часто применяют обычные кадровые буферы для графики и предостав-

ляют акселерированное графическое оборудование только как дорогостоящие расширения системы).

Вашей машине потребуется минимум 16 Мбайт физической оперативной памяти и 32 Мбайт виртуальной памяти (например, 16 Мбайт физической и 16 Мбайт подкачки). Помните, что чем больше у вас физической памяти, тем меньше система будет пользоваться подкачкой на диске при нехватке памяти. Поскольку свопинг всегда работает медленно (диски значительно медленнее, чем оперативная память), для комфортной работы с XFree86 необходимо 16 Мбайт и более оперативной памяти. Система с 16 Мбайт физической памяти может работать *значительно* медленнее (возможно, даже в 10 раз медленнее), чем система с 32 Мбайт или больше.

Установка XFree86

Двоичный дистрибутив XFree86 для Linux можно найти на многих FTP-сайтах. На *ftp://ftp.xfree86.org* он находится в каталоге */pub/XFree86/4.2.0/binaries*, где в соответствующих подкаталогах вы найдете системы для различных архитектур. (Когда писалась эта книга, текущей версией была версия 4.2.0; периодически выпускаются новые версии.)

Весьма вероятно, что вы получили XFree86 в составе дистрибутива Linux. В этом случае нет необходимости загружать программу из сети. Если же вы загружаете XFree86 с FTP, взгляните на табл. 10.1 и 10.2, где перечислены файлы в дистрибутиве XFree86-4.2.0.

Чтобы узнать, какие файлы вам необходимо загрузить, настоятельно советуем загрузить сначала небольшой сценарий оболочки с именем *Xinstall.sh* и запустить его следующей командой:

```
sh Xinstall.sh -check
```

В результате вы получите перечень двоичных файлов, которые вам необходимо загрузить с сервера.

Необходимые файлы перечислены в табл. 10.1.

Таблица 10.1. Файлы, необходимые для установки XFree86

Файл	Описание
Xinstall.sh	Программа установки
extract	Утилита для распаковки
Xbin.tgz	Клиенты и утилиты X, а также библиотеки этапа исполнения
Xlib.tgz	Файлы данных, необходимые на этапе исполнения
Xman.tgz	Страницы руководства
Xdoc.tgz	Документация
Xfnts.tgz	Основные шрифты
Xfenc.tgz	Базовый набор данных кодировки шрифтов
Xetc.tgz	Файлы конфигурации этапа исполнения

Таблица 10.1 (продолжение)

Файл	Описание
Xvar.tgz	Данные этапа исполнения
Xxserv.tgz	Сам X-сервер
Xmod.tgz	Различные модули драйверов

Есть также ряд дополнительных файлов, загружаемых только в случае необходимости и перечисленных в табл. 10.2.

Таблица 10.2. Дополнительные файлы для установки XFree86

Файл	Описание
Xfsrv.tgz	Сервер шрифтов
Xnest.tgz	Вложенный X-сервер
Xprog.tgz	Заголовочные файлы X, файлы конфигурации и библиотеки, необходимые во время компиляции
Xprt.tgz	Сервер печати X
Xvfb.tgz	Сервер X, использующий виртуальный буфер кадров
Xf100.tgz	Шрифты в разрешении 100 dpi
Xfcyr.tgz	Кириллические шрифты
Xfscl.tgz	Масштабируемые шрифты
Xhtml.tgz	HTML-версия документации
Xps.tgz	PostScript-версия документации
Xjdoc.tgz	Японская версия документации

Каталог XFree86 должен содержать файлы *README* и замечания по установке текущей версии.

Получите эти файлы и сохраните их в каталоге */var/tmp* (вы можете использовать другой каталог; просто измените соответствующим образом путь к нему в следующих примерах). Затем снова запустите сценарий установки, на этот раз без параметра *-check*:

```
sh Xinstall.sh
```

Вам будет задан ряд вопросов. Обычно можно нажимать <Enter>, чтобы принять значения по умолчанию. В частности, на вопрос, нужно ли создавать ссылки для OpenGL, ответьте утвердительно, и будет установлена совместимая версия. OpenGL – библиотека для ускоренной 3D-графики, которой часто пользуются приложения для трехмерной графики и никакие другие. Если вы собираетесь запускать какие-либо приложения с 3D-графикой, эту библиотеку нужно включить. Программа установки также спросит, нужно ли создавать ссылку на утилиту *rstart*. Если вы не знаете, что это такое, можете смело отказаться.

Когда сценарий завершит свою работу, XFree86 окажется установлена, и можно будет приступить к ее настройке, о которой рассказывается в следующем разделе.

Настройка XFree86

Настроить XFree86 в большинстве случаев несложно. Однако если драйвер для вашего оборудования находится в процессе разработки, или вы хотите добиться лучшей производительности или разрешения от видеокарты с графическим акселератором, настройка XFree86 может потребовать некоторого времени.

В этом разделе мы опишем, как создать и отредактировать файл *XF86Config*, который настраивает сервер XFree86. В текущей версии XFree86 этот файл находится в каталоге */etc/X11/*, но раньше он находился в */etc/*. Кроме того, при переходе от версии 3 к версии 4 изменился формат этого файла. Предлагаемое описание относится к текущей, четвертой версии (вдобавок, некоторые версии Red Hat называют этот файл *XF86Config-4*). В большинстве случаев лучше всего начать с «основной» конфигурации XFree86, где используется низкое разрешение. Хорошим выбором будет разрешение 640×480, поддерживаемое всеми видеокартами и мониторами. Когда XFree86 уже работает, используя стандартное низкое разрешение, вы можете затем подстроить файл конфигурации для использования всех возможностей, предоставляемых видеокартой. Смысл этого заключается в том, чтобы убедиться в работоспособности XFree86 на вашей системе и в успешной установке, прежде чем пытаться настроить XFree86 для реального использования (иногда это бывает сложно). С оборудованием, используемым в настоящее время, вы можете легко получить разрешение до 1024×768 пикселов.

Но прежде чем вы начнете писать файл *XF86Config* вручную, попробуйте воспользоваться одной из программ конфигурирования. Во многих случаях можно обойтись без хитроумных настроек, описанных на следующих страницах. Вот некоторые программы, способные помочь в этом:

Средства конфигурации, специфичные для дистрибутива

Некоторые дистрибутивы имеют свои собственные средства конфигурации. Например, у SuSE есть *SaX*, а у Red Hat есть *Xconfigurator*. Обычно это очень удобные средства, и следует попытаться ими воспользоваться. Но если попытка окажется неудачной, полезно помнить, что у вас в распоряжении есть другие средства. Большинство программ установки автоматически запускает эти средства, но если настройка X-сервера осуществляется после установки, нужно знать имя такой программы, чтобы правильно указать его в командной строке. Если возникает неясность, обращайтесь к документации по дистрибутиву.

xf86cfg

Эта графическая программа конфигурации, как и описываемая ниже *xf86config*, создана самими разработчиками XFree86. Она запускает ми-

нимальный X-сервер (который работает практически на любом видеооборудовании), а затем позволяет выбрать видеокарту, тип монитора, тип мыши и другие параметры. В конце программа пытается запустить сервер в выбранной конфигурации, и если результат удовлетворительный, предлагает записать файл конфигурации. Во многих случаях эта программа оказывается очень удобной и надежной.

Параметр -configure

Если выполнить команду

```
XFree86 -configure
```

в качестве суперпользователя, X-сервер попытается выяснить как можно больше информации о вашем оборудовании и запишет соответствующий файл *XF86Config* в */root/XF86Config.new* (т. е. не перезапишет старый файл конфигурации, если таковой имеется). Чтобы опробовать полученную конфигурацию, можно запустить X-сервер вручную:

```
XFree86 -xf86config /root/XF86Config.new
```

Эта команда использует вновь созданный файл конфигурации вместо установленного по умолчанию.

xf86config

Эта программа, предоставляемая разработчиками XFree86, работает в текстовом режиме и, задавая множество вопросов, генерирует файл конфигурации *XF86Config*, основываясь на ваших ответах. Ее интерфейс не очень удобен, но бывают случаи, что она работает там, где другие варианты терпят неудачу.

Если одно из этих средств смогло настроить ваш X-сервер, им и следует пользоваться, избежав тем самым множества неприятностей. Однако если все они потерпят неудачу или вы хотите выполнить тонкую настройку X-сервера, вам необходимо знать, как редактировать файл *XF86Config* самостоятельно.

В дополнение к приведенной здесь информации следует ознакомиться со следующей документацией:

- Документация по XFree86, которая находится в каталоге */usr/X11R6/lib/ X11/doc* (в пакете *Xdoc*).
- Файл *README* (если таковой существует) для вашего видеочипсета, находящийся в каталоге */usr/X11R6/lib/X11/doc*. Такие файлы имеют имена типа *README.i740* или *README.cyrix*.
- Страницы руководства по *XFree86*.
- Страницы руководства по *XF86Config*.

Основным файлом конфигурации, который вам необходимо создать, является */etc/X11/XF86Config* (в старых версиях это */etc/XF86Config*). Этот файл содержит информацию о мыши, параметрах видеокарты и т. д. В качестве примера в дистрибутиве XFree86 содержится файл *XF86Config.eg*. Скопируйте этот файл в *XF86Config* и начните его редактировать.

Страница руководства по *XF86Config* подробно объясняет формат этого файла. Прочтите сейчас это руководство, если вы еще не сделали этого.

Сейчас мы рассмотрим пример файла *XF86Config*, участок за участком. Этот файл может выглядеть не совсем так, как файл, входящий в дистрибутив XFree86, но их структура совпадает. Формат файла *XF86Config* может изменяться с каждой версией XFree86; приводимая информация относится только к XFree86 версии 4.2.0.

Также не следует просто копировать файл конфигурации, приведенный здесь, и пытаться использовать его на вашей системе. Попытка использования файла конфигурации, не соответствующего оборудованию, может заставить монитор работать со слишком высокой для него частотой; были сообщения о выходе из строя мониторов (особенно мониторов с фиксированной частотой) при использовании неверно сконфигурированного файла *XF86Config*. Вывод отсюда один: следует быть абсолютно уверенным в том, что файл *XF86Config* соответствует установленному оборудованию, прежде чем пытаться использовать его.

После сделанного предупреждения хотим отметить, что настройка XFree86 стала значительно менее опасной, чем несколько лет назад, поскольку теперь X-сервер очень хорошо обнаруживает недопустимые настройки.

Каждая секция (section) файла *XF86Config* начинается со строки `Section "section-name"` и заканчивается строкой `EndSection`. Первая часть файла называется `Files` и выглядит следующим образом:

```
Section "Files"
    FontPath    "/usr/X11R6/lib/X11/fonts/misc:unscaled"
    FontPath    "/usr/X11R6/lib/X11/fonts/Type"
    RgbPath     "/usr/X11R6/lib/X11/rgb"
EndSection
```

Таких строк может быть много. Строка `RgbPath` устанавливает путь к базе данных цветов RGB для X11R6, а каждая строка `FontPath` определяет путь к каталогу, содержащему шрифты X11. Обычно не следует изменять эти строки; необходимо только убедиться, что для каждого установленного типа шрифта есть запись `FontPath` (т. е. для каждого каталога в */usr/X11R6/lib/X11/fonts*). Если в `FontPath` добавить строку `:unscaled`, шрифты из этого каталога не будут масштабироваться. Это часто полезно, т. к. значительно увеличенные шрифты выглядят ужасно. В эту секцию помимо `FontPath` и `RgbPath` можно добавить `ModulePath` для указания на каталог с динамически подгружаемыми модулями. Эти модули в настоящее время используются для некоторых специальных устройств ввода, а также для расширений PEX и XIE.

Следующей секцией является `ServerFlags`, которая определяет несколько глобальных параметров для сервера. Как правило, эта секция пустая или очень короткая:

```
Section "ServerFlags"
  Option       "AllowMouseOpenFail"
EndSection
```

Здесь сказано, что мы хотим, чтобы X-сервер запускался даже тогда, когда он не может обнаружить мышь. Дополнительные параметры можно найти на странице руководства *XF86Config*(1). Часто параметры автоматически обнаруживаются при запуске сервера, и их не нужно здесь перечислять.

Далее следует раздел `Module`, служащий для динамической загрузки дополнительных модулей X-сервера, таких как поддержка специальных устройств или графических библиотек вроде PEX. Если вам не нужно использовать специальные модули (для которых должна быть своя документация), можете убрать этот раздел целиком.

Следующие секции имеют название `InputDevice`. Обычно их не менее двух – одна для клавиатуры и одна для мыши. Если есть другие устройства, например, графический планшет, для них создаются отдельные секции.

```
Section "InputDevice"
        Identifier  "Keyboard1"
        Driver      "Keyboard"
        Option            "AutoRepeat" "250 30"
        Option            "XkbRules" "xfree86"
        Option            "XkbModel" "pc105"
        Option            "XkbLayout" "us"
EndSection
```

```
Section "InputDevice"
  Driver       "mouse"
  Identifier   "Mouse[1]"
  Option       "Device" "/dev/mouse"
  Option       "Emulate3Buttons" "on"
  Option       "Emulate3Timeout" "50"
  Option       "InputFashion" "Mouse"
  Option       "Name" "Autodetection"
  Option       "Protocol" "ps/2"
  Option       "Vendor" "Sysp"
EndSection
```

Есть и другие опции. Приведенные настройки клавиатуры относятся к клавиатуре для США. Если у вас другая клавиатура, нужно изменить соответствующие строки.

В секции `mouse` X-серверу сообщается, каким образом подключена мышь (в данном случае это */dev/mouse*, что обычно является ссылкой на соответствующий последовательный порт, например */dev/ttyS0*), какого типа эта мышь (параметр `"Protocol"`) и некоторые другие детали. Важно правильно указать протокол, но упоминавшиеся программы конфигурирования обычно должны автоматически определять протокол. Список поддерживаемых протоколов есть в */usr/X11R6/lib/X11/doc/README.mouse*.

Для шинной мыши Logitech нужно указать `BusMouse`. Для более старых мышей Logitech, которые не являются шинными, нужно задавать `Logitech`, но для более новые мышей Logitech, которые не являются шинными, нужно указывать протокол `Microsoft` или `Mouseman`. Это пример того, что протокол не обязательно имеет какое-то отношение к названию изготовителя мыши.

Если у вас современная мышь последовательного порта, можно попробовать задать Auto для автоматического выбора драйвера мыши.

Если X запущен, легко проверить, правильно ли выбран драйвер мыши: при перемещении мыши ее указатель на экране должен совершать соответствующее перемещение. Если это так, то настройки, скорее всего, корректные. Если нет – попробуйте другой драйвер и проверьте, правильно ли указано устройство.

Следующая секция файла *XF86Config* – Device. Здесь указываются параметры видеокарты. Если видеокарт несколько, должно быть несколько секций Device.

```
Section "Device"
  BoardName    "SiS630 GUI Accelerator+3D"
  BusID        "1:0:0"
  Driver       "sis"
  Identifier   "Device[0]"
  Screen       0
  VendorName   "SiS"
EndSection
```

Первая строка, BoardName, служит просто напоминанием о том, какая карта здесь конфигурируется (существенно, если карт несколько!). Аналогично, VendorName – строка в свободном формате, служащая чисто описательным целям. Даже строка Identifier может быть выбрана произвольно, но ей должна соответствовать строка Device в последующих секциях файла конфигурации. Обычно здесь используют такие названия, как Device "0", Device "1" и т. д.

BusID указывает фактическую графическую карту в терминах встроенной аппаратуры на шине PCI. PCI 1:0:0 или, короче, 1:0:0 обычно будет правильным значением, если вариант только один. Если есть сомнения, запустите X-сервер строкой

```
XFree86 -scanpci
```

и внимательно изучите вывод; в нем должна быть хотя бы одна видеокарта (возможно, среди других устройств, которые в данный момент нас не интересуют). Например, строка

```
(1:0:0) Matrox unknown card (0x19d8) using a Matrox MGA G400 AGP
```

сообщает о наличии карты Matrox MGA G400 с установленным соединением AGP. Первые цифры в скобках указывают идентификатор шины PCI, о котором говорилось выше.

Секция Screen обязательна для видеокарт, допускающих подключение нескольких мониторов; для прочих здесь указывается 0.

Строка Driver очень важна, поскольку определяет фактический графический драйвер, который должен быть загружен X-сервером. Узнать название правильного драйвера можно с помощью описывавшихся выше программ конфигурирования или из результатов, которые дает запуск X-сервера строкой

```
XFree86 -probeonly
```

Здесь можно выяснить, какую информацию X-сервер собрал о вашей аппаратуре, включая драйвер, который, по его мнению, должен быть использован.

В этом файле можно задать массу других параметров, включая набор микросхем видеокарты, RAMDAC и другие свойства аппаратуры, но X-сервер неплохо определяет их сам, поэтому обычно делать это не требуется. Если вы все же хотите задать параметры сами, прочтите файл *README* для конкретного драйвера, в котором должны быть перечислены параметры этого драйвера и их возможные значения.

Следующая секция, `Monitor`, определяет характеристики вашего монитора. Как и в случае с другими секциями, в файле *XF86Config* может быть более одной секции `Monitor`. Это полезно, когда к вашей системе подключено несколько мониторов, или один и тот же файл *XF86Config* используется с несколькими конфигурациями оборудования. Однако обычно необходима только одна секция `Monitor`:

```
Section "Monitor"
  Option       "CalcAlgorithm" "CheckDesktopGeometry"
  HorizSync    31-65
  Identifier   "Monitor[0]"
  ModelName    "1024X768@70HZ"
  VendorName   "--> LCD"
  VertRefresh  58-78
  UseModes     "Modes[0]"
EndSection
```

Строка `Identifier` используется для задания произвольного имени записи `Monitor`. Это может быть любая строка; она используется потом для ссылки на секцию `Monitor` в файле *XF86Config*.

`HorizSync` указывает возможные частоты горизонтальной развертки для вашего монитора в килогерцах. Если у вас многочастотный монитор (multisync), можно указать интервал значений (или несколько интервалов, разделенных запятыми). Для мониторов с фиксированными частотами указывается список фиксированных значений, например:

```
HorizSync    31.5, 35.2, 37.9, 35.5, 48.95
```

В руководстве для монитора эти значения должны быть описаны в разделе технических спецификаций. Если у вас нет этой информации, следует получить ее у производителя или продавца вашего монитора. Кроме этих источников информации имеются и другие; они будут перечислены позже.

Следует проявлять осторожность, устанавливая эти значения. Значения `VertRefresh` и `HorizSync` (они описаны ниже) помогают убедиться, что ваш монитор не будет поврежден неправильными настройками. Если они указаны неправильно, вы вряд ли будете довольны вашей установкой X. Возможны нестабильное изображение, мерцание или обычные видеопомехи типа «снега».

`VertRefresh` указывает допустимые частоты вертикальной развертки (или частоты вертикальной синхронизации) для вашего монитора в килогерцах. Как и в случае с `HorizSync`, это может быть диапазон значений или список

фиксированных значений. Допустимые значения должны быть указаны в руководстве для монитора.

HorizSync и VertRefresh используются только для дополнительного контроля того, что разрешения монитора, которые вы указали, находятся в допустимых диапазонах. Это уменьшает риск повреждения монитора при попытке его использования на частотах, для которых он не предназначен.

Директивы Modeline и Mode используются, чтобы задать режимы разрешения монитора. В отличие от более старых версий XFree86, они не являются обязательными. В приведенной секции Monitor, взятой с лэптопа, их нет. Эта информация переместилась в следующий раздел, Modes.

Секция Modes, которая должна быть отдельно для каждого настраиваемого монитора, перечисляет различные видеорежимы, которые должен поддерживать X-сервер, например:

```
Section "Modes"
  Identifier   "Modes[0]"
  Modeline         "1024x768" 69.35 1024 1040 1216 1400 768 768 778 802
  Modeline         "1024x768" 79.55 1024 1040 1216 1400 768 768 778 802
  Modeline         "800x600" 42.43 800 816 928 1072 600 600 610 626
  Modeline         "800x600" 48.67 800 816 928 1072 600 600 610 626
EndSection
```

Строка Identifier ссылается на имя, указанное в секции Monitor. Каждая из строк Modeline задает видеорежим. Формат Modeline следующий:

```
Modeline name dot-clock horiz-values vert-values
```

Здесь *name* – это произвольная строка, которую в дальнейшем можно использовать для ссылки на режим разрешения в этом файле, *dot-clock* – частота генератора, или *dot clock*, связанная с режимом разрешения. Обычно она указывается в мегагерцах и определяет скорость, с которой видеокарта должна посылать пикселы на монитор при данном разрешении. Группы значений *horiz-values* и *vert-values*, состоящие из 4-х чисел каждая, определяют, когда электронная пушка должна включаться и когда должны проходить импульсы горизонтальной и вертикальной синхронизации во время развертки луча.

Как выяснить значения строки Modeline для вашего монитора? Это нелегко, особенно потому, что значительная часть прежней документации по XFree86 более не поставляется, возможно, потому, что она устарела и пока еще не обновлена. Лучше всего воспользоваться одной из программ, генерирующих файлы конфигурации и описанных в предыдущем разделе, чтобы получить первоначальный набор значений и корректировать их, пока настройки не станут удовлетворительными. Например, если во время работы XFree86 изображение на экране несколько смещено или мерцает, нужно понемногу изменять значения, стараясь исправить картинку. Что именно нужно исправлять, сказать трудно, поскольку это зависит от графического устройства, но после некоторых экспериментов обычно можно добиться хороших результатов. Кроме того, нужно не забывать про ручки контроля на самом мо-

ниторе! Часто необходимо изменить горизонтальный или вертикальный размер изображения после запуска XFree86, чтобы изображение находилось в центре и было правильного размера. Другой вариант – воспользоваться программой *xvidtune* (о том, как ею пользоваться, см. страницу руководства), которая помогает получить все числа, необходимые для Modeline, опробовать их и даже отменить, если что-то сделано ошибочно.

Кроме того, в XFree86 входят так называемые стандартные режимы VESA, благодаря которым можно обойтись вообще без секции Modes. Синхронизация VESA дает стандартные значения для Modeline, которые поддерживаются большинством мониторов, хотя не позволяют полностью использовать возможности конкретных устройств.

Обратите внимание, что аргумент *name* в Modeline (в данном случае "640x480") – это произвольная строка. Принято называть режим по значению разрешения экрана, но можно использовать любое название, которое описывает для вас разрешение монитора.

Для каждой строки Modeline сервер проверяет, попадают ли указанные для режима значения в интервал значений, определенных с помощью HorizSync и VertRefresh. Если они не совпадают, сервер выдает предупреждение при попытке запустить X (подробнее об этом позже).

Не следует пробовать использовать параметры синхронизации и другие значения Modeline, предназначенные для других мониторов, отличающихся от вашей модели. Если вы попытаетесь запустить монитор на частоте, для которой он не предназначен, то можно повредить его или полностью вывести из строя.

Следующая секция файла *XF86Config* называется Screen и задает комбинации мониторов/видеокарт, используемые с конкретным сервером:

```
Section "Screen"
  DefaultDepth 16
  SubSection "Display"
    Depth      15
    Modes      "800x600"
  EndSubSection
  SubSection "Display"
    Depth      16
    Modes      "1024x768"
  EndSubSection
  SubSection "Display"
    Depth      24
    Modes      "800x600"
  EndSubSection
  SubSection "Display"
    Depth      32
    Modes      "800x600"
  EndSubSection
  SubSection "Display"
    Depth      8
    Modes      "800x600"
  EndSubSection
```

```
    Device       "Device[0]"
    Identifier   "Screen[0]"
    Monitor      "Monitor[0]"
  EndSection
   EndSection
```

Эта секция связывает вместе определения устройств, экранов и мониторов и указывает глубину цвета, используемую с разными видеорежимами.

Наконец, секция ServerLayout завершает картину, определяя одну активную конфигурацию, которая состоит из одной или нескольких секций Screen, и одной или нескольких секций InputDevice. Если у вас система с несколькими мониторами (multihead system), в которой несколько графических карт, к каждой из которых подключен монитор, либо одна графическая карта, допускающая подключение нескольких мониторов, то в этой секции также задается их относительное расположение. Вот пример:

```
Section "ServerLayout"
  Identifier   "Layout[all]"
  InputDevice  "Keyboard[0]" "CoreKeyboard"
  InputDevice  "Mouse[1]" "CorePointer"
  Option       "Clone" "off"
  Option       "Xinerama" "off"
  Screen       "Screen[0]"
EndSection
```

Существуют и другие секции, но они совершенно необязательны; X-сервер может работать и без них.

Запуск XFree86

Когда файл *XF86Config* готов, вы можете запустить X-сервер. Сначала убедитесь, что */usr/X11R6/bin* входит в список каталогов поиска.

XFree86 запускается командой:

```
startx
```

Это внешний интерфейс к команде *xinit* (в случае если вы привыкли использовать *xinit* на других Unix-системах). Вы можете по-прежнему использовать *xinit*, предоставляющую полный контроль над запуском программы, но вам придется запускать все необходимые программы вручную.

Эта команда запускает X-сервер и выполняет команды, найденные в файле *.xinitrc* в домашнем каталоге пользователя. Файл *.xinitrc* – это лишь сценарий оболочки, содержащий клиенты X, которые должны быть запущены. Если этот файл не существует, по умолчанию выполняется системный файл */usr/X11R6/lib/X11/xinit/xinitrc*.

Задав в домашнем каталоге другой файл *.xinitrc*, можно изменить параметры начального отображения, устанавливаемые при запуске X Window System. В следующей главе рассказывается о том, что можно поместить в этот файл.

Если вы новичок в среде X Window System, мы настоятельно рекомендуем обратиться к книге типа «The X Window System User's Guide» Валери Кэрсия (Valerie Quercia) и Тима О'Рейли (Tim O'Reilly).

Возможные проблемы

Часто при начальной загрузке X-сервера что-нибудь работает не совсем правильно. Практически всегда это связано с ошибками в файле *XF86Config*. Как правило, неверно указаны значения синхронизации монитора или частоты видеоплаты. Если изображение на экране крутится или размыты края, это указывает на то, что эти значения установлены неверно. Проверьте также, правильно ли указан чипсет видеокарты и другие параметры в секции `Device` в *XF86Config*. В версии 4 XFree86 есть только один исполняемый файл сервера, который загружает модуль, необходимый для конкретной видеокарты. То, какой модуль загружается, зависит от настроек в секции `Device`.

Если ничто другое не помогает, попробуйте запустить «чистый» X-сервер:

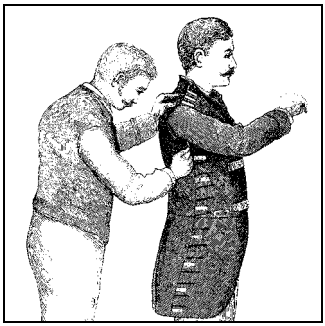
```
X > /tmp/x.out 2>&1
```

Вы можете затем завершить работу X-сервера (комбинацией клавиш <Ctrl>+<Alt>+<Backspace>) и проверить содержимое файла */tmp/x.out*. Сервер запишет туда все предупреждения и ошибки, например, о том, что видеокарта не поддерживает необходимую для монитора частоту. */tmp/x.out* может быть весьма полезен для диагностики проблем любого сорта. Внимательно изучите его, если X-сервер не запускается вовсе, не обеспечивает нужные вам разрешения, либо показывает неустойчивую, со «снегом» или чем-то другим, не удовлетворяющую вас картинку. И даже если все работает нормально, вы можете заглянуть в этот файл, чтобы выяснить, как X-сервер распознал ваше оборудование. Строки, начинающиеся с (**), содержат данные, введенные вами в файл конфигурации, а строки, начинающиеся с (--), содержат данные, которые X-сервер выяснил самостоятельно.

Не забывайте, что можно использовать комбинацию <Ctrl>+<Alt> с клавишами плюс и минус на цифровой клавиатуре для переключения между режимами разрешения монитора, перечисленными в строке `Modes` в секции `Screen` файла *XF86Config*. Если режим с высоким разрешением работает некорректно, попробуйте перейти на меньшее разрешение. По крайней мере вы выясните, что настройки для низких разрешений работают правильно.

Попробуйте также использовать кнопки управления на мониторе. Часто эти настройки необходимо подкорректировать при запуске X, например, если изображение съезжает в одну сторону.

Обсуждению вопросов по XFree86 посвящена телеконференция *comp.windows.x.i386unix*. Просматривая сообщения, относящиеся к такой же конфигурации видеосистемы, как у вас, вы, возможно, обнаружите, что кто-то уже сталкивался с теми же проблемами. Если это не помогает, свяжитесь с вашим дистрибьютором Linux; возможно, его служба технической поддержки вам поможет.



11

Настройка среды X

Система X работает. Но это далеко не конец истории. В предыдущей главе мы рассказали, как установить X Window System, чтобы система распознавала ваш монитор и видеоадаптер. Очевидно, это необходимо, но это только полдела. В этой главе мы расскажем о второй половине – настройке среды X. В чем важность настройки? При работе в X Window System иногда требуется менять настройки, потому что, например, вы стали работать по-другому, появились новые и лучшие варианты среды или вам просто надоело старое окружение. Некоторые из сред достаточно продвинутые. Например, они позволяют запускать программы со всеми необходимыми параметрами одним нажатием кнопки или щелчком мыши или позволяют для распечатки текста перетаскивать значок файла на принтер, а также много других вещей, благодаря которым повышается продуктивность вашего труда.

В этой главе мы сначала обсудим основные аспекты настройки X, а затем познакомимся с двумя наиболее заметными настольными средами для Linux: KDE и GNOME. Читателям, более глубоко интересующимся материалом или испытывающим потребность в приложениях, не входящих в KDE или GNOME, будет полезен рассказ об общих способах настройки (более старых) приложений X, X-ресурсов и некоторых общих приложений, выполняющихся независимо от какой-либо настольной среды.

До недавнего времени проблема использования X на Unix-системах вообще и Linux в частности заключалась в отсутствии интеграции. Можно было использовать менеджер окон и некоторое количество приложений X, но они все выглядели и работали по-разному, а об интеграции не было и речи. Например, банальная для Windows или Macintosh операция drag-and-drop с трудом поддерживалась в Linux, и даже если она была, трудно было найти пару приложений, которые могли бы ее совместно использовать.

Задачу создания приличного графического интерфейса для Unix призван решить относительно новый класс программ – так называемые настольные среды. Они должны предоставить интегрированный удобный «рабочий стол», где все программы выглядят и работают одинаковым образом и даже имеют одинаковое меню (где это возможно).

В настоящее время есть две основные настольные среды для Linux: K Desktop Environment (KDE) и GNOME. KDE немного старше, стабильнее и предлагает значительно больше функциональных возможностей. Эта среда помогает пользователям, пришедшим с других ОС, чувствовать себя как дома, а также предоставляет постоянным пользователям Unix более продуктивное и дружественное рабочее окружение. GNOME, с другой стороны, обладает приятным внешним видом, но менее стабильна, чем KDE, а отдельные программы хуже интегрированы. KDE устанавливается по умолчанию большинством дистрибутивов, но почти во все дистрибутивы входят обе среды. Если в ваш дистрибутив входит только одна из них, значит, создатель дистрибутива пытается за вас решить, чем лучше пользоваться. Мы расскажем об обеих средах – KDE и GNOME.

В Red Hat Linux версии 8.0 для внешнего вида KDE и GNOME выбрана тема Red Hat, что делает обе среды весьма похожими одна на другую. Задача была в том, чтобы придать всем приложениям единообразный вид независимо от выбора KDE или GNOME. Такое сходство должно облегчить поддержку Red Hat Linux и придать дистрибутиву узнаваемый внешний вид независимо от того, которая из графических сред используется. В некоторых случаях даже трудно определить, какая из сред действует в данный момент – KDE или GNOME. Если новый внешний вид вам не нравится, довольно просто вернуть настройки к прежнему виду. Помимо этих настроек есть очень немного отличий от того программного обеспечения, которое поставляется самими проектами KDE и GNOME.

Основы настройки X Window

Перед запуском приложений в X неплохо разобраться с основами настройки этой среды, чтобы не зависеть от настроек по умолчанию (зачастую малопривлекательных). Обратите внимание, что следующие несколько разделов обычно не применяются к упомянутым выше настольным средам, в которых есть более дружественные пользователям основанные на GUI средства конфигурирования приложений и среды. Однако знать старые способы настройки все же полезно, потому что вы вполне можете столкнуться с приложениями, которые по-прежнему следуют этой схеме.

xinit

X запускается командой *startx*. Это оболочка для программы *xinit*, ответственной за запуск X-сервера и различных X-клиентов. Программа *xinit* (через *startx*) выполняет сценарий оболочки *.xinitrc* в вашем домашнем каталоге пользователя. Этот сценарий просто содержит команды, которые желательно выполнить при запуске X, например, *xterm*, *xclock* и т. д. Если отсутствует пользовательский файл *.xinitrc*, вместо него по умолчанию используется системный файл */usr/lib/X11/xinit/xinitrc*.

При использовании KDE в файле *.xinitrc* часто находится лишь одна команда *startkde*. Это поставляемый с KDE сценарий оболочки, который запускает все необходимые службы. Однако поставщик дистрибутива может добавить в *.xinitrc* и другие команды.

Мы приведем пример файла *.xinitrc* для «голой» системы X без какой-либо настольной среды и объясним, что он делает. Можно использовать этот *.xinitrc* в качестве основы для собственного *.xinitrc* или скопировать для этой цели файл, используемый системой по умолчанию:

```
1  #!/bin/sh
2  # Sample .xinitrc shell script
3
4  # Start xterms
5  xterm -geometry 80x40+10+100 -fg black -bg white &
6  xterm -geometry -20+10 -fn 7x13bold -fg darkslategray -bg white &
7  xterm -geometry -20-30 -fn 7x13bold -fg black -bg white &
8
9  # Other useful X clients
10 oclock -geometry 70x70+5+5 &
11 xload -geometry 85x60+85+5 &
12 xbiff -geometry +200+5 &
13 xsetroot -solid darkslateblue &
14
15 # Start the window manager
16 exec fvwm2
```

Этот сценарий запускает ряд программ; очевидно, что они должны быть предварительно установлены. Если вы не вмешивались в ход установки своего дистрибутива, то, скорее всего, была установлена одна из сред KDE или GNOME, но не эти более старые программы. Однако они должны быть на установочном носителе в готовности быть инсталлированными. Хотя несколько менеджеров окон не могут работать одновременно, устанавливать их в системе можно в произвольном количестве.

Все должно быть достаточно ясно, даже если вы не знакомы с X. Первые две строки идентифицируют сценарий оболочки. Строки 5–7 запускают три клиента *xterm* (*xterm* – это эмулятор терминала). Другие клиенты запускаются в строках 10–13, а в строке 16 запускается менеджер окон *fvwm*. Результат запуска *startx* при наличии такого *.xinitrc* приведен на рис. 11.1.[1]

Давайте посмотрим на детали. В строке 5 *xterm* запускается с несколькими параметрами: *-geometry*, *-fg* и *-bg*. Большинство X-клиентов поддерживают эти стандартные параметры наряду с другими. Параметр *-geometry* позволяет указать размер и позицию окна на экране и имеет следующий формат:

```
xsizexysize+xoffset+yoffset
```

[1] Да, согласны, это не произведение искусства, но нам требовалось нечто простое, что может работать на всех дисплеях.

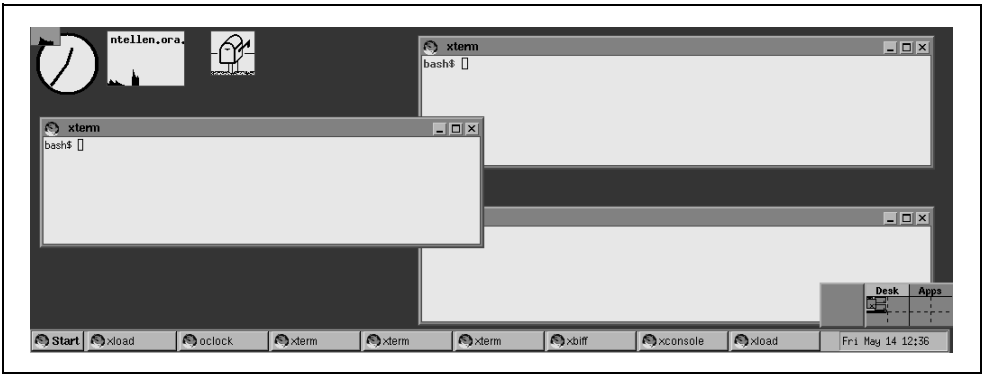


Рис. 11.1. *Экран, создаваемый файлом .xinitrc из примера*

В нашем примере параметр *-geometry 80x40+10+100* поместит окно в позицию (10,100) (где (0,0) – это верхний левый угол экрана) и задаст ширину и высоту, равными 80 и 40 символам, соответственно. Обратите внимание, что *xterm* измеряет размеры окна в *символах*, а не в пикселах. Реальный размер окна в пикселах определяется используемым шрифтом.

Параметры *-fg* и *-bg* позволяют указать цвета переднего (цвет текста) и заднего плана для окна *xterm*. Здесь используются скучные цвета – черный и белый, но результат будет одинаков на цветных и монохромных дисплеях. В X Window цвета, как правило, указываются по именам, хотя можно указать и значения RGB. Список названий цветов (и соответствующих значений RGB) приводится в файле */usr/lib/X11/rgb.txt*. Запустив *xcolors*, вы увидите эти цвета вместе с их именами.

Строка 6 запускает еще одно окно *xterm*, но при этом используются несколько другие параметры:

```
xterm -geometry -20+10 -fn 7x13bold -fg darkslategray -bg white &
```

Прежде всего, в параметре *geometry* мы видим только `-20+10`. Если не задан размер, *xterm* использует размер по умолчанию, который обычно равен `80x25`. Также мы видим, что *xoffset* используется с префиксом «–» вместо «+». В этом случае окно будет расположено в 20 пикселах от *правого* края экрана. Аналогично значение `-20-30` (в строке 7) определяет положение окна в 20 пикселах от правого края экрана и в 30 пикселах от нижней границы. В этом случае расположение окон меньше зависит от конкретного разрешения экрана.

Параметр *-fn* в строках 6 и 7 указывает на то, что *xterm* следует использовать шрифт `7x13bold`. С помощью команды *xlsfonts* можно посмотреть полный список шрифтов на вашей системе; X-клиент *xfontsel* позволяет выбирать шрифты интерактивно (см. подробнее о шрифтах ниже).

В строке 10 запускается клиент *oclock* – простые аналоговые часы. Строка 11 запускает программу *xload*, отображающую график *средней загрузки* системы (число запущенных процессов), который постоянно изменяется. Строка 12

запускает клиент *xbiff*, оповещающий о наличии свежей почты. Наконец, в строке 13 заменяем светло-серый цвет фона на роскошный темно-синий. (Не пугайтесь: украшательство в X предстоит большее, чем в этом примере.)

Следует отметить, что все X-клиенты, запущенные в строках 6–13, выполняются в фоновом режиме (на это указывает амперсанд в конце каждой строки). Если вы забудете поместить клиенты в фоновый режим, *xinit* запустит первый *xterm*, дождется завершения его работы, выполнит запуск следующего *xterm* и т. д. Наличие амперсанда заставляет клиенты работать одновременно.

В строке 16 запускается *fvwm* (версия 2) – оконный менеджер, используемый на многих системах Linux. Как уже отмечалось, менеджер окон обеспечивает возможность оформления окон, перемещения их мышью и т. д. Он запускается командой:

```
exec fvwm2
```

При этом процесс *fvwm2* замещает процесс *xinit*. Таким образом, завершив процесс *fvwm*,[1] вы завершите работу X-сервера. Это более лаконичный эквивалент комбинации клавиш <Ctrl>+<Alt>+<Backspace>. После завершения работы X-сервера вы вернетесь в командную строку или будет показано новое графическое окно регистрации в системе, что зависит от ее конфигурации.

Обычно следует добавлять амперсанд к строке запуска каждого X-клиента в *.xinitrc* и помещать строку *exec* для менеджера окон в конце файла. Конечно, есть и другие способы, но многие пользователи применяют эту технику.

Менеджер окон *fvwm2* долгое время использовался по умолчанию в большинстве дистрибутивов Linux. Он все еще в ходу, но в основном его вытеснили очень развитый менеджер окон *kwin*, поставляемый с KDE, и более современные менеджеры окон типа *blackbox* и *sawfish*.

В страницах руководства по *xterm* и другим X-клиентам можно найти значительно больше параметров командной строки. Как мы говорили, в X практически все можно сконфигурировать. *fvwm* (версия 2) использует собственный файл конфигурации *.fvwm2rc*, для которого есть страница руководства. (Если у вас нет файла *.fvwm2rc*, вместо него по умолчанию будет использован системный файл */usr/lib/X11/fvwm2/system.fvwmrc*.) Более новые менеджеры окон по-прежнему используют файлы конфигурации, но обычно предоставляют программы настройки с графическим интерфейсом, позволяющие выбрать установки, а затем сохранить их в файле конфигурации в правильном формате. Подробную информацию по настройке отдельных клиентов можно найти на страницах руководства, а также в книгах по использованию X (таких как «X Window System User's Guide», O'Reilly).

---

[1] Если вы поработаете с *fvwm*, то обнаружите, что щелчок левой кнопкой мыши в области фона экрана вызывает появление всплывающего меню. Чтобы завершить *fvwm*, выберите в этом меню опцию *Quit fvwm*.

K Desktop Environment

K Desktop Environment – это проект open source, призванный предоставить удобный, полнофункциональный, современный и дружественный рабочий стол для Unix и, следовательно, для Linux. Со времени начала проекта в октябре 1996 года были достигнуты значительные успехи. Частично этот прогресс обязан выбору очень высококачественного средства разработки Qt, а также последовательному использованию C++ и его объектно-ориентированных возможностей.

Сразу следует заметить, что KDE *не* является менеджером окон, как *fvwm*, а представляет собой целую систему рабочего окружения, которая может быть использована с любым менеджером окон. Однако в состав KDE входит свой собственный оконный менеджер, именуемый *kwin*, который дает наилучшие результаты, и поэтому здесь мы будем говорить именно о нем.

Текущая версия KDE, так же как и готовящийся набор офисных приложений для KDE (см. *http://koffice.kde.org*), во многом основаны на KParts – компонентной технологии, которая в числе прочего позволяет встраивать офисные компоненты, например, средство просмотра файлов PDF – в веб-броузер, чтобы беспрепятственно просматривать загруженные PDF-файлы.

KDE постоянно развивается, и каждые несколько месяцев команда разработчиков KDE выпускает так называемый официальный релиз, который считается устойчивым и подходящим для конечных пользователей. Эти релизы доступны и в скомпилированном виде, и в виде исходных текстов. Скомпилированные версии поставляются в различных форматах, часто специально адаптированных для основных дистрибутивов Linux. Но если вам не терпится поработать с самой свежей версией KDE и вас не пугают эпизодически обнаруживающиеся ошибки, можно скачать ежедневно обновляющийся вариант KDE, но это развлечение не для слабонервных. Когда писалась эта книга, текущей стабильной версией была 3.0.2, а выход 3.0.3 был не за горами.

Общие характеристики

Мы уже несколько раз отмечали, что настройка менеджеров окон и приложений X долгое время предполагала изучение меняющегося синтаксиса файлов конфигурации и редактирование этих файлов, что для опытных пользователей Linux считается само собой разумеющимся, однако новичков часто отпугивает. Поэтому одна из задач, стоявших перед командой разработчиков KDE, заключалась в реализации возможности конфигурирования всего, что может быть сконфигурировано в KDE, через графические окна диалогов. При желании вы можете, как и прежде, редактировать файлы конфигурации, но это не обязательно. Даже самые опытные пользователи обычно соглашаются с тем, что для того чтобы выполнить такое простое действие, как изменение цвета фона, быстрее все-таки нажать несколько кнопок, чем искать в руководстве синтаксис команды для задания цвета фона, открывать файл конфигурации, редактировать его и перезапускать менеджер окон.

Помимо легкости в конфигурировании KDE предоставляет некоторые другие возможности, о которых ранее в Linux никто не слышал. Например, система полностью интегрирует в рабочий стол доступ в Интернет. При этом менеджер файлов выступает в роли веб-броузера, и просмотр файлов на каком-либо FTP-сайте почти не отличается от навигации по файлам на локальном диске. Можно перетаскивать на рабочий стол значки, представляющие адрес интернет-ресурса, и обращаться к ним позже. KDE интегрирует с рабочим столом поисковые механизмы и другие ресурсы Интернета и даже позволяет с легкостью определить свои предпочтительные поисковые механизмы и ссылки в Интернете. Кроме того, почти все приложения KDE способны открывать и сохранять файлы на удаленных системах.

Функция drag-and-drop, обычная в Windows или Macintosh, также широко применяется в KDE. Например, чтобы открыть файл в текстовом редакторе, можно найти его значок в окне менеджера файлов и перетащить в окно редактора. При этом не важно, где находится файл. Если он расположен на удаленном сервере, KDE автоматически скачает файл для работы с ним в текстовом редакторе или в любом другом приложении, которое вам потребуется. То же относится к мультимедийным файлам. Щелкнув по значку MP3-файла на удаленном сервере, можно загрузить его в фоновом режиме и воспроизвести на локальной машине.

Хотя страницы руководства очень хорошо подходят для быстрого доступа к краткой информации о системных библиотеках, которыми обычно пользуются программисты, они не очень удобны для документации, предназначенной конечному пользователю. Поэтому KDE использует стандартные HTML-файлы и поставляется с программой просмотра справки. Эта программа позволяет просматривать также страницы руководства man и файлы Info, так что вы можете обращаться ко всей документации в вашей системе из одного приложения. Кроме того, большинство приложений KDE поддерживают контекстную справку.

В последние версии X Window включена возможность, называющаяся *управлением сеансом*. Эта возможность позволяет выходить из среды X (например, при выключении или перезагрузке компьютера), а по возращении в X все ранее открытые приложения, которые поддерживают управление сеансом, появляются вновь на своих местах и в той же конфигурации. К сожалению, эта очень удобная возможность редко поддерживалась X-приложениями, но зато KDE использует ее весьма интенсивно. KDE предоставляет менеджер, управляющий сеансом, и все приложения KDE пишутся с учетом этой возможности.

В KDE включен свой менеджер окон *kwin*, причем отличный. Но это лишь часть KDE. Помимо менеджера окон в состав KDE входят файловый менеджер, веб-броузер, панель, пейджер, центр управления настройками рабочего стола и многое другое. При желании можно даже запустить KDE с другим оконным менеджером, но при этом потеряются некоторые из возможностей интеграции. Кроме того, вместе с KDE поставляется тьма приложений, от полного комплекта офисных программ до средств просмотра PostScript и PDF, мультимедийных программ и игр.

Вы можете подумать: «Это все звучит красиво, но у меня есть пара обычных приложений X, которые я хочу запускать». В этом случае вам будет приятно услышать, что такая возможность у вас сохраняется. Все ваши приложения X можно запускать на рабочем столе KDE, при этом KDE даже предоставляет, насколько это возможно, некоторые средства для интегрирования их в общую рабочую среду. Например, по вашему желанию KDE может попытаться перенастроить ваши приложения на использование тех же цветов, которыми пользуется весь рабочий стол. Конечно, приложения, не поддерживающие KDE, не поддерживают и некоторые из продвинутых возможностей KDE, такие как drag-and-drop или управление сеансом, но вы можете продолжать пользоваться привычными программами, пока не появятся какие-нибудь приложения KDE, выполняющие такие же функции (или, возможно, KDE-версии ваших любимых программ).

Установка KDE

Большинство дистрибутивов Linux сегодня поставляется вместе с KDE, но если у вас не такой дистрибутив или вам нужна более новая версия KDE, можно скачать ее из Интернета. Все связанное с KDE, включая документацию, снимки экранов и адреса для загрузки программ, вы найдете на *http://www.kde.org*. Поскольку FTP-сайт проекта KDE *ftp://ftp.kde.org* часто перегружен, может быть удобнее попробовать обратиться к зеркалам.

В состав KDE входят следующие пакеты:

kdesupport

Этот пакет содержит библиотеки других производителей, которые не являются частью KDE, но которые KDE использует. Рекомендуется установить этот пакет, чтобы гарантировать установку корректных версий всех библиотек.

aRts

aRts (от real time-sequencer) составляет основу большинства мультимедийных приложений KDE.

kdelibs

Пакет содержит библиотеки KDE. Сюда входит основной каркас приложения, ряд графических элементов GUI, система конфигурации, система работы с HTML и многие другие вещи. Без этого пакета KDE работать не будет.

kdebase

Здесь находятся основные приложения KDE, которые превращают обычный рабочий стол в рабочий стол KDE, включая файловый менеджер/веб-броузер, менеджер окон и панель. Этот пакет определенно необходим для использования KDE.

kdegames

Пакет содержит игры, включая карточные, стратегические и прочие. Этот пакет, возможно, захотят установить многие (исключительно для знакомства с системой, конечно).

kdegraphics

Программы, связанные с графикой, такие как программы просмотра файлов dvi, PostScript и редактор пиктограмм.

kdeutils

Этот пакет содержит некоторые рабочие средства, такие как текстовые редакторы, калькулятор, менеджер печати и адресную книгу.

kdemultimedia

Пакет содержит программы для работы с мультимедиа, такие как CD-плеер, MIDI-плеер и даже Karaoke-плеер.

kdenetwork

Здесь вы найдете программы для работы в Интернете, включая программу чтения почты, телеконференций и некоторые средства управления сетью.

kdeadmin

Этот пакет содержит некоторые программы для системного администратора, включая программы управления пользователями, редактор уровней исполнения и программу для резервного копирования.

kdepim

В этот пакет входит такое примечательное приложение, как *korganizer*, представляющее собой полнофункциональный менеджер персональной информации, который поддерживает даже синхронизацию с Palm Pilot.

kdeedu

Как подсказывает название, этот пакет содержит ряд образовательных программ – от средств, помогающих заучивать слова, до программ, рассказывающих о движении звезд и планет.

koffice

Это законченный пакет полноценных офисных программ. Кое-где в нем встречаются шероховатости, но многие уже используют его в повседневной деятельности.

Цикл выпуска новых версий KOffice отделен от цикла обновления KDE. На момент написания книги текущая версия имела номер 1.1.2. Вся информация о KOffice есть на *http://koffice.kde.org*.

Кроме этих пакетов, которые официально поставляются командой разработчиков KDE, есть буквально сотни других программ для KDE. Текущий список доступных приложений находится по адресу: *http://www.kde.org/applications.html*.

Когда пакеты выбраны, их надо установить. Способ установки зависит от вашего дистрибутива Linux и от того, какой вариант вы предпочитаете: установку скомпилированного пакета или компиляцию из исходных кодов. Если ваш дистрибутив содержит KDE, можно также выбрать установку KDE при установке всей системы.

Когда программа уже загружена на жесткий диск, осталось выполнить всего несколько шагов. Сначала следует убедиться, что каталог, где содержатся

приложения KDE, включен в переменную окружения PATH. По умолчанию исполняемые программы KDE находятся в */opt/kde3/bin*, но если вы установили KDE в другое место, то необходимо указать соответствующий путь в переменной окружения PATH.[1] Добавить каталог в переменную PATH можно командой:

```
export PATH=/opt/kde3/bin:$PATH
```

Для того чтобы эта установка действовала постоянно, надо добавить эту строку в файл конфигурации *.bashrc* в вашем домашнем каталоге или в системный файл конфигурации */etc/profile*.

Затем то же самое нужно проделать в отношении каталога с библиотеками KDE (по умолчанию */opt/kde3/lib*) и переменной окружения LD_LIBRARY_PATH:

```
export LD_LIBRARY_PATH=/opt/kde3/lib:$LD_LIBRARY_PATH
```

Почти все сделано. Осталось указать X, что вы хотите запускать рабочий стол KDE при запуске X Window. Это делается в файле *.xinitrc* в вашем домашнем каталоге. Сначала лучше сделать резервную копию, а затем ввести вместо всего содержимого файла единственную строку:

```
exec startkde
```

startkde – это сценарий оболочки, поставляемый с KDE, который просто запускает менеджер окон *kwin* и ряд системных служб. Обычно дистрибутивы устанавливают более сложные файлы *.xinitrc*, из которых могут даже запускаться приложения и службы, не входящие в KDE.

Использование KDE

Пользоваться KDE весьма просто. Большинство вещей интуитивно понятны, поэтому часто можно просто догадаться, что надо сделать. Однако мы дадим вам несколько советов по использованию KDE, чтобы воодушевить вас на дальнейшее исследование рабочего стола KDE.

Панель KDE и K-меню

При первом запуске среды KDE она выглядит, как на рис. 11.2. Вдоль нижнего края экрана видна так называемая *панель*. Панель служит нескольким целям, включая быстрый доступ к установленным приложениям. Вдоль верхней границы разместилась *панель задач*. Здесь показаны все открытые окна, и ее можно использовать для быстрого доступа к запущенным приложениям. При первом запуске KDE также откроет программу настройки, которая позволит изменить начальные установки.

KDE предоставляет несколько рабочих экранов (workspaces), к которым можно переходить с помощью кнопок в середине панели; по умолчанию они помечены номерами от одного до восьми. Попробуйте нажать эти кнопки. Вы увидите, что открытые вами окна видны только на первом рабочем экране, а нижняя панель и панель задач присутствуют постоянно. Теперь попро-

[1] Некоторые дистрибутивы помещают программы KDE в другое место, например в */usr/bin*.

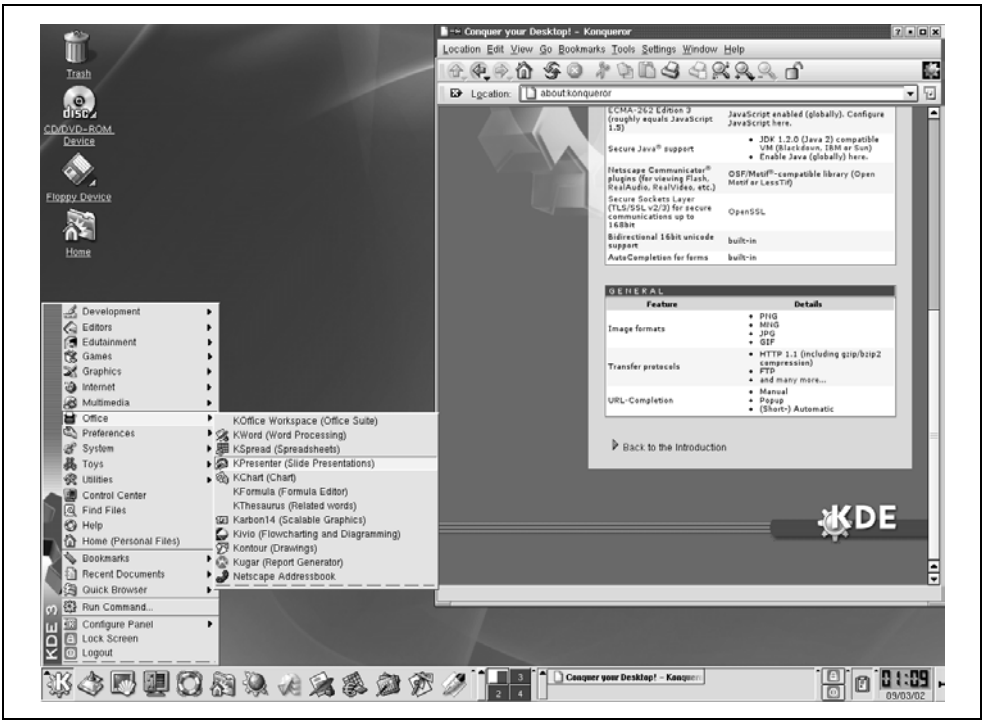


Рис. 11.2. *Рабочий стол KDE при первом запуске*

буйте, находясь в рабочем экране «2», запустить терминал, щелкнув по его пиктограмме на панели. После этого снова перейдите в другой рабочий экран. Вы обнаружите, что терминал виден только, когда вы находитесь в рабочем экране «2», но его значок виден на панели задач во всех рабочих экранах. Если находясь в любом другом рабочем экране, вы щелкните по значку терминала на панели задач, то немедленно перейдете обратно к экрану, где показывается терминал.

Другая удобная функция – это маленький значок с изображением кнопки в углу окна терминала. Нажмите ее и перейдите в другой экран. Терминал виден теперь в каждом рабочем экране – вы «прикрепили» его к фону рабочего стола.

Если такое прикрепленное окно вам надоело, нажмите кнопку снова и «отцепите» окно, а если вы хотите совсем его закрыть, нажмите кнопку в правом верхнем углу с маленьким крестиком.

Есть еще много вещей, которые можно проделывать с окнами в KDE, но сейчас мы перейдем к краткому введению в так называемое *K-меню*. Оно открывается щелчком по значку в левом углу панели, где нарисована буква K с шестеренкой. Помимо некоторых параметров для настройки самого K-меню и панели, вы найдете там все установленные приложения KDE, сгруппированные по подменю. Чтобы запустить одно из таких приложений, просто выберите элемент меню.

Мы обещали, что вы сможете запускать ваши старые приложения X на вашем рабочем столе KDE. Вы можете запустить их, открыв окно терминала и введя название приложения в командной строке либо нажав <Ctrl>+<F2> и введя название приложения в маленькой командной строке, появившейся в центре экрана. Однако немного поработав, вы сможете включить приложения, не являющиеся приложениями KDE, в K-меню и панель, в результате чего они будут представлены значками, щелчком по которым вы сможете запустить связанные с ними программы.

В зависимости от того, как вы установили KDE, очень может быть, что среди элементов меню уже есть подменю Non-KDE programs, содержащее приложения, не поддерживающие KDE. Если этого подменю нет, запустите приложение *KAppfinder*, находящееся в подменю System. Эта программа ищет в вашей системе приложения, которые входят в ее базу данных, и интегрирует их в рабочий стол KDE, создавая для каждого приложения файл *.desktop*. Если нужной программы нет в базе данных *KAppfinder*, вам придется самостоятельно написать такой файл *.desktop*, однако для этого, как, впрочем, и всегда в KDE, вам будет нужно лишь ввести требуемую информацию в ряде диалоговых окон. См. документацию KDE на *http://www.kde.org/documentation/index.html*.

По умолчанию панель уже содержит значки для запуска наиболее часто используемых программ, но можно легко добавить собственные. Для этого надо снова открыть K-меню и выбрать подменю Configure Panel→Add→Button. Появится копия K-меню. Найдите приложение, значок которого вы хотите добавить на панель, и выберите его так, как если бы вы хотели его запустить. KDE добавит на панель значок этого приложения. Можно даже добавлять на панель целые подменю, выбирая в дереве Add/Button первую запись в подменю. В этом случае к значку добавляется небольшая черная стрелка, указывающая на то, что щелчок по значку откроет другое меню, а не запустит приложение.

Пространство на панели не бесконечно, поэтому может возникнуть необходимость удалить некоторые значки программ, которыми вы не часто пользуетесь. Щелкните правой кнопкой мыши по значку и выберите Remove. При этом удаляется не сама программа, а ее значок. Вообще, в KDE щелчок правой кнопкой мыши обеспечивает доступ к множеству функций.

Центр управления KDE

Теперь мы покажем, как настроить рабочий стол KDE по своему вкусу. Как мы обещали, для этого не потребуется редактировать никакие файлы конфигурации.

Настройка производится через центр управления KDE (Control Center), который запускается из K-меню. Все параметры настройки сгруппированы по уровням. Начав работу с центром управления, вы увидите группы верхнего уровня. Нажав значок «+», можно открыть группу и обратиться к ее элементам.

Настройка фона. Для примера мы поменяем цвет фона. Для этого надо открыть группу Look&Feel и выбрать Background. Появится окно для настройки фона (рис. 11.3).

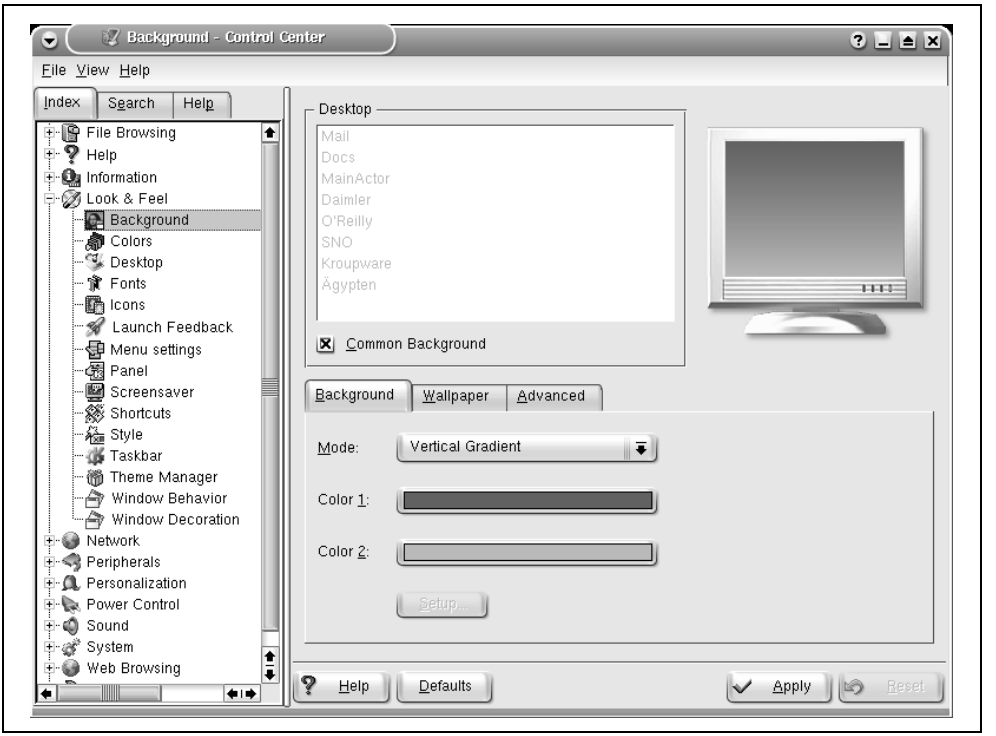


Рис. 11.3. *Настройка фона для рабочего стола KDE*

Можно выбрать одноцветный фон, двухцветный фон и градиент для плавного перехода одного цвета в другой, «обои» (готовые или из вашей собственной графики) и эффект слияния, в котором объединены несколько вариантов. Чтобы выбрать цвет, щелкните по кнопке Color 1 или Color 2. Появится диалоговое окно, где можно выбрать цвет по своему вкусу. Когда вы закроете это окно, новый цвет будет показан в области предварительного просмотра в верхнем правом углу окна настройки. При настройке KDE вы часто будете встречать такую возможность предварительного просмотра вносимых изменений. Однако можно посмотреть выбранные настройки и на полном экране, для чего нужно нажать кнопку Apply в нижней части окна конфигурации, и изменения будут немедленно автоматически применены. При этом перегружать рабочий стол не надо.

Если вы предпочитаете одноцветный экран, выберите Flat в выпадающем списке Mode. Вы увидите, что кнопка Color 2 стала неактивной; нужный цвет выберите кнопкой Color 1.

Можно также выбрать фоновый рисунок на вкладке Wallpaper. Вместе с KDE поставляется много фоновых рисунков, но вы можете выбрать собственную картинку (например, фотографию своей семьи). Можно даже выбрать несколько рисунков, которые будут меняться с заданной периодичностью. Наконец, на вкладке Advanced вы можете выбрать различные способы смешива-

ния. Их трудно описать словами, поэтому лучше попробовать самостоятельно, наблюдая в маленьком окошке, какой эффект достигается. Настраивать фон можно и дальше, но сейчас мы займемся настройкой стилей и цвета окон.

Настройка стилей и цвета окон. В обычных менеджерах окон настраивается только цветовое оформление окна, но не его содержимого. KDE – это не обычный менеджер окон, а интегрированный рабочий стол, поэтому цвет и другие установки применяются и к оформлению окна, контролируемому оконным менеджером, и к содержимому окна, которое контролируется приложениями. Сейчас мы немного поработаем над их внешним видом.

В центре управления откройте группу Look&Feel и выберите меню Colors. Вы увидите окно предварительного просмотра и список, в котором можно выбрать цветовую схему. KDE работает не через настройку отдельных цветов, а через определение так называемых цветовых схем. Это делается потому, что изменять только один цвет бессмысленно: все цвета должны сочетаться друг с другом, чтобы на них можно было смотреть без содрогания.

KDE позволяет создавать собственные цветовые схемы, но это требует некоторых знаний из области психологии человеческого зрения. Поэтому мы предлагаем выбрать одну из имеющихся цветовых схем. После того как вы оценили схему в окне предварительного просмотра, нажмите кнопку Apply и посмотрите, как все запущенные приложения вздрогнут и внезапно изменят цвет – без всякой перезагрузки. Для пользователей Windows в этом нет ничего необычного, а в Unix до появления KDE ничего подобного не было.

Такие же функции применяются и в отношении других установок. Например, откройте группу Look&Feel и далее Style. Вам представится большой выбор так называемых «стилей». Стиль определяет способ отображения элементов интерфейса пользователя, например, как в Windows (стиль Qt Windows), как в Motif (стиль Qt Motif), как в RISC OS (стиль RISC OS), или что-нибудь оригинальное, скажем «Light». Изменив установку, можно нажать кнопку Apply и посмотреть, как все запущенные приложения изменят свой стиль. Кстати, то же самое относится и к шрифтам, которые можно выбрать на странице Fonts.

Интернационализация. Мы не можем описывать все, что допускает настройку в KDE, ибо тогда в книге не осталось бы места для остальных тем. Но еще одну из настроек мы не можем обойти вниманием. Она вам особенно понравится, если английский язык не является вашим родным языком или если вы часто пользуетесь другими языками.

Перейдите на страницу Country & Language в группе Personalization (рис. 11.4). Здесь можно выбрать национальные настройки и язык, которые будут использоваться рабочим столом KDE и приложениями KDE. На текущий момент KDE позволяет выбирать более чем из 80 национальных настроек и языков. Учтите, что для выбора конкретного языка должен быть установлен модуль языка. Эти модули можно загрузить с FTP-сервера KDE (как описано выше) или установить с носителя дистрибутива.

Вы можете удивиться, зачем может понадобиться выбирать более одного языка. Причина в том, что программы KDE переводятся добровольцами.

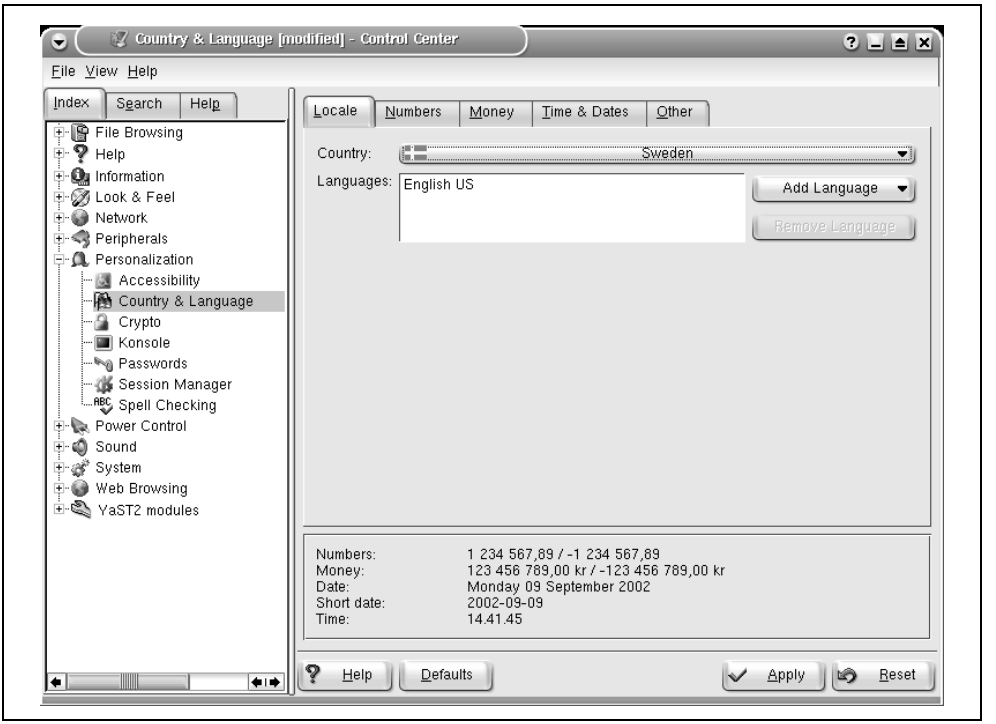


Рис. 11.4. *Настройка языка для рабочего стола KDE*

При этом не все приложения переводятся одновременно, а некоторые и вовсе не переведены. Поэтому какое-либо приложение может оказаться недоступным на том языке, который выбран в качестве основного (верхнего в списке Language). В этом случае для него автоматически используется второй из выбранных языков, а в случае неудачи будет выбран третий и т. д. Если ни с одним из этих трех языков приложение работать не может, будет выбран английский, который всегда присутствует в системе.

Можно еще долго говорить об удобствах рабочего стола KDE, но лучше начать самостоятельное изучение. Кроме очевидных и интуитивных функций есть и менее очевидные, но, тем не менее, очень полезные возможности, поэтому не стесняйтесь и почитайте документацию на *http://www.kde.org/documentation/index.html.*

Приложения KDE

Для KDE существуют тысячи программ – от базовых утилит, таких как *konsole* (эмулятор терминала) и *OClock* (часы), до редакторов, средств разработки приложений, игр и мультимедийных приложений. В нашей книге мы можем охватить лишь малую часть из того множества программного обеспечения, которое доступно для KDE. В этом разделе мы расскажем о тех приложениях, пользоваться которыми должны уметь все пользователи KDE. Это

не обязательно самые замечательные и впечатляющие программы, но их определенно следует иметь в своем арсенале рабочих приложений. Кроме того, нужно помнить, что если для решаемой вами задачи нет KDE-программы, можно воспользоваться классическим X-приложением, если таковое имеется. Оно не будет выглядеть так изящно и так хорошо интегрироваться, но все равно будет функционировать на рабочем столе KDE.

Кроме того, не забывайте, что приложений KDE значительно больше, чем мы здесь сможем перечислить. Некоторые из них, такие как KWord (процессор слов) или KMail (почтовый агент пользователя), встретятся вам в других местах книги. Однако другим, таким как менеджер персональной информации KOrganizer, места в книге не нашлось. Поэтому покопайтесь в архивах Linux, и вы найдете сотни замечательных KDE-программ.

konsole: ваше главное окно

Мы начнем наше исследование приложений X с рабочей лошадки – в терминале вы, возможно, проводите большую часть времени. Это просто окно, в котором содержится оболочка Unix. Оно выводит приглашение, принимает команды и прокручивается подобно терминалу.

Традиционно классическим эмулятором терминала Unix являлся *xterm*. В настольной среде KDE его заменил *konsole*.

Возможно, некоторые из вас усмехнутся, глядя на то, как, купив дорогой цветной монитор и установив мегабайты графического программного обеспечения, мы сидим перед эмулятором старого терминала VT100. Но не надо забывать, что это Linux, а не просто операционная система типа «укажи и щелкни». Здесь есть и замечательные графические приложения, но вам наверняка придется уделять много времени работе с текстами, а интерфейс командной строки – это самое мощное средство для таких задач.

Итак, посмотрим на окно *konsole* (рис. 11.5), содержащее несколько команд.

Рис. 11.5. *Окно konsole*

Запуск konsole

Запустить *konsole* можно несколькими способами, как и все программы KDE:

- Запустить из панели, если там есть значок *konsole*. В большинстве терминалов он устанавливается по умолчанию.
- Выбрать в К-меню, где *konsole* можно найти в Utilities/System/Konsole.
- Набрать <Alt>+<F2> и в открывшемся маленьком окне команд ввести «konsole».
- Если *konsole* уже открыта, можно ввести в ней «konsole» и получить еще одну консоль.

При открытии окна *konsole* появляется окно Tip of the Day, в котором даются полезные советы относительно *konsole*. Его можно отключить, но мы советуем на некоторое время его оставить, чтобы узнать некоторые полезные вещи. Можно просмотреть все советы, многократно щелкая по кнопке Next в этом окне. У многих приложений KDE есть такое же окно «Tip of the Day».

konsole позволяет запускать несколько сеансов в одном окне *konsole*. Новый сеанс можно запустить, просто выбрав его тип в меню Session или щелкнув по кнопке New в панели инструментов. После этого можно переключаться между сеансами с помощью панели инструментов или меню View. Если вы не видите панели инструментов, выберите в меню пункт Settings/Show Toolbar.

Операции удаления и вставки

На самом деле *konsole* предлагает значительно больше, чем терминал VT100. Одной из возможностей является мощная операция удаления и вставки.

Взглянем еще раз на рис. 11.5. Допустим, нам не нужен каталог *notes*; вместо этого мы хотим посмотреть на *~/perl/_example/for_web_site*.

Выберем сначала интересующую нас часть команды *cd*. Поместите курсор мыши слева от `c` в `cd`. Нажав левую кнопку мыши, перемещайте мышь, пока не будет выделен слэш после «example». Результат показан на рис. 11.6.

Рис. 11.6. *Выделенный текст в konsole*

Когда выделенная область охватывает нужное количество символов, нажмите среднюю кнопку мыши. *konsole* вставит выделенный текст в следующую командную строку (рис. 11.7). Теперь можно набрать оставшуюся часть пути к каталогу *for_website* и нажать клавишу <Enter> для выполнения команды.

Рис. 11.7. *Окно konsole после вставки текста*

В окне можно выбрать все что угодно: как ввод, так и результаты выполнения команд. Для выбора целых слов вместо символов нужно выполнить двойной щелчок левой кнопкой мыши. Целая строка выбирается тройным щелчком. Вы можете выбрать также и несколько строк. Это не требуется при вводе команд, но удобно, если используется редактор *vi* и надо часто переносить фрагменты текста между окнами.

Будьте внимательны: в случае длинной строки в выделенный текст попадет символ новой строки, даже если вы не нажимали <Enter> при вводе строки. И учтите, что если вам привычнее копировать текст путем буксировки (drag-and-drop), то *konsole* поддерживает и такой стиль.

Дополнительные приемы работы с konsole

Очень многие характеристики *konsole* можно настраивать. Можно выбирать шрифты, цветовые схемы, показывать полосу прокрутки слева или справа или вообще убрать ее и т. д. К наиболее часто используемым настройкам можно обращаться через меню Settings, а если вы не можете найти требуемое, перейдите в Settings/Configure Konsole. Там можно установить интервал между строками, мерцание курсора и т. д.

Очень полезна такая функция *konsole*, как возможность сторожить вывод или его отсутствие в одном из сеансов. Для этого нужно включить демон вывода в уже упомянутом окне настроек на странице Write Daemon.

Когда можно использовать *konsole* watcher (сторож)? Представьте себе, что вы разрабатываете большую программу, которая долго компилируется. Непрограммисты могут представить себе загрузку большого файла в окне терминала с помощью *wget* или расчет сложного образа POVRAY. Пока выпол-

няется компиляция, хотелось бы заняться чем-то другим (в конце концов, для чего существуют многозадачные операционные системы?), скажем, начать составление письма в почтовом клиенте KDE. Обычно вы должны время от времени переключаться в окно консоли и проверять, не закончилась ли компиляция, а затем продолжать разработку программы. С помощью сторожа можно получать зрительное или звуковое уведомление по завершении компиляции. Просто переключитесь в сеанс, за которым вы хотите следить, и выберите View/Monitor for Silence. Вы получите уведомление, как только компилятор в течение некоторого времени не будет выдавать сообщения, и это переключит ваше внимание от почтового клиента к окну *konsole*. Конечно, можно следить и за появлением вывода, а не за его прекращением, например, при длительной сетевой операции, которая не выводит индикатор хода процесса.

Часы

Разве может ваш экран быть полным, если его не украшают небольшие часы, показывающие, сколько времени вы убили, настраивая внешний вид вашего экрана? Часы могут быть самыми разными: квадратными или круглыми, со стрелкой или электронными, маленькими или большими. Вы даже можете заставить их бить в куранты.

В KDE есть часы нескольких типов, но обычно запускают небольшое приложение на панели, поскольку, независимо от разрешения, место на экране всегда в цене. По умолчанию часы появляются в правом нижнем углу экрана, в пределах панели (это называется панельным апплетом, т. е. маленьким приложением, выполняемым внутри панели). Если дистрибутив создал другие настройки, можно щелкнуть правой кнопкой в любом месте фона панели и выбрать в меню Panel→Add→Applet→Clock, в результате чего часы появятся на панели. Если вы хотите поместить часы в другом месте панели, щелкните правой кнопкой на маленькой полосатой ручке слева от часов, выберите Move в появившемся контекстном меню и переместите часы мышью в нужное положение. Остальные объекты панели автоматически освободят место для часов.

У апплета часов есть несколько режимов, которые устанавливаются путем щелчка правой кнопкой по самим часам и выбора Type, а затем нужного режима в контекстном меню. Есть простые, цифровые, аналоговые и, что примечательно, нечеткие (fuzzy) часы. Нечеткие часы – для тех, кто не любит, чтобы его подгоняло время. Например, если запустить «нечеткие» часы, они покажут *Middle of the week* (*середина недели*). Если это покажется слишком нечетким, можно выбрать в контекстном меню часов Preferences/Fuzzy Clock и установить степень нечеткости. Например, я ввожу этот текст в 9:53 A.M. в четверг, и четыре степени нечеткости дадут «без десяти пять», «десять часов», «почти полдень» и уже упомянутую «середину недели».

Приложение часов позволяет также настроить формат показа даты и времени, а также установить системное время (для этого потребуются права root; если вы зарегистрированы как обычный пользователь, появится окно с просьбой ввести пароль root).

Наконец, если вы предпочитаете видеть на экране обычные большие часы, можете запустить KDE-версию классических «OClock» через пункт K-меню Utilities/OClock.

KGhostview: просмотр файлов PostScript

Стандарт Adobe PostScript стал одним из самых популярных форматов для обмена документами в компьютерном мире. В этом формате распространяется множество научных работ. Руководства из Linux Documentation Project тоже распространяются в виде PostScript среди прочих форматов. Этот формат особенно удобен для тех, у кого нет времени на форматирование или достаточная полоса пропускания сети для передачи огромных файлов. Создавая собственные документы с помощью *groff* или TEX, вы, наверное, захотите увидеть их на экране, прежде чем портить дорогостоящую бумагу, распечатывая их.

KGhostview, приложение KDE, предлагает приятную среду для просмотра файлов PostScript в X Window System, в которой помимо файлов PostScript можно также просматривать файлы в формате Adobe PDF. KGhostview в действительности является лишь удобным интерфейсом к более старому приложению Ghostview, поэтому описываемые здесь функции можно осуществлять и с помощью Ghostview. Однако пользователь значительно комфортнее чувствует себя в KGhostview, поэтому мы здесь и опишем эту программу.

Программа KGhostview очень проста; она вызывается с именем файла, который надо просмотреть, например:

```
eggplant$ kghostview article.ps
```

или можно просто щелкнуть по значку любого файла PostScript или PDF в каком-либо месте KDE.

Поскольку сейчас мы интересуемся только просмотром имеющихся файлов, нас не должны беспокоить преимущества самих PostScript и PDF. Оба могут считаться стандартами для многих программ, способных записывать файлы в этих форматах (а некоторые могут и читать); оба созданы одной компанией, Adobe Systems. PDF несколько более переносим между платформами и самодостаточен, поскольку может даже содержать шрифты, необходимые для показа документа. Кроме того, PDF лучше известен в Microsoft Windows и Macintosh, поэтому в Интернете у вас больше шансов встретить файлы PDF, а не файлы PostScript. И наконец, PostScript в действительности предназначен для печати, тогда как в PDF есть некоторые функции для интерактивного просмотра, такие как значки страниц, гиперссылки и т. п.

KGhostview не является совершенным средством просмотра PDF, хотя для большинства документов его достаточно. Если возникают проблемы с конкретным документом, можно попробовать воспользоваться Adobe Acrobat Reader (не являющимся свободно распространяемым программным обеспечением, но допускающим бесплатную загрузку с *www.adobe.com*) либо утилитой *xpdf*, которая, возможно, входит в состав вашего дистрибутива.

Окно KGhostview огромно; оно может легко занять большую часть вашего экрана. Первая страница документа при необходимости имеет полосы прокрутки. Как и в большинстве программ KDE, имеются панели меню и инструментов, а также средства прокрутки страниц и список страниц в левой части окна.

Как и большинство приложений X, KGhostview предлагает для основных функций и опции меню, и быстрые клавиши. Таким образом, чтобы перейти к следующей странице, можно выбрать в меню View пункт Next Page. Либо можно просто нажать клавишу PgDn (или клавишу <пробел> в отсутствие PgDn, как на лэптопе).[1]

Вернуться к предыдущей странице можно, выбрав Previous Page в меню View. Для перехода к произвольной странице надо левой кнопкой мыши нажать соответствующую цифру в колонке Page Number. Для выхода выберите Quit из меню File или просто нажмите <Ctrl>+<q>.

В разных странах используются различные размеры страниц. Ghostview по умолчанию использует американский стандарт letter (его можно закомментировать в файле PostScript, что часто делается инструментами PostScript в дистрибутивах Linux, настроенных для работы в Европе). Вы можете указать другой размер в подменю Paper Size меню View.

Ghostview позволяет увеличивать или уменьшать размер страницы. Это очень полезная функция для детального просмотра результатов форматирования. (Но учтите, что экранные шрифты отличаются от шрифтов в принтере, и поэтому расположение символов на экране Ghostview не будет точно таким же, как на бумажной копии.) Для увеличения некоторой части страницы нажмите <Ctrl>+<+>; для уменьшения используйте <Ctrl>+<->. Можно также воспользоваться кнопками панели инструментов и пунктами Zoom In/ Zoom Out в меню View.

Можно настроить размер окна, чтобы оно точно соответствовало ширине страницы документа, выбрав Fit To Page Width в меню View. Для распечатки страницы выберите Print в меню File или нажмите <Ctrl>+ <P> в любом месте окна. Появится стандартное окно печати KDE, которое помимо прочего позволяет выбрать принтер.

Можно также распечатать только текущую страницу или диапазон страниц; достаточно задать свой выбор в диалоговом окне печати. Можно соединить это с функцией PageMarks. Меню PageMarks позволяет помечать и снимать отметки с отдельных страниц или их групп. Помеченные страницы показываются с маленьким красным флажком в списке страниц. Если есть помеченные страницы и выбирается функция печати, во всплывающем диалоговом окне автоматически заполняется список печатаемых страниц. Конечно, его можно изменить, прежде чем отправить документ на печать.

---

[1] Есть тонкое различие между клавишами <пробел> и PgDn: клавиша PgDn всегда переносит вас на следующую страницу, тогда как клавиша <пробел> сначала переносит в нижнюю часть текущей страницы, если размеров окна недостаточно для показа на экране целой страницы. Второе нажатие на <пробел> переводит на следующую страницу.

Чтение документации с помощью Konqueror

Konqueror – не только высококлассный веб-броузер и файловый менеджер, но и выполняет функцию чтения документации. Документация KDE поставляется в формате HTML, однако Konqueror способен отображать документацию и в другом формате, например Info и страницы руководства man. Например, чтобы показать страницу руководства для команды *ls*, откройте мини-окно командной строки, нажав <Alt>+<F2> и введя:

```
man:ls
```

KDE поймет, что вы хотите прочесть страницу руководства для команды *ls*, откроет окно Konqueror и покажет страницу руководства. Результат будет гораздо красивее отформатирован, чем это сделала бы команда man (или ее аналог в X11 *xman*).

Аналогичное происходит со страницами Info. Например, документация для компилятора GNU C *gcc* поставляется в формате info. Просто введите:

```
info:gcc
```

в командной мини-строке или строке ввода URL Konqueror, и появится требуемая страница Info (если, конечно, она установлена). Если ранее вас раздражала действительно неудобная программа командной строки *info* и программы типа *xinfo* тоже не радовали, данная возможность окажется благодеянием.

Но Konqueror этим не ограничивается, когда нужно получить информацию. Хотите воспользоваться поисковым механизмом в Интернете? Чтобы найти страницы, касающиеся Tux (талисман Linux), например, на AltaVista, просто введите в командной мини-строке или строке ввода URL Konqueror:

```
av:tux
```

и появится окно Konqueror с 1 319 135 (на момент написания этих строк) результатами поиска. Можно воспользоваться и другими поисковыми механизмами. Взгляните на некоторые наиболее популярные поисковые механизмы и их префиксы.

Таблица 11.1. Популярные поисковые механизмы и их префиксы

Поисковый механизм	Префикс
AltaVista	av:
Lycos	ly:
SourceForge	sf:
Excite	ex:
Google	gg:

Если ваш любимый поисковый механизм не включен в конфигурацию (что в действительности весьма маловероятно), можно сделать это самостоятельно, открыв окно Konqueror и выбрав Settings→Configure Konqueror→Enhanced Browsing.

Раздел Enable Web Shortcuts на этой странице конфигурации содержит все заранее настроенные поисковые механизмы и позволяет добавлять собственные.

Настольная среда GNOME

Настольная среда GNOME была задумана в 1999 как альтернатива KDE, уходя корнями в GPL и LGPL. Как и в случае KDE, задачей GNOME было предоставить современные простые в использовании приложения, взаимодействующие друг с другом и с существующими приложениями X.

Мы собираемся познакомить вас с GNOME и ее обычным внешним видом, но следует учитывать, что это универсальная структура с практически неограниченной гибкостью. Например, мы показываем текущий менеджер окон Sawfish, но можно установить другой менеджер окон с совершенно другим поведением и внешним видом. Как и библиотеки X, библиотеки GNOME лежат в основе всех компонентов и появились в командной строке и даже в выполняемых на сервере приложениях, как и в графической настольной среде. Кроме того, в проекте GNOME разработан ряд мощных приложений офисных , таких как электронные таблицы и адресные книги. Любое приложение X может выполняться в GNOME (хотя оно должно быть написано в среде GNOME, чтобы использовать мощнейшие функции настольной среды, такие как виртуальная файловая система и темы). В частности, многие приложения KDE прекрасно работают в GNOME, и наоборот.

Конечно, нас более всего интересуют основы настольной среды и связанные с ней приложения. В последующих разделах мы коснемся внешнего вида GNOME, поговорим о существующих в этой среде возможностях настройки, а затем кратко познакомимся с основными приложениями, такими как Evolution и Gnumeric. Но сначала необходимо установить или обновить нужное программное обеспечение.

Установка и обновление GNOME

GNOME входит в большинство дистрибутивов Linux, но если вы не устанавливали его или хотите получить более новую версию, можете взять исходный код на *http://gnome.org* или исполняемые файлы на *http://ximian.com*. Ximian предлагает готовый дистрибутив этой настольной среды для большинства распространенных дистрибутивов Linux. Для его установки нужно сделать следующее:

1. Открыть окно терминала.
2. С помощью команды `su` получить права `root`.
3. Выполнить команду `lynx -source http://go-gnome.com |sh`.

Эта команда загружает графическую программу установки, показанную на рис. 11.8. Следуйте инструкциям, показываемым на экране, и через несколько минут все будет установлено. Программа установки попросит вас выйти из системы, а когда вы снова зарегистрируетесь в ней, мастер на-

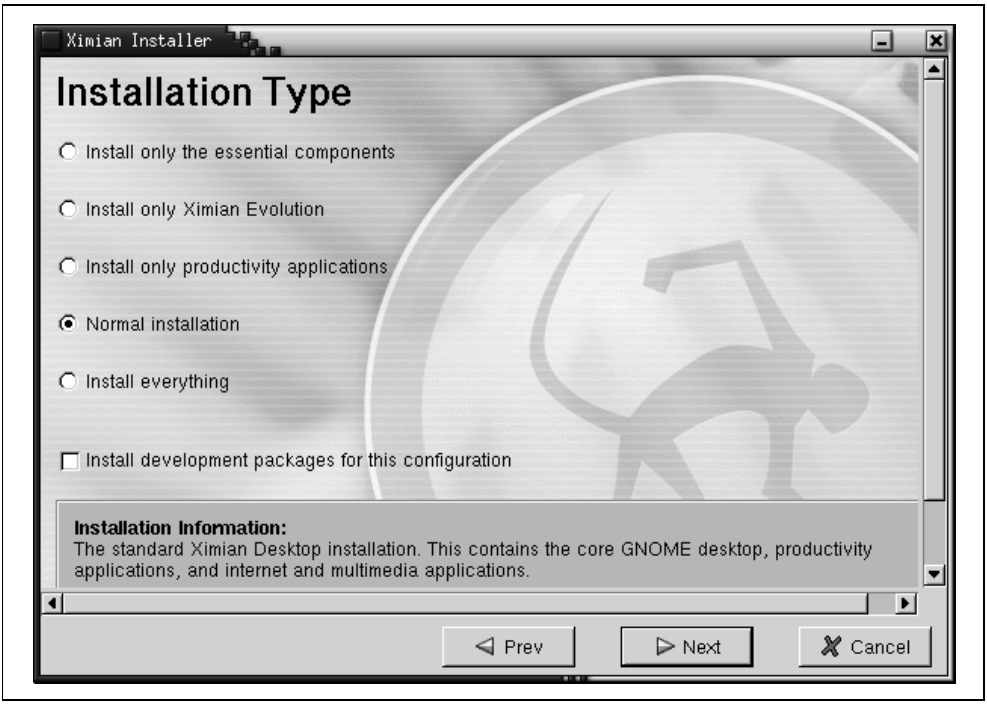


Рис. 11.8. Установка GNOME

стройки проведет вас через процедуру установки некоторых параметров вашего нового рабочего стола.

Обновление GNOME также выполняется легко: утилита Red Carpet (введите `red-carpet` в командной строке или щелкните System, а затем Get Software в панели меню GNOME) проверяет наличие обновлений для всех программ системы, включая GNOME, и предлагает их установить. Программное обеспечение делится на четкие каналы – один для дистрибутива, другой для настольной среды GNOME, еще один для дополнительных программ типа веб-броузера Opera или инструментов CodeWeavers WINE. Если вы подписываетесь на канал, Red Carpet будет проверять наличие обновлений для программного обеспечения из этого канала, которое есть в вашей системе. В каждом канале можно также добавлять или удалять программы с помощью кнопок Update, Install и Remove, расположенных в верхнем правом углу окон каналов.

Базовый интерфейс рабочего стола

Рабочий стол GNOME спроектирован так, чтобы быть понятным всякому, кто ранее имел дело с компьютерами. Хотя можно изменить практически любые настройки, при типичной установке появляются рабочий стол со значками и панели вдоль верхнего и нижнего краев экрана. Панели принад-

лежат к наиболее важным инструментам GNOME, поскольку они универсальны и предоставляют многообразные возможности взаимодействия с системой. Панели могут располагаться вдоль края экрана, как панель управления Windows; вдоль части его, как Macintosh Control Strip; в произвольном месте экрана, как стыкуемая панель в NeXT; использовать комбинацию стилей. Панели могут содержать меню, кнопки и маленькие приложения, например часы, списки окон, сетевые и системные мониторы и даже маленькие игры.

Некоторые характеристики имеют небольшие отличия от других графических интерфейсов. Например, можно иметь несколько рабочих пространств (экранов), что знакомо пользователям *fvwm*, но не часто встречается в других менеджерах экранов. Определенные архитектурные украшения есть в файловом менеджере Nautilus. По ходу дела мы расскажем о некоторых из них, но большинство достаточно невелики, и их можно исследовать самостоятельно.

Ниже следует краткое описание того, как выполнять стандартные задачи. Освоившись с ними, вы, вероятно, догадаетесь, как делать все остальное.

Перемещение элементов по рабочему столу

Щелкнуть и перетащить левой кнопкой мыши.

Перемещение элементов по панели

Щелчок и перетаскивание левой кнопкой мыши действует для запускающих объектов, но в некоторых апплетах левая кнопка используется для управления апплетом. В таком случае щелкните и перетаскивайте средней кнопкой. То же касается перемещения окон захватом границы – щелчок левой кнопкой раскрывает окно, но средней кнопкой – перемещает его.

Упорядочение элементов на рабочем столе

Щелкнуть правой кнопкой по фону рабочего стола и выбрать `Clean Up by Name`. Элементы будут упорядочены в алфавитном порядке, за двумя исключениями: первый элемент в левом верхнем углу всегда будет домашним каталогом пользователя, а последний элемент в списке – всегда мусорная корзина.

Открыть или активизировать элемент на рабочем столе

Дважды щелкнуть. Двойной щелчок по значку папки открывает ее в файловом менеджере Nautilus. Если дважды щелкнуть по документу электронной таблицы, запустится приложение электронной таблицы Gnumeric, в котором откроется документ.

Открыть или активизировать элемент на панели

Щелкнуть один раз левой кнопкой.

Получить список опций или задать свойства любого объекта

Щелкнуть правой кнопкой мыши, чтобы получить список опций для любого объекта. Например, можно изменить фон рабочего стола, щелкнув по нему правой кнопкой и выбрав `Change Desktop Background`. Другие общие

предпочтения можно задать через управляющий центр GNOME, который вызывается через System→Settings или Applications→Desktop Preferences или вводом `gnome-control-center` в командной строке.

Вставить текст в любую текстовую область

Сначала выделите текст, который нужно вставить. Затем щелкните средней кнопкой (если кнопок только две, нажмите обе, что эмулирует среднюю кнопку) в области, в которую вы хотите поместить текст.

Панель

Первоначальная конфигурация во многих системах задает узкую панель вверху и внизу экрана. На верхней панели есть ряд меню в левой части и несколько кнопок и часы – в правой. Нижняя панель содержит апплет списка окон, который должен показаться знакомым пользователям Microsoft Windows.

Для того чтобы создать новую панель, щелкните по свободному месту на имеющейся панели и выберите Panel→Create New Panel, затем выберите тип нужной панели. Чтобы изменить свойства панели, например размер и цвет, щелкните по ней правой кнопкой и выберите Properties (у панели меню вверху экрана нет доступных свойств; для нее установлены одно положение и размер). Поэкспериментируйте с разными видами панелей и различными размерами, чтобы узнать, какие вам нравятся больше всего. Если у вас маленький экран типа лэптоп, то, скорее всего, лучше подойдет меньший размер панели, чем при наличии большого экрана.

Для того чтобы добавить на панели кнопки запуска приложений, можно перетащить их из меню или щелкнуть по панели правой кнопкой и выбрать Panel→Add to Panel→Launcher. Затем нужно выбрать имя запускаемого приложения и пиктограмму для него. Можно также выбрать описание для кнопки, которое будет выводиться во всплывающем окошке при нахождении курсора мыши над значком в панели. Если вы хотите запускать приложение из терминала, установите флажок Run in Terminal.

Для получения дополнительной информации о панели щелкните правой кнопкой по свободному месту на ней и выберите Panel→Panel Manual.

Панельные апплеты – это маленькие приложения, выполняемые внутри панели. Поместить их на панель можно через меню Add to Panel или просто запустить, щелкнув Applications→Applets. Разновидностей панельных апплетов великое множество – от игр до утилит. Вот наиболее часто использующиеся:

CPU Load

График загрузки системных ресурсов в течение нескольких последних секунд.

Workspace Switcher

В большинстве установок этот апплет уже запущен, когда вы входите в систему, и обычно настроен на четыре рабочих пространства. Каждое рабочее пространство служит эквивалентом нового экрана рабочего стола;

их можно открыть в любом количестве. Переключатель рабочих пространств показывает все созданные вами виртуальные рабочие пространства и выводит каждое окно на рабочем столе в виде крошечного прямоугольника. С помощью левой кнопки мыши можно перетаскивать окно из одного рабочего пространства в другое. Чтобы изменить количество и расположение рабочих пространств, щелкните правой кнопкой и выберите пункт меню Properties.

Window List

Как и апплет рабочих пространств, список окон включается в большинство конфигураций. Он отображает открытые вами окна, чтобы можно было легко переходить из одного в другое, даже когда они минимизированы. Если у некоторого приложения несколько окон, они группируются под одной записью. Чтобы отключить эту функцию или установить другие опции апплета, щелкните правой кнопкой по Window List и выберите Properties.

Battery Charge Monitor

Монитор заряда аккумулятора показывает, сколько времени осталось работать источнику питания в портативных системах. Варианты отображения – график, процентное отношение и оценка оставшегося времени.

SlashApp

Показывает в панели заголовки Slashdot или веб-сайта новостей GNOME.

Eyes

Этот апплет показывает пару глаз, следящих за перемещением мыши. Польза его сомнительна, но он явно занятный.

Nautilus: менеджер рабочего стола и файлов

Nautilus – название менеджера рабочего стола и файлов в GNOME. Он управляет показом фонового рисунка и файлов на рабочем столе, позволяет обрабатывать файлы без участия терминала и следит за мусорной корзиной. Иными словами, это эквивалент Windows Explorer, Macintosh Finder и Konqueror из KDE для GNOME.

В большинстве случаев Nautilus работает, когда вы регистрируетесь в системе. Если нет (вы узнаете это по отсутствию значков на рабочем столе), можно запустить его из окна терминала, введя `nautilus`. Если вы решите, что Nautilus вам вообще не нужен, можете удалить его из сеанса с помощью утилиты Session Properties в Control Center.

Как и прочие приложения, Nautilus позволяет перетаскивать объекты с одного места на другое. Можно также копировать файлы с помощью <Ctrl>+<C>, вырезать с помощью <Ctrl>+<X> и вставлять с помощью <Ctrl>+<V>.

Быстрее всего начать работу с Nautilus, дважды щелкнув по значку домашнего каталога в левом верхнем углу рабочего стола, помеченную «your home». При этом откроется ваш домашний каталог.

Обычно файлы, расположенные на рабочем столе, хранятся в ~/*.gnome-desktop*. Однако, если вы пожелаете, чтобы на столе отображался ваш домашний каталог, выберите Preferences, Edit Preferences и в категории Windows and Desktop установите флажок Use your home folder as your desktop.

Nautilus очень похож на веб-броузер и может использоваться в качестве такового. В верхней части находится панель инструментов с кнопками, с помощью которых можно перемещаться по структуре каталогов: назад, вперед, вверх, обновить и в домашний каталог. Панель адреса, как в веб-броузере, описывает местоположение файла или каталога, которые вы рассматриваете в этом окне. Если ваше имя пользователя `jdoe` и вы щелкнете по значку Home, то увидите в панели адреса `/home/jdoe`.

Левая часть окна Nautilus кажется пустой. Однако если посмотреть на нижний левый край, обнаружатся закладки Notes, Tree, Help, History и News:

- Notes представляет собой виртуальный блокнот, в котором можно набросать любое краткое напоминание.
- Tree показывает дерево каталогов в системе – щелкните по треугольнику рядом с каталогом, и он развернется, показав свое содержимое. С помощью этого средства можно перемещаться по структуре каталогов или быстро перемещать элементы.
- Закладка Help предлагает указатель нескольких систем подсказки – страницы помощи для приложений GNOME и KDE, а также более традиционные страницы руководства и Info, поставляемые с приложениями командной строки для Linux.
- Закладка History поможет проследить ваши предыдущие действия. Не помните, где вы оставили этот файл? Можно узнать здесь.
- Закладка News показывает заголовки, получаемые с различных веб-сайтов. Можно выбрать источники новостей и частоту обновления через диалоговое окно Preferences.

Щелкните по закладке, и она расширится, заполнив всю панель.

У Nautilus есть ряд интересных дополнительных функций, которые вы не найдете в других приложениях, например:

- Вместо общих пиктограмм для графических файлов Nautilus использует уменьшенные картинки самих изображений. Благодаря этому легко организовывать каталоги графических образов, например, фотографий, полученных цифровой камерой.
- При нахождении курсора мыши над музыкальным файлом осуществляется его воспроизведение.
- Для текстовых файлов значок простого документа украшается фактическим текстовым содержанием файла. Благодаря этому можно вспомнить содержание файла, не открывая его, даже если имя файла не слишком содержательно.

- Пиктограмму файла можно растянуть, если щелкнуть по ней правой кнопкой и выбрать пункт Stretch Icon. Если растянуть значок текстового файла в достаточной мере, можно увидеть целиком его содержание и использовать как настольный блокнот.
- Любой графический файл можно перетащить на фон левой панели, и он станет использоваться как фон левой панели для этого каталога.
- Для большинства каталогов можно выбрать представление их содержимого в виде списка или группы значков. Однако некоторые файлы предоставляют более широкий спектр возможностей. Например, если каталог содержит звуковые файлы, можно выбрать View, а затем View as Music, и воспроизводить файлы в любом порядке – Nautilus даже покажет имя исполнителя, название и время воспроизведения отдельных файлов. То же относится к файлам HTML: Nautilus может показывать их исходный текст или как отформатированные веб-страницы.

В целом, Nautilus представляет собой универсальный инструмент, которым можно научиться пользоваться в результате небольшой практики. Дополнительную подсказку можно получить, выбрав Help и Nautilus User Manual из любого окна Nautilus.

Приложения GNOME

Освоившись с рабочим столом и основными операциями на нем, рассмотрим некоторые приложения, созданные для работы с ним. Обратите внимание, что использование этих приложений не ограничивается рабочим столом GNOME, и это не единственные приложения, которые можно выполнять на рабочем столе GNOME – просто они построены из одних и тех же материалов и исключительно хорошо взаимодействуют друг с другом.

Ximian Evolution: почта, календарь и контакты

Ximian Evolution – это то, что называют приложениями для рабочих групп (groupware suite): здесь объединены электронная почта, календарь и адресная книга, благодаря чему связь и планирование задач оказываются в одном удобном пакете. У нас нет места, чтобы подробно разбирать все три компоненты, но есть полное руководство, включенное в меню Help и доступное в сети на *http://support.ximian.com*.

Запустить Evolution можно через соответствующий пункт в меню Applications либо путем ввода `evolution` в командной строке. При этом должен появиться экран, подобный показанному на рис. 11.9.

При первом запуске Evolution программа просит создать учетную запись электронной почты, для чего нужно ввести информацию о себе и доступе по электронной почте. Эти сведения можно взять из имеющейся почтовой программы, либо узнать у системного администратора или ISP.

Рис. 11.9. *Evolution на рабочем столе GNOME*

Evolution работает со стандартными протоколами почтовых серверов и может действовать практически в любом сетевом окружении. Пакет позволяет хранить почту на сервере (протокол IMAP), загружать в локальную систему (протокол POP) или пользоваться спулингом почты на локальной системе, если у вас работает собственный почтовый сервер. Ximian также распространяет коммерческое расширение Evolution под названием Ximian Connector, которое позволяет подключаться к серверам Microsoft Exchange 2000 для расширения возможностей взаимодействия.

После создания учетной записи показывается основное окно Evolution в режиме Summary. Суммарное представление Evolution показывает сводку погоды, новости, список текущих задач и встреч и количество новых сообщений в почтовых ящиках.

В левой части окна Evolution находится панель с кнопками сокращенного вызова. Щелчок по одной из этих кнопок перемещает вас в Inbox, Contacts или Calendar. Для просмотра имеющихся ресурсов в другом представлении выберите View, а затем Folder Bar. В этом режиме представление более напоминает список каталогов, что особенно удобно, если вы создали много папок.

Следующие разделы описывают три основные функции Evolution.

Электронная почта Evolution

Для того чтобы начать работать с почтой Evolution, щелкните по кнопке Inbox или выберите любую почтовую папку в панели папок. Окно представления почты делится на две части: в верхней располагается список сообщений, а в нижней показывается выбранное сообщение. Соотношение частей можно изменить, перетащив разделяющую их серую полосу.

В целом функции почты довольно просты: щелкните по кнопке Send and Receive, чтобы проверить поступление новой почты и отправить почту, помещенную в очередь для отложенной отправки, и по кнопке New Message, чтобы составить новое сообщение.

Что отличает Evolution от других почтовых программ, так это скорость поиска, мощь и простота фильтров и уникальная характеристика vFolders – своего рода комбинация поиска с фильтрами.

Панель поиска располагается вверху списка сообщений. Для поиска в почте перейдите в любую почтовую папку, выделите ту часть сообщения, в которой нужно провести поиск (тело сообщения, отправителя, все сообщение и т. д.), введите в текстовое окно слово и нажмите <Enter>. Evolution выполняет предварительное индексирование почты, поэтому результат возвращается быстрее, чем в других программах.

Фильтры добавляют в конце поиска операцию: при каждом получении почты Evolution осуществляет в новых сообщениях заданный поиск и выполняет действия, зависящие от его результатов. Чаще всего фильтры применяют для автоматического размещения сообщений в зависимости от отправителей и удаления сообщений, помеченных как спам.

Чтобы создать фильтр, перейдите в любое представление почты и откройте список фильтров с помощью Tools→Filters. Затем сделайте следующее:

1. Щелкните по кнопке Add, чтобы добавить фильтр.
2. В верхней части окна выберите группу критериев отбора сообщений для фильтра. Например, если выбрать в первом выпадающем списке Sender Contains и ввести `gnome.org` в появившемся рядом текстовом окне, то фильтр будет применяться ко всем сообщениям, поступающим с почтовых адресов *gnome.org*.
3. В нижней части окна выберите одно или несколько действий, выполняемых над сообщениями. Например, если выбрать Move to Folder, появится кнопка с надписью Click to Select Folder. Щелкните по ней, и вы сможете выбрать почтовую папку, в которую будет помещаться вся почта с адресов *gnome.org*.
4. Щелкните OK в окне создания фильтра и OK в списке фильтров. Создание фильтра закончено.

Если вам покажется недостаточной гибкость, предоставляемая фильтрами, можете воспользоваться vFolders. vFolder, или виртуальная папка, по суще-

ству, представляет собой результат сложного поиска, выглядящий как папка. Это означает, что хотя почтовое сообщение может располагаться только в одной обычной папке, оно может находиться в нескольких виртуальных vFolder.

При создании vFolder устанавливается критерий отбора, так же, как это делается для фильтра, но затем, вместо указания действий над сообщениями, указывается место, где они должны храниться. Созданная vFolder появляется в списке виртуальных папок в нижней части дерева папок. Каждый раз, когда вы открываете такую папку, в почтовых папках выполняется поиск сообщений, отвечающих заданному при создании папки критерию. Таким образом, если вы создаете фильтры для размещения почты в зависимости от отправителя, то можете создать виртуальную папку, которая содержит сообщения с заданной темой вне зависимости от того, кто их отправил.

Календарь Evolution

Календарь Evolution предоставляет большую гибкость в создании и просмотре расписаний. Чтобы начать с ним работать, щелкните по кнопке Calendar в панели кнопок сокращенного доступа или выберите папку Calendar в панели папок. Вам будет представлено пустое расписание на неделю без каких-либо назначенных встреч.

Чтобы увидеть меньший или больший промежуток времени, можно выбрать диапазон дат в календаре, находящемся в правом верхнем углу, или щелкнуть на панели инструментов по одному из предустановленных диапазонов дат: сегодня, одни сутки, пять дней, неделя или месяц.

Освоившись с перемещением по своему календарю, можно начать планировать события. Чтобы создать событие, щелкните по кнопке New Appointment. Введите краткое описание события, выберите время и дайте (если хотите) более подробное описание.

В нижнем правом углу расположен список категорий, к которым можно отнести событие. События, принадлежащие к одной из категорий, периодические события и напоминания отображаются в календаре с помощью маленьких значков: будильник для напоминаний, зацикленные стрелки для периодических событий, торт для дней рождения и т. д.

Можно также планировать напоминания и периодические события. Например, если на следующую неделю у вас назначена важная встреча, можно запланировать напоминание, которое всплывет за 15 минут до начала, чтобы вы могли приготовиться. Для этого щелкните по вкладке Reminder и выберите время и тип напоминания, а затем щелкните по Add, чтобы добавить его к списку. С периодическими событиями процедура аналогична: щелкните по вкладке Recurrence и выберите, как часто должно повторяться событие. Будет ли это только в четверг на этой неделе и во вторник на следующей? Будет ли оно каждую среду с сегодняшнего дня и до Рождества? Или это праздник, случающийся раз в год? Выберите правила повторения, щелкните по Save and Close, и событие окажется помещенным в ваш календарь.

Осталось лишь скоординировать данное событие с другими людьми. Выберите Actions, а затем Forward as iCalendar, чтобы создать почтовое сообщение, к которому прикреплено событие. Когда адресат получит сообщение, ему достаточно будет один раз щелкнуть по кнопке, чтобы добавить событие в свой календарь и отправить вам сообщение, подтверждающее его согласие встретиться.

Адресная книга Evolution

Менеджер контактов Evolution, или адресная книга, представляет собой, вероятно, наименее яркую часть комплекта. Однако она тесно переплетена со средствами электронной почты. Карточки контактов создаются щелчком по кнопке New Contact в представлении контактов, но можно создать карточку, щелкнув правой кнопкой по любому почтовому адресу в полученном сообщении электронной почты.

Если вы ищете в адресной книге чей-то почтовый адрес, можете щелкнуть правой кнопкой по его карточке и выбрать отправку сообщения, либо послать его карточку кому-то другому, сделав для этого два щелчка.

Чтобы посмотреть на менеджер контактов, щелкните по кнопке Contacts в панели сокращенного вызова или выберите какую-нибудь папку с контактами в панели папок. Вы увидите простой список карточек. Если вы хотите упорядочить контакты, например список телефонов, выберите View, Current View и Phone List. Можно также вывести список по организациям, а не по именам.

Электронная таблица Gnumeric

Если вы работаете в офисе и пользуетесь электронными таблицами, весьма вероятно, что с этими таблицами должны также работать пользователи Microsoft Excel. Скорее всего, вы сами пользовались Excel, и это приложение электронных таблиц лучше всего вам знакомо.

Эти факты жизни не помешают вам воспользоваться Linux в своем офисе, поскольку программа Gnumeric спроектирована так, чтобы пользователи Excel чувствовали себя комфортно в этом приложении. В действительности, не зная, в чем дело, эти приложения легко спутать. Gnumeric легко экспортирует и импортирует файлы Excel, а их возможности и обработка данных вполне сопоставимы.

Окошки в теле таблицы отображают значения формул или числа, а поле формул в верхней части окна показывает ту формулу, по которой получен результат.

Например, большинству пользователей электронных таблиц знакома процедура добавления колонки чисел: нужно щелкнуть мышью по любому полю и ввести число в поле формулы вверху окна. Нажмите <Return>, чтобы перейти к следующей ячейке, и введите новое число. Введя несколько чисел, перейдите в пустую ячейку и нажмите кнопку, помеченную греческой буквой «сигма», чтобы вычислить сумму. Затем введите диапазон ячеек, значения которых нужно сложить, или просто выделите их мышью.

К Gnumeric прилагается пространное руководство, которое открывается для чтения при выборе Help, а затем Gnumeric Manual.

gPhoto, инструмент для цифровой камеры

gPhoto, инструмент GNOME для цифровой камеры, позволяет копировать снимки из цифрового фотоаппарата на жесткий диск, упорядочивать их и создавать веб-страницы галерей.

gPhoto требует некоторой настройки, которая зависит от того, каким фотоаппаратом вы пользуетесь. Когда вы впервые запускаете gPhoto, программа спрашивает, какая модель камеры у вас и через какой порт к ней обращаться. Если вы уже запускали gPhoto ранее и хотите снова ввести эту информацию, выберите Configure→Select Port→Camera Model.

Некоторые системы в состоянии автоматически идентифицировать фотоаппарат, а в качестве имени порта использовать */dev/camera*. В других даже устанавливается в качестве стандартного каталога */mnt/camera*, и gPhoto можно адресовать туда, как если бы это был каталог, выбирая File, Open и Directory.

В иных случаях придется немного потрудиться. Если вы пользуетесь кабелем последовательного порта (на его конце видны иголки), то, вероятно, подключили фотоаппарат к */dev/ttyS0* или */dev/ttyS1*. Если у вас кабель USB, вероятно, вы подключили камеру к */dev/usb*. Для FireWire (называемого иначе iLink или IEEE-1394) это может быть / *dev/sga0* или */dev/sga1*.

Если не работает ни одно из этих соединений, проверьте наличие прав чтения/записи для устройства цифровой камеры; это можно сделать с помощью Nautilus или команды *chmod* в командной строке. Можно также попробовать смонтировать устройство камеры, как если бы это был жесткий диск, действуя согласно инструкциям для вашей операционной системы. Одним камерам это требуется, другим – нет.

После того как закончена установка, можно идти дальше. Чтобы загрузить каталог всех снимков в камере, нажмите <Ctrl>+<I> или выберите Camera→ Download Index→Thumbnails. После этого можно решить, какие снимки сохранить, а какие выбросить. Выделите то, что вас интересует (или выберите Select→All, чтобы выделить все), затем нажмите <Ctrl>+<G> или выберите Camera→ Download Selected→Images→Save to Disk, чтобы сохранить снимки на жестком диске.

Чтобы поместить снимки в веб-галерею, сначала выделите в каталоге те снимки, которые вы хотите включить. Затем выберите File→Export→HTML Gallery либо нажмите <Ctrl>+<M>. Вам будет предложено выбрать стиль страниц и адрес для сохранения результатов. Следите за тем, чтобы выбрать новый пустой каталог, а не свой домашний, иначе вы получите небольшой веб-сайт в своем домашнем каталоге.

gPhoto позволяет также поворачивать, изменять масштаб и настраивать цвета для отдельных изображений. Однако это не специальная программа

обработки графики; для серьезных задач редактирования лучше воспользоваться GIMP. Для получения информации о пользовании средствами редактирования изображений или о других возможностях gPhoto выберите Help→ User's Manual либо нажмите <Ctrl>+<H>.

В планы разработки на 2002-2003 годы включена существенная модификация gPhoto. В gPhoto2 можно будет установить как gPhoto, так и инструмент «GnoCam», управлять соединением фотоаппарата через центр управления GNOME и работать со снимками в камере через представления значков в Nautilus. gPhoto2 обещает упростить процедуру подключения камеры и разрешить использование других графических интерфейсов в зависимости от выбора настольной среды. Подробности на *http://gphoto.sourceforge.net*.

Текстовый редактор Abiword

Откройте текстовый процессор Abiword, введя в терминале `abiword` или выбрав Abiword в меню Applications. Abiword, как и Gnumeric, спроектирован так, чтобы в нем привычно чувствовали себя те, кто в прошлом пользовался аналогичными приложениями. Если вы работали в каком-то другом текстовом процессоре, то, вероятно, догадаетесь, что кнопка b в панели инструментов делает текст полужирным, а значок с дискетой сохраняет файл на диске.

Для работы с Abiword может потребоваться установить шрифты или подправить путь для поиска шрифтов. Чтобы узнать, как это делается, зайдите на *http://abisource.com*.

Дополнительные приложения и ресурсы

Существуют десятки, если не сотни, других приложений GNOME, включая средства программирования, игры, инструменты создания блок-схем и построения графиков. Узнать о них лучше всего, посетив веб-сайт *http://gnome.org*, либо установив какие-либо приложения через канал *Red Carpet* GNOME.

Есть несколько мест, куда можно обратиться за помощью в случае возникновения трудностей. Помимо системы подсказки Nautilus и веб-сайта *gnome.org* можно поискать помощи в чатах. Разработчиков можно найти на *irc.gnome.org* в *#gnome*, поэтому туда следует обратиться, если возникли вопросы по разработке программного обеспечения. Если нужна помощь в работе с приложением, посетите *http://support.ximian.com*, подпишитесь на списки почтовой рассылки на *http://lists.gnome.org* или попробуйте воспользоваться чат-системой взаимопомощи пользователей, встроенной в приложения Ximian, для чего нужно выбрать Help, а затем Help Chat.

Отличная коммерческая поддержка GNOME и приложений GNOME доступна также на *support.ximian.com* – как для физических лиц, так и для корпораций.

Другие приложения X

В этом разделе мы немного расскажем о приложениях X, не являющихся приложениями для KDE или GNOME. Как теперь должно быть понятно, они могут без проблем выполняться в любой настольной среде, даже если и не интегрированы в нее.

Особое внимание мы уделим одному аспекту более старых X-приложений: ресурсам X – очень мощному, но в то же время более сложному и трудному в освоении способу настройки X-приложений.

Прежде чем заняться X-ресурсами, отметим, что многие из программ, описываемых в разных частях этой книги, являются X-приложениями и выполняются в среде X (KDE, GNOME или какой-либо другой) точно так же, как описываемые здесь. А в Сети таких программ можно найти буквально десятки тысяч.

База данных ресурсов X

Если вы не пользуетесь рабочим столом или вам приходится иметь дело с приложениями, не вполне интегрированными с рабочим столом, возникнет потребность обратиться непосредственно к ресурсам X. Они упоминаются практически в каждой странице руководства. Ресурсы X предоставляют более гибкий и мощный способ настройки X-клиентов по сравнению с заданием параметров командной строки, таких как *-geometry* и *-fg*. Ресурсы позволяют определить значения по умолчанию для целых классов клиентов. Например, можно установить шрифт по умолчанию `7x13bold` для всех клиентов *xterm* вместо того, чтобы каждый раз указывать его в командной строке.

В последнее время ресурсы X потеряли расположение разработчиков. Хотя они действительно очень гибкие, с ними не просто работать, и они больше похожи на древний реликт. Теперь все большее число программ настраивается не с помощью ресурсов X, а через удобный оконный интерфейс. Однако знание ресурсов X не помешает, т. к. рано или поздно вы с ними столкнетесь.

Для использования ресурсов X необходимы две вещи. Во-первых, надо создать файл со значениями по умолчанию для ресурса X. Обычно этот файл называется *.Xdefaults* и находится в вашем домашнем каталоге. Во-вторых, вам необходимо с помощью *xrdb* загрузить ресурсы на сервер, что сделает их доступными для использования. Как правило, *xrdb* запускается из *.xinitrc* перед запуском любых клиентов.

Для примера возьмем разные параметры командной строки, используемые клиентами в предыдущем примере в файле *.xinitrc*, и определим их как ресурсы X. Затем мы покажем, какие изменения надо внести в *.xinitrc*, чтобы активизировать использование ресурсов.

Для начала несколько слов о ресурсах и их работе. Каждое приложение X является частью определенного *класса приложений*. Например, *xterm* является членом класса `XTerm`. *xclock* и *oclock* входят в класс `Clock`. Ресурсы для класса `Clock` влияют на все приложения этого класса. Поскольку *xclock*

(квадратные часы) и *oclock* (овальные часы) схожи между собой, они принадлежат к одному классу и используют одинаковые ресурсы. Большинство приложений являются членами своего собственного эксклюзивного класса: *xload* является единственным членом класса XLoad. Однако если будет написано другое приложение, похожее на *xload*, оно может также входить в класс XLoad. Помещая X-клиенты в классы приложений, вы получаете возможность определять ресурсы для всех приложений в этом классе. (В руководстве для каждого клиента указан класс, к которому он принадлежит.)

Стандартные X-клиенты используют такие ресурсы, как foreground, background, geometry и font. Кроме того, многие X-клиенты имеют свои собственные специфические ресурсы. Например, *xterm* определяет ресурс logFile, который позволяет указать файл для протоколирования сессии терминала. Доступные ресурсы можно найти в том же руководстве.

Более того, сами ресурсы упорядочены в иерархию классов. Например, ресурс background является членом класса Background. Классы ресурсов позволяют множеству отдельных ресурсов входить в один класс, для которого можно установить значения ресурсов для класса в целом. Например, ресурс background обычно определяет основной цвет фона окна. Однако если у приложения есть несколько областей или панелей, можно установить цвет фона для каждой панели отдельно. В этом случае у каждой панели может быть свой ресурс, например, background1, background2 и т. д., но все они будут входить в класс ресурсов Background. Установив значение ресурса для класса Background, вы установите значения для всех ресурсов этого класса.

Обычно нет необходимости беспокоиться о разнице между классом ресурсов и самими ресурсами из этого класса. В большинстве случаев проще установить значения ресурса для всего класса (например, Background), чем для его отдельных членов.

Теперь давайте посмотрим, как устанавливаются значения ресурсов в базе данных ресурсов X. Полная спецификация ресурса имеет следующий вид:[1]

```
(ApplicationClass|applicationName)*(ResourceClass|resourceName) : value
```

Вертикальная черта означает «исключающее или». Допустим, мы хотим установить цвет фона окна *xterm*. *Полная* спецификация ресурса может выглядеть так:

```
xterm*background: darkslategray
```

Однако здесь мы определили только отдельный ресурс background (не все из ресурсов, которые могут входить в класс Background) и только для клиента *xterm*, вызываемого как *xterm* (далее мы вернемся к этому вопросу). Поэтому лучше использовать классы ресурсов:

---

[1] На самом деле спецификации ресурсов использует более сложный синтаксис, и правила для установки связи между ресурсами и значениями довольно запутанные, но для облегчения понимания мы упростили модель ресурсов приложения. Любопытных читателей мы направляем к книгам по использованию X, таким как «X Window System User's Guide»

```
XTerm*Background: darkslategray
```

Эта спецификация ресурса будет применяться ко *всем* клиентам *xterm* и всем ресурсам класса Background, используемым *xterm*.

Теперь давайте переведем параметры из нашего файла *.xinitrc* в ресурсы приложений. Создайте файл *.Xdefaults* в домашнем каталоге. Для соответствия файлу *.xinitrc* из предыдущего примера он должен содержать такие строки:

```
1  Clock*Geometry:         70x70+5+5
2  XLoad*Geometry:         85x50+85+5
3  XBiff*Geometry:          +200+5
4
5  ! Defaults for all xterm clients
6  XTerm*Foreground:       white
7  XTerm*Background:       black
8
9  ! Specific xterms
10 xterm-1*Geometry:       80x40+10+110
11
12 xterm-2*Geometry:       -20+10
13 xterm-2*Font:           7x13bold
14 xterm-2*Background:     darkslategray
15
16 xterm-3*Geometry:       80x25-20-30
17 xterm-3*Font:           7x13bold
```

Строки 1–3 устанавливают класс ресурса Geometry для классов приложений Clock, XLoad и XBiff. В строках 6–7 мы установили классы ресурсов Foreground и Background для класса XTerm в целом. Все клиенты *xterm* будут по умолчанию использовать эти значения для Foreground и Background.

В строках 10–17 мы установили ресурсы, специфичные для каждого экземпляра *xterm*. Это необходимо, поскольку не все запущенные клиенты *xterm* одинаковы; например, каждый из них имеет свои спецификации geometry. Поэтому каждому индивидуальному клиенту *xterm* мы дали имя: xterm-1, xterm-2 и xterm-3. Ресурс Geometry для каждого из них установлен в строках 10, 12 и 16. Мы также определили класс Font для xterm-2 и xterm-3 и установили класс Background для xterm-2 в значение darkslategray.

Правила привязки ресурсов X работают таким образом, что определенные привязки имеют преимущества перед другими. В нашем случае ресурс для конкретного экземпляра *xterm* (например, xterm-2*Background в строке 14) имеет преимущество перед установками ресурса для всего класса XTerm (XTerm*Background в строке 7). Обычно значения для класса приложения или ресурса имеют *более низкий* приоритет, чем значения ресурса для конкретных экземпляров этого класса. Благодаря этому можно устанавливать значения по умолчанию для всего класса в целом, а затем переопределять их для конкретных экземпляров класса.

Теперь рассмотрим изменения, которые необходимо внести в *.xinitrc*, чтобы использовать наши ресурсы X Window. Сначала надо добавить команду

xrdb, загружающую ресурсы приложений на сервер. И мы можем избавиться от различных параметров командной строки, которые мы заменили ресурсами:

```
#!/bin/sh
# Sample .xinitrc shell script

# Load resources
xrdb -load $HOME/.Xdefaults

# Start xterms
xterm -name "xterm-1" &
xterm -name "xterm-2" &
xterm -name "xterm-3" &

# Other useful X clients
oclock &
xload &
xbiff &
xsetroot -solid darkslateblue &

# Start the window manager
exec fvwm2
```

Параметр *-name*, заданный для трех экземпляров *xterm*, позволяет указать имя приложения, используемое *xterm*, чтобы найти ресурс. Большинство X-клиентов не поддерживает параметр *-name*; обычно используется имя, под которым вызывается программа. Но поскольку многие пользователи запускают сразу несколько клиентов *xterm*, будет полезным различать их при установке ресурсов.

Теперь вы способны в некоторой степени изменять вашу среду X. Конечно, умение настраивать X частично зависит от того, знакомы ли вы с разными X-клиентами, а также с менеджером окон (и его настройкой).

Emacs и другие редакторы

Функций X в Emacs становится все больше и больше. Теперь они включают выпадающие меню, различные шрифты для разных частей окна и полную интеграцию с функциями работы с выделенным текстом среды X.

Большинство дистрибутивов в настоящее время также включают XEmacs – версию Emacs, которая еще лучше интегрирована в систему X Window и имеет более приятный и дружественный внешний вид. Большинство вопросов, рассматриваемых в этом разделе, относится и к XEmacs.

Давайте начнем с определения цветов для разных частей окна Emacs. Попробуйте выполнить команду:

```
eggplant$ emacs -bg ivory -fg slateblue -ms orangered -cr brown
```

Этим вы установили цвет фона, цвет текста, цвет указателя мыши и цвет курсора, соответственно. Курсор – это маленький прямоугольник, появляю-

щийся в окне, который представляет собой так называемую «точку» (point) в Emacs, т. е. то место, где вы набираете текст. К цветам мы еще вскоре вернемся.

При запуске окно Emacs содержит строку меню вверху и полосу прокрутки с правой стороны окна (рис. 11.10).

```
Buffers Files Tools Edit Search Mule Emacs-Lisp Help
(display-time)

(define-key global-map "\C-xr" 'replace-regexp)

(setq load-path
      (append (list
        (expand-file-name "~andyo/Emacs"))
         load-path))

(read-abbrev-file "" t)

; so that shell window doesn't jump unexpectedly when I enter a command
(setq comint-scroll-to-bottom-on-input nil)

; I don't like my From and Subject lines in reverse video in mail messages
(setq rmail-highlighted-headers "^\\'x")

--:--  .emacs         12:14PM   (Emacs-Lisp)--L1--All--------------------------
```

Рис. 11.10. *Окно Emacs*

Полоса прокрутки работает так же, как полоса прокрутки в *xterm*. Строка меню предлагает обычные функции. В некоторых режимах редактирования, таких как C и TEX, есть свои собственные выпадающие меню. Меню не документированы, поэтому для выяснения особенностей их работы вам придется поэкспериментировать.

Меню приходит на помощь, когда вы хотите воспользоваться функцией, для которой нет простой комбинации клавиш, или вы забыли эту комбинацию. Например, если вы редко прибегаете к поиску с помощью регулярных выражений (очень мощная возможность, которая стоит времени, потраченного на ее изучение), самый простой способ вызвать эту функцию – это выбрать Regexp Search в меню Edit.

Другой полезный элемент – Choose Next Paste из меню Edit, позволяющий получить то, что недоступно другими способами: список всех фрагментов текста, которые вы недавно вырезали. Иными словами, вам показывается «кольцо удалений». Вы можете выбрать текст, который хотите вставить, и при следующем нажатии C-y он будет помещен в буфер.

Если вы устали от полосы прокрутки и меню, попрощайтесь с ними и поместите следующий код LISP в ваш файл *.emacs*:

```
(if (getenv "DISPLAY")
    (progn (menu-bar-mode -1)
    (scroll-bar-mode -1))
   )
```

Следующие интересные возможности X относятся к мыши. Вы можете вырезать и вставлять текст так же, как и в *xterm*. Можно обмениваться текстом между различными окнами; если вам надо поместить результаты какой-либо команды *xterm* в файл, вы можете скопировать их из *xterm* и вставить в буфер Emacs. Кроме того, с любым скопированным обычным образом текстом (например, через C-w) можно проделать то же, что и с текстом, вырезанным мышью. Поэтому вы можете вырезать несколько слов из вашего буфера Emacs и вставить их в *xterm*.

Правая кнопка мыши работает немного необычно. Если выделяется текст левой кнопкой мыши, то скопировать его можно, щелкнув один раз правой кнопкой. Второй щелчок правой кнопкой мыши удаляет текст. Чтобы вставить его обратно, нажмите среднюю кнопку мыши. Текст будет вставлен прямо перед символом, на котором находится указатель мыши. Ошиблись? Ничего страшного, со всеми бывает; команда «undo» вернет все, как было, так же как и в случае всех остальных функций Emacs. (Выберите Undo из меню Edit или просто нажмите C-_.)

Если вам по душе работа с мышью, можно привязать к кнопкам любые нужные вам функции точно так же, как и к клавиатуре. Попробуйте вставить в ваш файл *.emacs* следующую команду. Теперь, если вы, удерживая нажатой клавишу <Shift>, нажмете левую кнопку мыши, должен появиться буфер для создания сообщения электронной почты:

```
(define-key global-map [S-mouse-1] 'mail)
```

Мы не рекомендуем переопределять существующие функции мыши, но массу неиспользованных возможностей предлагают клавиши <Shift>, <Ctrl> и <Meta>. В привязках можно комбинировать S-, C- и M- любым образом:

```
(define-key global-map [S-C-mouse-1] 'mail)
```

Теперь давайте поиграем с окнами. Окна применялись в Emacs задолго до появления системы X Window. Поэтому окна Emacs – это не то же самое, что окна X Window. То, что в X считается окном, Emacs называет *фреймом*.

Хотите редактировать один и тот же файл сразу в двух фреймах? Нажмите C-x 5 2, и появится другой фрейм. Новый фрейм – это просто еще одно представление редактируемого документа. Вы можете редактировать разные буферы в двух фреймах, но все, что вы сделаете в одном фрейме, будет повторено в соответствующем буфере другого фрейма. Когда вы выйдете из Emacs, нажав C-x C-c, исчезнут оба фрейма; если вы хотите закрыть только один фрейм, нажмите C-x 5 0.

В завершение нашего исследования Emacs в X Window посмотрим на возможности цветового оформления. Изменять его можно прямо в сессии Emacs, что делает достаточно простыми эксперименты с различными возможностями. Нажмите M-x, затем наберите set-background-color и нажмите <Enter>. В ответ на запрос наберите ivory или любой другой цвет. (Помните, что Emacs использует соглашение M-x там, где в книге мы используем <Meta>+<x> или <Alt>+<x>.)

Постарайтесь выбрать цвета текста и фона так, чтобы они достаточно отличались, и можно было видеть текст! Помимо `set-background-color` Emacs предлагает команды `set-foreground-color`, `set-cursor-color` и `set-mouse-color`.

Прежде чем завершить этот раздел, мы хотим отметить, что если Emacs или XEmacs кажутся вам слишком сложными, то вам будет приятно узнать, что в KDE входит целый ряд текстовых редакторов – от совсем простых, до достаточно усложненных. Ни один из них не обладает такими размерами или мощью, как (X)Emacs, но любого из них может оказаться достаточно.

Три текстовых редактора KDE следующие (в порядке роста сложности): *KEdit*, *KWrite* и *Kate*. Последнее имя происходит от KDE Advanced Text Editor. *Kate* можно использовать как полнофункциональный редактор программиста, с выделением цветом синтаксиса, одновременным открытием нескольких файлов и т. д. *KEdit* по богатству (или бедности) функций аналогичен редактору *Notepad* в Windows, а *KWrite* занимает промежуточное положение. Все три могут быть найдены в K-меню, в подменю `Editors`.

По договору между издательством «Символ-Плюс» и Интернет-магазином «Books.Ru – Книги России» единственный легальный способ получения данного файла с книгой ISBN 5-93286-069-3, название «Запускаем Linux, 4-е издание» – покупка в Интернет-магазине «Books.Ru – Книги России». Если Вы получили данный файл каким-либо другим образом, Вы нарушили международное законодательство и законодательство Российской Федерации об охране авторского права. Вам необходимо удалить данный файл, а также сообщить издательству «Символ-Плюс» (piracy@symbol.ru), где именно Вы получили данный файл.

12

Совместимость с Windows и Samba

Linux – чрезвычайно эффективная операционная система, которая во многих случаях может полностью заменить MS-DOS/Windows. Однако среди нас всегда найдутся такие, кто пожелает использовать наряду с Linux другие операционные системы или, по крайней мере, иметь возможность напрямую обмениваться с ними файлами. Linux удовлетворяет такие сокровенные желания с помощью внутренних расширений, позволяющих получать доступ к посторонним файловым системам и работать с их файлами. Linux имеет возможность монтировать на жестком диске разделы MS-DOS/Windows или обращаться к файлам и принтерам, выделяемым для совместного использования в сети серверами Windows. Linux также может выполнять приложения DOS и Windows с помощью утилит совместимости, позволяющих вызывать MS-DOS или Windows.

В этой главе мы используем названия MS-DOS и Windows в некотором общем смысле в отношении любых основанных на DOS операционных систем производства Microsoft или совместимых с ними. В это число входят MS-DOS, PC-DOS и DR-DOS/Novell DOS (с Windows 3.x, работающей поверх или без нее), а также различные версии собственно Windows, независимо от того, основаны ли они на установленной отдельно DOS, как Windows 3.x, или имеют встроенное ядро DOS, как Windows 95/98/ME. Windows NT/2000/XP – особая статья, и некоторые описываемые здесь вещи не будут там работать или будут работать по-другому.

Одна из причин, по которым чаще всего возникает потребность в Windows, заключается в том, что она часто лучше поддерживает новые аппаратные устройства. Если вы установили Windows из-за того, что вам нужно работать с неким устройством, которое поддерживается Windows, но для которого нет драйвера в Linux, не отчаивайтесь. Большинство основных аппаратных устройств, поддерживаемых Windows, в конечном счете, получают поддержку в Linux, хотя, возможно, приходится некоторое время ждать. Например, драйверы Linux для устройств USB одно время были редкими и сырыми, но сейчас многие распространенные устройства USB прекрасно ра-

ботают в Linux. Самые свежие данные о том, какие устройства USB работают под Linux, можно получить на *http://www.linux-usb.org*.

Windows может потребоваться вам и для работы со «стандартными» приложениями, такими как Photoshop или Microsoft Office. В обоих случаях можно воспользоваться бесплатными приложениями open source (а именно: The Gimp, KOffice и OpenOffice), которые могут составить конкуренцию и даже превзойти свои фирменные эквиваленты с закрытым кодом. Однако иногда все же приходится запускать Windows, чтобы получить доступ к программным продуктам, для которых в Linux нет равноценных или отсутствует полная совместимость.

В сущности, есть четыре способа взаимодействия Linux и Windows:

1. Совместное использование CD и дискет («sneakernet»).
2. Совместное использование компьютера при установке в разных разделах диска.
3. Совместное использование данных в сети.
4. Одновременное выполнение на одном компьютере с помощью эмулятора или виртуальной машины.

Когда Windows и Linux работают на разных машинах, не связанных сетью, можно записать дискету или CD (CD-R или CD-RW) на одной машине и прочесть на другой. Как Windows, так и Linux могут читать и писать CD в стандартном формате ISO9660. Программа *cdrecord*, работающая под Linux и другими разновидностями Unix, может создавать CD с использованием расширения стандарта ISO9660 под названием Joliet, разработанного Microsoft, благодаря чему у Windows не возникает проблем с форматом диска.

Хотя на дискетах помещается значительно меньше данных, чем на CD, они могут быть удобны, когда требуется перенести лишь несколько небольших файлов. Linux и Windows могут совместно использовать данные на дискетах в формате MS-DOS, отформатированных утилитами MTools, описываемыми далее в этой главе.

Более выгодной экономически является установка Windows и Linux на одной и той же машине в разных разделах жесткого диска. Во время начальной загрузки пользователю дается возможность выбрать операционную систему, которая должна быть запущена. В разделе «Начальная загрузка системы» главы 5 описывается, как настроить загрузку альтернативных операционных систем.

Для доступа к файлам на разделах Windows во время работы Linux можно использовать MTools, но значительно удобнее смонтировать раздел Windows непосредственно к файловой системе Linux. В результате появляется возможность обращаться к файлам Windows как к обычным файлам Unix.

Для компьютеров, объединенных в сеть, самым замечательным средством заставить Linux и Windows работать вместе является пакет Samba – программное обеспечение Open Source, позволяющее получать доступ к файлам и принтерам Unix из Windows. Серверы Linux, на которых работает Samba,

могут в некоторых случаях обслуживать компьютеры Windows даже быстрее, чем сами серверы Windows! Кроме того, пакет Samba оказался очень стабильным и надежным.

В пакет Samba входят также программы, работающие с файловой системой *smbfs*, поддерживаемой Linux, которая позволяет выделенные Windows для совместного использования каталоги монтировать в файловой системе Linux. Мы достаточно подробно обсудим *smbfs* и Samba, чтобы вы могли монтировать совместно используемые каталоги и получить действующий функциональный сервер.

Наконец, есть способы непосредственного запуска приложений Windows и даже самой Windows под Linux. Wine – проект open source, имеющий задачей непосредственную поддержку приложений Windows без установки последней. Другой подход применяется в коммерческом приложении VMware, которое позволяет одновременную работу нескольких операционных систем – Windows, Linux, FreeBSD или др. При работе Windows под VMware данные используются совместно с хостом Linux с помощью средств Samba.

Следует проявлять известный скептицизм в отношении некоторых заявлений о совместимости. Например, вы можете обнаружить, что необходимо вдвое больше дискового пространства для поддержки двух операционных систем и связанных с ними файлов и прикладных программ, плюс средства преобразования файлов и графических форматов и т. д. Может оказаться, что аппаратные устройства, настроенные для работы с одной ОС, не настроены для работы с другой, или даже после установки и правильной настройки всего необходимого программного обеспечения остаются небольшие неразрешимые проблемы совместимости.

Совместный доступ к дискам с помощью MTools

Пакет MTools представляет собой совокупность команд, позволяющих манипулировать файлами и каталогами на одном или нескольких дисках MS-DOS. Эти команды действуют аналогично стандартным командам MS-DOS, хотя между ними есть существенные различия, к которым нужно привыкнуть.

Файл */etc/mtools.conf* используется для вызова MTools, и, возможно, он уже настроен у вас для идентификации первичного и вторичного приводов гибких дисков как приводов MS-DOS A: и B:

```
drive a: file="/dev/fd0" exclusive 1.44m mformat_only
drive b: file="/dev/fd1" exclusive 1.44m mformat_only
```

Если на диске есть раздел DOS, можно обращаться к нему как к C: с помощью такой строки в *mtools.conf*:

```
drive c: file="/dev/hda1"
```

Добавляя эту строку, укажите вместо */dev/hda1* имя файла, соответствующее вашему разделу MS-DOS. Будьте очень осторожны при задании раздела на диске. Ошибка может легко привести к разрушению всего раздела Linux или других ценных данных.

Иногда можно встретить дискету, которая не в формате MS-DOS. Чтобы использовать ее, нужно сначала выполнить форматирование нижнего уровня с помощью команды *fdformat*, которая не входит в MTools. Затем с помощью команды MTools *mformat* можно поместить на дискету FAT и структуру каталогов:

```
$ fdformat /dev/fd0
Double-sided, 80 tracks, 18 sec/track. Total capacity 1440 kB.
Formatting ... done
Verifying ... done
$ mformat a:
```

Теперь команда *mdir*, служащая для вывода содержимого каталога MS-DOS, покажет:

```
$ mdir a:
 Volume in drive A has no label
 Volume Serial Number is 4077-6090
Directory for A:/

No files
                          1 457 664 bytes free
```

Сейчас дискета готова принять файлы, копируемые на нее командой *mcopy*:

```
$ mcopy /etc/hosts a:
$ mdir
 Volume in drive A has no label
 Volume Serial Number is 4077-6090
Directory for A:/

hosts              948 10-02-2002  19:28  hosts
        1 file                  948 bytes
                          1 456 640 bytes free
```

Этот файл на дискете может быть прочтен любой системой – MS-DOS или Windows. Приведенные примеры дают базовое представление о том, как работают утилиты MTools. Команда *mcopy* типична тем, что может принимать в качестве аргументов имена файлов MS-DOS или Linux, выполняя соответствующим образом операцию копирования. В число некоторых других полезных команд MTools входят *mmd* для создания каталога и *mcd* для изменения каталога. Когда вы изменяете рабочий каталог на диске MS-DOS с помощью *mcd*, утилиты MTools следят за рабочим каталогом, моделируя обычное поведение MS-DOS. Рассмотрим, например, такую последовательность команд:

```
$ mmd a:dir1
$ mcd a:dir1
$ copy c:file1 a:
$ mdir a:
 Volume in drive A has no label
 Volume Serial Number is 4077-6090
Directory for A:/dir1
```

```
.            <DIR>     10-02-2002  19:42
..           <DIR>     10-02-2002  19:42
file1    TXT       948 10-02-2002  19:50  file1.txt
        3 files                 948 bytes
                          1 454 080 bytes free
```

В результате выполнения этих команд файл *file1.txt* на диске C: копируется в каталог *dir1* на диске A:, и команда *mdir* показывает содержимое *A:\dir1*. Если у вас нет опыта работы с командами MS-DOS/Windows, будьте осторожны, потому что такой режим отличается от стандарта, принятого в Unix.

Обратите внимание, что в приведенном списке содержимого каталога имя каталога выводится как `A:/dir1` с использованием обычной косой черты вместо обратной. Это служит напоминанием, что при работе с утилитами MTools пути каталогов нужно задавать через прямой слэш, например:

```
$ mdel c:/dir1/dir2/readme.txt
```

Кроме того, в командах MTools для задания параметров используется черточка, а не косая черта. Например, вывести краткий список содержимого диска A: можно командой:

```
$ mdir -b a:
A:/hosts
A:/dir1/
A:/file1
```

Ниже следует краткое описание всех команд MTools. Подробнее о них см. на страницах руководства для *mtools*(1) и для каждой отдельной команды.

mattrib

Изменяет атрибуты MS-DOS-файла, находящегося в текущем рабочем каталоге MS-DOS.

mbadblocks

Тестирует диск и делает в FAT (таблице размещения файлов) записи, помечающие сбойные блоки.

mcd

Изменяет рабочий каталог MS-DOS.

mcopy

Копирует файлы.

mdel

Удаляет файл.

mdeltree

Удаляет каталог MS-DOS и его содержимое, включая все подкаталоги.

mdir

Показывает содержимое каталога MS-DOS. По умолчанию это рабочий каталог MS-DOS.

mdu

Сообщает об объеме пространства на диске MS-DOS, занимаемом каждым файлом, аналогично команде Unix *du*.

mformat

Создает пустую файловую систему MS-DOS (FAT) на диске. Диск уже должен пройти форматирование MS-DOS низкого уровня (например, командой *fdformat*).

minfo

Выводит информацию низкого уровня о диске MS-DOS.

mkmanifest

Создает файл, содержащий таблицу трансляции имен файлов из MS-DOS в Unix для диска MS-DOS. Полезна, когда на диск MS-DOS копируются несколько файлов Unix, которые позднее нужно будет скопировать обратно, дав им исходные имена в системе Unix.

mlabel

Записывает метку тома на диск MS-DOS.

mmd

Создает каталог на диске MS-DOS.

mmount

Монтирует диск MS-DOS на файловую систему Linux. Диск должен уже пройти форматирование MS-DOS низкого уровня и содержать файловую систему MS-DOS. Чтобы это действовало, в файле */etc/fstab* должна быть запись типа:

```
/dev/fd0     /mnt/floppy     auto    noauto,owner    0 0
```

С помощью этой записи команда *mmount a:* смонтирует файловую систему флоппи-диска на каталог */mnt/floppy*, к которому Linux сможет потом обращаться непосредственно. Для размонтирования диска используйте команду Linux *umount*.

mmove

Перемещает или переименовывает файл или каталог на диске MS-DOS.

mrd

Удаляет один или несколько каталогов с диска MS-DOS. Каталог должен быть пустым. Для удаления каталога MS-DOS и его содержимого используйте команду *mdeltree*.

mren

Переименовывает файл или каталог на диске MS-DOS.

mtype

Показывает содержимое файла, который может быть на диске MS-DOS или локальной системе Linux.

Совместное использование дисковых разделов

Как было показано в предыдущем разделе, с помощью утилит MTools можно получить доступ к разделу жесткого диска, отформатированного в DOS, и даже монтировать отформатированный в DOS диск в файловой системе Linux с помощью *mmount*. Эти методы можно использовать в системе с загрузкой альтернативных ОС, содержащей раздел MS-DOS, но есть более удачный способ.

Как говорилось в разделе «Монтирование файловых систем» главы 6, доступ к разделам на жестких дисках можно получить, смонтировав их в каталоги файловой системы Linux. Чтобы иметь возможность чтения и записи в конкретной файловой системе, ее поддержка должна быть включена в ядро Linux.

В Linux есть драйверы для чтения и записи файлов в обычной файловой системе FAT и более новой системе VFAT, которая появилась в Windows 95 и поддерживает длинные имена файлов. Linux может читать (и, с некоторыми ограничениями, записывать) файловую систему NTFS Windows NT/2000/XP.

В разделе «Сборка нового ядра» главы 7 вы научились собирать собственное ядро. Для доступа к разделам DOS (используется MS-DOS и Windows 3.x) и VFAT (используется Windows 95/98/ME) необходимо включить `DOS FAT fs support` в секции filesystems во время конфигурирования ядра. После установки этого параметра можно выбрать `MSDOS fs support` и `VFAT (Windows-95) fs support`. Первый параметр позволяет монтировать разделы FAT, а второй – разделы FAT32.

Если требуется доступ к файлам в разделе Windows NT с файловой системой NTFS, то нужен другой драйвер. Во время конфигурирования ядра включите опцию `NTFS filesystem support`. Это позволит монтировать разделы NTFS, указав в качестве типа файловой системы *ntfs*. Учтите, однако, что в настоящее время драйвер NTFS поддерживает доступ только по чтению. Существует версия этого драйвера, которая поддерживает и запись, но на момент написания книги она все еще находилась на стадии разработки и не гарантировала надежной работы при записи в раздел NTFS. Внимательно прочтите документацию, прежде чем устанавливать ее и работать с ней!

Во время работы Linux можно монтировать разделы Windows так же, как любые другие типы разделов. Если, например, третий раздел на первом диске IDE содержит установленную копию Windows 98, вы можете сделать его файлы доступными с помощью следующей команды, которую надо выполнить, зарегистрировавшись в качестве `root`:

```
mount -t vfat /dev/hda3 /mnt/windows98
```

Здесь */dev/hda3* указывает на диск, соответствующий диску Windows 98, а параметр */mnt/windows98* может быть изменен на любой каталог, который вы создали для доступа к этим файлам. Но как узнать, что вам нужен именно */dev/hda3*, а не какой-либо другой каталог? Если вы уже знакомы с соглашениями об именах для файловых систем Linux, то знаете, что *hda3* – это главный диск на первом IDE-порту. Можно облегчить себе жизнь, записав

имена разделов во время создания их с помощью *fdisk*, но если вы этого не сделали, можно снова запустить *fdisk* и посмотреть на таблицу разделов.

Драйверы файловых систем *msdos* и *vfat* поддерживают ряд параметров, которые можно задавать с помощью ключа *-o* команды *mount*. Параметры описаны на страницах руководства *mount*(8), отдельные разделы посвящены параметрам, специфичным для файловых систем *fat* и *ntfs*, два из которых представляют особый интерес.

Параметр *check* указывает ядру, что делать с именами файлов, не разрешенными в MS-DOS. Это относится только к созданию и переименованию файлов. Вы можете указать три значения для *check*: `relaxed` позволит сделать с именем файла практически все что угодно. Если имя не отвечает соглашению 8.3 для файлов MS-DOS, оно будет соответствующим образом укорочено. `normal`, являющееся значением по умолчанию, также сократит длинные имена файлов, но, кроме того, удалит спецсимволы, такие как `*` и `?`, не допускающиеся в именах файлов MS-DOS. Наконец, `strict` запрещает и длинные имена файлов и спецсимволы. Для того чтобы потребовать от Linux строгости по отношению к именам файлов в смонтированном ранее разделе, команду *mount* можно использовать следующим образом:

```
# mount -o check=strict -t msdos /dev/sda5 /mnt/dos
```

Этот параметр применим только к файловым системам *msdos;* ограничения на длину файла не относятся к файловой системе *vfat*.

Второй параметр, *conv*, может оказаться полезен, но не так широко, как покажется на первый взгляд. Системы Windows и Unix пользуются разными соглашениями относительно того, как указывать конец строки в текстовых файлах. Windows использует и возврат каретки, и символ перевода строки, в то время как Unix использует только символ перевода строки. Это не значит, что файл из одной системы нельзя прочесть на другой, но может сильно мешать. Чтобы ядро автоматически выполняло преобразование форматов текстовых файлов Windows и Unix, необходимо выполнить команду *mount* с параметром *conv*. Здесь возможны три значения: `binary`, являющееся значением по умолчанию, не выполняет никакого преобразования; `text` преобразует все файлы; и `auto` пытается по расширению имени файла определить, является ли файл текстовым или двоичным. Если расширение входит в список «известных двоичных расширений», файл не преобразуется, иначе преобразование выполняется.

Обычно не рекомендуется использовать *text*, т. к. это приведет к повреждению любых двоичных файлов, включая графические файлы и файлы, написанные в текстовых процессорах, электронных таблицах и других программах. Может оказаться опасным и *auto*, т. к. механизм распознавания, основанный на расширениях, слишком прост. Поэтому мы не рекомендуем применять параметр *conv*, если только раздел не содержит исключительно текстовые файлы. Устанавливайте режим *binary* (значение по умолчанию) и преобразовывайте файлы по мере необходимости. Ниже в этой главе есть раздел «Утилиты преобразования файлов» с указаниями, как это делать.

Файловые системы MS-DOS и NTFS, как и все остальные, можно автоматически монтировать во время начальной загрузки, для чего нужно поместить в файл */etc/fstab* соответствующую запись. Например, следующая строка в */etc/fstab* монтирует раздел Windows 98 в каталоге */win*:

```
/dev/hda1    /win    vfat    defaults,umask=002,uid=500,gid=500    0  0
```

При доступе из Linux к одной из файловых систем *msdos*, *vfat* или *ntfs*, системе приходится каким-то способом присваивать файлам права доступа и владения, предусматриваемые Unix. По умолчанию права доступа и владения определяются с помощью UID, GID и маски, устанавливаемой по умолчанию вызывающим процессом. Такой принцип приемлем при выполнении команды *mount* из оболочки, но при выполнении из загрузочных сценариев владельцем файлов будет назначаться суперпользователь, что может быть нежелательно. В приведенном выше примере параметр *umask* устанавливает маску, которую система будет использовать при создании файлов и каталогов. Параметр *uid* задает владельца (в виде числового UID, а не текстового имени), а параметр *gid* задает группу (в виде числового GID). Все файлы на монтируемом разделе будут иметь в Linux владельца и группу. Поскольку системы с загрузкой альтернативных операционных систем обычно используются как рабочие станции, принадлежащие одному пользователю, параметрам *uid* и *gid* можно дать значения UID и GID этого пользователя.

Монтирование разделяемых ресурсов Windows

Когда Linux и Windows работают на разных машинах, объединенных в сеть, можно организовать их совместный доступ к файлам друг друга. В Windows встроена поддержка предоставления ресурсов в совместное сетевое использование с помощью протокола, называемого SMB (Server Message Block), известного также как Common Internet File System (CIFS). Linux поддерживает протокол SMB с помощью Samba и файловой системы Linux *smbfs*.

В этом разделе мы рассмотрим предоставление ресурсов в совместное использование в одном направлении: доступ к файлам на системах Windows из Linux. В следующем разделе мы расскажем, как добиться обратного, т. е. сделать отдельные файлы на Linux доступными пользователям систем Windows.

Утилиты *smbmount* и *smbmnt* из дистрибутива Samba совместно с драйверами файловой системы *smbfs* управляют взаимодействием между Linux и Windows и монтируют в Linux каталог, выделенный Windows в качестве общего ресурса. С одной стороны, это похоже на монтирование разделов Windows, о котором говорилось в предыдущем разделе, с другой стороны – на монтирование файловой системы NFS.

Устанавливать какие-либо новые программы в Windows при этом не требуется, потому что система Linux будет обращаться к машине с Windows так же, как это делают другие машины Windows. Однако важно, чтобы на машине с Windows работал протокол TCP/IP, а не NetBEUI или Novell (IPX/SPX). Можно организовать работу и при использовании NetBEUI и/или IPX/SPX, но гораздо лучше их избегать. Когда используются другие протоколы помимо TCP/IP, возможны конфликты имен и похожие проблемы.

В настройках протокола TCP/IP на машине под Windows должны быть правильно установлены IP-адрес и маска подсети. Кроме того, должны быть установлены рабочая группа (или домен) и имя компьютера. Легко проверить доступность системы Windows из Linux по имени компьютера:

```
$ ping maya
PING maya.metran.cx (172.16.1.6) from 172.16.1.3 : 56(84) bytes of data.
64 bytes from maya.metran.cx (172.16.1.6): icmp_seq=2 ttl=128 time=362 usec
64 bytes from maya.metran.cx (172.16.1.6): icmp_seq=3 ttl=128 time=368 usec

--- maya.metran.cx ping statistics ---
2 packets transmitted, 2 packets received, 0% packet loss
round-trip min/avg/max/mdev = 0.344/0.376/0.432/0.038 ms
```

Результат показывает, что Linux может связаться с Windows-сервером `maya`, и разрешение имен и базовая связь TCP/IP действуют.

Теперь допустим, что может быть установлено соединение TCP/IP между компьютерами Linux и Windows и что на машине Windows есть каталог, выделенный как общий ресурс. Подробные инструкции о том, как настроить работу сети и выделение общих ресурсов в Windows 95/98/ME и Windows NT/2000/XP можно найти в книге «Using Samba» Роберта Экштейна (Robert Eckstein) и Дэвида Колье-Брауна (David Collier-Brown), O'Reilly.

На машине Linux необходимо сделать три шага:

1. Скомпилировать поддержку файловой системы *smbfs* в ядре.
2. Установить утилиты Samba *smbmount* и *smbmnt*, и создать хотя бы минимальный файл конфигурации.
3. Смонтировать совместно используемые каталоги командой *mount* или *smbmount*.

В вашем дистрибутиве Linux *smbfs* и Samba могут быть уже установлены. Если нет, пройдем указанные этапы поочередно. Первый шаг совсем простой. В секции filesystems/Network File Systems при конфигурировании ядра выберите SMB file system support (to mount Windows shares etc.). Скомпилируйте и установите ядро или установите и загрузите модуль.

Затем нужно установить утилиты *smbmount* и *smbmnt* из пакета Samba. Установить Samba можно, руководствуясь указаниями в следующем разделе, а если Samba уже стоит на какой-то системе Linux, можно просто скопировать оттуда эти команды. Может также потребоваться скопировать некоторые другие утилиты Samba, например *smbclient* и *testparm*.

Программа *smbmount* запускается из командной строки или как *mount* с параметром *-t smbfs*. В любом случае *smbmount* вызывает *smbmnt*, которая выполняет фактическую операцию монтирования. Когда совместно используемый каталог смонтирован, процесс *smbmount* продолжает выполняться, и если выполнить *ps ax*, можно увидеть процесс *smbmount* для каждого смонтированного ресурса.

Программа *smbmount* читает файл конфигурации Samba, хотя ей нужно оттуда не много информации. В действительности можно обойтись совершен-

но пустым файлом конфигурации! Главное, чтобы файл конфигурации находился в правильном месте, иначе будут получены сообщения об ошибках. Чтобы узнать, где находится файл конфигурации, выполните программу *testparm*. (Если вы скопировали обе необходимые утилиты с другой системы Linux, выполните *testparm* на той машине.) В первой строке результата показывается местонахождение файла конфигурации, как в следующем примере:

```
$ testparm
Load smb config files from /usr/local/samba/lib/smb.conf
... удалены ...
```

О том, как составлять файл конфигурации, будет рассказано в следующем разделе; для наших текущих задач достаточно поместить в него следующее содержание:

```
[global]
    workgroup = NAME
```

Замените *NAME* именем своей рабочей группы, как оно задано в системах Windows вашей сети.

Наконец, нужно смонтировать совместно используемый каталог. Пользоваться *smbmount* совсем просто. Синтаксис команды следующий:

```
smbmount share_name mount_point options
```

Здесь *mount_point* задает каталог, как в команде *mount*. *share_name* следует формату Windows Universal Naming Convention (UNC), за исключением того, что обратная косая черта заменяется прямой. Например, если нужно монтировать общий ресурс SMB на компьютере с именем maya, экспортируемый под именем *mydocs* в каталог */windocs*, можно воспользоваться командой:

```
# smbmount //maya/mydocs/ /windocs
```

Если для доступа к ресурсу необходимы имя пользователя и/или пароль, *smbmount* предложит их ввести. Теперь рассмотрим более сложный пример команды *smbmount*:

```
# smbmount //maya/d /maya-d/ \
-o credentials=/etc/samba/pw,uid=jay,gid=jay,fmask=600,dmask=700
```

Здесь мы воспользовались параметром *-o*, чтобы задать опции для монтирования ресурса. В строке параметра мы сначала задаем файл регистрационных данных, содержащий имя пользователя и пароль, необходимые для доступа к ресурсу. Это устраняет необходимость каждый раз вводить их интерактивным образом. Формат файла регистрационных данных очень прост:

```
username=USERNAME
password=PASSWORD
```

Здесь *USERNAME* и *PASSWORD* заменяются именем пользователя и паролем, необходимыми для аутентификации на сервере рабочей группы или домена Windows. Параметры *uid* и *gid* указывают владельца и группу, которые должны применяться к файлам общего ресурса, как при монтировании раздела MS-

DOS, описанном в предыдущем разделе. Разница в том, что здесь разрешается использовать как имена пользователя и группы, так и числовые UID и GID. Параметры *fmask* и *dmask* позволяют соединять маски прав доступа с правами, назначаемыми системой, предоставляющей совместный ресурс, через логическое AND. Более подробное описание этих параметров и их использования можно найти на странице руководства *smbmount*(8).

Одна из проблем *smbmount* связана с тем, что при неудачной попытке смонтировать совместно используемый каталог программа не сообщает о том, что именно произошло. Для диагностики проблемы попробуйте получить доступ к общему ресурсу через программу *smbclient*, которая тоже входит в пакет Samba. *smbclient* позволяет вывести содержимое совместно используемого каталога и копировать файлы в него и из него, а также позволяет получить несколько более подробные сообщения об ошибках. Дополнительные сведения можно найти на странице руководства для *smbclient*(1).

После успешного монтирования совместно используемого каталога с помощью *smbmount* можно включить в файл */etc/fstab* запись, позволяющую автоматически монтировать ресурс во время начальной загрузки системы. Для этого нужно просто скопировать аргументы приведенной команды *smbmount*, создав из них в */etc/fstab* запись такого вида:

```
//maya/d  /maya-d  smbfs/
credentials=/etc/samba/pw,uid=jay,gid=jay,fmask=600,dmask=700 0 0
```

Использование Samba для обслуживания разделяемых ресурсов SMB

Мы научились монтировать совместно используемые ресурсы Windows на машине Linux. Теперь рассмотрим организацию противоположной функции: предоставление файлов, хранящихся в Linux, клиентам Windows, находящимся в сети. Это также делается с помощью Samba.

Samba имеет множество применений и хорошо масштабируется. Можно воспользоваться этим пакетом, чтобы сделать файлы Linux-системы доступными одному клиенту Windows (например, при выполнении Windows в среде виртуальной машины на портативном компьютере под Linux). Можно организовать с помощью Samba надежный и высокопроизводительный сервер файлов и печати для сети с тысячами клиентов Windows.

Хотим предостеречь вас перед погружением в замечательный мир Samba. Протокол SMB весьма сложен, а т. к. Samba должна работать со всеми его возможностями, существует огромное количество настроек. В этом разделе мы покажем простую настройку Samba, использующую максимум значений по умолчанию. Если вы действительно серьезно настроились обслуживать большое количество пользователей, использующих все типы Windows, и не ограничиваться базовыми функциями Samba, мы настоятельно рекомендуем тщательно прочитать документацию по Samba и может даже почитать хорошую книгу об этой программе, например «Using Samba» (O'Reilly).

Настройка Samba включает следующие шаги:

1. Компиляция и установка программ Samba, если они еще не установлены на вашей системе.
2. Написание файла конфигурации Samba *smb.conf* и проверка его на корректность.
3. Запуск двух демонов Samba – *smbd* и *nmdb*.

Если вы успешно установили сервер Samba, он появится в списках просмотра пользователей Windows вашей локальной сети вместе с выделенными для совместного доступа каталогами; обычно доступ к списку осуществляется щелчком по пиктограмме Network Neighborhood (Сетевое окружение) или My Network Places на рабочем столе Windows. Пользователи этих систем смогут осуществлять чтение и запись файлов из сетевых каталогов точно так же, как они это делают на своих локальных машинах или сервере Windows, но в соответствии с вашими настройками безопасности. Сервер Samba будет виден как еще одна Windows-машина в сети и работать почти так же.

Установка Samba

Установить Samba на машине Linux можно двумя способами:

- Из двоичного пакета, например Red Hat RPM (применяемого также в SuSE и некоторых других дистрибутивах) или в формате пакетов .deb дистрибутива Debian.
- Скомпилировав дистрибутив исходного кода Samba.

Большинство дистрибутивов Linux содержит Samba, позволяя установить этот пакет, просто выбрав нужный параметр при установке Linux. Если Samba не устанавливалась вместе с системой, обычно нетрудно провести установку пакета позднее. В любом случае файлы пакета Samba обычно размещаются следующим образом:

- Демоны помещаются в */usr/sbin*
- Утилиты командной строки помещаются в */usr/bin*
- Файлы настройки помещаются в */etc/samba*
- Файлы журналов помещаются в */var/log/samba*

Возможны некоторые отклонения. Например, в старых версиях файлы журналов могут находиться в */var/log*, а файлы конфигурации Samba – в */etc*.

Если в ваш дистрибутив не входит Samba, можно загрузить исходный код, скомпилировать его и установить самостоятельно. В этом случае все файлы, входящие в Samba, устанавливаются в подкаталоги */usr/local/samba*.

В любом случае можно взглянуть в перечисленные каталоги, чтобы проверить, установлена ли Samba на вашей машине, и если да, то каким образом.

Если вы – не единственный системный администратор на вашей машине Linux, будьте осторожны. Другой администратор может использовать дистрибутив исходного кода для обновления версии, установленной из двоичного пакета, и наоборот. В таких случаях файлы могут обнаружиться в двух разных местах, и вы не сразу определите, которая из установок является активной в данный момент.

При необходимости установить Samba можно воспользоваться одним из пакетов, созданных для вашего дистрибутива, либо выполнить установку из исходного кода. Устанавливать двоичную версию обычно удобнее, но двоичные пакеты Samba поставщиков дистрибутивов Linux обычно значительно отстают от последних разработок. Даже если в вашей системе Linux уже стоит и работает Samba, можно обновить ее до последней стабильной версии исходного кода.

Для установки из исходного кода зайдите на веб-сайт Samba на *http://www.samba.org* и щелкните по одной из ссылок для загрузки с ближайшего к вам сайта. Вы попадете на один из зеркальных сайтов для загрузки по FTP. Самая свежая стабильная версия исходного кода находится в файле *samba-latest.tar.gz*. После загрузки этого файла распакуйте его и прочтите файл *docs/htmldocs/UNIX_INSTALL.html*, в котором будут подробные инструкции относительно компиляции и установки Samba. Если говорить кратко, вам потребуется выполнить следующие команды:

```
$ tar xvfz samba-latest.tar.gz
$ cd samba-VERSION
$ su
Password:
# ./configure
# make
# make install
```

Перед выполнением сценария *configure* зарегистрируйтесь как суперпользователь. Samba несколько более требовательна в этом отношении, чем большинство других пакетов Open Source. После выполнения перечисленных выше команд файлы Samba окажутся в следующих каталогах:

- Исполняемые модули в */usr/local/samba/bin*
- Файлы конфигурации в */usr/local/samba/lib*
- Файлы журналов в */usr/local/samba/log*
- Файл smbpasswd в */usr/local/samba/private*
- Страницы руководства в */usr/local/samba/man*

Вам потребуется добавить каталог */usr/local/samba/bin* в переменную окружения PATH, чтобы запускать утилиты Samba, не указывая полного пути к ним. Кроме того, в файл */etc/man.config* нужно будет добавить следующие две строки, чтобы команда *man* смогла найти страницы руководства Samba:

```
MANPATH /usr/local/samba/man
MANPATH_MAP /usr/local/samba/bin /usr/local/samba/man
```

Настройка Samba

Следующий шаг – создание файла *smb.conf*, подходящего для вашей системы. Многие программы в дистрибутиве Samba читают файл конфигурации, и хотя некоторые из них могут обойтись минимумом находящейся в нем ин-

формации (и даже пустым файлом), демоны, ответственные за совместный доступ к файлам, требуют полного задания файла конфигурации.

Имя и местонахождение файла конфигурации Samba зависят от того, как компилировался и устанавливался пакет. Это можно легко выяснить с помощью команды *testparm*, как было показано ранее в этой главе в разделе, посвященном монтированию совместно используемых каталогов. Обычно этот файл называется *smb.conf*, и мы будем пользоваться в дальнейшем этим именем.

Формат файла *smb.conf* похож на формат файлов *.ini*, используемых в Windows 3.x; в нем есть записи типа:

```
key = value
```

При работе с Samba ключи почти всегда называют *параметрами* или *опциями*. Параметры группируются в секции, которые начинаются с меток, образуемых именем секции, заключенным в квадратные скобки. Название секции размещается на отдельной строке, например:

```
[section-name]
```

Каждый каталог или принтер, предоставляемый в совместное использование, в терминологии сетей Windows называется разделяемым *ресурсом* (*share*) или *сервисом* (*service*). Каждый сервис можно задать отдельно с помощью секции с особым именем, но мы покажем несколько способов, как упростить файл конфигурации и организовать поддержку множества сервисов всего в нескольких секциях. Специальная секция `[global]` содержит параметры, применяющиеся по умолчанию ко всем сервисам и к серверу в целом. Хотя Samba понимает буквально сотни параметров, скорее всего, вы будете использовать лишь несколько, т. к. большинство из них имеют разумные значения по умолчанию. Если вы хотите узнать, какие ключи имеются вообще, или вы ищете какой-то особый параметр, можно посмотреть страницы руководства для *smb.conf*(5). Но для начала создадим следующий файл *smb.conf*:

```
[global]
    workgroup = METRAN
    encrypt passwords = yes
    wins support = yes
    local master = yes

[homes]
    browsable = no
    read only = no
    map archive = no

[printers]
    printable = yes
    printing = BSD
    path = /var/tmp

[data]
```

```
path = /export/data
read only = no
map archive = no
```

Это очень простая конфигурация, Но в большинстве случаев ее может оказаться достаточно. Мы опишем каждую секцию в порядке ее появления в файле, чтобы вы смогли разобраться с ее значением и сделать изменения, необходимые для вашей системы. Те части, которые, скорее всего, потребуется изменить, выделены жирным шрифтом.

Секция `[global]` содержит параметры, настраивающие Samba в целом. Например, параметр `workgroup` указывает рабочую группу, в которую входит сервер Samba. Название рабочей группы `METRAN` нужно заменить реальным именем группы. Если на ваших машинах Windows уже задана рабочая группа, нужно воспользоваться ее именем. Если нет – задайте здесь имя новой рабочей группы и настройте все машины Windows для его использования. Не пользуйтесь устанавливаемым Windows по умолчанию именем `WORKGROUP`, чтобы избежать конфликтов с неправильно сконфигурированными машинами.

В качестве имени компьютера для сервера (называемого также именем NetBIOS) воспользуемся предлагаемым в Samba по умолчанию именем хоста машины. Это значит, что если полное доменное имя машины *dolphin.example.com*, то в Windows она будет видна как *dolphin*. Убедитесь, что имя хоста вашей машины установлено в соответствии с этим.

Параметр `encrypt passwords` сообщает Samba, что клиенты будут посылать пароли в «зашифрованном» виде, а не открытым текстом. Это необходимо для работы Samba с Windows 98, Windows NT Service Pack 3 и последующими версиями. Если вы пользуетесь Samba версии 3.0 и выше, то эта строка необязательна, поскольку новые версии Samba используют зашифрованные пароли по умолчанию.

Параметр `wins support` требует от Samba действовать как сервер WINS, разрешая имена компьютеров в IP-адреса. Это необязательно, но способствует эффективной работе сети.

Параметр `local master` тоже необязателен. Он определяет, что Samba будет функционировать в качестве главного броузера подсети и хранить главный список компьютеров, действующих как серверы SMB, и их разделяемых ресурсов. Обычно лучше предоставить эту роль Samba, а не машине под Windows.

Остальные секции в нашем примере *smb.conf* необязательны и задают ресурсы, которые Samba предоставляет сети.

Секция `[homes]` настраивает Samba на автоматическое выделение в совместное пользование личных каталогов пользователей. При подключении клиента к серверу Samba происходит поиск его имени пользователя в файле Linux */ etc/passwd* и определяется, есть ли у клиента учетная запись в системе. Если учетная запись есть и с ней связан домашний каталог, то этот каталог предлагается пользователю в качестве разделяемого ресурса. Имя пользователя станет именем этого ресурса, который будет выглядеть как папка у клиента Windows. Например, если у пользователя `diane` есть учетная запись

на машине Samba, и он соединяется с сервером Samba, этот пользователь увидит свой домашний каталог, находящийся на машине Linux, как папку *diane* с совместным доступом.

Параметры секции `[homes]` определяют режим, в котором будут предоставляться домашние каталоги пользователей. Для того чтобы папка *homes* не показывалась в списке просмотра, необходимо задать `browsable = no`. По умолчанию Samba предоставляет разделяемые папки с правом доступа только на чтение. При установке `read only = no` папка и ее содержимое могут быть предоставлены клиенту в режиме чтения/записи. Устанавливая такие права доступа в определении совместного ресурса для файлов Linux, вы лишь налагаете дополнительные ограничения: файл, который разрешено только читать на сервере, не станет доступным для записи через сеть, если установить для него параметр `read only`, равным `no`. Аналогично, если в системе Linux файл имеет разрешение на чтение/запись, то при обращении к нему сетевых клиентов Samba действует доступ Samba по умолчанию, разрешающий только чтение.

Samba выполняет не всегда простую задачу – заставить файловую систему Unix выглядеть как файловая система Windows для клиентов Windows. Одно из различий между файловыми системами Windows и Unix состоит в том, что в Windows есть атрибут «archive», с помощью которого программы резервного копирования определяют, был ли некий файл модифицирован после предыдущего копирования. В случае инкрементального резервного копирования создаются копии только тех файлов, у которых установлен бит «archive». В Unix такая информация обычно следует из временной отметки модификации файла, и прямого аналога атрибута «archive» нет. Samba моделирует атрибут архивации с помощью бита выполнения для владельца файла Unix. Это позволяет программам резервного копирования Windows корректно работать с разделяемыми ресурсами Samba, но побочный эффект состоит в том, что файлы данных выглядят в Linux-системе как выполняемые модули. Мы устанавливаем параметр `map archive` равным `no`, потому что предполагаем, что вас больше интересует организация правильной работы вашей Linux-системы, чем возможность выполнять резервное копирование с помощью приложений Windows.

В секции `[printers]` Samba получает указание сделать серверы, подключенные к машине Linux, доступными клиентам сети. Каждая секция *smb.conf*, включая данную, в которой определен совместно используемый принтер, должна содержать строку `printable = yes`. Для доступа к принтеру нужно создать запись в системном файле Linux */etc/printcap*. Как говорилось в разделе «Управление службами печати» главы 8, файл printcap перечисляет все файлы в системе и способ доступа к ним. Принтер будет виден пользователям в сети под именем, которое указано для него в файле printcap.

Если вы уже сконфигурировали принтер для работы, при выделении его в сетевой ресурс он может работать неправильно. Обычно при настройке принтера под Linux очередь печати связывается с драйвером принтера, который транслирует данные, получаемые от приложений, в коды, имеющие значение для конкретного используемого принтера. Однако у клиентов Windows

есть свои драйверы принтеров, и они предполагают, что принтер удаленной системы принимает файлы необработанных данных, предназначенные для использования непосредственно принтером без какой-либо промежуточной обработки. Решение заключается в том, чтобы добавить для принтера еще одну очередь печати (или создать ее, если принтер еще не сконфигурирован), которая передает данные непосредственно на принтер. Иногда это называется «raw mode» (режим неформатированных данных).

Первый раз, когда клиент Windows обращается к принтеру, ему требуется установить Windows-драйвер принтера. Эта процедура не отличается от установки принтера, подключенного непосредственно к машине клиента. Когда документ печатается на машине-клиенте Windows, он обрабатывается драйвером принтера, а затем посылается Samba. Samba просто помещает файл в очередь печати принтера, а система печати Linux завершает остальное. Исторически в большинстве дистрибутивов Linux устанавливается система печати в стиле BSD, поэтому мы задаем `printing = BSD`, чтобы известить Samba об использовании системы BSD. После этого Samba действует соответствующим образом, выдавая надлежащие команды системе печати. В последнее время некоторые дистрибутивы Linux стали использовать систему печати LPRng или CUPS. Если дистрибутив использует LPRng, установите `printing = LPRNG`. Если используется CUPS, установите `printing = CUPS`, а также `printcap name = CUPS`.

Мы установили параметр `path`, равным */var/tmp*, чтобы сообщить Samba, куда временно помещать двоичные файлы, получаемые от сетевого клиента, прежде чем помещать их в очередь печати системы. При желании можно использовать другой каталог. Этот каталог должен быть общедоступен по записи, чтобы все клиенты имели доступ к принтеру.

Разделяемый ресурс `[data]` в нашем примере показывает, как открыть совместный доступ к каталогу. Следуя этому примеру, можно добавить сколько угодно разделяемых каталогов, указав для каждого свое имя секции и значение `path`. Имя секции используется в качестве имени разделяемого ресурса и показывается клиентам Windows как папка с таким именем. Как и в предыдущих секциях, мы воспользовались `read only = no`, чтобы разрешить доступ к ресурсу по чтению/записи, и `map archive = no`, чтобы для файлов не устанавливался бит выполнения. Параметр `path` сообщает Samba, какой каталог в системе Linux выделяется для совместного использования. Каталог может быть любым, но он должен существовать, и права доступа к нему должны соответствовать предполагаемому назначению. Для нашего разделяемого ресурса `[data]` каталог */export/data* имеет разрешения на чтение, запись и выполнение, установленные для всех пользователей, группы и прочих, поскольку это совместно используемый каталог общего значения, которым может пользоваться каждый.

Закончив создание файла *smb.conf*, выполните программу *testparm*, которая проверит наличие ошибок и несовместимостей в *smb.conf*. Если файл *smb.conf* правильный, *testparm* должна возвратить удовлетворительные сообщения, например:

```
$ testparm
Load smb config files from /usr/local/samba/lib/smb.conf
Processing section "[homes]"
Processing section "[printers]"
Processing section "[data]"
Loaded services file OK.
Press enter to see a dump of your service definitions
```

Если при создании файла *smb.conf* сделаны существенные ошибки, вы получите сообщения об ошибках вперемешку с показанной выдачей. Не требуется делать в этом случае дамп определений служб, поэтому просто введите <Ctrl>+<C> и завершите *testparm*.

Добавление пользователей

Сетевые клиенты должны быть аутентифицированы Samba, чтобы получить доступ к совместным ресурсам. В конфигурации данного примера Samba использует безопасность на уровне пользователя; от клиентов требуется передать имя пользователя и пароль, которые должны соответствовать имеющимся на хосте Linux. Первый шаг в добавлении нового пользователя Samba – обеспечить наличие у этого пользователя учетной записи Linux, а если в вашем *smb.conf* есть ресурс `[homes]`, то существование у этой учетной записи исходного каталога.

Кроме того, в Samba есть свой файл паролей, служащий для проверки зашифрованных паролей, получаемых от клиентов. Для каждого пользователя Samba нужно выполнить команду *smbpasswd*, чтобы добавить учетную запись Samba для этого пользователя:

```
# smbpasswd -a username
New SMB password:
Retype new SMB password:
```

Следите, чтобы имя пользователя и пароль, передаваемые *smbpasswd*, были такими же, как в учетной записи пользователя Linux. Предлагаем начать с добавления собственной учетной записи, которой можно будет потом воспользоваться для тестирования правильности установки.

Запуск демонов Samba

В дистрибутиве Samba содержится две программы-демон, *smbd* и *nmbd*, и обе они должны выполняться для функционирования Samba. Запуск этих демонов прост:

```
# smbd
# nmbd
```

Если в файле *smb.conf* нет ошибок, то затруднений с запуском демонов быть не должно. Тем не менее можно запустить команду *ps ax* и проверить, что эти демоны есть в списке активных процессов. Если их нет, проверьте, нет ли сообщений об ошибках в файлах журналов Samba, *log.smbd* и *log.nmbd*.

Чтобы остановить демоны, можно выполнить команду *killall* и послать им сигнал SIGTERM:

```
# killall -TERM smbd nmbd
```

Если вы уверены в правильности конфигурации, то, вероятно, захотите запускать демоны Samba во время начальной загрузки вместе с другими системными демонами. Если вы установили двоичный дистрибутив Samba, то в каталоге */etc/init.d* может быть сценарий, запускающий и останавливающий Samba. Например, в Red Hat и SuSE Linux можно запустить Samba следующей командой:

```
# /etc/init.d/smb start
```

Остановить или перезапустить Samba можно с помощью сценария *smb*, заменив аргумент `start` на *stop* или *restart*. Имя и местонахождение этого сценария могут быть различными в разных дистрибутивах. В Debian 3.0 этот сценарий именуется *samba*, а в более старых версиях Red Hat он находится в */etc/rc.d/init.d*.

Если вы выполнили установку из дистрибутива исходного кода, вам придется написать и установить собственный сценарий, который может выполнять функции запуска и останова. (Либо можно скопировать сценарий из двоичного пакета Samba в вашем дистрибутиве.) При запуске из сценария *smbd* и *nmbd* могут принимать параметр *-D*, который определяет их выполнение в качестве демонов.

После того как вы протестировали сценарий и уверены в том, что он работает, создайте соответствующие символические ссылки в своих каталогах */etc/rcN.d*, чтобы запускать Samba на обычном для вас уровне исполнения и останавливать Samba при переходе на другие уровни исполнения.

Теперь, после установки, настройки и запуска Samba, попробуйте выполнить команду *smbclient*, чтобы получить доступ к одному из совместно используемых каталогов:

```
$ smbclient //localhost/data
added interface ip=172.16.1.3 bcast=172.16.1.255 nmask=255.255.255.0
Password:
Domain=[METRAN] OS=[Unix] Server=[Samba 2.2.5]
smb: \>
```

В ответ на приглашение `smb: \>` можете ввести любую команду *smbclient*. Попробуйте ввести команду *ls* и перечислить содержимое каталога. Затем попробуйте команду *help*, которая покажет вам все имеющиеся команды. Программа *smbclient* весьма похожа на *ftp*, поэтому, если вы привыкли к *ftp*, то не должны испытывать трудностей. Теперь завершите *smbclient* (с помощью команды *quit* или *exit*) и попробуйте другие варианты. Сначала задайте вместо `localhost` имя хоста сервера и убедитесь в правильном функционировании разрешения имен. Затем попробуйте получить доступ к своему домашнему каталогу, используя имя пользователя вместо `data`.

А теперь нечто более интересное: перейдите на Windows-машину и зарегистрируйтесь с именем пользователя и паролем вашей учетной записи на Samba. (В Windows NT/2000/XP нужно добавить учетную запись для нового пользователя с именем и паролем из учетной записи Samba.) Сделайте двойной щелчок по значку Network Neighborhood или My Network Places на рабочем столе. Найдите в сети свою рабочую группу и дважды щелкните по ее значку. Вы должны увидеть в открывшемся окне значок для своего сервера Samba. Дважды щелкнув по нему, вы откроете окно, в котором будут показаны ваш исходный каталог, принтер и разделяемые ресурсы *data*. Теперь можно перетаскивать мышью файлы между вашим исходным каталогом и разделяемыми ресурсами, а после установки драйвера для совместно используемого принтера отправлять задания печати Windows на ваш принтер Linux!

Мы коснулись лишь малой части того, что может делать Samba, но и по ней можно получить представление, почему Samba способствовала известности Linux, хотя этот пакет создан не специально для Linux.

Утилиты преобразования файлов

Одна из самых значительных проблем, возникающих при совместном использовании файлов в Linux и Windows, связана с различием принятых окончаний строк в текстовых файлах. К счастью, есть несколько способов решения этой проблемы:

- Если вы обращаетесь к файлам на смонтированном разделе на той же машине, можно разрешить ядру автоматически преобразовывать файлы, как это описано в разделе «Совместный доступ к разделам» в этой главе. Но пользуйтесь этим аккуратно!
- Стандартные редакторы, такие как Emacs и *vi*, могут автоматически выполнять преобразование во время загрузки или сохранения файла.
- Имеется большое количество утилит, выполняющих преобразование файлов из одного формата окончания строк в другой. Некоторые из них могут выполнять и другие типы преобразований
- Напишите собственную утилиту преобразования на том языке программирования, который вам больше нравится.

Если вас интересует только преобразование символов перевода строки, то написать программу для преобразования файлов очень просто. Чтобы преобразовать из формата DOS в формат Unix, замените все вхождения CRLF (`\r\f` или `\r\n`) в файле на символ перевода строки (`\n`). В обратном направлении, замените все переводы строки на CRLF. Приведем две программы на Perl, которые решают эту задачу. Первая из них, которую назовем *d2u*, преобразует файлы из формата DOS в формат Unix:

```
#!/usr/bin/perl
while (<STDIN>) { s/\r$//; print }
```

Следующая программа (которую мы назовем *u2d*) преобразует файл из формата Unix в формат DOS:

```
#!/usr/bin/perl
while (<STDIN>) { s/$/\r/; print }
```

Обе команды читают входной файл со стандартного устройства ввода и пишут выходной файл на стандартное устройство вывода. Легко модифицировать примеры с тем, чтобы имена входного и выходного файла задавались в командной строке. Если вам лень писать такие утилиты самому, можете поискать в своем дистрибутиве Linux программы *dos2unix* и *unix2dos*, которые действуют аналогично нашим простым *d2u* и *u2d*, а также принимают имена файлов в командной строке. Другая аналогичная пара утилит – *fromdos* и *todos*. Если не найдете ни одной из них, попробуйте команду *flip*, которая может преобразовывать файлы в обоих направлениях.

Если эти простенькие утилиты кажутся вам недостаточно мощными, можете попробовать *recode*. Эта программа преобразует практически любые стандарты текстовых файлов.

Простейший способ использования *recode* предполагает указание прежнего и нового набора символов (кодировки текстового файла) и файла для преобразования. *recode* заменит старый файл на преобразованный файл с тем же именем. Например, для конвертации текстового файла из Windows в Unix можно ввести:

```
recode ibmpc:latin1 textfile
```

textfile будет заменен на преобразованную версию. Нетрудно догадаться, что для преобразования этого файла обратно в кодировку Windows надо ввести:

```
recode latin1:ibmpc textfile
```

В добавление к кодировке `ibmpc` (используемой в Windows) и `latin1` (для Unix) имеются и другие возможные варианты, среди которых есть, например `latex` для кодировки диакритических знаков в стиле LATEX (см. главу 9) и `texte` для кодировки почтовых сообщений на французском языке. Полный список можно получить, введя команду:

```
recode -l
```

Если вам не нравится привычка *recode* заменять ваш старый файл новым, можно воспользоваться тем фактом, что *recode* может также читать из стандартного потока ввода и писать в стандартный поток вывода. Для преобразования файла *dostextfile* в *unixtextfile* без удаления *dostextfile* можно выполнить:

```
recode ibmpc:latin1 < dostextfile > unixtextfile
```

Другие форматы документов

С помощью описанных выше утилит можно достаточно комфортно управляться с текстовыми файлами, но это только начало. Например, растровая графика в Windows обычно сохраняется как файлы *bmp*. К счастью, имеются средства преобразования файлов *bmp* в графические файлы формата *png* или *xpm*, которые чаще используются на Unix. Среди них есть Gimp, который, возможно, входит в ваш дистрибутив.

Ситуация осложняется, когда дело доходит до других файловых форматов, например, используемых в офисных приложениях. В то время как в Windows разные инкарнации формата *doc*, используемого текстовым процессором Word, превратились de facto в общепринятый язык текстовых процессоров Windows, до последнего времени, прочесть такие файлы в Linux было почти невозможно. К счастью, недавно появились программы, способные читать (и иногда даже выводить) файлы *.doc*. Среди них офисный пакет KOffice, свободно распространяемый OpenOffice, и коммерческий пакет StarOffice 6.0, близкий родственник OpenOffice. Однако следует знать, что такое преобразование никогда не будет безупречным, и, скорее всего, после него придется редактировать файл вручную. Даже в Windows преобразования не бывают корректными на 100%; если вы попытаетесь импортировать файл Microsoft Word в WordPerfect (или наоборот), то поймете, о чем мы говорим.

Вообще, чем более популярным является формат файла в Windows, тем больше вероятность, что разработчики Linux предоставят способы для чтения или даже записи файлов этого формата. Другой выход – перейти при создании файлов в Windows на открытые файловые форматы, такие как RTF (Rich Text Format) или XML (Extensible Markup Language). В эпоху Интернета, где потоки информации текут свободно, использование закрытых недокументированных файловых форматов – это анахронизм.

Выполнение программ MS-DOS и Windows под Linux

Если вы работаете в Windows главным образом потому, что она поддерживает специфическое периферийное или аппаратное устройство, то обычно лучше всего установить систему с загрузкой альтернативных ОС или выполнять Windows на отдельной машине, где можно будет непосредственно обращаться к аппаратным ресурсам. Но если задача в том, чтобы выполнять программы, написанные для Windows, то идеальным решением было бы выполнять эти приложения под Linux, чтобы не требовалось перегружать систему или переходить на другой компьютер.

Был сделан целый ряд попыток решить эту задачу, как группами разработчиков Open Source, так и коммерческими разработчиками. Простейшее решение предоставляет Dosemu (*http://www.dosemu.org*), который эмулирует аппаратные средства PC достаточно хорошо для выполнения MS-DOS (или совместимой системы типа PC-DOS или DR-DOS). DOS все равно требуется инсталлировать в эмуляторе, но поскольку DOS фактически выполняется внутри эмулятора, это обеспечивает хорошую совместимость приложений. С некоторыми ограничениями, но можно выполнять даже Windows 3.1

Wine (*http://www.winehq.com*) – это более амбициозный пакет, ставящий задачей воспроизвести Microsoft Win32 API, чтобы позволить программам Windows выполняться непосредственно под Linux без накладных расходов, связанных с эмуляцией. Это означает, что Windows-приложениям для работы не потребуется Windows. Однако, хотя группа разработчиков Wine до-

стигла поразительного прогресса, сложность решаемой задачи столь велика, что количество приложений, способных выполняться под Wine, очень ограничено.

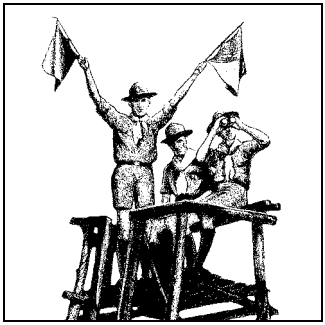
Есть другой проект open source под названием Bochs (*http://bochs.sf.net*), который эмулирует аппаратную часть PC достаточно хорошо для работы Windows и других операционных систем. Однако, поскольку каждая инструкция 386-го процессора эмулируется программно, производительность сокращается в итоге до малой части той, которую можно было бы получить при непосредственном выполнении операционной системы на той же аппаратуре.

Проект Plex86 (*http://savannah.nongnu.org/projects/plex86*) использует еще один подход, реализуя виртуальную среду, в которой могут выполняться Windows или другая операционная система (или их приложения). Программы, выполняемые на виртуальной машине, работают без снижения скорости за исключением тех случаев, когда нужно обращаться к аппаратным устройствам. Это весьма напоминает Dosemu, но реализация более надежна и не ограничивается выполнением DOS.

На момент написания этой книги все проекты, которые обсуждались в данном разделе, были довольно незрелыми и имели существенные ограничения. Грубо говоря, здесь вполне применимы выражения «как повезет» и «за что заплатишь, то и получишь».

Возможно, вам более повезет с коммерческим продуктом, таким как VMware (*http://www.vmware.com*) или Win4Lin (*http://www.win4lin.com*). Оба основаны на реализации среды виртуальной машины (как в Plex86), поэтому для выполнения Windows-приложений нужно установить экземпляр Windows. По крайней мере в отношении VMWare известно, что степень совместимости очень высока. VMWare поддерживает версии DOS/Windows от MS-DOS до .NET, включая все промежуточные. Можно даже установить несколько из известных дистрибутивов Linux, чтобы запускать на одном компьютере несколько экземпляров Linux. С разной степенью ограничений могут выполняться и другие операционные системы, в том числе FreeBSD, Netware и Solaris. Хотя при этом неизбежны некоторые накладные расходы, но современные процессоры с гигагерцевыми тактовыми частотами могут обеспечить приемлемые уровни производительности для большинства стандартных приложений, таких как офисные программы.

Win4Lin – более новая разработка, чем VMWare. Когда писалась эта книга, Windows и ее приложения быстрее выполнялись под Win4Lin, чем под VMWare, но поддержка существовала только для Windows 95/98/ME, а не для Windows NT/2000/XP. Как и в отношении других проектов, описывавшихся в данном разделе, мы советуем следить за состоянием разработки этого продукта, чтобы определить, когда он созреет до такой степени, что будет способен удовлетворить ваши потребности.



13

Языки программирования

Использование Linux предполагает значительно больше, чем простую работу в системе. Одно из преимуществ свободно распространяемого программного обеспечения состоит в том, что его можно модифицировать в соответствии с потребностями пользователя. Это в равной степени относится как к многочисленным свободно распространяемым программам для Linux, так и к самому ядру системы.

Linux поддерживает развитый интерфейс программирования, используя компиляторы и средства GNU, такие как компилятор *gcc*, отладчик *gdb* и другие. Поддерживается ряд языков программирования, включая Perl, Python и LISP. В чем бы вы ни программировали, Linux является хорошей средой разработки приложений для Unix. Поскольку доступен полный исходный код библиотек и ядра Linux, те программисты, которым необходимо покопаться во внутренностях системы, могут это сделать.[1]

Linux является идеальной платформой для разработки приложений, которые должны выполняться в X Window System. Дистрибутив Linux X, как отмечалось в главе 10, является полной реализацией со всем необходимым для разработки и поддержки приложений X. Само программирование для X позволяет получать переносимый код, поэтому ваши приложения в той части, которая относится к X, должны корректно компилироваться на других Unix-системах.

В этой главе мы исследуем среду программирования Linux и дадим беглый обзор многочисленных средств, которые она предоставляет. Успех программирования в Unix наполовину зависит от знания того, какие имеются инструменты и как их эффективно использовать. Часто самые полезные функции этих средств не очевидны новичкам.

---

[1] Во многих Unix-системах авторы сталкивались с недостаточностью имеющейся документации. В Linux можно изучать сам исходный код ядра, библиотек и системных утилит. Возможность доступа к исходному коду более важна, чем предполагает большинство программистов.

Поскольку программирование на C лежит в основе большинства крупных программных проектов (хотя в настоящее время оно все более уступает программированию на C++) и C знаком большинству современных программистов не только в Unix, но и в других операционных системах, мы начнем с рассказа о том, какие средства есть для этого языка. В первых разделах этой главы предполагается, что вы программируете на C.

Однако существуют и другие средства, которые приобретают все большее значение, особенно для системного администрирования. В этой главе мы рассмотрим одно из них – Perl. Perl – язык сценариев, которые используются в оболочках Unix. Он берет на себя тяжелую работу по управлению памятью, благодаря чему вы можете сосредоточиться непосредственно на своей задаче. Perl достаточно развит, что обеспечивает ему большую мощь, чем могут предоставить языки оболочки, и делает пригодным для решения различных задач программирования.

Многие программисты в восторге от работы с Java – новым языком, разработанным Sun Microsystems. Хотя для большинства программистов Java ассоциируется с интерактивными программами (апплетами) на веб-страницах, тем не менее, это язык общего назначения с многочисленными возможностями использования в Интернете. В одном из последующих разделов мы сравним Java с более старыми языками программирования и покажем, как начать с ним работать.

Программирование для gcc

Язык программирования C используется значительно чаще других при разработке программ для Unix. Возможно, это связано с тем, что сама система Unix была первоначально разработана на C: это родной язык Unix. Компиляторы C для Unix традиционно определяют стандарты интерфейсов для других языков и средств, таких как компоновщики, отладчики и другие. Соглашения, предложенные в первоначальных компиляторах C, довольно последовательно соблюдаются во всех средствах программирования для Unix.

Компилятор GNU C – *gcc* – является одним из наиболее гибких и развитых из имеющихся компиляторов. В отличие от других компиляторов C (например, поставляемых в дистрибутивах AT&T или BSD или сторонними разработчиками), *gcc* поддерживает все современные стандарты C, такие как стандарт ANSI C, а также собственные многочисленные расширения. К счастью, *gcc* совместим со старыми компиляторами C и старыми стилями программирования на C. Есть даже средство с названием *protoize*, позволяющее писать прототипы функций для программ C в старом стиле.

gcc является также компилятором C++. Для тех, кто предпочитает сложную объектно-ориентированную среду, поддерживается C++ со всеми его художествами, включая возможности, появившиеся при выходе стандарта C++, такие как шаблоны методов. Имеются также полные библиотеки классов C++, такие как стандартная библиотека шаблонов (STL).

Для тех, кто имеет склонность к тому, что понятно лишь посвященным, *gcc* предоставляет Objective-C – объектно-ориентированную разновидность C, не получившую особого распространения, но получившую второе дыхание благодаря применению в Mac OS X. Кроме того, есть *gcj*, компилирующий код Java в машинный код. И этим, как мы увидим, дело не ограничивается.

В этом разделе мы рассмотрим использование *gcc* для компиляции и компоновки программ в Linux. Мы предполагаем, что вы знакомы с программированием на C/C++, но не предполагаем знакомства со средой программирования Unix. Ее мы и собираемся здесь представить.

На момент написания книги последняя версия *gcc* имела номер 3.0.4. Однако серия 3.0 все еще весьма нестабильна, поэтому официальной стандартной версией по-прежнему считается 2.95.3. Мы советуем пользоваться ею, если нет серьезных оснований для иного.

Краткий обзор

Прежде чем сообщить необходимые подробности собственно о *gcc*, мы приведем простой пример, на котором покажем те шаги, через которые проходит компиляция программ на C в Unix.

В качестве примера рассмотрим фрагмент кода избитой программы «Hello, World!»:

```
#include <stdio.h>
int main() {
  (void)printf("Hello, World!\n");
  return 0; /* Для приличия */
}
```

Для того чтобы скомпилировать эту программу в живой выполняемый модуль, нужно сделать несколько шагов. Большую часть их можно осуществить единственной командой *gcc*, но на подробностях мы остановимся в конце главы.

Прежде всего, компилятор *gcc* должен создать *объектный файл* из этого *исходного кода*. Объектный файл является, в сущности, эквивалентом исходного текста C в машинных кодах. В нем содержится код для организации стека вызова *main*(), вызова функции *printf*() и код, возвращающий значение 0.

Следующим шагом является *компоновка* объектного файла с целью создания выполняемого модуля. Эту работу выполняет компоновщик (*linker*, или *редактор связей*). Его задача – взять объектные файлы, слить их с программами, содержащимися в библиотеках, и выдать исполняемый файл. Объектный код, полученный из исходного, не является законченной исполняемой программой. Прежде всего, нужно присоединить код для *printf*(). Кроме того, к выполняемому модулю нужно присоединить некоторые программы инициализации, невидимые простому программисту.

Откуда берется код для *printf*()? Из библиотек. Не удастся долго рассказывать о *gcc*, не упомянув библиотеки. Библиотека состоит, в сущности, из

многочисленных объектных файлов и указателя на них. При поиске кода для *printf*() компоновщик просматривает указатели всех библиотек, компоновку с которыми ему указано произвести. Он находит объектный файл, содержащий функцию *printf*(), извлекает этот объектный файл (весь объектный файл целиком, который может содержать значительно больше, чем одну функцию *printf*()) и присоединяет его к исполняемому модулю.

На самом деле все гораздо сложнее. Как упоминалось, в Linux есть два вида библиотек: *статические* (*static*) и *совместного доступа* (*shared*). В этом примере мы описывали статические библиотеки, т. е. библиотеки, фактический код из которых для вызываемых подпрограмм присоединяется к выполняемому модулю. Однако код для таких подпрограмм, как *printf*(), может быть весьма объемным. Поскольку многие программы используют одни и те же библиотечные подпрограммы, неразумно включать библиотечный код в каждый выполняемый модуль. Вот здесь и появляются библиотеки совместного доступа.[1]

При использовании библиотек совместного доступа код общих подпрограмм содержится в «файле образа» библиотеки на диске. При компоновке программы с библиотекой совместного доступа в исполняемый модуль вместо кода подпрограммы включается *заглушка* (*stub code*). Код, содержащийся в заглушке, сообщает загрузчику программы, где на диске в файле образа можно найти фактический код подпрограммы во время выполнения. Поэтому когда запускается наша дружелюбная программа «Hello, World!», загрузчик обращает внимание, что программа скомпонована с библиотекой совместного доступа. Тогда он находит образ библиотеки совместного доступа и загружает код библиотечных функций, таких как *printf*(), вместе с кодом самой программы. Код заглушки сообщает загрузчику, где в файле образа находится код для *printf*().

Даже такая картина является упрощением того, что происходит в действительности. Библиотеки совместного доступа под Linux используют *таблицы переходов* (*jump tables*), позволяющие обновлять библиотеки и перемешивать их содержимое, при этом нет необходимости перекомпоновывать программы, использующие эти библиотеки. Код заглушки в исполняемом модуле фактически сам ищет другую ссылку в библиотеке, т. е. в таблице переходов. Благодаря этому можно изменять содержимое библиотек и соответствующих таблиц переходов таким образом, что код заглушки может оставаться неизменным.

У библиотек совместного доступа есть еще одно преимущество: возможность обновления. Если кто-нибудь устранит программную ошибку в *printf*() (или брешь в системе защиты, что важнее), вам нужно обновить лишь одну библиотеку, а не перекомпоновывать заново все программы в системе.

---

[1] Следует отметить, что некоторые умные программисты считают, что библиотеки совместного доступа вредны. Основания для такого мнения слишком сложны, чтобы обсуждать их здесь. Они заявляют, что не стоит мучиться, когда на большинстве машин стоят диски по 20 Гбайт и не менее 128 Мбайт оперативной памяти.

Но не позволяйте себе одурманиться этими абстрактными сведениями. В свое время мы возьмем реальный пример и покажем, как компилировать, компоновать и отлаживать программы. На самом деле это очень просто. О большинстве проблем позаботится вместо вас сам компилятор *gcc*. Однако неплохо понимать, что же происходит за кулисами.

Характеристики gcc

У *gcc* столько функций, что мы просто не сможем перечислить их в этой книге. Страница руководства по *gcc* и документ Info дадут вам массу интересных сведений об этом компиляторе. Далее в этом разделе мы дадим обстоятельный обзор наиболее полезных для начала функций *gcc*. С его помощью вы сами сможете определить, как заставить работать в ваших интересах и другие возможности.

Для начинающих *gcc* поддерживает «стандартный» синтаксис C, используемый в настоящее время и определяемый, по большей части, стандартом ANSI C. Самая важная особенность этого стандарта – прототипирование функций. Это означает, что функцию *foo*(), возвращающую значение типа `int` и принимающую два аргумента – `a` (типа `char *`) и `b` (типа `double`), можно определить следующим образом:

```
int foo(char *a, double b) {
  /* здесь находится ваш код... */
}
```

Это отличается от старого синтаксиса (без прототипов) определения функций, который выглядел так:

```
int foo(a, b)
char *a;
double b;
{
  /* здесь находится ваш код... */
}
```

Такой синтаксис тоже поддерживается *gcc*. Конечно, в ANSI C определены многие другие соглашения, но это наиболее заметное для начинающего программиста. Любой, кто знаком со стилем программирования на C по современным книгам, таким как второе издание «The C Programming Language» Кернигана и Ричи, без проблем сможет программировать с помощью *gcc*.

Компилятор *gcc* имеет развитую оптимизацию. В то время как большинство компиляторов C позволяет задавать для оптимизации только один ключ *-0*, *gcc* поддерживает различные уровни оптимизации. На высшем уровне оптимизации *gcc* совершает такие фокусы, как возможность совместного доступа к коду и статическим данным. Это означает, что если в вашей программе есть статическая строка, такая как `Hello, World!`, и ASCII-кодировка этой строки случайно совпала с последовательностью команд в вашей программе, то *gcc* позволяет хранить данные строки и соответствующий код по одному адресу. Неплохо?

Конечно, *gcc* позволяет включать в объектные файлы отладочную информацию, что помогает отладчику, а значит, и программисту, осуществлять трассировку программы. Компилятор вставляет в объектный файл маркеры, дающие возможность отладчику находить в скомпилированной программе нужные строки, переменные и функции. Поэтому при использовании отладчика, например *gdb* (о котором мы поговорим ниже), можно одновременно пошагово выполнять программу и видеть исходный текст.

Среди других приемов *gcc* следует отметить возможность одним щелчком (буквально говоря) сгенерировать код ассемблера. Можно потребовать от *gcc* не компилировать исходный текст в машинный код, а остановиться на уровне языка ассемблера, который значительно легче понять. Это прекрасный способ изучить сложности программирования под Linux на языке ассемблера в защищенном режиме процессора: напишите некоторый код на C, заставьте *gcc* оттранслировать его в ассемблер и изучайте то, что получилось.

В *gcc* входит собственный ассемблер (который может использоваться независимо от *gcc* и называется *gas*) на случай, если вам понадобится ассемблировать этот код на языке ассемблера. Можно встраивать код на языке ассемблера в исходный текст C, если нужно осуществить какие-то особые чудеса, но вы не хотите писать на чистом ассемблере.

Основы работы с gcc

Вероятно, вам уже не терпится узнать, как пользоваться всеми этими замечательными функциями. Важно знать, как эффективно пользоваться *gcc*, особенно новичкам в Unix и C. Использование компилятора командной строки типа *gcc* существенно отличается от работы в такой среде, как Visual Studio или C++ Builder под Windows[1]. Несмотря на сходство в синтаксисе языка, методы, используемые для компиляции и сборки программ, различны.

Вернемся к нашему невинному примеру «Hello, World!». Как бы вы скомпилировали и собрали эту программу?

Сначала, конечно, нужно ввести исходный код. Это делается с помощью текстового редактора, такого как Emacs или *vi*. Программист должен ввести исходный код и сохранить его в файле с именем, скажем, *hello.c*. (Как и большинство компиляторов C, *gcc* очень разборчив в отношении расширений имен файлов; благодаря расширениям он может различать исходные тексты C, исходные тексты ассемблера, объектные файлы и т. д. Для стандартного исходного кода C должно использоваться расширение *.c*.)

Для того чтобы скомпилировать и собрать программу в исполняемый модуль *hello*, нужно выполнить команду:

```
papaya$ gcc -o hello hello.c
```

[1] Ряд интегрированных сред (IDE) есть сейчас и для Linux. В их число входят как коммерческие (Kylix, Linux-версия Delphi), так и open source, например KDevelop, о чем мы расскажем в следующей главе.

и (если нет ошибок) в один прием *gcc* скомпилирует исходный код в объектный файл, скомпонует его с соответствующими библиотеками и выдаст исполняемую программу *hello*, готовую к запуску. Подозрительный программист может попытаться протестировать эту работу:

```
papaya$ ./hello
Hello, World!
papaya$
```

Настолько дружественно, насколько можно ожидать.

Очевидно, что при выполнении этой единственной команды *gcc* было произведено немало невидимых снаружи действий. Прежде всего, *gcc* нужно было скомпилировать ваш исходный файл *hello.c* в объектный файл *hello.o*. Затем скомпоновать *hello.o* со стандартными библиотеками и создать выполняемый модуль.

По умолчанию *gcc* предполагает, что вы не только хотите скомпилировать указанные исходные файлы, но и скомпоновать их вместе (друг с другом и со стандартными библиотеками) для создания выполняемого модуля. Сначала *gcc* компилирует исходные файлы в объектные. Затем он автоматически вызывает компоновщик, чтобы склеить все объектные файлы и библиотеки в выполняемый модуль. (В действительности компоновщик является отдельной программой с именем *ld*, а не частью самого *gcc*, хотя можно сказать, что *gcc* и *ld* являются близкими друзьями.) *gcc* также знает о «стандартных» библиотеках, используемых большинством программ, и он сообщает *ld* о необходимости компоновки с ними. Можно, конечно, рядом способов переопределить это поведение по умолчанию.

Можно указать несколько имен файлов в одной команде *gcc*, но в больших проектах оказывается более удобным скомпилировать сразу несколько файлов и хранить объектные файлы. Если нужно скомпилировать исходный код в объектный файл, но воздержаться от компоновки, используйте ключ *-c*:

```
papaya$ gcc -c hello.c
```

В результате будет создан лишь файл *hello.o* и ничего больше.

По умолчанию компоновщик создает исполняемый модуль с неожиданным именем *a.out*. Это пережиток ранних реализаций Unix, на котором не стоит останавливаться. С помощью ключа *-o* в *gcc* можно присвоить выходному модулю другое имя, в данном случае *hello*.

Использование нескольких исходных файлов

Следующим шагом в вашем исследовании *gcc* будет выяснение того, как компилируются программы из нескольких исходных файлов. Предположим, ваша программа состоит из двух исходных файлов – *foo.c* и *bar.c*. Естественно, вы используете один или несколько заголовочных файлов (например, *foo.h*), содержащих объявления функций, используемых обеими программами. Благодаря этому код в *foo.c* знает о функциях в *bar.c*, и наоборот.

Для того чтобы скомпилировать эти два файла и скомпоновать их вместе (а также с библиотеками), создав исполняемый модуль *baz*, можно выполнить команду:

```
papaya$ gcc -o baz foo.c bar.c
```

Это примерно эквивалентно трем командам:

```
papaya$ gcc -c foo.c
papaya$ gcc -c bar.c
papaya$ gcc -o baz foo.o bar.o
```

gcc действует в качестве удобного интерфейса к компоновщику и другим «скрытым» утилитам, вызываемым во время компиляции.

Конечно, компиляция программы с использованием нескольких исходных файлов в одной команде может потребовать много времени. Если, например, у вас в программе пять или более файлов с исходным кодом, то *gcc* должен поочередно перекомпилировать каждый из них, прежде чем компоновать в исполняемую программу. Это может вызвать большой и непроизводительный расход времени, особенно если вслед за последней компиляцией вы сделали изменения только в одном файле. Нет смысла перекомпилировать остальные исходные файлы, поскольку их объектные файлы не повреждены и сохранили актуальность.

Эту проблему решает использование администратора проекта, такого как *make*. О *make* мы поговорим позже в разделе «Файлы проектов Make».

Оптимизация

Потребовать от *gcc* оптимизировать ваш код при компиляции просто. Для этого нужно лишь использовать ключ *-O* в командной строке *gcc*:

```
papaya$ gcc -O -o fishsticks fishsticks.c
```

Как мы недавно отмечали, *gcc* поддерживает несколько уровней оптимизации. Если использовать *-O2*, а не *-O*, то будут включены некоторые «дорогостоящие» виды оптимизации, из-за которых компиляция может протекать медленнее, но, надо надеяться, значительно увеличит производительность вашей программы.

При работе с Linux вы могли обратить внимание на то, что ряд программ компилируется с ключом *-O6* (примером служит ядро Linux). Текущая версия *gcc* не поддерживает оптимизацию до уровня *-O6*, поэтому (в настоящее время) этот ключ эквивалентен *-O2*. Однако *-O6* иногда используется для совместимости с будущими версиями *gcc* и обеспечения использования наивысшего уровня оптимизации.

Включение отладочной информации

Ключ *-g* заставляет *gcc* включить отладочную информацию в скомпилированные объектные файлы. Это означает, что в объектный файл и конечный исполняемый модуль добавляются данные, которые позволят трассировать

программу с помощью отладчика, такого, например, как *gdb*. Недостатком использования отладочной информации является то, что она значительно увеличивает размер получаемых объектных файлов. Ключ *-g* обычно следует использовать во время разработки и отладки, и не применять при «окончательной» компиляции.

К счастью, включение отладочной информации совместимо с оптимизацией. Это означает, что можно без опасений использовать команду:

```
papaya$ gcc -O -g -o mumble mumble.c
```

Однако некоторые виды оптимизации, осуществляемые по *-O* или *-O2*, могут вызвать хаотическое поведение программы при работе под отладчиком. Обычно лучше не использовать одновременно ключи *-O* и *-g*.

Еще о библиотеках

Прежде чем расстаться с *gcc*, нужно сказать несколько слов о компоновке и библиотеках. Во-первых, легко создать собственные библиотеки. Если у вас есть набор часто используемых функций, вы можете создать набор соответствующих исходных файлов, откомпилировать их все в объектные файлы и сделать из объектных файлов библиотеку. Это избавит вас от необходимости отдельно компилировать эти функции для каждой программы, в которой они используются.

Предположим, что у вас есть набор часто используемых функций, например, такой:

```
float square(float x) {
  /* Код для square()... */
}

int factorial(int x, int n) {
  /* Код для factorial()... */
}
```

и т. д. (конечно, в стандартных библиотеках *gcc* имеются аналоги этих распространенных функций, и пусть вас не смущает их выбор в нашем примере). Допустим, исходный текст *square*() находится в файле *square.c*, а исходный текст *factorial*() – в файле *factorial.c*.

Для того чтобы создать библиотеку, содержащую эти функции, нужно лишь скомпилировать все файлы исходного текста следующим образом:

```
papaya$ gcc -c square.c factorial.c
```

В результате получатся *square.o* и *factorial.o*. Теперь создадим из объектных файлов библиотеку. Оказывается, библиотека является просто архивным файлом, создаваемым *ar* (близким аналогом *tar*). Назовем нашу библиотеку *libstuff.a* и создадим ее следующим образом:

```
papaya$ ar r libstuff.a square.o factorial.o
```

При обновлении такой библиотеки может понадобиться предварительно удалить старую библиотеку *libstuff.a*. Последним шагом является создание

указателя для библиотеки, который позволит компоновщику находить в ней функции. Для этого используется команда *ranlib:*

```
papaya$ ranlib libstuff.a
```

Эта команда добавляет данные к самой библиотеке и не создает отдельного файла указателя. Оба шага выполнения, *ar* и *ranlib,* можно объединить, используя в *ar* команду `s`:

```
papaya$ ar rs libstuff.a square.o factorial.o
```

Теперь у вас есть статическая библиотека *libstuff.a*, содержащая ваши функции. Для того чтобы компоновать с ней программы, нужно создать файл заголовков, описывающий содержимое библиотеки. Например, можно создать *libstuff.h* со следующим содержимым:

```
/* libstuff.h: routines in libstuff.a */
extern float square(float);
extern int factorial(int, int);
```

Каждый исходный файл, использующий функции из *libstuff.a*, должен содержать строку `#include "libstuff.h"`, как для стандартных файлов заголовков.

Теперь, когда у нас есть библиотека и файл заголовка, как компилировать программы, чтобы их использовать? Прежде всего, нужно поместить библиотеку и файл заголовка в такое место, где компилятор сможет их найти. Многие пользователи помещают личные библиотеки в каталог *lib* своего домашнего каталога, а личные включаемые файлы в каталог *include*. Предположив, что так мы и сделали, скомпилируем гипотетическую программу *wibble.c* с помощью команды:

```
papaya$ gcc -I../include -L../lib -o wibble wibble.c -lstuff
```

Параметр *-I* предлагает *gcc* добавить каталог *../include* к пути поиска для включаемых файлов. Аналогично *-L* указывает *gcc* на необходимость добавить *../lib* к пути поиска библиотек.

Последний аргумент командной строки *-lstuff* сообщает компоновщику о необходимости присоединения библиотеки *libstuff.a* (где бы она ни нашлась по пути поиска библиотек). Предполагается, что имена файлов библиотек начинаются с *lib*.

Когда вы хотите, чтобы компоновка производилась с библиотеками, отличными от стандартных, используйте ключ *-l* в командной строке *gcc*. Например, если вы хотите использовать математические функции (определенные в *math.h*), следует добавить в конец команды *gcc* ключ *-lm*, в результате чего компоновка будет осуществляться с библиотекой *libm*. Обратите внимание на то, что существенен *порядок* параметров *-l*. Например, если в нашей библиотеке *libstuff* есть такие же функции, как в *libm*, то *-lm* нужно включать в командную строку после *-lstuff*:

```
papaya$ gcc -Iinclude -Llib -o wibble wibble.c -lstuff -lm
```

Это вынуждает компоновщик присоединять *libm* после *libstuff* для нахождения ссылок, которые не были найдены в *libstuff*.

Где *gcc* ищет библиотеки? По умолчанию поиск происходит в нескольких местах, главным из которых является */usr/lib*. Если взглянуть на содержание */usr/lib*, то можно заметить, что в этом каталоге содержится много библиотечных файлов, причем имена одних оканчиваются на *.a*, а других – на *.so.version*. Файлы *.a* являются статическими библиотеками, такими как наша *libstuff.a*. Файлы *.so* являются совместно используемыми библиотеками, содержащими код, присоединяемый во время исполнения, а также код заглушки, необходимый компоновщику времени исполнения (*ld.so*) для определения местоположения библиотеки.

Во время исполнения загрузчик программ ищет образы совместно используемых библиотек в разных местах, в том числе в */lib*. Если заглянуть в */lib*, можно увидеть файлы типа *libc.so.6*. Это файл образа, содержащий код совместно используемой библиотеки *libc* (одной из стандартных библиотек, с которыми компонуется большинство программ).

По умолчанию компоновщик пытается осуществлять компоновку с совместно используемыми библиотеками, однако в некоторых случаях используются статические библиотеки, например, когда в пути поиска нет библиотеки совместного доступа с указанным именем. Можно также потребовать от *gcc* компоновки со статическими библиотеками с помощью ключа *-static*.

Создание совместно используемых библиотек

Теперь, когда вы знаете, как создавать и использовать статические библиотеки, очень легко перейти к созданию совместно используемых библиотек. У совместно используемых библиотек есть ряд преимуществ. Они сокращают расход памяти, если используются более чем одним процессом, и уменьшают размер исполняемого модуля. Более того, они облегчают разработку: при использовании совместно используемых библиотек и внесении изменений в библиотеку не требуется всякий раз перекомпилировать приложение. Перекомпиляция требуется только при внесении несовместимых изменений, например, при добавлении к вызову новых аргументов или изменении размеров структуры.

Пока вы не начали делать все свои разработки с помощью совместно используемых библиотек, мы должны вас предупредить, что отлаживаться с ними несколько сложнее, чем со статическими библиотеками, поскольку у отладчика *gdb*, обычно используемого в Linux, есть некоторые проблемы с совместно используемыми библиотеками.

Программный код, включаемый в совместно используемую библиотеку, не должен зависеть от положения. Это соглашение для объектного кода, которое позволяет использовать код в совместно используемых библиотеках. *gcc* создает независимый от положения код, если передать ему в командной строке ключей *-fpic* или *-fPIC*. Предпочтительнее использовать первый ключ, если только модули не выросли настолько, что таблица перемещаемого кода оказалась слишком мала. В этом случае компилятор выдает сообщение об ошибке, и приходится использовать ключ *-fPIC*. Повторим наш пример из предыдущего раздела:

```
papaya$ gcc -c -fPIC square.c factorial.c
```

После этого создать совместно используемую библиотеку уже совсем просто:[1]

```
papaya$ gcc -shared -o libstuff.so square.o factorial.o
```

Обратите внимание на ключ компилятора *-shared*. Создавать указатель, как в случае статических библиотек, не требуется.

Использовать нашу вновь созданную совместно используемую библиотеку еще проще. В команду компилятора не нужно вносить никаких изменений:

```
papaya$ gcc -I../include -L../lib -o wibble wibble.c -lstuff -lm
```

Вы можете спросить, что делает компоновщик, если имеются как совместно используемая библиотека *libstuff.so*, так и статическая библиотека *libstuff.a*. В таком случае компоновщик всегда выбирает совместно используемую библиотеку. Для использования статической библиотеки ее нужно точно назвать в командной строке:

```
papaya$ gcc -I../include -L../lib -o wibble wibble.c libstuff.a -lm
```

Очень полезным инструментом при работе с библиотеками совместного доступа является *ldd*. С помощью этой утилиты можно узнать, какие библиотеки совместного доступа использует исполняемая программа. Вот пример:

```
papaya$ ldd wibble
        libstuff.so => libstuff.so (0x400af000)
        libm.so.5 => /lib/libm.so.5 (0x400ba000)
        libc.so.5 => /lib/libc.so.5 (0x400c3000)
```

Три поля в каждой строке – это имя библиотеки, полный путь к используемому экземпляру библиотеки и адрес, по которому библиотека отображена в виртуальном адресном пространстве.

Если для некоторой библиотеки *ldd* сообщает `not found`, то дело плохо, и запустить эту программу вы не сможете. Придется поискать эту библиотеку. Возможно, она входит в состав вашего дистрибутива, но от ее установки вы отказались, либо она находится на жестком диске, но загрузчик (часть системы, загружающая все исполняемые программы) не может ее найти.

В последнем случае попытайтесь найти библиотеку самостоятельно и определить, не находится ли она в нестандартном каталоге. По умолчанию загрузчик осуществляет поиск только в */lib* и */usr/lib*. Если у вас есть библиотеки, находящиеся в другом каталоге, создайте переменную окружения `LD_LIBRARY_PATH` и укажите в ней каталоги, разделив их двоеточиями. Если вы уверены, что все настроено правильно, а требуемую библиотеку все равно не удается найти, выполните команду *ldconfig* в качестве `root`; это обновит кэш компоновщика.

---

[1] На заре создания Linux создание разделяемых библиотек было устрашающей задачей, которой побаивались даже знатоки. С появлением несколько лет назад формата объектных файлов ELF задача свелась к выбору подходящего ключа при компиляции. Дело явно идет к лучшему!

Использование C++

Если вы предпочитаете объектно-ориентированное программирование, то *gcc* обеспечивает полную поддержку C++, а также Objective-C. При программировании на C++ с использованием *gcc* нужно учитывать несколько обстоятельств.

Во-первых, имена исходных файлов C++ должны иметь расширение *.cpp* (используется чаще всего), *.C* или *.cc*. Это отличает их от файлов с исходным текстом обычного C, имеющих расширение *.c*.

Во-вторых, при компиляции кода C++ вместо *gcc* нужно использовать сценарий оболочки *g++*. Сценарий *g++* вызывает *gcc* с рядом дополнительных параметров, в частности определяющих компоновку со стандартными библиотеками C++. *g++* использует те же ключи и параметры, что и *gcc*.

Если вы не хотите использовать *g++*, то нужно обеспечить компоновку с библиотеками C++, чтобы использовать базовые классы C++, такие как объекты ввода/вывода `cout` и `cin`. Убедитесь также, что вы действительно установили библиотеки C++ и включаемые файлы. В некоторых дистрибутивах содержатся только стандартные библиотеки C. *gcc* сможет откомпилировать ваши программы C++, но без библиотек C++ дело закончится ошибками компоновщика, если вы пытаетесь использовать стандартные объекты.

Файлы проектов

Работая с Linux, вам, вероятно, придется столкнуться с утилитой *make*, даже если вы не собираетесь заниматься программированием. Вам, вероятно, понадобится устанавливать патчи или перекомпилировать ядро, а для этого придется запустить *make*. Если повезет, то вам не придется пачкаться с файлами проектов (makefiles), но мы постарались сделать эту книгу полезной не только для счастливчиков. Поэтому в данном разделе мы разъясним тонкий синтаксис *make*, чтобы вас не пугали файлы проектов.

В некоторых примерах мы будем использовать текущий файл проекта для ядра Linux. В нем используется множество расширений из мощной версии GNU *make*, поэтому мы опишем некоторые из них наряду со стандартными функциями *make*. Хорошее введение в *make* дано в книге «Managing Projects with make» Эндрю Орэма (Andrew Oram) и Стива Талботта (Steve Talbott), O'Reilly. Расширения GNU хорошо документированы в руководстве по GNU *make*.

Большинство пользователей считает, что *make* – это средство создания объектных файлов и библиотек из исходного кода, а также создания исполняемых модулей из объектных файлов. В более общем смысле *make* является универсальной программой создания *целевых объектов* (*targets*) на основе *зависимостей* (*dependencies*). Целевым объектом может являться выполняемая программа, документ в формате PostScript или что-то другое. Исходными данными могут быть текст на C, текстовый файл для TEX и т. д.

Можно, конечно, писать простые сценарии оболочки для выполнения команд *gcc*, которые будут создавать выполняемую программу, но особенность *make* состоит в том, что она знает, какие объекты нужно создавать заново, а какие – нет. Объектный файл нужно компилировать заново, только если изменился соответствующий исходный код.

Допустим, у вас есть программа, состоящая из трех исходных файлов на C. Если бы вы создавали исполняемую программу с помощью команды:

```
papaya$ gcc -o foo foo.c bar.c baz.c
```

то при каждом изменении одного из исходных файлов все три файла подвергались бы перекомпиляции и повторной сборке в исполняемый модуль. Если изменения касаются только одного файла, то это бесполезная трата времени (это особенно плохо, если программа состоит из большого числа исходных файлов). На самом деле вам нужно перекомпилировать только один изменившийся исходный файл в объектный файл и заново скомпоновать все объектные файлы. *make* может автоматизировать этот процесс.

Что делает make

Основная задача *make* – позволить создавать целевой файл маленькими шагами. Если конечный исполняемый модуль создается из многих исходных файлов, то можно, изменив один исходный файл, собрать исполняемую программу, не перекомпилируя все подряд. Чтобы обеспечить подобную гибкость, *make* ведет учет файлов, необходимых для создания целевого объекта.

Приведем пример тривиального файла проекта. Назовите его *makefile* или *Makefile* и поместите в один каталог с файлами исходного кода:

```
edimh: main.o edit.o
        gcc -o edimh main.o edit.o

main.o: main.c
        gcc -c main.c

edit.o: edit.c
        gcc -c edit.c
```

Этот файл создаст программу с именем *edimh* из двух исходных файлов с именами *main.c* и *edit.c*. В файлах проектов вы не ограничены программированием на C; команды могут быть любыми.

В файле имеется три записи. В каждой из них есть строка зависимостей, показывающая, как строится файл. Таким образом, первая строка говорит, что *edimh* (имя перед двоеточием) строится из двух объектных файлов *main.o* и *edit.o* (имена после двоеточия). Эта строка сообщает *make*, что она должна выполнить следующую строку с командой *gcc*, если изменится один из этих объектных файлов. Строки, содержащие команды, должны начинаться с символов табуляции (не с пробелов).

Команда

```
papaya$ make edimh
```

выполняет строку с командой *gcc*, если не существует файл с именем *edimh*. Однако эта строка выполняется и в случае, если *edimh* существует, но один из объектных файлов имеет более позднее время создания. *edimh* мы называем *целевым объектом* (*target*). Файлы после двоеточия называются *зависимостями* (*dependents*), или *пререквизитами* (*prerequisites*).

Следующие две записи делают то же самое в отношении объектных файлов. Файл *main.o* создается, если он не существует или ассоциированный с ним исходный файл имеет более позднее время создания. *edit.o* строится из *edit.c*.

Как *make* узнает о том, что файл является новым? Она смотрит на временную метку, которую файловая система связывает с каждым файлом. Эти временные отметки можно увидеть, выполнив команду *ls -l*. Поскольку точность временной метки составляет одну секунду, она достаточно надежно сообщает *make* о том, подвергался ли файл изменениям со времени последней компиляции или компилировался ли объектный файл со времени последней сборки выполняемого файла.

Давайте испытаем файл проекта и посмотрим, что он делает:

```
papaya$ make edimh
gcc -c main.c
gcc -c edit.c
gcc -o edimh main.o edit.o
```

Если мы отредактируем *main.c* и повторно выполним команду, то перестроены будут только необходимые файлы, что сэкономит часть времени:

```
papaya$ make edimh
gcc -c main.c
gcc -o edimh main.o edit.o
```

Порядок расположения этих трех записей в файле проекта не имеет значения: *make* определяет зависимости файлов и выполняет команды в нужном порядке. Запись для *edimh* делается первой для удобства, т. к. в результате *edimh* становится файлом, создаваемым по умолчанию. Другими словами, ввод `make` становится равносильным вводу `make edimh`.

Вот более пространный файл проекта. Попробуйте разобраться в том, что он делает:

```
install: all
        mv edimh /usr/local
        mv readimh /usr/local

all: edimh readimh

readimh: read.o edit.o
        gcc -o readimh main.o read.o

edimh: main.o edit.o
        gcc -o edimh main.o edit.o

main.o: main.c
        gcc -c main.c
edit.o: edit.c
```

```
        gcc -c edit.c

read.o: read.c
        gcc -c read.c
```

Прежде всего, мы видим целевой объект `install`. Он не создает никакого файла и называется *фиктивным объектом*, поскольку существует только для того, чтобы могли быть выполнены команды, перечисленные под ним. Но прежде чем выполнится `install`, должен выполниться `all`, поскольку `install` зависит от `all`. (Напомним, что порядок записей в файле не имеет значения.)

Итак, обратимся к объекту `all`. Под ним нет команд (это вполне допустимо), но он зависит от `edimh` и `readimh`. Это уже реальные файлы: каждый из них является исполняемой программой. Поэтому `make` прослеживает зависимости в обратном направлении, пока не придет к файлам *.c*, которые уже ни от чего не зависят. После этого она старательно перестраивает каждый из целевых объектов.

Вот образец прогона этого файла (вам могут понадобиться привилегии суперпользователя, чтобы установить файлы в каталоге */usr/local*):

```
papaya$ make install
gcc -c main.c
gcc -c edit.c
gcc -o edimh main.o edit.o
gcc -c read.c
gcc -o readimh main.o read.o
mv edimh /usr/local
mv readimh /usr/local
```

Таким образом, этот файл проекта вызывает полную компиляцию и установку. Сначала строятся файлы, необходимые для создания *edimh*. Затем строится дополнительный объектный файл, необходимый для создания *readmh*. После создания этих двух исполняемых файлов определяется целевой объект `all`. Теперь *make* может перейти к созданию целевого объекта `install`, что означает перемещение двух исполняемых файлов к месту их окончательного расположения.

Во многих файлах проектов, включая те, которые используются при компиляции Linux, содержатся фиктивные объекты для осуществления некоторых стандартных действий. Например, файл проекта для ядра Linux содержит команды для удаления временных файлов:

```
clean:  archclean
        rm -f kernel/ksyms.lst
        rm -f core `find .  -name '*.[oas]' -print`
        .
        .
        .
```

и создания списка объектных и заголовочных файлов, от которых они зависят (это сложная, но важная задача; если изменяется файл заголовка, необходимо перекомпилировать файлы, имеющие ссылки на него):

```
depend dep:
      touch tools/version.h
      for i in init/*.c;do echo -n "init/";$(CPP) -M $$i;done > .tmpdep
      .
      .
      .
```

Некоторые из этих команд оболочки становятся довольно сложными. Мы рассмотрим команды файла проекта далее в этой главе в разделе «Множественные команды».

Некоторые синтаксические правила

Самым сложным в работе с файлами проектов, по крайней мере, на этапе освоения, является соблюдение правильного синтаксиса. Честно говоря, синтаксис *make* довольно бестолковый. Если использовать пробелы там, где должны быть символы табуляции, или наоборот, то файл проекта не обработается должным образом. А сообщения об ошибках просто ставят в тупик.

Всегда ставьте символ табуляции, а не пробелы, перед командой. И не используйте табуляцию в начале других строк.

В любом месте строки можно поставить символ (#), чтобы начать комментарий. Символы после решетки игнорируются.

Обратная косая черта в конце строки означает продолжение на следующей строке. Это применяется для длинных командных строк и в других случаях.

Теперь рассмотрим некоторые мощные функции *make*, образующие своего рода язык программирования.

Макросы

Когда имя файла или другая строка используется в файле проекта неоднократно, их можно включить в макрос, т. е. сделать строкой, которую *make* «раскрывает» в другую строку. Например, в начале нашего тривиального файла проекта можно было поместить такое макроопределение:

```
OBJECTS = main.o edit.o

edimh: $(OBJECTS)
        gcc -o edimh $(OBJECTS)
```

make просто вставит `main.o edit.o` во все места, где указано `$(OBJECTS)`. Если вам нужно добавить к проекту другой объектный файл, укажите его в первой строке. В результате строка зависимости и командные строки будут соответствующим образом изменены.

Не забывайте круглые скобки при ссылке на `$(OBJECTS)`. Макросы похожи на переменные окружения, такие как `$HOME` и `$PATH`, но это не одно и то же.

Один макрос можно определить через другой, поэтому допустимы выражения типа:

```
ROOT = /usr/local
HEADERS = $(ROOT)/include
SOURCES = $(ROOT)/src
```

В этом случае HEADERS превращается в каталог */usr/local/include*, а SOURCES — в */usr/local/src*. Если вы устанавливаете пакет в системе и не хотите, чтобы он попал в */usr/local*, просто выберите другой путь и измените строку, определяющую ROOT.

Кстати, использовать верхний регистр в макроопределениях не обязательно, но это является общепринятым соглашением.

Расширение GNU *make* позволяет делать дополнения к макроопределению. При этом вместо знака равенства используется :=:

```
DRIVERS        =drivers/block/block.a

ifdef CONFIG_SCSI
DRIVERS := $(DRIVERS) drivers/scsi/scsi.a
endif
```

Первая строка является обычным макроопределением, назначающим DRIVERS имя файла drivers/block/block.a. Следующее определение добавляет к нему имя файла drivers/scsi/scsi.a. Но это происходит только тогда, когда определен макрос CONFIG_SCSI. В этом случае полное определение становится следующим:

```
drivers/block/block.a drivers/scsi/scsi.a
```

А как определить CONFIG_SCSI? Можно вставить его в файл проекта, назначив ему произвольную строку:

```
CONFIG_SCSI = yes
```

Однако может оказаться более удобным определить его в командной строке *make*:

```
papaya$ make CONFIG_SCSI=yes target_name
```

Одна из тонкостей использования макросов состоит в возможности оставить их неопределенными. Если они не определены, то вместо них подставляется пустая строка. При этом у вас появляется возможность задать макрос в качестве переменной окружения. Например, можно не определять в файле проекта CONFIG_SCSI, а сделать это в файле конфигурации *.bashrc*, используемом оболочкой *bash*:

```
export CONFIG_SCSI=yes
```

Если вы используете оболочку *csh* или *tcsh*, определение можно вставить в файл *.cshrc*:

```
setenv CONFIG_SCSI yes
```

В результате при любой сборке CONFIG_SCSI будет определен.

Правила использования суффиксов и шаблонов

Для таких рутинных задач, как сборка объектного файла из исходного, не хотелось бы указывать в файле проекта все отдельные зависимости. И не нужно. Компиляторы Unix используют простой стандарт (компилирование файла с суффиксом *.c* для создания файла с суффиксом *.o*), а *make* имеет функцию, называемую правилом суффиксов, работающую с такими файлами.

Вот простое правило суффиксов для компиляции исходных файлов C, которое можно вставить в файл проекта:

```
.c.o:
        gcc -c $(CFLAGS) $<
```

Строка `.c.o:` означает: «использовать исходный файл *.c* для создания файла *.o*». `CFLAGS` является макросом, в который можно вставить нужные параметры компилятора, например *-g* для отладки или *-O* для оптимизации. Строка `$<` является хитрым способом указать исходный файл. Таким образом, при выполнении этой команды в это место будет вставлено имя вашего файла *.c*.

Вот простой пример использования правила суффиксов. В командной строке передаются параметры *-g* и *-O*:

```
papaya$ make CFLAGS="-O -g" edit.o
gcc -c -O -g edit.c
```

Фактически нет необходимости указывать это правило суффиксов в вашем файле проекта, поскольку нечто аналогичное уже встроено в *make*. Даже переменная `CFLAGS` в ней определена, поэтому параметры компиляции можно определить, просто установив значение этой переменной. Файл проекта для сборки ядра Linux содержит такое определение с множеством параметров *gcc*:

```
CFLAGS = -Wall -Wstrict-prototypes -O2 -fomit-frame-pointer -pipe
```

Говоря о флагах компилятора, надо особенно упомянуть одну группу. Это параметр *-D*, используемый для задания символов в выражениях исходного кода. Поскольку в условных выражениях типа `#ifdef` встречается масса различных символов, может потребоваться передавать в файл проекта множество таких параметров, как *-DDEBUG* или *-DBSD*. Если вы делаете это через командную строку *make*, то не забывайте весь набор заключать в двойные кавычки или апострофы, поскольку необходимо, чтобы оболочка передала весь набор как один аргумент:

```
papaya$ make CFLAGS="-DDEBUG -DBSD" . . .
```

GNU *make* позволяет воспользоваться так называемыми *правилами шаблонов*, которые еще удобнее, чем правила суффиксов. Правило шаблонов использует знак процента, означающий «любую строку». Поэтому исходные тексты C можно компилировать с помощью такого правила:

```
%.o: %.c
        gcc -c -o $@ $(CFLAGS) $<
```

Здесь сначала указан выходной файл *%.o*, а затем после двоеточия – исходный файл *%.c*. Правило шаблонов аналогично обычной строке зависимости, но вместо точных имен файлов содержит знаки процента.

Мы видим строку `$<`, относящуюся к исходному файлу, а также строку `$@`, указывающую на выходной файл, поэтому вместо нее вставляется имя объектного файла *.o*. Обе строки являются встроенными макроопределениями; *make* определяет их при обработке каждой записи.

Другим часто используемым встроенным макроопределением является `$*`, которое ссылается на имя исходного файла без расширения. Поэтому если исходным файлом является *edit.c*, строка `$*.s` означает *edit.s* (исходный файл на языке ассемблера).

Рассмотрим полезную вещь, которую можно сделать с помощью правила шаблонов, но нельзя сделать с помощью правила суффиксов, а именно: добавить строку `_dbg` к имени выходного файла для того, чтобы потом было видно, что он скомпилирован с отладочной информацией:

```
%_dbg.o: %.c
        gcc -c -g -o $@ $(CFLAGS) $<

DEBUG_OBJECTS = main_dbg.o edit_dbg.o

edimh_dbg: $(DEBUG_OBJECTS)
        gcc -o $@ $(DEBUG_OBJECTS)
```

Теперь все объекты можно создавать двумя различными способами: с отладочной информацией и без нее. У них будут разные имена, поэтому их можно хранить в одном каталоге:

```
papaya$ make edimh_dbg
gcc -c -g -o main_dbg.o  main.c
gcc -c -g -o edit_dbg.o  edit.c
gcc -o edimh_dbg  main_dbg.o edit_dbg.o
```

Множественные команды

В проект *make* могут включаться любые команды оболочки. Однако дело осложняется тем, что *make* выполняет каждую команду в отдельной оболочке, поэтому, например, не будет работать такой проект:

```
target:
        cd obj
        HOST_DIR=/home/e
        mv *.o $HOST_DIR
```

Ни команда *cd*, ни определение переменой `HOST_DIR` не оказывают эффекта на последующие команды. Все команды должны быть объединены в одну строку. Оболочка использует в качестве разделителя между командами точку с запятой, поэтому строка может выглядеть так:

```
target:
        cd obj ; HOST_DIR=/home/e ; mv *.o $$HOST_DIR
```

Еще одно изменение связано с тем, что переменная оболочки в команде должна начинаться с удвоенного символа доллара, что позволяет *make* отличить ее от макроса.

Файл будет легче читаться, если разбить команды на несколько строк с помощью обратной косой черты; при этом *make* продолжает считать такие строки одной командой:

```
target:
        cd obj ; \
        HOST_DIR=/home/e ; \
        mv *.o $$HOST_DIR
```

Иногда файлы проектов *make* содержат собственные команды *make*, т. е. являются рекурсивными проектами, например:

```
linuxsubdirs: dummy
        set -e; for i in $(SUBDIRS); do $(MAKE) -C $$i; done
```

Макрос `$(MAKE)` вызывает *make*. Есть несколько причин использовать вложенные *make*. Одна из них – это выполнение сборки в нескольких каталогах, как в данном примере (каждый из них должен содержать собственный файл проекта *make*). В других случаях макросы определяются в командной строке, что позволяет выполнять проекты с различными макроопределениями.

GNU *make* предлагает в качестве расширения оболочки другой мощный интерфейс. Можно выполнить команду оболочки и присвоить ее вывод макросу. Ряд примеров такого рода есть в файле проекта для ядра Linux, но мы приведем простой пример:

```
HOST_NAME = $(shell uname -n)
```

Эта команда присваивает сетевое имя вашей машины, возвращаемое командой *uname -n*, макросу `HOST_NAME`.

make предлагает ряд других соглашений, которые иногда полезны. Например, знак коммерческое at (@) перед командой подавляет отображение выполняемых команд:

```
@if [ -x /bin/dnsdomainname ]; then \
   echo #define LINUX_COMPILE_DOMAIN \"`dnsdomainname`\"; \
else \
   echo #define LINUX_COMPILE_DOMAIN \"`domainname`\"; \
fi >> tools/version.h
```

Согласно другому соглашению дефис, помещенный перед именем команды, сообщает *make*, что обработка проекта должна быть продолжена, даже если команда не могла быть выполнена. Это полезно, если вы хотите продолжить работу после неудачи с командой *mv* или *cp*:

```
- mv edimh /usr/local
- mv readimh /usr/local
```

Включение других файлов проекта

Большие проекты часто разбивают на несколько файлов. Это облегчает использование различными проектами в различных каталогах общих объектов, особенно макроопределений. Строка

```
include filename
```

читает содержимое файла *filename*. В файле проекта ядра Linux можно увидеть такую строку:

```
include .depend
```

Заглянув в файл *.depend*, вы обнаружите множество записей – строк, определяющих зависимость объектных файлов от заголовочных. (Кстати, *.depend* может изначально не существовать, а создаваться другой записью в проекте.)

Иногда строки `include` ссылаются на макросы, а не на имена файлов, например:

```
include ${INC_FILE}
```

В данном случае `INC_FILE` должна быть определена как переменная окружения или как макрос, что дает больше возможностей управлять именем включаемого файла.

Интерпретация сообщений make

Сообщения об ошибках, выводимые *make*, могут быть весьма загадочными, поэтому поможем вам разобраться с ними. Следующие пояснения относятся к большинству стандартных сообщений.

**** No targets specified and no makefile found. Stop.*

Обычно это означает, что в компилируемом каталоге нет файла проекта. По умолчанию *make* сначала ищет файл *GNUmakefile*; в случае неудачи она ищет *Makefile* и, наконец, *makefile*. Если ни одного из этих файлов не существует, выводится это сообщение об ошибке. Если по каким-либо причинам вы хотите использовать файл проекта с другим именем (или в другом каталоге), можете задать его с помощью параметра командной строки *–f*.

*make: *** No rule to make target 'blah.c', needed by 'blah.o'. Stop.*

Это означает, что *make* не может найти требуемый исходный объект (в данном случае *blah.c*), чтобы построить целевой (в данном случае *blah.o*). Как упоминалось, *make* сначала ищет исходный объект среди целевых объектов в файле проекта, а если таких нет, то ищет файл с именем исходного объекта. Если и его не существует, выводится это сообщение об ошибке. Обычно это означает, что не хватает каких-то файлов исходного кода или в файле проекта сделана опечатка.

**** missing separator (did you mean TAB instead of 8 spaces?). Stop.*

Текущие версии *make* достаточно дружественны к пользователю, чтобы спросить его, не совершил ли он распространенной ошибки: пропустил

символ табуляции перед командой. В более старых версиях *make* вы получите лишь *missing separator*. В этом случае проверьте, действительно ли у вас есть табуляция перед каждой командой и нигде более.

Autoconf, Automake и другие инструменты генерации Makefile

Создание больших файлов проектов обычно является утомительной и долгой задачей, особенно если предполагается компиляция на нескольких платформах. Проект GNU предоставляет два средства, называемые *Autoconf* и *Automake*, которые нелегко освоить, однако, овладев ими, можно сильно упростить задачу создания переносимых файлов проектов. Кроме того, создание переносимых библиотек совместного доступа весьма облегчается благодаря использованию *libtool*. Эти инструменты могут быть на CD вашего дистрибутива, либо их можно загрузить с *ftp:// ftp.gnu.org/gnu/*.

С точки зрения пользователя, использование *Autoconf* заключается в запуске программы *configure*, которая должна поставляться с пакетом исходного кода, который вы хотите скомпилировать. Эта программа анализирует вашу систему и настраивает файлы проектов пакета соответственно вашей системе и настройкам. Перед тем как реально запустить сценарий *configure*, неплохо выполнить такую команду:

```
owl$ ./configure --help
```

Она выводит все ключи командной строки, воспринимаемые программой *configure*. Многие пакеты допускают различную установку – например, могут включаться разные модули, и выбрать их можно с помощью параметров *configure*.

С точки зрения программиста вы пишете не файлы проекта, а файлы с именем *makefile.in*. В них могут присутствовать символы-заместители, которые будут заменены фактическими значениями при выполнении пользователем программы *configure*, что создает файлы проектов, выполняемые затем *make*. Кроме того, вы должны написать файл с именем *configure.in*, который описывает ваш проект и то, что нужно проверить на целевой системе. После этого *Autoconf* генерирует программу *configure* из этого файла *configure.in*. К сожалению, написать файл *configure.in* слишком сложно, чтобы эту процедуру можно было здесь описать, но в пакете *Autoconf* содержится необходимая документация по этому поводу.

Создание файлов *makefile.in* остается утомительной и долгой задачей, но даже ее можно, в основном, автоматизировать с помощью пакета *Automake*. Используя этот пакет, вы пишете не файлы *makefile.in*, а файлы *makefile.am*, синтаксис которых значительно проще, а текст короче. С помощью *automake* файлы *makefile.am* преобразуются в *makefile.in*, включаемые в дистрибутивы исходного кода и, в свою очередь, преобразуемые в *makefiles* при настройке пакета для системы пользователя. Описание правил составления файлов *makefile.am* также выходит за рамки этой книги. Здесь следует воспользоваться документацией, имеющейся в пакете.

В настоящее время большинство пакетов open source используют для генерации файлов проектов инструмент *libtool/automake/autoconf*, но это не значит, что этот довольно сложный и запутанный метод – единственный имеющийся. Есть и другие средства генерации файлов проектов, например *imake*, используемый для настройки X Window System. Существует еще один инструмент, не обладающий мощью комплекта *Autoconf* (хотя и позволяющий делать большую часть из того, что нужно для генерации файлов проектов), но чрезвычайно простой в применении – он даже может генерировать собственные файлы описаний с самого начала. Имеется в виду *qmake* в составе библиотеки Qt C++ GUI (можно загрузить с *http://www.trolltech.com*).

Программирование на языке оболочки

В разделе «Оболочки» главы 4 «Основные команды и понятия Unix» мы обсуждали различные оболочки для Linux, но необходимо отметить, что оболочки могут также быть мощным и идеально гибким средством программирования. Различия между оболочками отчетливее всего проявляются при создании сценариев. Командные языки оболочки Борна и оболочки C несколько различны, но отличие не столь заметно при обычном интерактивном использовании. На практике, многие различия проступают только при использовании необычных и малоизвестных функций каждой из оболочек, таких как подстановка слов или некоторые непрямые функции расширения параметров.

Самое заметное различие между оболочками Борна и C состоит в форме различных структур ветвления, в том числе `if ...then` и циклов `while`. В оболочке Борна структура `if ...then` имеет вид:

```
if list
then
  commands
elif list
then
  commands
else
  commands
fi
```

где *list* является просто последовательностью команд, которые должны использоваться в качестве условного выражения для команд `if` и `elif` (сокращение от «else if»). Считается, что условие имеет значение «истина», если код выхода из *list* равен нулю (в отличие от булевых выражений в C; в терминологии оболочки код выхода, равный нулю, означает успешное завершение). *commands* в операторе условного выполнения является списком команд, выполняемых при возвращении истины соответствующим *list*. Слово `then` после каждого списка *list* должно располагаться на отдельной строке, чтобы отличаться от самого списка. Список *list* можно также заканчивать символом «;». То же справедливо для *commands*.

Приведем пример:

```
if [ "$PS1" ]; then
  PS1="\h:\w% "
fi
```

Эта последовательность операторов проверяет, была ли оболочка запущена после регистрации пользователя в системе (т. е. установлена ли переменная PS1), и если так, то устанавливает приглашение в \h:\w%, что при расширении дает имя машины, за которым следует имя рабочего каталога, например:

```
loomer:/home/loomer/mdw%
```

Условие [...], помещенное после if, является встроенной командой *bash* – сокращением для команды *test*. Команда *test* и ее эквивалентное сокращение предоставляют удобный механизм для проверки значений переменных оболочки, эквивалентности строк и т. д. После if вместо [...] можно вызвать любую последовательность команд, и значением условия будет код выхода последней команды.

В *tcsh* составной оператор if ...then выглядит следующим образом:

```
if (expression) then
  commands
else if (expression) then
  commands
else
  commands
endif
```

Различие заключается в том, что выражение *expression* после if является арифметическим или логическим выражением, вычисляемым *tcsh*, в то время как в *bash* условное выражение является командой, и его значение становится true или false в зависимости от статуса выхода из команды. В *bash* использование *test* или [...] аналогично использованию арифметического выражения в *tcsh*.

Если при работе в *tcsh* вы хотите выполнить в выражении *expression* внешние команды, то должны заключать команду в фигурные скобки: {*command*}.

В *tcsh* эквивалентом предшествующей последовательности *bash* для установления приглашения будет:

```
if ($?prompt) then
  set prompt="%m:%/%% "
endif
```

где используются собственные специальные символы *tcsh* для приглашения. Как можно видеть, в *tcsh* используется командный синтаксис, близкий к языку C, и выражения имеют арифметическую и логическую направленность. В *bash* же почти все является фактическими командами, а выражения вычисляются на основе кода выхода команд. В обеих оболочках есть сходные функции, но подход несколько различен.

Аналогичные различия имеются в цикле `while`. В *bash* он имеет вид:

```
while list
do
  commands
done
```

Можно добиться обратного эффекта, заменив `while` на `until`. Список *list* и здесь является последовательностью команд, которые нужно выполнить, а результат определяется кодом выхода (ноль в случае успеха и не ноль в случае неудачи). В *tcsh* цикл имеет следующий вид:

```
while (expression)
  commands
end
```

где *expression* является логическим выражением, которое должно быть вычислено в *tcsh*.

Этого примера должно быть достаточно для понимания общего различия между сценариями оболочек *bash* и *tcsh*. Мы рекомендуем прочесть страницы руководства по *bash*(1) и *tcsh*(1) (хотя они могут служить скорее справочником, чем учебником), и Info-страницы, если они у вас есть. Имеются также различные книги и учебники по этим двум оболочкам; в действительности подойдет любая книга по программированию в оболочке, а сведения о расширенных функциях *bash* и *tcsh*, по сравнению со стандартными оболочками Борна и С, можно почерпнуть на страницах руководства. Полезными приобретениями стали бы «Learning the bash Shell» Камерона Ньюхема (Cameron Newham) и Билла Розенблатта (Bill Rosenblatt), а также «Using csh and tcsh» Пола Дюбуа (Paul DuBois).

Использование Perl

Появление Perl, возможно, является самым благодатным событием в мире программирования Unix за многие годы. Его ценность не умаляется доступностью только в Linux.[1] Perl ориентирован на работу с текстами и файлами, и первоначально создан для просмотра и обработки больших объемов текста, а также создания красивых отчетов из этих данных. Однако по мере своего становления Perl превратился в язык сценариев общего назначения, пригодный для любых задач, от управления процессами до сетевых коммуникаций по TCP/IP. Perl, являясь свободным программным обеспечением, был разработан Ларри Уоллом (Larry Wall), гуру Unix, которому мы обязаны существованием программы чтения телеконференций *rn* и многих распространенных утилит, таких как *patch*. В настоящее время его поддерживают Ларри и группа добровольцев.

---

[1] Справедливости ради следует отметить, что теперь Perl есть и на других системах, в том числе Windows. Но его популярность и общеупотребительность не идут ни в какое сравнение с тем, как им пользуются в Linux.

Основная сила Perl заключается в том, что он объединил наиболее широко используемые возможности таких языков, как C, *sed*, *awk* и различных оболочек в один интерпретируемый язык сценариев. В прошлом, для того чтобы выполнить сложную работу, требовалось создавать из этих языков артистические конструкции, в которых зачастую сценарии на *sed* передавали данные в сценарии на *awk*, которые, в свою очередь, передавали результат в сценарии оболочки, а в конечном итоге – в программы на C. Perl освобождает от обычной философии Unix, заключающейся в использовании многочисленных маленьких программ для обработки мелких частей большой задачи. На Perl делается вся задача, причем есть возможность сделать ее несколькими способами. Фактически эта глава была написана программой искусственного интеллекта, разработанной на Perl. (Это шутка, Ларри.)

Perl предоставляет удобный программный интерфейс для многих функций, которые часто трудно использовать в других языках. Например, распространенной задачей, решаемой сценариями системного администрирования в Unix, является просмотр большого объема текста, выбор полей из каждой строки текста на основе шаблона, обычно представленного в виде регулярного выражения, и составление отчета по этим данным. Допустим, вы хотите обработать вывод Unix-команды *last*, которая выводит журнал времени регистрации всех пользователей системы в таком виде:

```
mdw       ttypf    loomer.vpizza.co Sun Jan 16 15:30 - 15:54  (00:23)
larry     ttyp1    muadib.oit.unc.e Sun Jan 16 15:11 - 15:12  (00:00)
johnsonm  ttyp4    mallard.vpizza.c Sun Jan 16 14:34 - 14:37  (00:03)
jem       ttyq2    mallard.vpizza.c Sun Jan 16 13:55 - 13:59  (00:03)
linus     FTP      kruuna.helsinki. Sun Jan 16 13:51 - 13:51  (00:00)
linus     FTP      kruuna.helsinki. Sun Jan 16 13:47 - 13:47  (00:00)
```

Если нужно просуммировать общее время регистрации каждого пользователя в системе (приведенное в скобках в последнем поле), то можно написать сценарий на *sed*, который выберет значения времени, поданные на вход, затем написать сценарий на *awk*, который отсортирует данные по пользователям и просуммирует время, и еще один сценарий на *awk*, который создаст из полученных данных отчет. Можно также написать довольно сложную программу на C, которая выполнит всю работу. Программа будет сложной, поскольку всем программистам на C известно, что функции обработки текста в этом языке не сильно развиты.

Эта задача легко решается простым сценарием Perl. Средства ввода-вывода, поиск по регулярным выражениям, сортировка по ассоциативным массивам и числовые задачи – все это с легкостью достигается в программах на Perl. Программы на Perl, как правило, коротки и конкретны и не фетишизируют технические проблемы на пути к тому, что ваша программа должна действительно *делать*.

Использование Perl в Linux не отличается от использования в других системах Unix. Имеется несколько хороших книг по Perl, изданных O'Reilly:

«Programming Perl»[1] Ларри Уолла (Larry Wall), Тома Кристиансена (Tom Christiansen) и Джона Орванта (Jon Orwant), «Learning Perl»[2] Рэндала Л. Шварца (Randal L. Schwartz) и Тома Феникса (Tom Phoenix), «Advanced Perl Programming» Шрирама Шринивасана (Sriram Srinivasan) и «Perl Cookbook»[3] Тома Кристиансена и Натана Торкингтона (Nathan Torkington). Тем не менее мы считаем, что Perl служит таким прекрасным инструментом, что заслуживает некоторого введения. В конце концов, Perl является таким же свободно распространяемым программным обеспечением, как и Linux; они идут рука об руку.

Пример программы

Что нам действительно нравится в Perl, так это возможность сразу перейти к решаемой задаче. Нет необходимости писать пространный код для настройки структур данных, открытия файлов и каналов, отводить память под данные и т. д. Обо всем этом за вас позаботятся, причем самым дружественным образом.

Для знакомства с рядом основных функций Perl вернемся к примеру с учетом времени регистрации. Сначала мы представим сценарий целиком (снабженный комментариями), а затем опишем, как он работает. Этот сценарий читает выходные данные команды *last* (см. предыдущий пример) и выводит запись по каждому пользователю системы, содержащую общее время работы в системе и число регистраций. (Номера строк слева проставлены для удобства ссылок):

```
1       #!/usr/bin/perl
2
3       while (<STDIN>) {   # Пока есть входные данные...
4         # Найти строки и запомнить имя пользователя, время регистрации
5         if (/^(\S*)\s*.*\((.*):(.*)\)$/) {
6           # Инкрементировать сумму часов, минут и число регистраций
7           $hours{$1} += $2;
8           $minutes{$1} += $3;
9           $logins{$1}++;
10        }
11      }
12
13      # Для каждого пользователя в массиве ...
14      foreach $user (sort(keys %hours)) {
15         # Вычислить часы по сумме минут
16         $hours{$user} += int($minutes{$user} / 60);
```

[1] Ларри Уолл, Том Кристиансен, Джон Орвант «Программирование на Perl», 3-е издание. – Пер. с англ. – СПб: Символ-Плюс, 2002.

[2] Рэндал Шварц и Том Феникс «Изучаем Perl», 3-е издание. – Пер. с англ. – СПб: Питер, 2002.

[3] Том Кристиансен и Натан Торкингтон «Perl: библиотека программиста». – Пер. с англ. – СПб: Питер, 2001.

```
17          $minutes{$user} %= 60;
18          # Вывести данные по пользователю
19          print "User $user, total login time ";
20          # В Perl тоже есть printf
21          printf "%02d:%02d, ", $hours{$user}, $minutes{$user};
22          print "total logins $logins{$user}.\en";
23      }
```

В строке 1 загрузчику сообщается, что сценарий должен выполняться через Perl, а не командным процессором. Программа начинается в строке 3. Это заголовок простого цикла `while`, с которым должны быть знакомы программисты, использующие C и оболочку: код в скобках в строках 4–10 должен выполняться, пока значение некоторого выражения истинно. Однако условное выражение `<STDIN>` выглядит любопытно. В действительности выражение читает одну строку со стандартного устройства ввода (представляемого в Perl именем `STDIN`) и делает эту строку доступной программе. Значение выражения истинно, пока есть входные данные.

Perl читает входные данные построчно (если не указано иное). По умолчанию ввод производится со стандартного устройства ввода, опять-таки, если не задано иное. Поэтому в данном цикле `while` будут непрерывно читаться данные со стандартного устройства ввода, пока не останется строк для чтения.

То, что вы видите в строке 5, является всего лишь оператором `if`. Как и в большинстве языков программирования, код в скобках (строки 6–9) будет выполнен, если выражение, следующее за `if`, истинно. Но что представляет собой выражение внутри скобок? Читатели, знакомые с такими утилитами Unix, как *grep* и *sed*, сразу определят его как *регулярное выражение*: загадочный, но эффективный способ представить шаблон, который должен быть найден во входном тексте. Регулярные выражения обычно ограничиваются с обеих сторон косыми чертами (/.../).

Данное регулярное выражение соответствует строкам вида:

```
mdw       ttypf    loomer.vpizza.co Sun Jan 16 15:30 - 15:54  (00:23)
```

Это выражение также «запоминает» имя пользователя (`mdw`) и общее время регистрации в системе для этой записи (`00:23`). Само выражение вас пусть не беспокоит: составление регулярных выражений – сложная тема. Пока все, что вам нужно знать, это то, что данный оператор `if` находит строки вида, приведенного в примере, и вычленяет имя пользователя и время работы для дальнейшей обработки. Имя пользователя присваивается переменной `$1`, часы – переменной `$2` и минуты – переменной `$3`. (В Perl переменные начинаются с символа `$`, но, в отличие от оболочки, `$` должен также использоваться и при присвоении значения переменной.) Это присваивание осуществляет само регулярное выражение при нахождении соответствия. (Все, что заключено в регулярном выражении в скобки, сохраняется в переменных от `$1` до `$9` и может быть в дальнейшем использовано).

В строках 6–9 происходит фактическая обработка этих данных. Она происходит интересным образом – через использование *ассоциативного массива*.

В то время как обычный массив индексируется числовым индексом, ассоциативный массив индексируется произвольной строкой. Это находит применение во многих мощных приложениях, позволяя связывать один набор данных с другим, собираемым динамически. В нашей короткой программе ключами служат имена пользователей, получаемые из выходных данных *last*. Мы храним три ассоциативных массива, каждый из которых индексируется именем пользователя: `hours`, в который записывается суммарное количество часов, в течение которых пользователь был зарегистрирован в системе; `minutes`, где записывается количество минут, и `logins`, где записывается общее число регистраций.

Например, ссылка на переменную `$hours{'mdw'}` возвращает общее количество часов, которое пользователь `mdw` провел в системе. Если имя пользователя `mdw` хранится в переменной `$1`, то ссылка на `$hours{$1}` имеет то же значение.

В строках 6–9 мы увеличиваем значения в этих массивах в соответствии с данными в текущей введенной строке. Например, если введена строка:

```
jem       ttyq2    mallard.vpizza.c Sun Jan 16 13:55 - 13:59  (00:03)
```

то строка 7 увеличит значение `hours`, индексированное `$1` (пользователь с именем `jem`), на количество часов, в течение которых пользователь `jem` был зарегистрирован (хранится в переменной `$2`).

В Perl оператор инкрементирования `+=` эквивалентен соответствующему оператору C. В строке 8 аналогичным образом инкрементируется число минут для соответствующего пользователя. В строке 9 на единицу увеличивается значение счетчика регистраций с использованием оператора `++`.

Ассоциативные массивы являются одной из наиболее полезных возможностей Perl. Они позволяют создавать сложные базы данных во время синтаксического анализа текста. Было бы почти невозможно использовать с этой целью обычные массивы. Пришлось бы сначала подсчитать количество различных пользователей во входном потоке, а затем выделить для массива память соответствующего размера, назначив каждому пользователю место в массиве (используя хеширование или иную схему индексации). Ассоциативный же массив позволяет индексировать данные непосредственно с помощью строк, не обращая внимания на размер массива. (Конечно, при работе с большими массивами возникают проблемы скорости обработки, но для большинства приложений они несущественны.)

Пойдем дальше. В строке 14 использован оператор Perl `foreach`, который вам может быть знаком, если вы пишете сценарии оболочки. (Цикл `foreach` на самом деле сводится к циклу `for`, сходному с имеющимся в C.) При каждом проходе цикла переменной `$user` присваивается очередное значение из списка, заданного выражением `sort(keys %hours)`. `%hours` ссылается на весь ассоциативный массив `hours`, который мы создали. Функция `keys` возвращает список всех ключей, использованных для индексирования массива, который в данном случае является списком имен пользователей. Наконец, функция `sort` сортирует список, возвращенный `keys`. В результате цикл перебира-

ет отсортированный список имен пользователей, поочередно присваивая имя пользователя переменной $user.

Строки 16 и 17 просто корректируют положение, когда число минут оказывается большим 60: определяется количество часов, содержащихся в поле minutes данного пользователя, и соответственно увеличивается количество часов в hours. Функция int возвращает целую часть своего аргумента (Perl обрабатывает и числа с плавающей запятой, поэтому необходимо использовать int).

Наконец, в строках 19–22 осуществляется вывод общего времени регистрации и количество входов в систему для каждого пользователя. Простая функция print просто выводит свои аргументы, как одноименная функция в *awk*. Обратите внимание, что вычисление переменных можно производить внутри оператора print, как сделано в строках 19 и 22. Однако если вы хотите осуществлять форматирование текста, нужно использовать функцию printf (аналогичную своему эквиваленту в C). В данном случае мы хотим установить минимальную ширину поля для вывода часов и минут в два символа и дополнить слева нулями недостающие позиции. Для этого используется команда printf в строке 21.

Если сохранить этот сценарий в файле *logintime*, то выполнить его можно следующей командой:

```
papaya$ last | logintime
User johnsonm, total login time 01:07, total logins 11.
User kibo, total login time 00:42, total logins 3.
User linus, total login time 98:50, total logins 208.
User mdw, total login time 153:03, total logins 290.
papaya$
```

Конечно, этот сценарий не вполне подходит в качестве учебника по Perl, но он должен дать вам представление о его возможностях. Мы советуем почитать одну из имеющихся прекрасных книг по Perl, чтобы узнать о нем больше.

Некоторые дополнительные функции

В предыдущем примере мы познакомились с некоторыми часто используемыми в Perl функциями, представив пример живой рабочей программы. Однако в Perl много других хорошо известных и не столь хорошо известных функций.

Как уже говорилось, Perl обеспечивает механизм создания отчетов, выходящий за рамки стандартных функций print и printf. С помощью этого механизма программист задает «формат», описывающий вид каждой страницы отчета. Например, мы могли бы включить такое определение формата в нашу программу:

```
format STDOUT_TOP =
User           Total login time     Total logins
-------------- -------------------- -------------------
```

```
format STDOUT =
@<<<<<<<<<<<<< @<<<<<<<<            @####
$user,         $thetime,           $logins{$user}
```

Определение STDOUT_TOP описывает заголовок отчета, который будет печататься вверху каждой страницы выдачи. Формат STDOUT описывает вид каждой строки выдачи. Описание каждого поля начинается с символа @; @<<<< указывает на выровненное по левому краю текстовое поле, а @#### задает числовое поле. Строка под описаниями полей задает имена переменных, которые должны выводиться в этих полях. Мы использовали переменную $thetime для хранения отформатированной строки времени.

Для того чтобы использовать при выводе этот отчет, заменим строки 19–22 в исходном сценарии на следующие:

```
$thetime = sprintf("%02d:%02d", $hours{$user}, $minutes{$user});
write;
```

Первая строка использует функцию sprintf для форматирования строки времени и сохранения ее в переменной $thetime; вторая строка является командой *write*, сообщающей Perl о необходимости вывести строку с использованием установленного формата отчета.

Используя такой формат, мы должны получить нечто вроде следующего:

```
User           Total login time     Total logins
-------------- -------------------- -------------------
johnsonm       01:07                   11
kibo           00:42                    3
linus          98:50                  208
mdw            153:03                 290
```

С помощью других форматов отчета можно получить другие (и лучше выглядящие) результаты.

Perl поставляется с огромным количеством модулей, которые можно вставлять в программы, чтобы получать быстрый доступ к очень мощным функциям. В популярном электронном архиве CPAN (Comprehensive Perl Archive Network – полный сетевой архив Perl) содержится еще больше модулей: сетевые модули, позволяющие отправлять электронную почту и выполнять другие сетевые задачи, модули для дампа данных и отладки, модули для обработки дат и времени, модули для математических функций. Этот список можно продолжить на многие страницы.

Узнав об интересном модуле, проверьте сначала, не установлен ли он уже в вашей системе. Вы можете просмотреть каталоги, где размещаются модули (вероятно, в */usr/lib/perl5*), или попытаться загрузить модуль и посмотреть, как он работает. Например, команда

```
$ perl -MCGI -e 1
Can't locate CGI in @INC...
```

с прискорбием извещает, что модуль *CGI.pm* отсутствует в вашей системе. Модуль CGI.pm весьма популярен и включается в стандартный дистрибутив

Perl, откуда его можно установить. Однако за многими другими модулями вам придется обратиться к CPAN (а некоторых нет и в CPAN). CPAN, сопровождаемый Ярко Хиетаньеми (Jarkko Hietaniemi) и Андреасом Кенигом (Andreas Konig), лежит на десятках зеркальных сайтов по всему миру, поскольку очень многие хотят загрузить его модули. Проще всего попасть в CPAN, посетив *http://www.perl.com/CPAN-local/*.

Следующая программа, которую мы хотели сделать короткой, и потому не стали искать практическую задачу, показывает два модуля, один из которых обрабатывает даты и время довольно сложным образом, а другой отправляет электронную почту. Недостатком использования таких мощных функций является необходимость загрузки для них огромного объема кода, что делает весьма большим размер программы при выполнении:

```
#! /usr/local/bin/perl

# Демонстрация работы модулей Date и Mail
use Date::Manip;
use Mail::Mailer;

# Иллюстрация использования модуля Date::Manip
if ( Date_IsWorkDay( "today", 1) )  {

    # Сегодня рабочий день
    $date = ParseDate( "today" );

}
else {

    # Сегодня нерабочий день, поэтому выберем очередной рабочий день
    $date=DateCalc( "today" , "+ 1 business day" );

}

# Преобразуем дату из компактной формы в читаемую "April  8"
$printable_date = UnixDate( $date , "%B %e" );

# Иллюстрация работы модуля Mail::Mailer
my ($to) = "the_person\@you_want_to.mail_to";
my ($from) = "owner_of_script\@system.name";

$mail = Mail::Mailer->new;

$mail->open(
            {
                From => $from,
                To => $to,
                Subject => "Автоматическое напоминание",
            }
           );

print $mail <<"MAIL_BODY";
Вы получите это почтовое сообщение,
если находитесь на работе
$printable_date,
или позже.
```

```
MAIL_BODY

$mail->close;

# Почта отправлена! (Предполагается, что не было ошибок.)
```

Пакеты очень просто использовать благодаря тому, что в версию Perl 5 добавлены функции объектно-ориентированного программирования. Модуль Date, использованный в предыдущем примере, не является объектно-ориентированным, в отличие от модуля Mail. Переменная `$mail` является объектом типа Mailer, упрощая работу почты благодаря таким методам, как `new`, `open` и `close`.

Для солидных задач, таких как синтаксический разбор HTML, нужно считать подходящий пакет CGI, использовать команду `new`, чтобы создать подходящий объект, и в вашем распоряжении окажутся все функции, необходимые для разбора HTML.

При желании снабдить свой Perl-сценарий графическим интерфейсом воспользуйтесь модулем Tk, который первоначально был разработан для применения с языком Tcl, или модулем Gtk, который использует более новый GIMP Toolkit (GTK), или модулем Gt, использующим Gt-инструментарий, являющийся основой и для KDE. В книге «Learning Perl/Tk» Нэнси Уолш (Nancy Walsh), O'Reilly, рассказано, как создавать графику с помощью этого модуля.

Другой скрытой возможностью Perl является его способность более или менее прямо осуществлять некоторые системные вызовы Unix, в том числе для связи между процессами. Например, Perl предоставляет доступ к функциям `msgctl`, `msgget`, `msgsnd` и `msgrcv` из System V IPC. Perl поддерживает также реализацию сокетов BSD, что позволяет осуществлять коммуникации по TCP/IP непосредственно из программы на Perl. Теперь C не является единственным языком для сетевых демонов и клиентов. Программа на Perl с функциями IPC может быть очень мощной, особенно с учетом того, что многие реализации систем клиент-сервер нуждаются в развитых функциях обработки текста, таких как предоставляемые Perl. Обычно проще делать разбор команд протокола, которыми обмениваются клиент и сервер в сценарии Perl, чем писать для этого сложную программу на C.

В качестве примера возьмем хорошо известный демон SMTP, управляющий отправкой и получением электронной почты. Протокол SMTP использует такие внутренние команды, как `recv from` и `mail to`, для предоставления клиенту возможности коммуникаций с сервером. Как клиент, так и сервер могут быть написаны на Perl, и иметь полный доступ к имеющимся в Perl функциям обработки текстов и файлов, а также важным функциям связи через сокеты.

Perl прочно обосновался в CGI-программировании, т. е. написании небольших программ, выполняемых на веб-сервере и делающих веб-страницы более интерактивными.

За и против

Одной из особенностей (некоторые говорят «проблем») Perl является его способность значительно сокращать – и делать непонятным – код. В предыдущем примере мы использовали несколько общепринятых сокращений. Например, входные данные в сценарии Perl считываются в переменную $_. Однако большинство операций производится с переменной $_ по умолчанию, поэтому нет необходимости обращаться к ней по имени.

Perl часто дает возможность сделать одно и то же разными способами, что может оказаться благодатью или проклятием, в зависимости от того, как на это посмотреть. В «Programming Perl» Ларри Уолл приводит такой пример короткой программы, которая просто выводит то, что поступает на ее стандартный ввод. Все нижеследующие операторы эквивалентны:

```
while ($_ = <STDIN>) { print; }
while (<STDIN>) { print; }
for (;<STDIN>;) { print; }
print while $_ = <STDIN>;
print while <STDIN>;
```

Программист может использовать наиболее подходящий в его ситуации синтаксис.

Perl популярен не только благодаря своей полезности. Программистов привлекает его эксцентричность. Программисты на Perl постоянно соревнуются в том, кто напишет более замысловатый код. Perl идеально подходит для интересных клуджей, изящных хаков и как очень хорошего, так и очень плохого стилей программирования. Программисты в Unix считают его сложным языком для работы, поскольку Perl относительно нов и не все его возможности разработаны. Даже если Perl на ваш вкус слишком причудлив, его все же можно оценить за артистичность. Возможность назвать себя «крутым программистом на Perl» (Perl hacker) служит предметом гордости в сообществе Unix.

Java

Java – это поддерживающий работу в сети объектно-ориентированной язык, разработанный Sun Microsystems. Java вызывает очень большой интерес в компьютерном сообществе, поскольку стремится предоставить безопасный язык для запуска приложений, загружаемых с сайтов World Wide Web.

Идея проста: позволить веб-броузерам загружать апплеты Java, которые могут выполняться на машине клиента. Популярный броузер Netscape (обсуждаемый в главе 1), а также его версия open source Mozilla, версия из GNOME – Galeon и веб-броузер из KDE Konqueror – все они содержат поддержку Java. Кроме того, Java Developer's Kit и другие инструменты перенесены на Linux. Но язык Java пригоден не только для создания таких апплетов. В последнее время он все чаще используется в качестве универсального языка программирования, не представляющего большой сложности для начинаю-

щих и часто используемого для программирования приложений в архитектуре клиент/сервер благодаря встроенным сетевым библиотекам. Ряд учебных заведений также выбрали Java для своих курсов программирования.

Перспективы Java

Все это может не вызвать у вас вдохновения. В конце концов, существует много объектно-ориентированных языков программирования, а с помощью подключаемых модулей Netscape можно загружать с веб-серверов выполняемые модули и запускать их на локальной машине.

Но Java дает больше. Одной из самых впечатляющих ее особенностей является *независимость от платформы*. Это означает, что можно написать на Java программу и скомпилировать ее, а затем установить почти на любой машине, будь то скромный 386-й компьютер под Linux, мощный Pentium II с последней раздутой системой Microsoft или мэйнфрейм IBM. Sun Microsystems называет это «Write Once, Run Anywhere», т. е. «однажды написав, выполняешь всюду». К сожалению, действительность сложнее, чем цели проектирования. Существуют небольшие, но срывающие все планы различия, из-за которых программа работает на одной платформе и отказывается работать на другой. С появлением новой библиотеки GUI Swing в Java 2 был сделан большой шаг для исправления этого положения.

Эта изящная возможность скомпилировать код один раз и иметь после этого возможность выполнять его на других машинах осуществляется благодаря виртуальной машине Java – Java Virtual Machine (JVM). Компилятор Java не создает объектный код для конкретного типа процессора и операционной системы, как это делает *gcc*, он генерирует код для JVM. Эта машина не существует в железе (пока), а является спецификацией. В этой спецификации сказано, какие так называемые коды операций (opcodes) понимает машина, и что она делает, когда встречает их в объектном файле. Программа распространяется в двоичном виде, содержащем так называемые *байт-коды*, соответствующие спецификации JVM.

Теперь все, что вам нужно, – это программа, реализующая виртуальную машину Java на вашем конкретном компьютере и операционной системе. В данный момент они имеются практически для каждой платформы – ни один поставщик не может позволить себе не предоставить виртуальную машину Java вместе со своим оборудованием или операционной системой. Эта программа называется также *интерпретатором* Java, поскольку интерпретирует коды операций, скомпилированные для JVM, и транслирует их в код для конкретной машины.

Эта отличительная особенность, делающая Java одновременно компилируемым и интерпретируемым языком, позволяет вам написать и скомпилировать свою Java-программу и передать ее кому-нибудь другому, и она сможет выполняться при наличии интерпретатора Java независимо от аппаратной платформы и операционной системы.

Увы, за независимость Java от платформы приходится платить. Поскольку объектный код не является объектным кодом для реально существующего аппаратного обеспечения, его необходимо дополнительно обработать, в результате чего написанные на Java программы выполняются в десять-двадцать раз медленнее, чем аналогичные программы, написанные, скажем, на C. В одних случаях это не имеет значения, а в других – просто недопустимо. Существуют так называемые компиляторы Just-In-Time (JIT), которые сначала транслируют объектный код для виртуальной машины Java в родной объектный код, а затем его выполняют. Когда тот же объектный код нужно выполнить вторично, можно использовать прекомпилированный собственный код (native code) без всякой интерпретации, в результате чего выполнение происходит быстрее, но все же скорости C-программ не достижимы. Sun Microsystems работает над технологией, которая якобы должна обеспечить скорость выполнения, сравнимую с программами на C, но возможность выполнения таких обещаний остается неясной.

В Java также различают *приложения* (*applications*) и *апплеты* (*applets*). Приложения являются самостоятельными программами, которые можно запустить из командной строки или с рабочего стола, и они будут вести себя как обычные приложения. Напротив, апплеты являются программами (обычно меньшего размера), выполняемыми внутри веб-броузера. (Для работы этих программ в броузер должен быть встроен интерпретатор Java.) При просмотре веб-страницы, содержащей апплет Java, веб-сервер посылает вам объектный код апплета, и броузер его выполняет. Это может быть использовано для любых задач – от простой мультипликации до полных интерактивных банковских систем.[1]

Читая об апплетах Java, вы могли подумать: «А что если в апплете содержится злонамеренный код, который шпионит за данными на моем жестком диске, а возможно, даже уничтожает или портит файлы?» Разумеется, это было бы возможно, если бы разработчики Java не предусмотрели многоэтапные контрмеры против таких атак: все апплеты Java выполняются в так называемой «песочнице», предоставляющей им только ограниченный доступ к ресурсам. Например, апплеты Java могут выводить текст на экран, но не могут читать данные из локальной файловой системы и даже записывать в нее, не получив на это вашего явного разрешения. Хотя эта парадигма песочницы уменьшает полезность апплетов, но в то же время повышает степень защищенности ваших данных. В последних версиях Java можно установить требуемый уровень защиты и получить благодаря этому дополнительную гибкость. Следует упомянуть, что есть сведения о наличии серьезных брешей в системе безопасности, вызванных использованием Java в броузерах, хотя все те, которые обнаружены, исправлены в современных веб-броузерах.

---

[1] Один из авторов все свои финансовые операции с банком производит через апплет Java, поставляемый банковским сервером при просмотре определенного раздела веб-сервера банка.

Если вы заинтересовались Java, мы можем порекомендовать книгу Брюса Эккеля (Bruce Eckel) «Thinking in Java» (Prentice Hall). Она освещает большинство вещей, которые нужно знать о Java, а также учит основным принципам программирования. Среди других книг по Java, в которые стоит заглянуть, отметим «Exploring Java» Пат Нимейер (Pat Niemeyer) и Джоша Пек (Josh Peck) (O'Reillly), а также «Core Java» Кэя Хорстмана (Cay Horstmann) и Гарри Корнелла (Gary Cornell) (Prentice Hall).

Как получить Java для Linux

К счастью, на Linux перенесен так называемый JDK (Java Developers Kit), поставляемый Sun Microsystems для Solaris и Windows и являющийся реализацией Java, на которую нужно ориентироваться. В прошлом существовал разрыв между появлением новой версии JDK для Solaris и Windows и наличием JDK для Linux. В настоящее время такой проблемы нет.

«Официальная» Java-реализация JDK содержит компилятор, интерпретатор и несколько сопутствующих программных средств. Существуют и другие комплекты для Linux, часто в виде Open Source. Здесь мы расскажем о JDK, поскольку он является стандартом. Есть и другие реализации для Linux, в том числе очень хорошая производства IBM. Возможно, одна из них есть в вашем дистрибутиве.

Следует отметить, что в большинстве дистрибутивов уже есть JDK для Linux, поэтому может оказаться проще установить тот, который содержится в пакете. Однако JDK быстро развивается, поэтому может оказаться предпочтительнее установить более новый пакет, чем тот, который содержится в вашем дистрибутиве.

Все необходимое для работы с Java в Linux можно найти по адресу *http://www.blackdown.org*. Здесь есть документация, новости о переносах для Linux и ссылки на адреса, откуда можно загрузить JDK для вашей машины.

После распаковки и установки JDK согласно имеющимся инструкциям в вашем распоряжении окажется несколько новых программ: *javac* – компилятор Java, *java* – интерпретатор, *appletviewer* – небольшая программа с графическим интерфейсом, позволяющая запускать апплеты без полномасштабного веб-броузера, и т. д.

Рабочий пример Java

Следующая программа является законченной Java-программой, которая может работать и самостоятельно, и в виде апплета. Это небольшая программа рисования, позволяющая чиркать на виртуальном холсте. В ней также реализованы некоторые элементы GUI, такие как кнопка и меню.

Та часть Java, которая позволяет использовать элементы GUI, такие как окна и меню, называется Abstract Window Toolkit (AWT). Она тоже соответствует обещанию Java «написав однажды, выполняешь всюду». Несмотря на то что оконные системы таких операционных систем, как Linux, Windows и Macintosh, совершенно различны, код интерфейса пользователя нужно написать

только один. AWT отображает независимые от платформы элементы в библиотеку GUI конкретной архитектуры. Поэтому ваша программа будет иметь вид, соответствующий системе окон машины, на которой вы ее выполняете.

AWT имеет ряд недостатков, например различный вид программы на разных платформах (некоторые считают это достоинством, поскольку программа выглядит родной для системы приложение), недостаточная скорость и многочисленные ошибки. Новый пакет Swing, появившийся с Java 2, устраняет многие из этих недостатков, например различный внешний вид (путем предоставления подключаемых стилей представления), и реализует значительно больше элементов интерфейса пользователя, например таблицы и древовидные структуры, однако работает еще медленнее.

Перейдем к делу. Вот код программы для рисования:

```
import java.applet.*;
import java.awt.*;
import java.awt.event.*;
/** Апплет, который может выполняться как самостоятельное приложение */
public class StandaloneScribble extends Applet {
  /**
   * Метод main(). Если программа вызвана как приложение,
   * этот метод создаст необходимое окно, добавит в него апплет
   * и вызовет init(). Обратите внимание, что Frame использует PanelLayout
   * по умолчанию.
   */
  public static void main(String[] args) {
    Frame f = new Frame();                      // Создать окно
    Applet a = new StandaloneScribble();        // Создать панель апплета
    f.add(a, "Center");                         // Добавить апплет в окно
    a.init();                                   // Инициализировать апплет
    f.setSize(400, 400);                        // Установить размер окна
    f.show();                                   // Сделать окно видимым
    f.addWindowListener(new WindowAdapter() {   // Обработка запросов
                                                // закрытия окна
      public void windowClosing(WindowEvent e) { System.exit(0); }
    });
  }
  /**
   * Метод init(). Если программа вызывается как апплет, броузер
   * отводит для него место на экране и вызывает этот метод для настройки.
   */
  public void init() {
    //Определить, создать экземпляр и зарегистрировать объект MouseListener
    this.addMouseListener(new MouseAdapter() {
      public void mousePressed(MouseEvent e) {
        lastx = e.getX();
        lasty = e.getY();
      }
    });
    //Определить, создать экземпляр, зарегистрировать объект MouseMotionListener
    this.addMouseMotionListener(new MouseMotionAdapter() {
```

```
        public void mouseDragged(MouseEvent e) {
          Graphics g = getGraphics();
          int x = e.getX(), y = e.getY();
          g.setColor(Color.black);
          g.drawLine(lastx, lasty, x, y);
          lastx = x; lasty = y;
        }
      });
      // Создать кнопку очистки
      Button b = new Button("Clear");
      // Определить, создать экземпляр и зарегистрировать функцию
      // обработки нажатий кнопки
      b.addActionListener(new ActionListener() {
        public void actionPerformed(ActionEvent e) {  // удалить каракули
          Graphics g = getGraphics();
          g.setColor(getBackground());
          g.fillRect(0, 0, getSize().width, getSize().height);
        }
      });

      // Добавить кнопку к апплету
      this.add(b);
    }
    protected int lastx, lasty;  // Координаты последнего щелчка мышью
  }
```

Сохраните этот код в файле с именем *StandaloneScribble.java*. Имя очень важно: оно должно совпадать с именем реализованного класса и иметь расширение *.java*. Для компиляции этого кода выполните команду:

```
$tigger javac StandaloneScribble.java
```

Компиляция может занять некоторое время. Компилятор Java не очень быстр, поскольку он сам написан, в основном, на Java.[1] В результате компиляции создается файл *StandaloneScribble.class* и некоторые другие файлы *.class*, о которых мы здесь говорить не будем.

Теперь можно запустить эту программу из командной строки, выполнив команду:

```
$tigger java StandaloneScribble
```

Если JDK установлен правильно, будет открыто окно, в котором можно рисовать. Обратите внимание, что в качестве аргумента команде Java передана строка *StandaloneScribble* без расширения *.class*. Это связано с тем, что формально интерпретатор выполняет не файл, а класс.

---

[1] Существуют другие, более быстрые компиляторы Java. Например, Jikes – быстрый, как молния (по крайней мере, в сравнении с *javac*), компилятор Java, который можно загрузить с *http://www.alphaworks.ibm.com*.

Вы можете также запустить эту программу из веб-броузера или программы просмотра апплетов JDK. Для этого нужен короткий код HTML. Достаточно будет такого:

```
<APPLET code="StandaloneScribble.class" width=150 height=100>
</APPLET>
```

Сохраните этот код в файле и откройте его в броузере типа Netscape Navigator или программе *appletviewer*, и вы увидите программу в окне броузера.

В завершение раздела рассмотрим некоторые наиболее интересные строки программы. В первых трех строках импортируются другие Java-классы, поступающие из JDK. Это можно сравнить с включением файлов заголовков в программу C, хотя в Java нет этапа компоновки. Во время выполнения программы интерпретатору Java необходимо найти импортируемые классы. Он делает это, осуществляя поиск в каталогах, относящихся к самому двоичному модулю интерпретатора и указанных в переменной окружения `CLASSPATH`, которую вы могли установить (что необязательно для данной простой программы).

Первый большой блок содержит метод `main()`. Когда программа выполняется как самостоятельное приложение, этот метод вызывается интерпретатором, чтобы начать выполнение программы. В этом методе создается окно, которое используется затем для вывода на экран.

Большую часть оставшегося кода занимает метод `init()`. Он вызывается из `main()` при самостоятельном выполнении программы или непосредственно броузером при работе в качестве апплета. В последнем случае `main()` вообще не выполняется.

Другие языки

Существует много других популярных (и не очень популярных) языков для Linux. По большей части они работают в Linux так же, как в других Unix-системах, поэтому много нового здесь не скажешь. Кроме того, их так много, что особенно подробно мы их не сможем осветить. Однако мы все же хотим проинформировать вас о том, что есть в наличии, и рассказать о некоторых различиях между языками и компиляторами.

В последнее время большое внимание привлек Python как мощное соединение различных парадигм и стилей программирования. Например, он является одним из очень немногих интерпретируемых и одновременно объектно-ориентированных языков (другим таким языком является Perl, но эти черты он приобрел относительно недавно). Поклонники Python утверждают, что он исключительно прост в изучении. Python был почти полностью спроектирован и написан Гвидо ван Россумом (Guido van Rossum). Язык получил такое название, поскольку автор во время написания интерпретатора смотрел по британскому телевидению шоу «Monty Python's Flying Circus». Введение в Python можно найти в «Learning Python» Марка Лутца (Mark Lutz) и

Дэвида Эшера (David Ascher), а детально познакомиться с ним по «Programming Python»[1] Марка Лутца (Mark Lutz) – обе книги изданы O'Reilly.

У Perl, как он ни прекрасен и полезен, есть один недостаток – по крайней мере, так считают многие: один и тот же код можно написать многими различными способами. С этим связано мнение о Perl, как языке, на котором просто написать код, но трудно прочесть его. (Имеется в виду, что другой программист может сделать все не так, как вы, а вам будет трудно разобраться с его стилем.)

Из этого следует, что Perl может оказаться не тем языком, на котором следует разрабатывать приложения, требующие сопровождения в течение последующих лет.

Если вы привыкли разрабатывать приложения на C, C++ или Java, но иногда вам нужно писать сценарии, синтаксис Perl может показаться вам слишком непривычным – например, нужно вводить знак доллара перед именем переменной:

```
foreach $user ...
```

Прежде чем более подробно рассматривать, что представляет собой Python, выскажем предположение, что выбор между Perl и Python в значительной мере является вопросом «религии» – так же как выбор между Emacs и *vi* или между KDE и GNOME. Perl и Python заполняют собой пробел между реальными языками типа C/C++/Java и языками сценариев типа tcx, которые встроены в *bash* или *tcsh*.

В противоположность Perl, Python с самого начала проектировался как реальный язык программирования, многие конструкции которого навеяны C. Из этого следует, что программы Python проще читать, чем написанные на Perl, хотя они получаются несколько длиннее.

Python объектно-ориентированный язык, но вам не обязательно программировать в объектно-ориентированном стиле, если вы этого не хотите. Благодаря этому можно начать писать сценарий, не используя объектной технологии, но по мере продвижения вперед и увеличения размеров и сложности сценария легко преобразовать его для использования объектов.

Сценарии Python интерпретируются, т. е. не требуется ждать конца долгого процесса компиляции. Программы Python динамически компилируются в байт-код, благодаря чему они все же работают с приличной скоростью. В повседневной работе этого можно и не заметить, за исключением того, что если вы напишете файл *.py*, Python создаст файл *.pyc*.

В Python есть списки, кортежи, строки и ассоциативные массивы (или, как принято называть в Python, «словари» (*dictionaries*)), которые встроены в синтаксис языка, что очень упрощает работу с этими типами.

---

[1] Марк Лутц «Программирование на Phyton», 2-е издание. – Пер. с англ. – СПб: Символ-Плюс, 2002.

Python поставляется с обширной библиотекой, близкой по мощи к той, которую мы видели в Perl. Полный справочник по библиотеке см. на *http://www.python.org/doc/current/lib/lib.html*.

Завершим это краткое знакомство с Python рассмотрением небольшого сценария Python. Надеемся, оно даст вам представление о том уровне абстракции, на котором работает Python. Этот сценарий выполняет ту же работу, что и предшествующий сценарий Perl. Первое впечатление – этот сценарий существенно длиннее, чем сценарий Perl. Учтите, что тут мы заставляем Python «соревноваться» с Perl в той области, где последний наиболее силен. Компенсацией служит то, что этот сценарий, по нашему мнению, проще читать.

Обратите также внимание на отступы. В то время как в большинстве языков отступы необязательны и просто улучшают читаемость кода, в Python они обязательны и составляют одну из его отличительных особенностей.

```
1    #!/usr/bin/python
2
3    import sys, re, string
4
5    minutes = {}
6    count = {}
7    line = sys.stdin.readline()
8    while line:
9      match = re.match( "^(\S*)\s*.*\(([0-9]+):([0-9]+)\)\s*$", line )
10     if match:
11       user = match.group(1)
12       time = string.atoi(match.group(2))*60 + string.atoi(match.group(3))
13       if not count.has_key( user ):
14         minutes[ user ] = 0
15         count[ user ]   = 0
16       minutes[ user ] += time
17       count[user] += 1
18     line = sys.stdin.readline()
19
20   for user in count.keys():
21     hour = `minutes[user]/60`
22     min = minutes[user] % 60
23     if min < 10:
24       minute = "0" + `min`
25     else:
26       minute = `min`
27     print "User " + user + ", total login time " + \
28           hour + ":" + minute + \
29           ", total logins " + `count[user]`
```

Этот сценарий должен быть самоочевиден, за некоторыми исключениями. В строке 3 импортируются необходимые для работы библиотеки. Например, импортировав `string`, мы можем в строке 12 воспользоваться этой библиотекой, применив входящий в нее метод *atoi*.

В строках 5 и 6 инициализируются два словаря. В отличие от Perl, мы должны инициализировать их, прежде чем присваивать им значения. Строка 7 читает строку из стандартного ввода. Когда строк для чтения больше нет, метод *readline* возвращает `None`, что эквивалентно нулевому указателю.

Строка 9 сравнивает строку, прочтенную из `stdin`, с регулярным выражением и возвращает объект соответствия. У этого объекта есть метод для доступа к отдельным элементам совпадения. В строке 21 результат деления `minutes[user]/60` преобразуется в строку. Это осуществляется с помощью двух обратных апострофов.

Все необходимое о Python можно прочесть на *http://www.python.org* или в «Learning Python» Марка Лутца (Mark Lutz) и Дэвида Эшера (David Ascher), а также в «Programming Python» Марка Лутца (Mark Lutz) – обе книги изданы O'Reilly.

Еще одна новая разработка в области языков сценариев – язык Ruby, созданный в Японии и нашедший там большое число поклонников. Это объектно-ориентированный язык сценариев, который еще шире, чем Python (если это возможно), использует объекты.

Tcl (Tool Command Language) – язык, предназначавшийся в качестве связующего между различными программами и прославившийся, главным образом, своим простым инструментарием создания оконного интерфейса, Tk.

LISP является интерпретируемым языком, используемым во многих приложениях – от искусственного интеллекта до статистики. Он применяется преимущественно в компьютерных исследованиях, поскольку определяет ясный логический интерфейс для работы с алгоритмами. (В нем также используется множество скобок, которые так любят теоретики вычислительной техники.) Это функциональный и очень обобщенный язык программирования. Многие операции определяются в терминах рекурсии, а не линейных циклов. Выражения являются иерархическими, а данные представляются списками элементов.

Существует несколько интерпретаторов LISP для Linux. Emacs LISP является довольно полной реализацией языка с рядом возможностей, позволяющих непосредственно взаимодействовать с Emacs – например, осуществлять ввод и вывод через буферы Emacs, но он может также использоваться и для приложений, не связанных с Emacs.

Следует упомянуть также CLISP – реализацию Common LISP Бруно Хайбле (Bruno Haible) из университета Карлсруе и Микаэля Штолля (Michael Stoll) из университета Мюнхена. В него входят интерпретатор, компилятор и подмножество CLOS (Common LISP Object System – объектно-ориентированное расширение LISP). Существует также CLX – интерфейс Common LISP к X Window System, который работает под CLISP. CLX позволяет писать на LISP приложения для X. Austin Kyoto Common LISP – еще одна реализация LISP, которая также совместима с CLX.

Существует еще SWI-Prolog – полная реализация Prolog Яна Вилемакера (Jan Wielemaker) из университета Амстердама. Prolog – это основанный на логике язык, позволяющий делать логические утверждения, определять эвристики для проверки этих утверждений и принимать основанные на них решения. Это полезный язык для создания приложений искусственного интеллекта.

Имеется также несколько интерпретаторов Scheme, из которых следует упомянуть MIT Scheme – полный интерпретатор Scheme, отвечающий стандарту R4. Scheme является диалектом LISP, предлагающим более гладкую и общую модель программирования. Это хороший диалект LISP для применения в теоретической вычислительной технике и изучения алгоритмов.

Существуют по крайней мере две реализации Ada: AdaEd – интерпретатор Ada и GNAT – GNU Ada Translator. GNAT является развитым оптимизирующим компилятором Ada. Для Ada это то же, что *gcc* для C и C++.

Кроме того, имеется еще два популярных транслятора для Linux: *p2c* – транслятор с Pascal в C и *f2c* – транслятор с FORTRAN в C. Если вы сомневаетесь, что эти два транслятора могут действовать как настоящие компиляторы, то напрасно. Как *p2c*, так и *f2c* показали себя зрелыми и полезными продуктами для серьезного применения с Pascal и FORTRAN.

f2c совместим с FORTRAN-77, а также имеет ряд вспомогательных утилит. *ftnchek* является программой контроля FORTRAN, аналогичной *lint*. С помощью *f2c* на Linux были перенесены библиотека численных методов LAPACK и библиотека FORTRAN многократно увеличенной точности – *mpfun*. *toolpack* является коллекцией утилит FORTRAN, таких как средство печати исходных текстов, преобразователь точности и программа проверки переносимости.

Среди других разнообразных языков, доступных в Linux, можно указать интерпретаторы APL, Rexx, Forth, ML, Eiffel и транслятор Simula-to-C. GNU-версии средств компиляции *lex* и *yacc* (переименованные в *flex* и *bison*, соответственно), используемые во многих программных пакетах, также перенесены на Linux. *lex* и *yacc* идеально подходят для создания разного рода синтаксических анализаторов или трансляторов и чаще всего используются при написании компиляторов.

14

Инструментарий программиста

Многие оценивают компьютерную систему по тому инструментарию, который она предоставляет программистам. По распространенному мнению Unix-системы выигрывают это соревнование, накопив с годами очень богатый набор таких средств. Возглавляет список отладчик GNU – *gdb*. В этой главе мы подробно познакомимся с этой ценной утилитой и рядом других вспомогательных средств, полезных для программистов на C.

Даже если вы не программист, стоит познакомиться с системой контроля ревизий (Revision Control System, RCS). Она предоставляет одно из наиболее надежных средств защиты, о котором может мечтать пользователь компьютера, а именно: создание файла с резервной копией всего, что вы делаете. Если вы случайно удалите файл или решите, что все, что вы делали в течение последней недели, было ошибкой и должно быть удалено, RCS поможет восстановить любую версию, которую вы пожелаете. Если вы работаете над большим проектом, в котором участвует большое число разработчиков или большое количество каталогов, то, возможно, больше подойдет система параллельных версий (Concurrent Versioning System, CVS). Изначально она была основана на RCS, но затем была переписана с самого начала и теперь содержит много дополнительных функций. В данное время ее еще раз переписывают с нуля. Новый инструмент будет носить название *Subversion*. Однако неизвестно, сможет ли он заменить CVS. Наконец, для самого ядра Linux используется еще одна система контроля версий под названием BitKeeper.

Отладка с помощью gdb

Не принадлежите ли вы к тем программистам, которые высмеивают саму мысль об использовании отладчика для трассировки кода? Вы считаете, что если код настолько сложен, что даже программист не может его понять, то не стоит его жалеть, если обнаруживаются ошибки? Проходите ли вы мысленно по своему коду, используя увеличительное стекло и зубочистку? Вызываются ли ошибки зачастую единственным пропущенным символом, например использованием оператора =, когда подразумевается += ?

Возможно, вам следует познакомиться с *gdb* – отладчиком GNU. Знаете вы об этом или нет, но *gdb* – ваш друг. Он может локализовать неясные и трудно обнаружимые ошибки, которые приводят к дампам памяти, утечке памяти и странному поведению (как программы, так и программиста). Иногда самые невинно выглядящие ошибки в вашей программе могут совершенно расстроить ее работу, и без помощи отладчика, такого как *gdb*, отыскать корень проблемы становится почти невозможно, особенно если в программе сотни и более строк. В этом разделе мы на нескольких примерах продемонстрируем использование наиболее полезных функций *gdb*. Кроме того, имеется выпущенная Фондом свободного программного обеспечения книга «Debugging with GDB», написанная Ричардом Столменом (Richard M. Stallman), Роландом Пешем (Roland Pesch), Стэном Шебсом (Stan Shebs) и др.

gdb может отлаживать программы во время выполнения или искать причину аварии программы по дампу памяти. Программы, отлаживаемые во время выполнения, могут запускаться из самого *gdb* или отдельно. Это означает, что *gdb* может подключиться к уже выполняемому процессу, чтобы исследовать его. Сначала мы обсудим, как отлаживать программы, запущенные внутри *gdb*, а затем перейдем к подключению к работающим процессам и изучению дампов памяти.

Трассировка программы

Нашим первым примером будет программа *trymh*, определяющая границы изображения в оттенках серого цвета. *trymh* принимает в качестве исходных данных графический файл, производит некоторые вычисления, основываясь на полученных данных, и выдает результат в виде другого графического файла. К сожалению, программа вызывает аварийную ситуацию сразу при запуске:

```
papaya$ trymh < image00.pgm > image00.pbm
Segmentation fault (core dumped)
```

Теперь мы могли бы с помощью *gdb* проанализировать полученный файл с дампом памяти, но в этом примере мы покажем, как трассировать программу во время выполнения.[1]

Прежде чем использовать *gdb* для трассировки выполняемого модуля *trymh*, необходимо обеспечить, чтобы он был скомпилирован с отладочной информацией (см. раздел «Языки программирования и утилиты» главы 1). Для этого нужно компилировать *trymh* с помощью *gcc* с ключом *-g*.

Обратите внимание, что одновременное использование ключей *-O* (для оптимизации) и *-g* (для включения отладочной информации) допустимо, но не рекомендуется. Например, если в двух различных местах функции есть одинаковые строки, то *gdb* может неожиданно перескочить ко второму включению строки вместо ожидавшегося первого. Это происходит из-за того, что

[1] Примеры программ, приведенные в этом разделе, вы вряд ли где-нибудь будете запускать; они были набросаны авторами в демонстрационных целях.

gcc объединил две строки в одну строку машинного кода, используемую в обоих случаях.

Некоторые виды автоматической оптимизации, осуществляемой *gcc*, могут сбивать с толку при использовании отладчика. Для отключения всякой оптимизации (даже той, которая производится без указания ключа *-O*) используйте при компиляции *gcc* ключ *-O0* (дефис-О-ноль).

Теперь можно запустить *gdb* для определения проблемы:

```
papaya$ gdb trymh
GNU gdb 20010316
Copyright 2001 Free Software Foundation, Inc.
GDB is free software, covered by the GNU GeneralPublic License, and you are
welcome to change it and/or distribute copies of it under certain conditions.
Type "show copying" to see the conditions.
There is absolutely no warranty for GDB. Type "show warranty" for details.
This GDB was configured as "i386-suse-linux".
(gdb)
```

(GDB является свободно распространяемым программным обеспечением, и вы можете свободно модифицировать и/или передавать его копии при некоторых условиях, для ознакомления с которыми введите «show copying». На GDB нет абсолютно никаких гарантий; подробности можно увидеть, введя «show warranty».)

Теперь *gdb* ждет команду. (Команда *help* выводит сведения об имеющихся командах.) Первое, что мы хотим сделать, – это запустить программу, чтобы наблюдать за ее поведением. Однако если мы немедленно выполним команду *run*, программа будет просто выполняться, пока не закончит работу или не приведет к аварийной ситуации.

Поэтому сначала необходимо установить *точку останова* (*breakpoint*) где-либо в программе. Точка останова – это то место в программе, где *gdb* должен остановиться и позволить нам посмотреть, как выполняется программа. Для простоты установим точку останова на первой строке кода, чтобы программа остановилась сразу после начала выполнения. Команда *list* выводит сразу несколько строк кода (количество можно изменять):

```
(gdb) list
12        main() {
13
14         FloatImage inimage;
15         FloatImage outimage;
16         BinaryImage binimage;
17         int i,j;
18
19         inimage = (FloatImage)imLoadF(IMAGE_FLOAT,stdin);
20         outimage = laplacian_float(inimage);
21
(gdb) break 19
Breakpoint 1 at 0x289c: file trymh.c, line 19.
(gdb)
```

Теперь точка останова установлена в строке 19 текущего исходного файла. В программе можно установить много точек останова. Точки останова могут

быть условными (т. е. срабатывать только при истинности некоторого выражения), безусловными, отложенными, временно отключенными и т. д. Можно установить точку останова на конкретной строке, конкретной функции, группе функций и массой других способов. Можно также при помощи команды *watch* установить *контрольную точку* (*watchpoint*), которая аналогична точке останова, но срабатывает при возникновении некоторого события, причем необязательно на указанной строке исходного кода программы. Подробнее о точках останова и контрольных точках см. далее в этой главе.

Теперь воспользуемся командой *run* для запуска программы. *run* принимает те же аргументы, которые вы передали бы в командной строке *trymh*; они могут включать в свой состав маску имени файла и перенаправление потока ввода/вывода, когда команда передается */bin/sh* для выполнения:

```
(gdb) run < image00.pgm > image00.pfm
Starting program: /amd/dusk/d/mdw/vis/src/trymh < image00.pgm > image00.pfm

Breakpoint 1, main () at trymh.c:19
19          inimage = (FloatImage)imLoadF(IMAGE_FLOAT,stdin);
(gdb)
```

Как и ожидалось, мы немедленно оказываемся в точке останова в первой строке кода. Теперь мы можем перехватить управление.

Наиболее полезными командами пошагового выполнения программы являются *next* и step. Обе команды выполняют следующую строку кода программы, за исключением того, что *step* входит во все вызовы функций в программе, а *next* переходит только к следующей строке кода в той же функции. *next* спокойно выполняет все вызовы функций, на которые натыкается, но не переходит в их код для изучения.

ImLoadF – это функция, которая загружает образ из дискового файла. Мы знаем, что с этой функцией проблем не будет (вам придется поверить, что это так), поэтому хотим перескочить через нее с помощью команды *next*:

```
(gdb) next
20          outimage = laplacian_float(inimage);
(gdb)
```

Здесь мы хотим отследить подозрительно выглядящую функцию *laplacian_float*, поэтому мы выполняем команду *step*:

```
(gdb) step
laplacian_float (fim=0x0) at laplacian.c:21
21           i = 20.0;
(gdb)
```

Выполним команду *list*, чтобы получить представление о том, где мы находимся:

```
(gdb) list
16        FloatImage laplacian_float(FloatImage fim) {
17
18           FloatImage mask;
```

```
19          float i;
20
21          i = 20.0;
22          mask=(FloatImage)imNew(IMAGE_FLOAT,3,3);
23          imRef(mask,0,0) = imRef(mask,2,0) = imRef(mask,0,2) = 1.0;
24          imRef(mask,2,2) = 1.0; imRef(mask,1,0) = imRef(mask,0,1) = i/5;
25          imRef(mask,2,1) = imRef(mask,1,2) = i/5; imRef(mask,1,1) = -i;
(gdb) list
26
27          return convolveFloatWithFloat(fim,mask);
28        }
(gdb)
```

Как видите, повторное выполнение *list* просто выводит код дальше. Поскольку мы не хотим проходить этот код пошагово вручную, и нас не интересует функция *imNew* в строке 22, продолжим выполнение до строки 27. Для этого выполним команду *until*:

```
(gdb) until 27
laplacian_float (fim=0x0) at laplacian.c:27
27          return convolveFloatWithFloat(fim,mask);
(gdb)
```

Прежде чем войти в функцию *convolveFloatWithFloat*, убедимся, что два параметра, `fim` и `mask`, имеют допустимые значения. Команда *print* показывает значение переменной:

```
(gdb) print mask
$1 = (struct {...} *) 0xe838
(gdb) print fim

$2 = (struct {...} *) 0x0
(gdb)
```

`mask` выглядит нормально, но `fim` (образ ввода) является нулевым (null). Очевидно, что *laplacian_float* был передан нулевой указатель вместо действительного образа. Если вы внимательно смотрели, то заметили это при входе в *laplacian_ float*.

Вместо того чтобы глубже залезать в программу (поскольку уже ясно, что возникли проблемы), продолжим выполнение до возврата из текущей функции. Это осуществляет команда *finish:*

```
(gdb) finish
Run till exit from #0  laplacian_float (fim=0x0) at laplacian.c:27
0x28c0 in main () at trymh.c:20
20       outimage = laplacian_float(inimage);
Value returned is $3 = (struct {...} *) 0x0
(gdb)
```

Теперь мы снова в *main*. Чтобы определить источник проблемы, посмотрим значение некоторых переменных:

```
(gdb) list
15        FloatImage outimage;
16        BinaryImage binimage;
17        int i,j;
18
19        inimage = (FloatImage)imLoadF(IMAGE_FLOAT,stdin);
20        outimage = laplacian_float(inimage);
21
22        binimage = marr_hildreth(outimage);
23        if  (binimage == NULL) {
24          fprintf(stderr,"trymh: binimage returned NULL\en");
(gdb) print inimage
$6 = (struct {...} *) 0x0
(gdb)
```

Переменная inimage, содержащая входной образ, возвращаемый *imLoadF*, имеет нулевое значение. В таком случае передача нулевого указателя в процедуры обработки изображения несомненно вызовет вывод дампа памяти. Однако нам известно, что *imLoadF* проверена и правильна, поскольку находится в хорошо протестированной библиотеке. Так в чем же дело?

Оказывается, наша библиотечная функция *imLoadF* возвращает NULL в случае неудачи, например при неверном формате входного файла. Поскольку мы не проверяли значение, возвращаемое *imLoadF*, прежде чем передать его *laplacian_float*, программа приходит в расстройство, когда inimage присваивается значение NULL. Для решения проблемы мы просто вставим код, вызывающий выход с сообщением об ошибке, когда *imLoadF* возвращает нулевой указатель.

Для выхода из *gdb* выполните команду *quit*. Если программа не закончила выполнение, *gdb* сообщит, что программа все еще выполняется:

```
(gdb) quit
The program is running.  Quit anyway (and kill it)? (y or n) y
papaya$
```

В следующих разделах мы рассмотрим некоторые специальные функции, предоставляемые отладчиком в ситуации, с которой мы столкнулись.

Исследование файла дампа памяти

Нравится ли вам, когда программа не только аварийно завершается, но еще оставляет в рабочем каталоге 10-мегабайтный файл дампа памяти, при том что рабочее пространство так необходимо? Не спешите удалять этот файл; он может быть очень полезен. Файл дампа памяти является копией образа памяти процесса в тот момент, когда произошла авария. Вы можете использовать этот файл с *gdb* для изучения состояния вашей программы (например, значений переменных и данных) и определения причины отказа.

Файл дампа памяти записывается на диск операционной системой, когда происходят некоторые виды отказов. Наиболее частой причиной отказа и последующего дампа памяти является нарушение границ памяти, т. е. попыт-

ка чтения или записи по адресам, к которым у вашей программы нет доступа. Например, попытка записи по нулевому адресу может вызвать *ошибку сегментации* (в этом случае сразу ясно, что вы «напортачили»). Ошибки сегментации являются частыми ошибками и возникают при попытке доступа (для записи или чтения) по адресу, не принадлежащему адресному пространству вашего процесса. В число этих адресов входит нулевой адрес, который часто возникает при пропуске инициализации указателя. Ошибка сегментации часто вызывается попыткой обращения к элементу массива, находящемуся за границами объявленного размера массива, обычно при превышении границы на единицу. Эти ошибки могут также быть обусловлены тем, что не была выделена память для некой структуры данных.

Другие ошибки, приводящие к дампам памяти, называются «ошибками шины» и «ошибками операций с плавающей запятой». Ошибки шины происходят при неправильном выравнивании данных и поэтому редко встречаются в архитектуре Intel, не предъявляющей таких строгих условий выравнивания, как другие архитектуры, например SPARC. Исключения, возникающие при выполнении операций с плавающей запятой, свидетельствуют о серьезных проблемах вычислений с плавающей запятой, таких как переполнение, но чаще всего ошибка состоит в делении на ноль.

Однако не все такие ошибки приводят к немедленному краху программы. Например, вы можете каким-то образом залезть в неверную область памяти, но программа продолжит выполнение, не видя разницы между фактическими данными и командами или мусором. Тонкие нарушения границ памяти могут вызвать неустойчивое поведение программ. Один из авторов был свидетелем случайных переходов в программе, но до трассировки ее с помощью *gdb* она казалась нормально работающей. Единственным признаком наличия ошибки в программе было то, что ее вывод, грубо говоря, показывал, что два плюс два не равно четырем. Оказалось, что ошибка состояла в попытке записать лишний символ в отведенный блок памяти. Ошибка в один байт вызвала долгие часы страданий.

Избежать проблем такого рода (даже хорошие программисты совершают такие ошибки!) можно с помощью пакета Valgrind – набора программ управления памятью, заменяющих часто используемые функции *malloc*() и *free*(), а также их аналоги в C++ – операторы *new* и *delete*. Мы обсудим Valgrind в разделе «Использование Valgrind».

Однако если ваша программа производит нарушение границ памяти, она вызовет аварийный останов и дамп памяти. В Linux файлы памяти так и называются – *core*. Файл *core* появляется в текущем рабочем каталоге выполняемого процесса, которым обычно является текущий каталог оболочки, запустившей программу, но иногда программы могут изменить свой рабочий каталог.

Некоторые оболочки предоставляют средства управления записью дампа памяти. Например, в *bash* файлы памяти по умолчанию не пишутся. Для того чтобы активировать запись дампов, следует выполнить команду

```
ulimit -c unlimited
```

вероятно, в файле инициализации *.bashrc*. Можно указать максимальный размер файлов дампа, отличный от `unlimited`, но укороченные файлы могут оказаться бесполезными при отладке приложений.

Для того чтобы файл памяти оказался полезным, программа должна быть откомпилирована с включением отладочной информации, как описано в предыдущем разделе. В большинстве двоичных файлов вашей системы отладочной информации нет, поэтому файл памяти будет иметь ограниченную ценность.

Примером использования *gdb* с файлом дампа памяти послужит еще одна мифическая программа с названием *cross*. Как и *trymh* из предыдущего раздела, *cross* принимает на входе файл с изображением, производит над ним вычисления и выводит другой файл с изображением. Однако при выполнении *cross* мы встречаемся с ошибкой сегментации:

```
papaya$ cross < image30.pfm > image30.pbm
Segmentation fault (core dumped)
papaya$
```

Для того чтобы использовать *gdb* с файлом дампа памяти, нужно указать не только имя файла дампа памяти, но и имя выполняемого модуля, с которым он получен, поскольку файл дампа памяти не содержит всей необходимой для отладки информации:

```
papaya$ gdb cross core
GDB is free software and you are welcome to distribute copies of it
 under certain conditions; type "show copying" to see the conditions.
There is absolutely no warranty for GDB; type "show warranty" for details.
GDB 4.8, Copyright 1993 Free Software Foundation, Inc...
Core was generated by `cross'.
Program terminated with signal 11, Segmentation fault.
#0  0x2494 in crossings (image=0xc7c8) at cross.c:31
31              if ((image[i][j] >= 0) &&
(gdb)
```

gdb сообщает, что этот файл дампа был создан, когда программа закончилась при получении сигнала с номером 11. *Сигнал* – это тип сообщения, которое посылается программе ядром, пользователем или самой программой. Сигналы обычно используются для завершения программы (и, возможно, вызывают дамп памяти). Например, при вводе символа прерывания работающей программе посылается сигнал, который должен ее убить.

В нашем случае ядро послало сигнал 11 процессу работающей программы *cross*, когда *cross* пыталась читать или писать в память, к которой у нее нет доступа. Этот сигнал привел к смерти *cross* и дампу памяти. *gdb* сообщает, что недопустимое обращение к памяти произошло в строке 31 исходного файла *cross.c*:

```
(gdb) list
26          xmax = imGetWidth(image)-1;
27          ymax = imGetHeight(image)-1;
28
```

```
29          for (j=1; j<xmax; j++) {
30            for (i=1; i<ymax; i++) {
31              if ((image[i][j] >= 0) &&
32                  (image[i-1][j-1] < 0) ||
33                  (image[i-1][j] < 0) ||
34                  (image[i-1][j+1] < 0) ||
35                  (image[i][j-1] < 0) ||
(gdb)
```

Здесь обращают на себя внимание несколько вещей. Прежде всего, есть цикл по двум переменным `i` и `j`, вероятно, для производства вычислений с исходным изображением. В строке 31 осуществляется попытка обратиться к данным двумерного массива `image[i][j]`. Когда программа производит дамп при доступе к данным из массива, это обычно указывает на выход одного из индексов за пределы границ. Проверим их:

```
(gdb) print i
$1 = 1
(gdb) print j
$2 = 1194
(gdb) print xmax
$3 = 1551
(gdb) print ymax
$4 = 1194
(gdb)
```

Здесь видна проблема. Программа пыталась сослаться на элемент `image[1][1194]`, однако массив простирается только до `image[1550][1193]` (напомним, что массивы в C индексируются от `0` до *max*-1). Иными словами, мы пытались читать 1195-ю строку изображения, в котором только 1194 строки.

Если взглянуть на строки 29 и 30, то мы видим, в чем проблема: значения `xmax` и `ymax` переставлены местами. Переменная `j` должна изменяться от `1` до `ymax` (поскольку это индекс строк массива), а `i` должна изменяться от `1` до `xmax`. Исправление двух циклов в строках 29 и 30 устраняет проблему.

Допустим, наша программа создает аварийную ситуацию в функции, вызываемой из разных мест программы, и вы хотите определить, откуда была вызвана функция и что привело к аварии. Команда *backtrace* выводит стек вызовов программы в момент отказа. Если вам, как и автору, лень постоянно вводить с клавиатуры *backtrace*, могу к вашему удовольствию сообщить, что можно использовать сокращение *bt*.

Стек вызовов является списком функций, приведших в текущую функцию. Например, если программа начинается с функции *main*, вызывающей функцию *foo*, которая вызывает *bamf*, то стек вызова будет выглядеть так:

```
(gdb) backtrace
#0  0x1384 in bamf () at goop.c:31
#1  0x4280 in foo () at goop.c:48
#2  0x218 in main () at goop.c:116
(gdb)
```

Каждая функция при своем вызове помещает некоторые данные в стек, например, содержимое регистров, аргументы функции, локальные переменные и т. д. Каждой функции отводится определенное пространство в стеке, которое она может использовать. Участок памяти, выделенный в стеке для данной функции, называется *кадром стека* (*stack frame*), а стек вызова является упорядоченным списком кадров стека.

В следующем примере мы видим дамп памяти программы анимации для X. Использование команды *backtrace* дает нам:

```
(gdb) backtrace
#0  0x602b4982 in _end ()
#1  0xbffff934 in _end ()
#2  0x13c6 in stream_drawimage (wgt=0x38330000, sn=4) at stream_display.c:94
#3  0x1497 in stream_refresh_all () at stream_display.c:116
#4  0x49c in control_update_all () at control_init.c:73
#5  0x224 in play_timeout (Cannot access memory at address 0x602b7676.
(gdb)
```

Это список кадров стека для процесса. Функции, вызванной последней, соответствует кадр 0. В нашем случае этой «функцией» является *_end*. Мы видим, что *play_timeout* вызвала *control_update_all*, которая вызвала *stream_refresh_all* и т. д. Каким-то образом программа перешла на *_end*, где и завершилась с ошибкой.

Однако *_end* не является функцией: это просто метка, указывающая на конец сегмента данных процесса. Когда программа переходит на такой адрес, как *_end*, не являющийся реальной функцией, это значит, что по каким-то причинам процесс сбился, и стек вызовов поврежден (на языке хакеров это называется «прыжком в гиперпространство»). На практике сообщение «`Cannot access memory at address 0x602b7676`» является другим свидетельством того, что произошло нечто странное.

Мы видим, однако, что последней «реальной» вызванной функцией была *stream_drawimage*, и можно предположить, что источник проблем лежит в ней. Чтобы изучить состояние *stream_drawimage*, нужно выбрать ее кадр стека (кадр 2) с помощью команды *frame*:

```
(gdb) frame 2
#2  0x13c6 in stream_drawimage (wgt=0x38330000, sn=4) at stream_display.c:94
94        XCopyArea(mydisplay,streams[sn].frames[currentframe],XtWindow(wgt),
(gdb) list
91
92        printf("CopyArea frame %d, sn %d, wid %d\en",currentframe,sn,wgt);
93
94        XCopyArea(mydisplay,streams[sn].frames[currentframe],XtWindow(wgt),
95                  picGC,0,0,streams[sn].width,streams[sn].height,0,0);
(gdb)
```

Не зная больше ничего об этой программе, мы не можем сказать, в чем здесь дело, если только переменная `sn`, используемая в качестве индекса массива `streams`, не вышла за допустимые границы. Данные вывода *frame* показыва-

ют, что *stream_drawimage* была вызвана со значением переменной `sn`, равным 4. (Параметры функций выводятся командой *backtrace*, а также при смене кадра.)

Переместимся в другой кадр, *stream_refresh_all*, чтобы посмотреть, как вызывалась *stream_display*. Для этого мы используем команду *up*, которая выбирает кадр стека, находящийся над текущим кадром:

```
(gdb) up
#3  0x1497 in stream_refresh_all () at stream_display.c:116
116         stream_drawimage(streams[i].drawbox,i);
(gdb) list
113     void stream_refresh_all(void) {
114       int i;
115       for (i=0; i<=numstreams; i++) {
116         stream_drawimage(streams[i].drawbox,i);
117
(gdb) print i
$2 = 4
(gdb) print numstreams
$3 = 4
(gdb)
```

Мы видим, что индексная переменная `i` изменяется от 0 до `numstreams`, и действительно `i` здесь равно 4 – второму параметру *stream_drawimage*. Однако `numstreams` тоже равно 4. Что происходит?

Цикл `for` в строке 115 выглядит странно; он должен выглядеть так:

```
for (i=0; i<numstreams; i++) {
```

Ошибка заключается в использовании оператора сравнения `<=`. Индекс массива `streams` лежит в пределах от 0 до `numstreams-1`, а не от 0 до `numstreams`. Эта простая ошибка смещения на единицу заставила программу обезуметь.

Как видите, использование *gdb* с дампом памяти позволяет просматривать образ аварийно завершившейся программы и находить ошибки. Вы больше не будете удалять эти надоедливые файлы дампов?

Отладка работающей программы

gdb позволяет производить отладку программы, которая уже выполняется, предоставляя возможность прервать ее, изучить, а затем возобновить процесс обычного выполнения. Это весьма сходно с запуском программы из *gdb*, и потребуется изучить лишь несколько новых команд.

Команда *attach* присоединяет *gdb* к работающему процессу. Для того чтобы использовать *attach*, у вас должен быть доступ к исполняемому модулю, соответствующему этому процессу.

Например, если вы запустили программу *pgmseq* с идентификатором процесса 254, то можете запустить *gdb*:

```
papaya$ gdb pgmseq
```

а затем уже в *gdb* выполнить команду:

```
(gdb) attach 254
Attaching program `/home/loomer/mdw/pgmseq/pgmseq', pid 254
__select (nd=4, in=0xbffff96c, out=0xbffff94c, ex=0xbffff92c, tv=0x0)
    at __select.c:22
__select.c:22: No such file or directory.
(gdb)
```

Ошибка `No such file or directory` выдается потому, что *gdb* не может найти исходный файл для *__select*. Это часто бывает с системными вызовами и библиотечными функциями, и не должно вызывать тревогу.

Можно также запустить *gdb* командой:

```
papaya$ gdb pgmseq 254
```

После того как *gdb* присоединится к работающему процессу, он временно прекратит выполнение программы и передаст вам управление, позволив выполнять команды *gdb*. Вы можете установить точки останова или контрольные точки (командами *break* и *watch* соответственно) и воспользоваться командой *continue*, чтобы продолжить выполнение до точки останова.

Команда *detach* отсоединяет *gdb* от текущего процесса. При необходимости можно подключиться к другому работающему процессу. Найдя ошибку, можно отсоединиться (*detach*) от текущего процесса, сделать изменения в исходном файле, перекомпилировать программу и использовать команду *file* для загрузки в *gdb* нового исполняемого модуля. Затем можно запустить новую версию программы и использовать команду *attach* для ее отладки. И все это – не покидая *gdb*!

Фактически *gdb* позволяет одновременно отлаживать три программы: одну, выполняемую непосредственно в *gdb*, вторую, трассируемую с помощью файла дампа памяти, и третью, работающую в качестве независимого процесса. Команда *target* позволяет выбрать программу, которую вы хотите отлаживать.

Изменение и изучение данных

Для изучения значений переменных в вашей программе можно использовать команды *print*, *x* и *ptype*. Команда *print* чаще всего используется для изучения данных. В качестве аргумента она принимает выражение на исходном языке (обычно C или C++) и возвращает его значение, например:

```
(gdb) print mydisplay
$10 = (struct _XDisplay *) 0x9c800
(gdb)
```

Эта команда выводит значение переменной `mydisplay` и ее тип. Поскольку данная переменная является указателем, ее содержимое можно изучить путем разадресации, как в C:

```
(gdb) print *mydisplay
$11 = {ext_data = 0x0, free_funcs = 0x99c20, fd = 5, lock = 0,
  proto_major_version = 11, proto_minor_version = 0,
  vendor = 0x9dff0 "XFree86", resource_base = 41943040,
  ...
  error_vec = 0x0, cms = {defaultCCCs = 0xa3d80 "",clientCmaps = 0x991a0 "`",
    perVisualIntensityMaps = 0x0}, conn_checker = 0, im_filters = 0x0}
(gdb)
```

mydisplay — это обширная структура, используемая программами для X. Для удобства чтения мы сократили вывод.

print может выводить значения почти любых выражений, в том числе вызовов функций C, которые она выполняет на лету в контексте работающей программы:

```
(gdb) print getpid()
$11 = 138
(gdb)
```

Конечно, так можно вызывать не все функции, а только те, которые скомпонованы с выполняемой программой. Если попытаться вызвать функцию, не скомпонованную с программой, *gdb* пожалуется, что такого символа нет в текущем контексте.

В качестве аргументов *print* могут использоваться более сложные выражения, включая присваивание значений переменным, например:

```
(gdb) print mydisplay->vendor = "Linux"
$19 = 0x9de70 "Linux"
(gdb)
```

присваивает элементу vendor структуры mydisplay значение "Linux" вместо "XFree86" (бесполезная модификация, интересная лишь в качестве примера). Таким способом можно интерактивно изменять данные в работающей программе для того, чтобы исправить ошибки или проверить необычные ситуации.

Обратите внимание, что после каждой команды *print* выведенное значение присваивается одному из регистров *gdb*, являющихся внутренними переменными *gdb*, используемыми для удобства. Например, чтобы вспомнить значение mydisplay в предыдущем примере, нужно лишь вывести значение $10:

```
(gdb) print $10
$21 = (struct _XDisplay *) 0x9c800
(gdb)
```

В команде *print* можно использовать такие выражения, как преобразование типа; годится почти все.

Команда *ptype* дает подробные (и часто многословные) сведения о типе переменной или определении структуры (struct) или типа (typedef). Для того чтобы получить полное определение структуры struct _Xdisplay, используемой переменной mydisplay, выполним команду:

```
(gdb) ptype mydisplay
type = struct _XDisplay {
    struct _XExtData *ext_data;
    struct _XFreeFuncs *free_funcs;
    int fd;
    int lock;
    int proto_major_version;
    ...
    struct _XIMFilter *im_filters;
} *
(gdb)
```

Если вы хотите более фундаментально изучить память, не ограничиваясь определенными типами, можете выполнить команду *x*. *x* принимает в качестве аргумента адрес памяти. Если передать ей переменную, то в качестве адреса используется *значение* этой переменной.

В качестве необязательного аргумента можно передать *x* количество и тип данных. Количество – это число выводимых объектов данного типа. Например, `x/100x 0x4200` выводит 100 байт данных, представленных в шестнадцатеричном формате, начиная с адреса 0x4200. Описание различных форматов вывода можно получить командой *help x*.

Для того чтобы выяснить значение `mydisplay->vendor`, можно выполнить команду:

```
(gdb) x mydisplay->vendor
0x9de70 <_end+35376>:   76 'L'
(gdb) x/6c mydisplay->vendor
0x9de70 <_end+35376>:   76 'L'  105 'i' 110 'n' 117 'u' 120 'x' 0 '_'
(gdb) x/s mydisplay->vendor
0x9de70 <_end+35376>:    "Linux"
(gdb)
```

Первое поле каждой строки показывает абсолютный адрес данных. Во втором поле адрес представлен как некоторый символ (в данном случае `_end`) плюс смещение в байтах. В остальных полях дается фактическое содержимое памяти по этому адресу: сначала как десятичное число, а затем как символ ASCII. Как уже говорилось, можно потребовать от *x* вывода данных в других форматах.

Получение информации

При помощи команды *info* можно получить данные о состоянии отлаживаемой программы. В *info* есть много подкоманд; чтобы вывести их список, выполните команду *help info*. Например, *info program* выводит статус выполнения программы:

```
(gdb) info program
Using the running image of child process 138.
Program stopped at 0x9e.
It stopped at breakpoint 1.
(gdb)
```

Другой полезной командой является *info locals*, которая выводит имена и значения всех локальных переменных в текущей функции:

```
(gdb) info locals
inimage = (struct {...} *) 0x2000
outimage = (struct {...} *) 0x8000
(gdb)
```

Это довольно беглое описание переменных. Команды *print* или *x* описывают их подробнее.

Аналогично *info variables* выводит список всех известных переменных в программе, определяемых исходным файлом. Обратите внимание, что многие выводимые переменные относятся к источникам, находящимся вне вашей фактической программы, например, в библиотечном коде. Значения этих переменных не выводятся, поскольку список составляется более или менее непосредственно из таблицы символов выполняемого модуля. Фактически *gdb* доступны только локальные переменные в текущем кадре стека и глобальные (статические) переменные. Команда *info address* дает вам сведения о том, где хранится некоторая переменная, например:

```
(gdb) info address inimage
Symbol "inimage" is a local variable at frame offset -20.
(gdb)
```

Значение `frame offset` говорит о том, что переменная `inimage` хранится на 20 байт ниже вершины кадра стека.

При помощи команды *info frame* можно получить данные о текущем кадре стека, например:

```
(gdb) info frame
Stack level 0, frame at 0xbffffaa8:
 eip = 0x9e in main (main.c:44); saved eip 0x34
 source language c.
 Arglist at 0xbffffaa8, args: argc=1, argv=0xbffffabc
 Locals at 0xbffffaa8, Previous frame's sp is 0x0

 Saved registers:
  ebx at 0xbffffaa0, ebp at 0xbffffaa8, esi at 0xbffffaa4, eip at\e
0xbffffaac
(gdb)
```

Такого рода данные полезны при отладке на уровне языка ассемблера при помощи команд *disass*, *nexti* и *stepi* (см. раздел «Отладка на уровне команд»).

Разные функции

Мы лишь поверхностно коснулись того, что может делать *gdb*. Это удивительная и очень мощная программа. Мы познакомили вас только с наиболее часто используемыми командами. В этом разделе будут рассмотрены другие функции *gdb*.

Если вы хотите больше узнать о *gdb*, рекомендуем почитать страницу руководства по *gdb* и руководство Free Software Foundation. Руководство имеется также и в виде Info-файла. (Файлы Info можно читать в Emacs или с помощью программы *info*; см. раздел «Учебное руководство и оперативная подсказка» главы 9.)

Точки останова и контрольные точки

Как мы и обещали, приведем еще примеры использования точек останова и контрольных точек. Точки останова устанавливаются командой *break;* аналогично, контрольные точки устанавливаются командой *watch*. Единственное различие между теми и другими состоит в том, что точки останова вызывают прерывание выполнения в определенном месте программы, например, на некоторой строке кода, а контрольные точки срабатывают, когда некоторое выражение становится истинным, независимо от места в программе, где это происходит. Несмотря на всю их мощь, контрольные точки могут быть крайне неэффективными: при каждом изменении состояния программы пересчитываются все контрольные точки.

Когда срабатывает точка останова или контрольная точка, *gdb* приостанавливает выполнение программы и передает вам управление. Точки останова и контрольные точки позволяют выполнять программу (используя команды *run* и *continue*) и останавливаться только в определенных ситуациях, избавляя от необходимости ручного прохода программы с помощью многочисленных команд *next* и *step*.

Существует много способов установки точки останова в программе. Можно указать номер строки, например в команде *break 20*. Или задать конкретную функцию, например *break stream_unload*. Можно также задать номер строки в другом исходном файле, например *break foo.c:38*. Полный синтаксис можно получить по команде *help break*.

Точки останова могут быть условными. Это означает, что точка останова срабатывает только при истинности некоторого выражения. Например, команда:

```
break 184 if (status == 0)
```

устанавливает условную точку останова в строке 184 текущего исходного файла, которая срабатывает только тогда, когда переменная `status` равна нулю. Переменная `status` должна быть либо глобальной, либо локальной в текущем кадре стека. Выражение может быть любым допустимым в языке исходного кода выражением, которое понятно *gdb*, подобно выражениям, используемым командой *print*. Изменить условие точки останова можно с помощью команды *condition*.

Команда *info break* выводит список всех точек останова и контрольных точек вместе с их статусом. Это позволяет удалять или отключать точки останова, используя команды *clear*, *delete* или *disable*. Отключенная точка останова просто бездействует до тех пор, пока не будет реактивирована командой *enable*. Напротив, при удалении точка останова удаляется из списка навсе-

гда. Можно также потребовать, чтобы точка останова срабатывала однократно, после чего отключалась.

Для установки контрольной точки выполните команду *watch*:

```
watch (numticks < 1024 && incoming != clear)
```

Контрольная точка может иметь условие, являющееся любым допустимым выражением исходного кода, таким же, как для точек останова.

Отладка на уровне инструкций

С помощью *gdb* можно производить отладку на уровне инструкций процессора, что позволяет самым пристальным образом рассматривать содержимое вашей программы. Однако для понимания того, что вы видите, требуется не только знание архитектуры процессора и языка ассемблера, но и некоторое понимание того, как операционная система организует адресное пространство процесса. Например, полезно понимать соглашения, используемые при создании кадров стека, вызове функций, передаче параметров и получении возвращаемых значений и т. д. Любая книга по программированию в защищенном режиме 80386/80486 подробно ознакомит вас с этими вопросами. Но имейте в виду, что программирование в защищенном режиме этого процессора совершенно отличается от программирования в реальном режиме (используемом в мире MS-DOS). Удостоверьтесь, что вы читаете именно о программировании в *защищенном режиме* 386, иначе будете периодически путаться.

Главными командами *gdb*, используемыми при отладке на уровне инструкций, являются *nexti*, *stepi* и *disass*. *nexti* эквивалентна *next*, но осуществляет переход к следующей инструкции, а не к следующей строке исходного кода. Аналогично, *stepi* является аналогом *step* на уровне инструкций.

Команда *disass* дизассемблирует переданный ей диапазон адресов. Этот диапазон может быть задан точным адресом или как имя функции. Например, для дизассемблирования функции *play_timeout* выполните команду:

```
(gdb) disass play_timeout
Dump of assembler code for function play_timeout:
to 0x2ac:
0x21c <play_timeout>:           pushl  %ebp
0x21d <play_timeout+1>:         movl   %esp,%ebp
0x21f <play_timeout+3>:         call   0x494 <control_update_all>
0x224 <play_timeout+8>:         movl   0x952f4,%eax
0x229 <play_timeout+13>:        decl   %eax
0x22a <play_timeout+14>:        cmpl   %eax,0x9530c
0x230 <play_timeout+20>:        jne    0x24c <play_timeout+48>
0x232 <play_timeout+22>:        jmp    0x29c <play_timeout+128>
0x234 <play_timeout+24>:        nop
0x235 <play_timeout+25>:        nop
...

0x2a8 <play_timeout+140>:       addb   %al,(%eax)
0x2aa <play_timeout+142>:       addb   %al,(%eax)
(gdb)
```

Это эквивалентно использованию команды *disass 0x21c* (где `0x21c` – точный адрес начала *play_timeout*).

Можно передать *disass* необязательный второй аргумент, который будет использован как адрес, по которому дизассемблирование прекратится. Команда *disass 0x21c 0x232* выведет только первые семь строк листинга ассемблера предыдущего примера (инструкция, начинающаяся с адреса `0x232`, выведена не будет).

При частом использовании *nexti* и *stepi* может пригодиться команда:

```
display/i $pc
```

Она вызывает вывод текущей команды после каждой выполненной *nexti* или *stepi*. *display* задает переменные, которые нужно просмотреть, или команды, которые нужно выполнить после каждой команды пошагового выполнения. `$pc` является внутренним регистром *gdb*, соответствующим программному счетчику процессора, указывающему на очередную инструкцию.

Использование Emacs с gdb

Emacs (описанный в разделе «Редактор Emacs» главы 9) предоставляет режим отладки, который позволяет запустить *gdb* или другой отладчик в интегрированной среде трассировки программ, предоставляемой Emacs. Эта так называемая библиотека «Grand Unified Debugger» (библиотека большого унифицированного отладчика) – очень мощная и позволяет редактировать и отлаживать программы целиком внутри Emacs.

Для запуска *gdb* под Emacs выполните команду Emacs `M-x gdb` и передайте ей в качестве аргумента имя исполняемого модуля. Для *gdb* будет создан буфер, что аналогично отдельному использованию *gdb*. После этого при желании вы можете использовать команду *core-file* для загрузки файла дампа памяти или команду *attach* для подключения к выполняемому процессу.

При переходе к новому кадру стека (например, при первой остановке в точке останова) *gdb* открывает отдельное окно, в котором выводится исходный код, соответствующий текущему кадру стека. Этот буфер может использоваться для редактирования исходного кода, так же как это обычно делается в Emacs, при этом текущая строка исходного текста отмечена стрелкой (`=>`). Это позволяет видеть исходный текст в одном окне и выполнять команды *gdb* в другом.

В окне отладки можно использовать несколько специальных последовательностей символов. Однако они достаточно длинные, и их использование может показаться не более удобным, чем непосредственный ввод команд *gdb*. В число наиболее часто употребляемых команд входят:

`C-x C-a C-s`

Эквивалент команды *gdb step*, соответствующим образом обновляющий окно с исходным кодом.

`C-x C-a C-i`

Эквивалент команды *stepi*.

`C-x C-a C-n`

Эквивалент команды *next*.

`C-x C-a C-r`

Эквивалент команды *continue*.

`C-x C-a <`

Эквивалент команды *up*.

`C-x C-a >`

Эквивалент команды *down*.

Если вы вводите команды в обычном стиле, то можно использовать комбинацию `M-p` для перемещения назад по буферу введенных ранее команд и `M-n` для перемещения вперед. Можно также перемещаться по буферу с использованием команд Emacs для поиска, перемещения курсора и т. д. В целом, использование *gdb* в Emacs более удобно, чем из оболочки.

Кроме того, можно редактировать исходный текст в буфере исходного кода *gdb*. Префиксная стрелка при сохранении текста записываться не будет.

Emacs очень легко настраивается, и многие расширения для этого интерфейса с *gdb* вы можете написать сами. В Emacs можно определить комбинации клавиш для часто используемых в *gdb* команд или изменить характер поведения окна исходного кода. (Например, можно каким-то образом выделить все точки останова или определить клавиши для отключения или удаления точек останова.)

Средства программирования

Наряду с языками и компиляторами существует изобилие программных средств, в том числе библиотеки, конструкторы интерфейсов, отладчики и прочие средства поддержки процесса программирования. В этом разделе мы поговорим о некоторых наиболее интересных «бантиках» среди этих средств, чтобы вы знали об имеющихся инструментах.

Отладчики

Существует несколько интерактивных отладчиков для Linux. Стандартом de facto является *gdb*, о котором мы только что подробно рассказали.

Кроме *gdb* есть несколько других отладчиков со схожими характеристиками. *xxgdb* является версией *gdb* с интерфейсом для X Window System, сходным с тем, который предоставляет отладчик *xdbx* в других Unix-системах. В окне отладчика *xxgdb* есть несколько панелей. Одна из них выглядит как обычный текстовый интерфейс *gdb*, позволяющий вручную вводить команды для взаимодействия с системой. В другой панели автоматически отображается текущий исходный файл с маркером, указывающим на текущую строку. Вы можете использовать панель исходного кода для установки и выбора точек останова, просмотра источника и т. д., передавая команды прямо

gdb. В окне *xxgdb* есть также ряд кнопок, обеспечивающих быстрый доступ к часто используемым командам, таким как *step*, *next* и другие. Кнопки позволяют использовать мышь в дополнение к клавиатуре для отладки вашей программы в простом интерфейсе X.

Стоит упомянуть о двух других графических интерфейсах для *gdb*. DDD (Data Display Debugger) имеет те же функции, что и *xxgdb*, но более приятный интерфейс пользователя Motif. Кроме того, он может графически отображать структуры и классы, что особенно полезно при исследовании структур данных в неизвестных программах. *kdbg* относится к проекту KDE: помимо предоставления тех же функций, что и в *xxgdb*, он полностью интегрирован в рабочий стол KDE.

Средства профилирования

Существует несколько утилит, позволяющих контролировать и измерять производительность программ. Эти средства помогают выявлять узкие места в коде, т. е. те места, которым недостает производительности. Они позволяют также получить схему структуры вызовов в вашей программе с указанием того, какие функции вызываются, откуда и как часто. (Все, что вы хотели знать о вашей программе, но боялись спросить.)

gprof является утилитой профилирования, предоставляющей детальный листинг статистики выполнения вашей программы, включая частоту вызова всех функций и мест, откуда они вызывались, суммарный объем времени, затраченный на выполнение каждой функции, и т. д.

Для того чтобы использовать *gprof* вместе с программой, нужно компилировать ее в *gcc* с ключом *-pg*. Благодаря этому в объектный файл добавляются данные для профилирования, и он компонуется со стандартными библиотеками, в которых есть данные для профилирования.

Скомпилировав программу с ключом *-pg*, просто запустите ее. Если она нормально завершит работу, то в рабочий каталог программы будет записан файл *gmon.out*. В нем содержатся данные по профилированию этого прогона, на основе которых *gprof* выведет таблицу статистики.

Для примера возьмем программу с именем *getstat*, собирающую статистические данные о графическом файле. Сначала мы скомпилируем ее с ключом *-pg*, а затем запустим:

```
papaya$ getstat image11.pgm > stats.dat
papaya$ ls -l gmon.out
-rw-------   1 mdw      mdw         54448 Feb  5 17:00 gmon.out
papaya$
```

Данные профилирования действительно были записаны в *gmon.out*.

Для изучения данных профилирования запустим *gprof*, передав ей имя исполняемого модуля и файла профилирования *gmon.out*:

```
papaya$ gprof getstat gmon.out
```

Если имя файла профилирования не задано, *gprof* предполагает имя *gmon.out*. Она также предполагает, что имя выполняемого модуля – *a.out*, если и оно не указано.

Вывод *gprof* довольно многословен, поэтому вы можете перенаправить его в файл или программу постраничного вывода. Вывод состоит из двух частей. Первая часть является «плоским профилем» с одной строкой для каждой функции, в которой указан процент времени работы, потраченного на эту функцию, время (в секундах) ее выполнения, количество вызовов и другие данные, например:

```
Each sample counts as 0.01 seconds.
  %   cumulative   self              self     total
 time   seconds   seconds    calls  ms/call  ms/call  name
 45.11     27.49    27.49       41   670.51   903.13  GetComponent
 16.25     37.40     9.91                             mcount
 10.72     43.93     6.54  1811863     0.00     0.00  Push
 10.33     50.23     6.30  1811863     0.00     0.00  Pop
  5.87     53.81     3.58       40    89.50   247.06  stackstats
  4.92     56.81     3.00  1811863     0.00     0.00  TrimNeighbors
```

Если какие-либо поля оказываются пустыми, значит, *gprof* не смогла определить дополнительные данные относительно этой функции. Обычно это связано с тем, что некоторые части кода компилировались без ключа *-pg*. Например, при вызове функций из нестандартных библиотек, компилировавшихся без ключа *-pg*, *gprof* сможет собрать об этих функциях не много данных. В приведенном примере функция *mcount*, вероятно, компилировалась без включения профилирования.

Как можно видеть, 45,11% общего времени прогона было потрачено на функцию *GetComponent*, т. е. в сумме 27,49 секунд. Но связано ли это с неэффективностью самой *GetComponent* или с тем, что *GetComponent* вызывает много других медлительных функций? Во время выполнения многократно вызывались функции *Push* и *Pop*. Не они ли виновники?

Здесь нам может помочь вторая часть отчета *gprof*. В ней приведен подробный «граф вызовов», описывающий вызовы одних функций другими и частоту этих вызовов. Например:

```
index % time    self  children    called     name
                                                 <spontaneous>
[1]     92.7    0.00   47.30                 start [1]
                0.01   47.29       1/1           main [2]
                0.00    0.00       1/2           on_exit [53]
                0.00    0.00       1/1           exit [172]
```

Первая колонка графа вызовов является индексом – уникальным номером, присвоенным каждой функции, позволяющим отыскивать ее в графе. Первая функция *start* неявно вызывается при запуске программы. *start* потребовала 92,7% общего времени работы (47,30 секунды), включая ее потомков, но для самой себя потребовала очень мало времени. Это связано с тем, что *start* является родительской для всех остальных функций программы,

включая *main*. Именно поэтому *start* вместе со своими потомками заняла такой процент времени.

Граф вызовов обычно отображает как потомков, так и родителей каждой функции графа. Мы видим, что *start* вызывала функции *main*, *on_exit* и *exit* (перечисленные под строкой для *start*). Однако у нее нет родителей (обычно перечисляемых над строкой *start*). Вместо этого мы видим зловещее слово <spontaneous>. Это означает, что *gprof* не смогла определить родителя *start*. Скорее всего, это вызвано тем, что *start* из самой программы не вызывалась, а была запущена операционной системой.

Перескакивая вниз к *GetComponent*, т. е. к подозреваемой функции, мы видим следующее:

```
index % time    self  children    called     name
                0.67    0.23       1/41          GetFirstComponent [12]
               26.82    9.30      40/41          GetNextComponent [5]
[4]     72.6   27.49    9.54      41         GetComponent [4]
                6.54    0.00 1811863/1811863     Push [7]
                3.00    0.00 1811863/1811863     TrimNeighbors [9]
                0.00    0.00       1/1           InitStack [54]
```

Родительскими функциями для *GetComponent* были *GetFirstComponent* и *GetNextComponent*, а дочерними – *Push*, *TrimNeighbors* и *InitStack*. Мы видим, что *GetComponent* вызывалась 41 раз (один раз из *GetFirstComponent* и 40 раз из *GetNextComponent*). В выдаче *gprof* содержатся примечания, описывающие отчет более подробно.

Самой *GetComponent* требуется для выполнения 27,49 секунд, и лишь 9,54 секунды потрачено на выполнение дочерних функций *GetComponent* (включая многочисленные вызовы *Push* и *TrimNeighbors*!). По всей видимости, *GetComponent* и, возможно, ее родительская функция *GetNextComponent* нуждаются в доработке; часто вызываемая функция *Push* не является единственной причиной проблемы.

gprof отслеживает также рекурсивные вызовы и «циклы» вызываемых функций и указывает время, потраченное на каждый вызов. Конечно, для эффективного использования *gprof* требуется, чтобы весь профилируемый код компилировался с параметром *-pg*. Требуется также знать программу, которую вы пытаетесь профилировать. *gprof* может только сообщить о том, что делается, а оптимизация неэффективного кода остается за программистом.

Следует отметить, что использование *gprof* может не дать осмысленных результатов в случае профилирования программы, содержащей малое количество вызовов и очень быстро выполняющейся. Единицы измерения временных затрат обычно достаточно грубы (возможно, сотые доли секунды), и если многие функции вашей программы выполняются быстрее, *gprof* не сможет выявить различие во времени их выполнения, округляя до сотых долей секунды. Для того чтобы получить правильную информацию профилирования в таком случае, можно выполнить программу в необычных условиях,

например, дав ей на обработку необычно большой объем данных, как в предыдущем примере.

Если *gprof* превосходит ваши потребности, то существует программа *calls*, выводящая дерево вызовов функций в вашем исходном коде C. Она может оказаться полезной при создании указателя всех вызываемых функций или при создании иерархического отчета высокого уровня о структуре программы.

Использование *calls* отличается простотой: вы сообщаете имена исходных файлов, для которых должна быть составлена карта, и выводится дерево вызовов функций. Например:

```
papaya$ calls scan.c
    1   level1 [scan.c]
    2           getid [scan.c]
    3                   getc
    4                   eatwhite [scan.c]
    5                           getc
    6                           ungetc
    7                   strcmp
    8           eatwhite [see line 4]
    9           balance [scan.c]
   10                   eatwhite [see line 4]
```

По умолчанию *calls* перечисляет только по одному экземпляру каждой вызываемой функции на каждом уровне дерева (поэтому если *printf* вызывается в данной функции пять раз, она будет перечислена лишь однажды). Ключ *-a* позволяет вывести все случаи вызовов. *calls* имеет и другие параметры, перечень которых можно получить при помощи *calls -h*.

Использование strace

strace выводит на экран системные вызовы, осуществляемые работающей программой.[1] Эта утилита может оказаться исключительно полезной при мониторинге работы программы в реальном времени, хотя при этом потребуется некоторое знание программирования на уровне системных вызовов. Например, когда в программе используется библиотечная функция *printf*, *strace* выводит данные только о выполнении лежащего в ее основе системного вызова *write*. Кроме того, вывод *strace* может оказаться очень пространным: в программе может выполняться большое количество системных вызовов, о которых не подозревает программист. Однако *strace* полезна для быстрого определения причины отказа программы или иного необычного поведения.

Возьмем программу «Hello, World!», приведенную в начале главы. Запуск *strace* с выполняемым модулем *hello* дает следующий результат:

---

[1] Полезным может также оказаться пакет *ltrace*. Это трассировщик библиотечных вызовов, отслеживающий все обращения к библиотекам, а не только к ядру. Он входит в состав ряда дистрибутивов. Пользователи остальных дистрибутивов могут загрузить последнюю версию ее исходного кода с *ftp://ftp.debian.org/debian/dists/unstable/main/source/utils/*.

```
papaya$ strace hello
execve("./hello", ["hello"], [/* 49 vars */]) = 0
mmap(0, 4096, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x40007000
mprotect(0x40000000, 20881, PROT_READ|PROT_WRITE|PROT_EXEC) = 0
mprotect(0x8048000, 4922, PROT_READ|PROT_WRITE|PROT_EXEC) = 0
stat("/etc/ld.so.cache", {st_mode=S_IFREG|0644, st_size=18612, ...}) = 0
open("/etc/ld.so.cache", O_RDONLY)      = 3
mmap(0, 18612, PROT_READ, MAP_SHARED, 3, 0) = 0x40008000
close(3)                                = 0
stat("/etc/ld.so.preload", 0xbffff52c)  = -1 ENOENT (No such file or directory)
open("/usr/local/KDE/lib/libc.so.5", O_RDONLY) = -1 ENOENT (No\
 such file or directory)
open("/usr/local/qt/lib/libc.so.5", O_RDONLY) = -1 ENOENT (No\
 such file or directory)
open("/lib/libc.so.5", O_RDONLY)        = 3
read(3, "ELF\1\1\1\0\0\0\0\0\0\0\0\0\3"..., 4096) = 4096
mmap(0, 770048, PROT_NONE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = \
0x4000d000
mmap(0x4000d000, 538959, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_\
FIXED, 3, 0) = 0x4000d000
mmap(0x40091000, 21564, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_\
FIXED, 3, 0x83000) = 0x40091000
mmap(0x40097000, 204584, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_\
FIXED|MAP_ANONYMOUS, -1, 0) = 0x40097000
close(3)                                = 0
mprotect(0x4000d000, 538959, PROT_READ|PROT_WRITE|PROT_EXEC) = 0
munmap(0x40008000, 18612)               = 0
mprotect(0x8048000, 4922, PROT_READ|PROT_EXEC) = 0
mprotect(0x4000d000, 538959, PROT_READ|PROT_EXEC) = 0
mprotect(0x40000000, 20881, PROT_READ|PROT_EXEC) = 0
personality(PER_LINUX)                  = 0
geteuid()                               = 501
getuid()                                = 501
getgid()                                = 100
getegid()                               = 100
fstat(1, {st_mode=S_IFCHR|0666, st_rdev=makedev(3, 10), ...}) = 0
mmap(0, 4096, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS,\
 -1, 0) = 0x40008000
ioctl(1, TCGETS, {B9600 opost isig icanon echo ...}) = 0
write(1, "Hello World!\en", 13Hello World!
)          = 13
_exit(0)                                = ?
papaya$
```

Это значительно больше, чем можно было ожидать от простой программы. Рассмотрим этот вывод и кратко объясним, что происходит.

Первый вызов *execve* запускает саму программу. Все вызовы *mmap*, *mprotect* и *munmap* исходят от системы управления памятью ядра и не очень нам здесь интересны. В трех последующих вызовах *open* загрузчик ищет библиотеку C и находит ее с третьей попытки. После этого считывается заголовок

библиотеки, и библиотека отображается в память. После нескольких дополнительных операций управления памятью и вызовов *getuid*, *geteuid*, *getgid* и *getegid*, которые извлекают привилегии процесса, следует вызов *ioctl*. *ioctl* является результатом библиотечного вызова *tcgetattr*, который программа использует для извлечения атрибутов терминала, прежде чем пытаться выводить на терминал. И наконец, вызов *write* выводит на терминал наше дружелюбное сообщение, а *exit* завершает программу.

Вызовы *munmap* (для отмены отображения в память части файла) и *brk* (для выделения памяти в куче) настраивают образ памяти работающего процесса. Обращение к *ioctl* происходит в результате вызова из библиотеки *tcgetattr* с целью получить атрибуты терминала перед началом вывода в него. Наконец, *write* выводит наше дружественное сообщение на терминал, а *exit* завершает выполнение программы.

strace посылает свою выдачу на стандартное устройство вывода ошибок, поэтому вы можете переназначить ее в файл отдельно от фактической выдачи программы (обычно посылаемой на стандартное устройство вывода). Как видите, *strace* сообщает не только имена системных вызовов, но также их параметры (в виде хорошо известных имен констант, а не просто чисел, если это возможно) и возвращаемые значения.

Использование библиотеки Valgrind

Библиотека Valgrind заменяет многочисленные процедуры распределения памяти, такие как *malloc*, *realloc* и *free*, используемые программами C, но поддерживает также программы C++. Она обеспечивает более интеллектуальные процедуры выделения памяти и код, позволяющий определять незаконный доступ к памяти, а также частые ошибки, такие как попытки повторного освобождения блока памяти. Valgrind выводит подробные сообщения об ошибках, когда ваша программа делает опасные попытки доступа к памяти, помогая перехватить ошибки сегментации до того, как они произойдут. Она может также выявлять утечки памяти, например, места в коде, где *malloc* выделяет новую память без последующего освобождения функцией *free* после ее использования.

Valgrind не просто заменяет *malloc* и другие функции. Она вставляет в программу код, проверяющий все операции чтения и записи в память. Эта библиотека обладает большой надежностью, и поэтому выполняется существенно медленнее, чем обычные функции *malloc*. Valgrind предназначена для использования во время разработки и тестирования программ. После устранения всех потенциальных ошибок обращения к памяти вы можете запустить свою программу без Valgrind.

Рассмотрим программу, которая выделяет некоторый объем памяти и пытается делать с ней нехорошие вещи:

```
#include <malloc.h>
int main() {
  char *thememory, ch;
```

```
    thememory=(char *)malloc(10*sizeof(char));

  ch=thememory[1];     /* Попытка чтения неинициализированной памяти */
  thememory[12]=' ';   /* Попытка записи после блока */
  ch=thememory[-2];    /* Попытка чтения перед блоком */
}
```

Чтобы найти эти ошибки, скомпилируем эту программу для отладки и запустим, введя предварительно в командной строке *valgrind*:

```
owl$ gcc -g -o nasty nasty.c
owl$ valgrind nasty
==18037== valgrind-20020319, a memory error detector for x86 GNU/Linux.
==18037== Copyright (C) 2000-2002, and GNU GPL'd, by Julian Seward.
==18037== For more details, rerun with: -v
==18037==
==18037== Invalid write of size 1
==18037==    at 0x8048487: main (nasty.c:8)
==18037==    by 0x402D67EE: __libc_start_main (in /lib/libc.so.6)
==18037==    by 0x8048381: __libc_start_main@@GLIBC_2.0 (in /home/kalle/tmp/nasty)
==18037==    by <bogus frame pointer> ???
==18037==    Address 0x41B2A030 is 2 bytes after a block of size 10 alloc'd
==18037==    at 0x40065CFB: malloc (vg_clientmalloc.c:618)
==18037==    by 0x8048470: main (nasty.c:5)
==18037==    by 0x402D67EE: __libc_start_main (in /lib/libc.so.6)
==18037==    by 0x8048381: __libc_start_main@@GLIBC_2.0 (in /home/kalle/tmp/nasty)
==18037==
==18037== Invalid read of size 1
==18037==    at 0x804848D: main (nasty.c:9)
==18037==    by 0x402D67EE: __libc_start_main (in /lib/libc.so.6)
==18037==    by 0x8048381: __libc_start_main@@GLIBC_2.0 (in /home/kalle/tmp/nasty)
==18037==    by <bogus frame pointer> ???
==18037==    Address 0x41B2A022 is 2 bytes before a block of size 10 alloc'd
==18037==    at 0x40065CFB: malloc (vg_clientmalloc.c:618)
==18037==    by 0x8048470: main (nasty.c:5)
==18037==    by 0x402D67EE: __libc_start_main (in /lib/libc.so.6)
==18037==    by 0x8048381: __libc_start_main@@GLIBC_2.0 (in /home/kalle/tmp/nasty)
==18037==
==18037== ERROR SUMMARY: 2 errors from 2 contexts (suppressed: 0 from 0)
==18037== malloc/free: in use at exit: 10 bytes in 1 blocks.
==18037== malloc/free: 1 allocs, 0 frees, 10 bytes allocated.
==18037== For a detailed leak analysis,  rerun with: --leak-check=yes
==18037== For counts of detected errors, rerun with: -v
```

Число в начале каждой строки указывает ID процесса. Если процесс порождает другие процессы, то и они будут выполняться под управлением Valgrind.

В случае любого нарушения работы с памятью Valgrind сообщает об ошибке и выводит данные о том, что произошло. Действительные сообщения Valgrind об ошибках включают в себя данные о том, в каком месте выполняется программа и где был размещен блок памяти. При желании вы можете полу-

чить от Valgrind более подробные сведения, а если дополнительно использовать отладчик, например *gdb*, можно легко локализовать ошибки.

Вы можете задать вопрос, почему операция чтения в строке 7, обращающаяся к инициализированному участку памяти, не заставила Valgrind выдать сообщение об ошибке. Это связано с тем, что Valgrind не выводит сообщений об ошибках, когда передается инициализированная память, но продолжает следить за ней. Как только вы воспользуетесь значением (например, при передаче функции операционной системы, или обработав его), вы получите ожидаемое сообщение об ошибке.

Valgrind предоставляет также сборщик и детектор мусора, который можно вызывать из вашей программы. Детектор мусора сообщает об утечках памяти, т. е. о тех местах, где *malloc* выделила блок памяти, но до выхода не была выполнена соответствующая функция *free*. Сборщик мусора просматривает «кучу» и очищает ее от результатов этих утечек. Вот образец вывода:

```
owl$ valgrind --leak-check=yes --show-reachable=yes nasty
...
==18081== ERROR SUMMARY: 2 errors from 2 contexts (suppressed: 0 from 0)
==18081== malloc/free: in use at exit: 10 bytes in 1 blocks.
==18081== malloc/free: 1 allocs, 0 frees, 10 bytes allocated.
==18081== For counts of detected errors, rerun with: -v
==18081== searching for pointers to 1 not-freed blocks.
==18081== checked 4029376 bytes.
==18081==
==18081== definitely lost: 0 bytes in 0 blocks.
==18081== possibly lost:   0 bytes in 0 blocks.
==18081== still reachable: 10 bytes in 1 blocks.
==18081==
==18081== 10 bytes in 1 blocks are still reachable in loss record 1 of 1
==18081==    at 0x40065CFB: malloc (vg_clientmalloc.c:618)
==18081==    by 0x8048470: main (nasty.c:5)
==18081==    by 0x402D67EE: __libc_start_main (in /lib/libc.so.6)
==18081==    by 0x8048381: __libc_start_main@@GLIBC_2.0 (in /home/kalle/tmp/nasty)
==18081==
==18081== LEAK SUMMARY:
==18081==    possibly lost:   0 bytes in 0 blocks.
==18081==    definitely lost: 0 bytes in 0 blocks.
==18081==    still reachable: 10 bytes in 1 blocks.
==18081==
```

Средства создания интерфейса

Ряд приложений и библиотек позволяет легко генерировать интерфейс пользователя для приложений, работающих в X Window System. Если вы не хотите утруждать себя сложностями программного интерфейса X, то альтернативой может быть использование одного из этих простых конструкторов интерфейса. Существуют и средства создания текстовых интерфейсов для программ, которые не требуют X.

В классической модели программирования X сделана попытка достичь максимальной общности, при этом вводится лишь самый минимум ограничений и допущений на интерфейс. Эта общность позволяет программистам создавать собственные интерфейсы с чистого листа, поскольку основные библиотеки X не делают предварительных допущений относительно интерфейса. X Toolkit Intrinsics (Xt) предоставляет рудиментарный набор графических элементов интерфейса (простые кнопки, полосы прокрутки и т. п.), а также общий интерфейс для создания собственных элементов в случае необходимости. К сожалению, это может потребовать большого объема работы от программистов, которые предпочли бы использовать набор готовых интерфейсных процедур. Имеется ряд наборов элементов Xt и программных библиотек для Linux, облегчающих программирование интерфейса пользователя.

Кроме того, ряд поставщиков предоставляет библиотеку и набор элементов Motif с дешевой лицензией для отдельных пользователей. Существуют также библиотека и интерфейсные элементы XView, что также является альтернативой использованию Xt для создания интерфейсов в X. XView и Motif являются двумя наборами библиотек программирования X, которыми в некоторых отношениях пользоваться проще, чем X Toolkit Intrinsics. Существует много приложений, использующих Motif и XView, например XVhelp (система генерации интерактивной гипертекстовой справки для программ). Двоичные файлы, статически связанные с Motif, могут свободно распространяться и использоваться теми, кто не имеет Motif в своей собственности.

Хочется предостеречь тех, у кого возникло желание делать разработки с помощью XView или Motif. XView был некоторое время коммерческим продуктом Sun Microsystems, однако сейчас он разработчиками больше не поддерживается. Кроме того, хотя некоторым нравится вид программ, написанных с использованием XView, они выглядят очень нестандартно. Напротив, Motif по-прежнему активно разрабатывается (хотя и довольно медленно), но с ним связаны свои проблемы. Во-первых, программирование на Motif не очень привлекательно. Связано это с тем, что оно отличается сложностью, возможностью совершения ошибок и утомительно, т. к. Motif API создавался без учета современных принципов проектирования GUI API. Во-вторых, созданные с помощью Motif программы выполняются очень медленно. По этой причине предпочтительнее может оказаться один из следующих вариантов:

Xaw3D

Модифицированная версия стандартного набора элементов Athena, предоставляющая 3-мерный Motif-образный вид.

Qt

Инструментальный набор C++ GUI, разработанный норвежской фирмой Troll Tech.

GTK

Инструментальный набор C GUI, созданный изначально для программы обработки изображений GIMP.

Многие жалуются, что элементы набора Athena имеют слишком простой вид. Xaw3D полностью совместим со стандартным набором Athena, и может даже заменить библиотеки Athena на вашей машине, придав современный вид программам, использующим элементы Athena. Xaw3D предоставляет также несколько элементов, отсутствующих в Athena, например, элемент структуры с TEX-образным интерфейсом для задания положений дочерних элементов.

Qt является отличным пакетом для разработки GUI на C++, использующим изобретательный механизм для связи действий пользователя с программным кодом, имеет очень быстрое ядро графики и обширный, но легкий в использовании API. Qt многими считается заменой Motif в качестве стандарта программирования GUI de facto, поскольку он лежит в основе рабочей среды KDE (см. раздел «K Desktop Environment» главы 11), которая занимает ведущее положение в качестве рабочего стола на современных системах Linux.

Qt является коммерческим продуктом, но он также выпускается под лицензией GPL, что означает возможность бесплатного использования при создании с его помощью бесплатного программного обеспечения для Unix (и, следовательно, Linux), которое тоже выпускается под лицензией GPL. Кроме того, есть коммерческие версии Qt для Windows и Mac OS X, что позволяет разрабатывать одновременно для Linux, Windows и Mac OS X и создавать приложение для другой платформы путем простой перекомпиляции. В результате предоставляется возможность вести разработку на своей любимой операционной системе Linux и все же быть в состоянии нацелиться на более обширный рынок Windows! Один из авторов, Калле, использует Qt для написания как бесплатного программного обеспечения (для упоминавшейся среды KDE), так и коммерческого (кросс-платформенные продукты, разрабатываемые для Linux, Windows и Mac OS X). Qt разрабатывается очень активно; более подробные сведения см. в «Programming with Qt» Калле Далхаймера (Kalle Dalheimer), изданной O'Reilly. Другим недавним замечательным дополнением к Qt является возможность ее работы на встроенных системах, не требующих X-сервера. А какую операционную систему будет она поддерживать на встроенных системах, если не Embedded Linux! Можно ожидать появления в скором будущем многих портативных устройств с графическими экранами, на которых работает Embedded Linux с Qt/Embedded.

Вместе с Qt поставляется конструктор GUI под названием Qt Designer, значительно облегчающий создание приложений GUI. Он включен и в GPL-версию Qt, поэтому, загрузив Qt (или установив со своего дистрибутива), вы получите и этот Designer.

Для тех, кто не любит программировать на C++, хорошим выбором может стать GTK (либо вы можете просто использовать привязки Python для Qt!). Программы GTK обычно демонстрируют такое же хорошее время реакции, как Qt, но сам набор инструментов не столь полон. Особенно ощущается недостаток в документации. Однако проектам, основанным на C, GTK представляет хорошую альтернативу, если вы не собираетесь перекомпилировать свой код для Windows. Недавно был разработан и перенос для Windows, но он пока далек от совершенства.

Многие программисты считают, что создание интерфейса пользователя, даже при наличии полного набора элементов и функций на C, требует больших затрат и может быть весьма затруднительно. Это вопрос соотношения гибкости и простоты программирования: чем проще создать интерфейс, тем меньшим контролем над ним обладает программист. Многие программисты считают, что готовые элементы удовлетворяют их потребностям, поэтому потеря гибкости не становится проблемой.

Одной из проблем при создании интерфейса и программировании для X является то, что трудно обобщить наиболее часто используемые элементы интерфейса пользователя в простой программной модели. Например, многие программы используют такие объекты, как кнопки, диалоговые окна, выпадающие меню и т. д., но почти все программы используют эти элементы в разном контексте. При упрощении задачи создания графического интерфейса средства генерации стараются предположить, что вам потребуется. Например, достаточно просто указать, что при нажатии кнопки должна выполняться некоторая процедура в программе. Но как быть, если вы хотите, чтобы при нажатии кнопки осуществлялось некоторое особое поведение, не предусмотренное интерфейсом программирования? Например, если вы хотите, чтобы нажатие левой и правой кнопками мыши приводило к разным результатам? Если система создания интерфейса не допускает такой степени общности, то от нее мало пользы программистам, которым требуется мощный настраиваемый интерфейс.

Комплект Tcl/Tk, состоящий из языка сценариев Tcl и инструментального набора графики Tk, приобрел некоторую популярность отчасти потому, что он прост в использовании и обеспечивает большую гибкость. Поскольку функции Tcl и Tk можно вызывать из интерпретируемых «сценариев», а также из программ на C, нетрудно связать функции интерфейса, предоставляемые этим языком и инструментальным набором, с функционированием программы. Использование Tcl и Tk в целом требует меньших затрат, чем изучение непосредственного программирования с использованием Xlib и Xt (а также множества наборов элементов). Однако следует заметить, что чем больше проект, тем более вероятно, что вам потребуется использовать такой язык, как C++, более пригодный для крупномасштабных разработок. По ряду причин крупные проекты становятся очень громоздкими, когда привлекается Tcl: использование интерпретируемого языка снижает скорость выполнения программы, архитектуру Tcl/Tk трудно масштабировать на большие проекты, отсутствуют важные средства обеспечения надежности, такие как проверка типов на этапах компиляции и компоновки. Проблему масштабируемости можно решить благодаря использованию пространств имен (способ предотвращения конфликтов имен в различных частях программы) и объектно-ориентированного расширения под названием [incr Tcl].

Tcl и Tk позволяют создавать интерфейс для X, укомплектованный окнами, кнопками, меню, полосами прокрутки и прочим, вокруг уже существующих программ. Доступ к интерфейсу может осуществляться из сценария Tcl (как описано в разделе «Другие языки» главы 13) или из программы на C.

Если вам требуется красивый текстовый интерфейс для программы, то имеется несколько возможностей на выбор. Библиотека GNU *getline* является набором функций, обеспечивающим развитое редактирование командной строки, вывод приглашений, буфер команд и другие функции, используемые во многих программах. Например, как *bash*, так и *gdb* используют библиотеку *getline* для ввода команд пользователя. *getline* предоставляет функции редактирования командной строки в стиле Emacs и *vi*, которые есть в *bash* и аналогичных программах. (Использование редактирования командной строки в *bash* описано в разделе «Сокращенный ввод с клавиатуры» главы 4.)

Еще одна возможность – написать ряд функций для взаимодействия вашей программы с Emacs. Примером является интерфейс *gdb* Emacs, в котором создается несколько окон, устанавливаются специальные последовательности клавиш и т. д. Этот интерфейс обсуждался выше в разделе «Использование Emacs с *gdb*». (Для его реализации не потребовалось изменений в коде *gdb*: посмотрите в библиотеке Emacs в файле *gdb.el* заметки по поводу того, как это было сделано.) Emacs позволяет запустить подпрограмму из текстового буфера и предоставляет много подпрограмм для синтаксического анализа и обработки текста в этом буфере. Например, в интерфейсе Emacs для *gdb* выдаваемый *gdb* листинг исходного текста перехватывается Emacs и превращается в команду, отображающую текущую строку кода в другом окне. Процедуры, написанные на Emacs LISP, обрабатывают выдачу *gdb* и осуществляют некоторые действия, основываясь на ней.

Преимущество использования Emacs для взаимодействия с текстовыми программами состоит в том, что Emacs сам по себе является мощным и настраиваемым интерфейсом пользователя. Пользователь может легко переопределять клавиши и команды, приспосабливая их к своим потребностям – вам не нужно самому обеспечивать эти функции настройки. Если текстовый интерфейс программы достаточно прост, чтобы взаимодействовать с Emacs, настройку выполнить несложно. Кроме того, многие пользователи предпочитают делать в Emacs буквально все – от чтения электронной почты и телеконференций до компиляции и отладки программ. Если вы снабдите свою программу интерфейсом Emacs, это облегчит ее использование людьми подобного склада ума. Это также позволит вашей программе взаимодействовать с другими программами, работающими под Emacs. Например, можно легко вырезать и переносить текст между различными текстовыми буферами Emacs. При желании можно писать целые программы на Emacs LISP.

Средства контроля ревизий – RCS

На Linux перенесена Revision Control System (RCS) – система контроля ревизий, представляющая собой комплект программ, позволяющих поддерживать «библиотеку» файлов, записывать историю ревизий, осуществлять блокировку файлов с исходным кодом (в случае работы нескольких людей над одним и тем же проектом) и автоматически отслеживать номера версий исходных файлов. RCS обычно используется с файлами исходного кода про-

граммы, но благодаря своей общности может применяться к любому типу файлов, для которых должны поддерживаться различные версии.

Зачем нужен контроль версий? Во многих больших проектах требуется тот или иной контроль ревизий для отслеживания многих мелких изменений в системе. Например, попытка разработки программы, состоящей из тысячи исходных файлов, бригадой из нескольких десятков программистов практически невозможна без использования какого-либо средства, аналогичного RCS. RCS в каждый данный момент разрешает модифицировать любой исходный файл только одному человеку, а все изменения вносятся в систему вместе с их описанием в журнале.

RCS основывается на понятии *файла RCS*, выполняющего функции «библиотеки», в которую файлы «сдаются» (check in) и из которой они «выдаются» (check out). Допустим, у вас есть исходный файл *importrtf.c*, который вы хотите поддерживать с помощью RCS. По умолчанию файл получает в RCS имя *importrtf.c,v*. Файл RCS содержит историю ревизий файла, что позволяет извлекать любую предшествующую версию файла, внесенную в него. Для каждой версии вы можете сделать запись в журнале.

При сдаче файлов в RCS ревизии добавляются к файлу RCS, а исходный файл по умолчанию удаляется. Чтобы получить доступ к первоначальному файлу, нужно выписать его из файла RCS. Если вы редактируете файл, то вряд ли хотите, чтобы одновременно с вами кто-либо еще редактировал этот же файл. Поэтому RCS блокирует файл на то время, когда он выдан для редактирования. Заблокированный файл может быть модифицирован только тем пользователем, который его выписал (это реализуется через права доступа к файлу). Сделав изменения в исходном файле, вы сдаете его обратно, что позволяет всем работающим над проектом, в свою очередь, получить его для дальнейшей работы. Получение файла без его блокировки не влечет таких ограничений. Обычно файлы выписываются с блокировкой только для редактирования, а просто для чтения – без блокировки (например, для использования исходного файла при компиляции программы).

RCS автоматически отслеживает все предыдущие версии в файле RCS и присваивает возрастающие номера версий каждой новой сдаваемой ревизии. При сдаче файла можно присвоить собственный номер версии, что позволяет начать новую «ветвь» ревизий. При этом несколько проектов могут развиваться, исходя из разных версий одного и того же файла. Это удобный способ использовать в проектах общий код и при этом быть уверенным, что изменения в одной ветви не затрагивают другие.

Приведем пример. Возьмем наш исходный файл *importrtf.c*, содержащий нашу дружелюбную программу:

```
#include <stdio.h>

int main(void) {
  printf("Hello, world!");
}
```

Сначала нужно сдать его в систему RCS с помощью команды *ci*:

```
papaya$ ci importrtf.c
importrtf.c,v  <--  importrtf.c
enter description, terminated with single '.' or end of file:
NOTE: This is NOT the log message!
>> Hello world source code
>> .
initial revision: 1.1
done
papaya$
```

Создан файл RCS *importrtf.c,v* и удален файл *importrtf.c*.

Для того чтобы снова начать работу с файлом исходного кода, нужно выполнить команду *co* («check out»). Например:

```
papaya$ co -l importrtf.c
importrtf.c,v  -->  importrtf.c
revision 1.1 (locked)
done
papaya$
```

выдает файл *importrtf.c* (из *importrtf.c,v*) и блокирует его. Блокировка файла позволяет отредактировать его и сдать обратно. Если файл нужен вам только для чтения (например, чтобы запустить *make*), можно опустить ключ *-l* в команде *co*, и файл останется незаблокированным. Нельзя сдать файл, который не заблокирован (если только это не новый файл, как в нашем примере).

Теперь можно сделать изменения в исходном коде и сдать файл обратно. В большинстве случаев желательно постоянно иметь файл «на руках», а команду *ci* использовать для записи последней ревизии в файл RCS и выдачи номера версии. Для этого можно использовать с *ci* ключ *-l*:

```
papaya$ ci -l importrtf.c
importrtf.c,v  <--  importrtf.c
new revision: 1.2; previous revision: 1.1
enter log message, terminated with single '.' or end of file:
>> Changed printf call
>> .
done
papaya$
```

В результате после сдачи файл автоматически выдается с блокировкой. Это полезный способ контроля ревизий, даже если вы один работаете над проектом.

Если вы часто используете RCS, то нагромождение в вашем каталоге всех этих файлов *importrtf.c,v* может выглядеть неопрятно. Создайте подкаталог *RCS* в вашем каталоге проекта, и команды *ci* и *co* будут помещать файлы RCS в него, не мешая остальному исходному коду.

Кроме того, RCS отслеживает все прежние ревизии вашего файла. Например, если произведенные в программе изменения приводят ее к аварии и вы хотите отменить ваши изменения, вернувшись к прежней версии, можно

указать конкретный номер версии, которую должна выдать команда *co*. Например:

```
papaya$ co -l1.1 importrtf.c
importrtf.c,v  -->  importrtf.c
revision 1.1 (locked)
writable importrtf.c exists; remove it? [ny](n): y
done
papaya$
```

выдает версию 1.1 файла *importrtf.c*. Чтобы просмотреть историю ревизий некоторого файла, можно воспользоваться программой *rlog*, которая выводит записи журнала ревизий (введенные в команде *ci*), а также другие данные, такие как дата, имя пользователя, сдавшего ревизию, и др.

При выдаче файла с исходным кодом RCS автоматически обновляет встроенные в него «строки ключевых слов». Например, если в исходном файле есть строка:

```
/* $Header: /work/linux/running4/RCS/ch14,v 1.1
        2002/09/20 20:51:50 sierra Exp sierra $ */
```

(*На самом деле это одна строка; она разбита, чтобы уместиться на странице.*)

то *co* заменит ее информативной строкой с датой ревизии, номером версии и т. д., например:

```
/* $Header: /work/linux/hitch/programming/tools/RCS/rcs.tex 1.2
        1994/12/04 15:19:31 mdw Exp mdw $ */
```

Есть и другие ключевые слова, такие как `$Author: jhawks $`, `$Date: 2002/09/24 15:30:14 $` и `$Log: ch14,v $`.

Многие программисты помещают в исходные тексты статические строки для определения версии программы после ее компиляции. Например, в каждом файле с исходным текстом вашей программы можно поместить строку:

```
static char rcsid[] = "\@(#)$Header: /work/linux/running4/RCS/ch14,v 1.3
        2002/09/24 15:30:14 andrews Exp ssherman $";
```

co заменяет ключевое слово `$Header: /work/linux/running4/RCS/ch14,v 1.3 2002/09/24 15:30:14 andrews Exp ssherman $` строкой приведенного вида. Эта статическая строка сохраняется в исполняемом файле, а команда *what* выводит такие строки для указанного двоичного файла. Например, скомпилировав *importrtf.c* в исполняемый файл *importrtf*, можно выполнить команду:

```
papaya$ what importrtf
importrtf:
        $Header: /work/linux/hitch/programming/tools/RCS/rcs.tex 1.2
              1994/12/04 15:19:31 mdw Exp mdw $
papaya$
```

what находит в файле строки, начинающиеся с символов `@(#)`, и выводит их. Если у вас есть программа, скомпилированная из большого числа исходных файлов и библиотек, и вы не знаете, насколько новы составляющие ее ком-

поненты, то можно с помощью *what* вывести строки версий для всех исходных файлов, использовавшихся при компиляции в двоичный файл.

Пакет RCS составляют несколько программ, в том числе *rcs*, используемая для сопровождения файлов RCS. Помимо прочего *rcs* может предоставлять другим пользователям право получения исходных файлов из файла RCS. Для получения дополнительной информации см. страницы руководства по *ci*(1), *co*(1) и *rcs*(1).

Средства контроля ревизий – CVS

CVS (Concurrent Version System) более сложна, чем RCS, и поэтому, вероятно, чрезмерна для проектов, разрабатываемых одним человеком. Но если над проектом работает более одного-двух программистов или исходный код размещен в нескольких каталогах, CVS предпочтительнее. Для хранения изменений CVS использует формат файла RCS, но применяет собственную структуру управления.

По умолчанию CVS работает с целыми деревьями каталогов, т. е. каждая команда CVS воздействует на текущий каталог и на все вложенные в него подкаталоги. Этот рекурсивный обход можно отключить с помощью параметра командной строки, либо можно указать отдельный файл, с которым должна работать команда.

CVS формализовала понятие «песочницы», используемое многими компаниями, разрабатывающими программное обеспечение. Эта концепция предполагает наличие так называемого *хранилища*, или *репозитория* (*repository*), содержащего «официальные» исходные коды, относительно которых известно, что они компилируются и работают (по крайней мере, частично). Ни одному разработчику не позволяется непосредственно редактировать файлы в этом хранилище. Вместо этого он выписывает файлы в локальное дерево каталогов, или *песочницу* (*sandbox*), где может редактировать исходные коды, сколько душе угодно, – вносить изменения, добавлять и удалять файлы, и заниматься всем тем, чем обычно занимаются разработчики. Удостоверившись, что сделанные им изменения компилируются и работают, разработчик передает их обратно в хранилище и делает доступными другим разработчикам.

Когда вы получаете файлы в локальное дерево каталогов, они все имеют разрешение для записи. Все необходимые изменения вы производите в файлах личного рабочего пространства. Закончив тестирование и чувствуя уверенность в своей работе, достаточную для того, чтобы поделиться своей работой с остальными разработчиками, вы записываете измененные файлы в хранилище с помощью команды CVS *commit*. CVS проверяет, не сделаны ли каким-либо другим разработчиком изменения с тех пор, как вы получили свое дерево каталогов. В этом случае CVS не разрешает вам внести свои изменения, а предлагает принять изменения, относящиеся к вашему локальному дереву и сделанные другими разработчиками. Во время этой операции обновления CVS применяет сложный алгоритм в попытке согласовать («merge») ваши изменения с изменениями, сделанными другими разработ-

чиками. Не всегда это удается сделать автоматически. В этом случае CVS сообщает о наличии конфликтов и просит разрешить их. Спорный файл размечается специальными символами, для того чтобы вы могли видеть место конфликта и решить, какую из версий следует использовать. При этом CVS допускает наличие конфликтов только в локальных каталогах разработчиков, а в хранилище всегда лежит непротиворечивая версия.

Создание хранилища CVS

Если вы участвуете в большом проекте, то, вероятно, кто-то уже установил все необходимое для работы CVS. Но если вы сами администрируете проект и хотите повозиться с CVS на своей локальной машине, вам придется самостоятельно создать хранилище.

Сначала укажите в переменной окружения `CVSROOT` каталог, в котором вы хотите разместить хранилище CVS. CVS может поддерживать в хранилище любое количество проектов и обеспечивать их независимое существование, поэтому вам нужно лишь однажды выбрать каталог для хранения всех проектов, поддерживаемых CVS, и вам не придется изменять его при переходе от одного проекта к другому. Вместо переменной `CVSROOT` можно во всех командах CVS использовать ключ `-d`, но поскольку вводить каждый раз имя каталога утомительно, мы предполагаем, что вы установили переменную `CVSROOT`.

Если определен каталог для хранилища, можно создать само хранилище следующей командой (в предположении, что CVS установлена на вашей машине):

```
$tigger cvs init
```

Существует несколько способов создания дерева проекта в хранилище CVS. Если у вас уже есть дерево каталогов, но оно еще не управляется RCS, можно просто импортировать его в хранилище командой:

```
$tigger cvs import directory manufacturer tag
```

где *directory* является именем каталога самого верхнего уровня проекта, *manufacturer* — именем автора кода (вы можете использовать здесь любое имя), а *tag* является так называемым тегом релиза, который может быть выбран произвольно. Вот пример:

```
$tigger cvs import dataimport acmeinc initial
... много выдачи ...
```

Если вы хотите начать новый проект, можно просто создать дерево каталогов командами *mkdir*, а затем импортировать это пустое дерево, как показано в предыдущем примере.

Если вы хотите импортировать проект, уже управляемый RCS, возникают некоторые дополнительные сложности, поскольку использовать *cvs import* нельзя. В этом случае вам придется создать необходимые каталоги прямо в хранилище, а затем скопировать в эти каталоги все файлы RCS (все файлы, оканчивающиеся на *,v*). Не используйте здесь подкаталоги RCS!

В каждом хранилище есть файл с именем *CVSROOT/modules*, содержащий названия проектов в хранилище. Для добавления нового модуля можно редактировать файл *modules*. Этот файл можно получать, редактировать и сдавать, как любой другой файл, в хранилище. Таким образом, чтобы добавить к списку ваш модуль, выполните следующие команды:

```
$tigger cvs checkout CVSROOT/modules
$tigger cd CVSROOT
$tigger emacs modules
... или любой другой редактор; что вводить, сказано ниже ...
$tigger cvs commit modules
$tigger cd ..
$tigger cvs release -d CVSROOT
```

Если вы не делаете чего-либо необычного, то формат файла *modules* очень прост: каждая строка начинается с имени модуля, за которым следуют пробел или знак табуляции и путь внутри хранилища. В файле *modules* можно делать много других вещей, о которых рассказано в документации по CVS, особенно в страницах Info или на *http://www.loria.fr/~molli/cvs-index.html*. Есть также короткая, но очень содержательная книга «CVS Pocket Reference» Грегора Пурди (Gregor N. Purdy), изданная O'Reilly.

Работа с CVS

В этом разделе мы предполагаем, что вы или ваш системный администратор установили модуль с именем `dataimport`. Вы можете получить локальное дерево каталогов этого модуля командой:

```
$tigger cvs checkout dataimport
```

Если для проекта, над которым вы собираетесь работать, не определен модуль, вам нужно узнать путь внутри хранилища. Например, может потребоваться что-то вроде:

```
$tigger cvs checkout clients/acmeinc/dataimport
```

Какую бы версию команды *checkout* вы ни использовали, CVS создаст каталог с именем *dataimport* в вашем текущем рабочем каталоге и выпишет в него из хранилища все файлы и каталоги, относящиеся к этому модулю. Все файлы доступны для записи, и вы можете немедленно начать их редактирование.

Произведя какие-то изменения, вы можете записать измененные файлы обратно в хранилище, выполнив единственную команду:

```
$tigger cvs commit
```

Конечно, можно записывать в хранилище и отдельные файлы:

```
$tigger cvs commit importrtf.c
```

Что бы вы ни делали, CVS так же, как это делает RCS, попросит ввести комментарии, которые должны быть введены вместе с вашими изменениями. Но CVS предоставляет дополнительные удобства по сравнению с RCS. Вме-

сто простого приглашения RCS вы получите полноэкранный редактор для ввода. Используемый редактор можно выбрать, установив переменную окружения CVSEDITOR. Если она не установлена, CVS ищет редактор в EDITOR, а если он не определен и там, CVS вызывает *vi*. Если вы выписали проект целиком, CVS будет использовать введенный вами комментарий для каждого каталога, в котором были изменения, но каждый раз будет заново запускать редактор, чтобы вы могли сделать изменения в комментарии.

Как уже говорилось, нет необходимости корректно устанавливать *CVSROOT* для получения файлов, поскольку при записи дерева в хранилище CVS создала каталог *CVS* в каждом рабочем каталоге. Этот каталог содержит все данные, необходимые CVS для работы, в том числе путь, по которому находится хранилище.

Вполне может оказаться, что пока вы работали над своими файлами, ваш коллега записал в хранилище некоторые из файлов, над которыми вы в настоящее время трудитесь. В этом случае CVS не позволит вам сохранить ваши файлы, а предложит сначала обновить ваше локальное дерево. Сделайте это командой:

```
$tigger cvs update
M importrtf.c
A exportrtf.c
? importrtf
U importword.c
```

(Можно также указать конкретный файл.) Внимательно изучите вывод этой команды: CVS выводит имена всех обрабатываемых файлов, каждому из которых предшествует один символ. Этот символ сообщает о том, что произошло во время операции обновления. Наиболее важные символы приведены в табл. 14.1.

Таблица 14.1. Ключевые символы для файлов в CVS

Символ	Объяснение
P	Файл обновлен. Символ P выводится, если файл уже добавлен в хранилище или он был изменен, но сами вы не сделали в этом файле изменений
U	Вы изменили этот файл, но никто другой этого не делал
M	Вы изменили этот файл, но кто-то еще записал более новую версию. Все изменения удалось благополучно объединить
C	Вы изменили этот файл, и кто-то еще записал более новую версию. При попытке слияния возникли конфликты
?	CVS не имеет сведений об этом файле, т. е. он не находится под контролем CVS

Наиболее важным из этих символов является C. CVS не смогла слить изменения и нуждается в вашей помощи. Загрузите файлы в редактор и поищите строку <<<<<<<. За этой строкой снова показывается имя файла, за которым

следует ваша версия, заканчивающаяся строкой, содержащей `=======`. Затем следует версия кода из хранилища, которая заканчивается строкой, содержащей `>>>>>>>`. Теперь вы должны установить, возможно, связавшись со своим коллегой, какая версия лучше, и можно ли объединить две версии вручную. Измените файл нужным образом и удалите метки CVS `<<<<<<<`, `=======` и `>>>>>>>`. Сохраните файл и еще раз выполните *commit*.

Если вы решили временно прекратить работу над проектом, проверьте, все ли изменения внесены в хранилище. Для этого перейдите в каталог над корневым каталогом вашего проекта и выполните команду:

```
$tigger cvs release dataimport
```

CVS проверит, внесли ли вы все изменения обратно в хранилище, и при необходимости предупредит вас. Полезно использовать параметр *-d*, который удаляет локальное дерево, если все изменения внесены в хранилище.

Работа с CVS через Интернет

CVS очень полезна для рассредоточенных бригад разработчиков,[1] поскольку предоставляет возможность доступа к хранилищам на других машинах.

В настоящее время есть как бесплатные (например, SourceForge), так и коммерческие сервисы, предоставляющие сервер CVS, благодаря чему вы можете начать сетевой программный проект, не заводя собственного круглосуточно работающего сервера.

Если вы можете с помощью *rsh* подключиться к машине, на которой находится хранилище, то у вас есть возможность использовать для доступа к хранилищу удаленную систему CVS. Для того чтобы выписать из хранилища модуль, выполните команду:

```
cvs -d :ext:user@domain.com:/path/to/repository checkout dataimport
```

Если вы не можете или не хотите использовать *rsh* по соображениям безопасности, можно использовать защищенную оболочку *ssh*. Вы можете сообщить CVS, что хотите использовать *ssh*, установив значение переменной окружения `CVS_RSH` в *ssh*.

Аутентификацию и доступ к хранилищу можно также осуществить по протоколу клиент/сервер. Для удаленного доступа необходимо, чтобы на машине с хранилищем работал сервер CVS. О том, как это сделать, читайте в документации по CVS. Если сервер установлен, подключиться к нему можно следующим образом:

```
cvs -d :pserver:user@domain.com:path/to/repository login
CVS password:
```

[1] Использование CVS расцвело вместе с числом проектов бесплатного программного обеспечения, разрабатываемого с помощью Интернета людьми с разных континентов.

Как видите, сервер CVS запросит пароль CVS, назначенный вам администратором сервера CVS. Эта процедура регистрации требуется только один раз для каждого хранилища. При получении модуля нужно указать машину, на которой находится сервер, ваше имя пользователя на этой машине и удаленный путь к хранилищу. Как и для локальных хранилищ, эти данные сохраняются в вашем локальном дереве. Поскольку пароль сохраняется в файле *.cvspass* в вашем исходном каталоге с минимальной криптографической защитой, возникает потенциальная угроза безопасности. В документации CVS об этом сказано более подробно.

При использовании CVS в Интернете и загрузке больших модулей может понадобиться использование параметра *-z* с дополнительным числовым значением, определяющим степень сжатия передаваемых данных в диапазоне от 1 до 9.

Обновление файлов с помощью patch

Допустим, вы пытаетесь сопровождать программу, которая периодически обновляется, но содержит столько исходных файлов, что невозможно при каждом обновлении выпускать полный дистрибутив исходного кода. Лучше всего производить инкрементальное обновление исходных файлов с помощью программы *patch*, созданной Ларри Уоллом, автором Perl.

Программа *patch* производит в файле контекстно-зависимые изменения с целью обновления его до следующей версии. Поэтому при изменении программы вы просто выпускаете файл-заплатку («патч») для исходного файла, который пользователь применяет с помощью *patch* и получает свежую версию. Например, Линус Торвальдс обычно выпускает новые версии ядра и в виде заплаток, и как полные дистрибутивы.

Замечательной особенностью *patch* является то, что она применяет обновления в контексте. Это означает, что если вы самостоятельно изменили исходный код, но все же хотите использовать изменения, предлагаемые файлом-заплаткой, *patch* обычно в состоянии правильно определить то место в исходном файле, которое нужно изменить. Благодаря этому ваши версии оригинальных файлов с исходным кодом не должны точно совпадать с теми, для которых создавалась заплатка.

Для создания файла-заплатки используется программа *diff*, которая вычисляет «контекстные различия» между двумя файлами. Возьмем, например, исходный код нашей «Hello World!»:

```
/* hello.c version 1.0 by Norbert Ebersol */
#include <stdio.h>

int main() {
  printf("Hello, World!");
  exit(0);
}
```

Допустим, мы хотим обновить этот код, чтобы он стал таким:

```
/* hello.c version 2.0 */
/* (c)1994 Norbert Ebersol */
#include <stdio.h>

int main() {
  printf("Hello, Mother Earth!\n");
  return 0;
}
```

Если вы хотите создать файл-заплатку для обновления исходного текста *hello.c* до новейшей версии, выполните *diff* с параметром *-c*:

```
papaya$ diff -c hello.c.old hello.c > hello.patch
```

В результате будет создан файл-заплатка *hello.patch*, в котором описано, как преобразовать исходный текст *hello.c* (сохраненный в нашем случае в файле *hello.c.old*) до новой версии. Вы можете поставлять этот файл-заплатку всем, у кого есть оригинальная версия «Hello, World!», и они могут делать обновление с помощью *patch*.

Использование *patch* отличается простотой: в большинстве случаев вы просто запускаете ее с файлом-заплаткой на входе:[1]

```
papaya$ patch < hello.patch
Hmm...  Looks like a new-style context diff to me...
The text leading up to this was:
--------------------------
|*** hello.c.old        Sun Feb  6 15:30:52 1994
|--- hello.c    Sun Feb  6 15:32:21 1994
--------------------------
Patching file hello.c using Plan A...
Hunk #1 succeeded at 1.
done
papaya$
```

patch выдает предупреждение, если ей кажется, что заплатка уже применялась. Если попытаться применить заплатку еще раз, *patch* спрашивает, не предполагалось ли задать ключ *-R*, который отменяет установленную заплатку. Это хороший способ отмены заплаток, которые вы не собирались применять. *patch* также сохраняет исходные версии обновляемых файлов в виде резервных копий, обычно именуемых как *filename~* (имя файла с приписанной тильдой).

Во многих случаях требуется обновить не отдельный файл, а файлы из целого дерева каталогов. *patch* позволяет обновлять группы файлов, исходя из одного файла-заплатки, полученного *diff*. Допустим, что у вас есть два дерева каталогов – *hello.old* и *hello*, в которых содержатся исходные файлы для

[1] Вывод, который здесь показан, получен для последнего релиза, выпущенного Ларри Уоллом, – версии 2.1. Если у вас более новая версия *patch*, то для получения такого же вывода нужно использовать флаг *--verbose*.

старой и новой версий программы, соответственно. Чтобы сделать файл заплатки для целого дерева, используйте ключ *-r*:

```
papaya$ diff -cr hello.old hello > hello.patch
```

Теперь посмотрим, что делается в системе, программное обеспечение в которой нуждается в обновлении. Предполагая, что исходный текст содержится в каталоге *hello*, можно применить заплатку такой командой:

```
papaya$ patch -p0 < hello.patch
```

Ключ *-p0* сообщает *patch* о необходимости сохранить пути обновляемых файлов (поэтому программа ищет исходные файлы в каталоге *hello*). Если исходные тексты, которые нужно обновить, находятся не в том каталоге, который указан в файле-заплатке, можно воспользоваться параметром *-p* без числа. Подробности ищите в страницах руководства по *patch*(1).

Создание отступов в коде

Если вы озабочены расстановкой отступов в коде, а редакторы, автоматически устанавливающие отступы во вводимом коде на лету, несколько раздражают вас, можно воспользоваться программой *indent* для красивого оформления кода по окончании его ввода. *indent* является интеллектуальным форматером программ на C и имеет много параметров, позволяющих указать желательный стиль отступов.

Рассмотрим пример ужасно отформатированного исходного кода:

```
double fact (double n) { if (n==1) return 1;
else return (n*fact(n-1)); }
int main () {
printf("Factorial 5 is %f.\n",fact(5));
printf("Factorial 10 is %f.\n",fact(10)); exit (0); }
```

В результате применения *indent* к этому исходному коду получим довольно красивый код:

```
#include <math.h>

double
fact (double n)
{
  if (n == 1)
    return 1;
  else
    return (n * fact (n - 1));
}
void
main ()
{

  printf ("Factorial 5 is %f.\n", fact (5));
  printf ("Factorial 10 is %f.\n", fact (10));
  exit (0);
}
```

В этом коде не только хорошо выставлены отступы строк, но и добавлены пробелы вокруг операторов и параметров функций, что делает их удобочитаемыми. Существует много способов определить, как должна выглядеть выдача *indent*. Если данный стиль отступов вам не нравится, *indent* может его изменить.

indent может также создавать из исходного файла код для *troff*, который пригоден для печати или включения в технический документ. В нем будут такие приятные характеристики, как выделение комментариев курсивом, выделение ключевых слов полужирным шрифтом и т. д. Выполнение команды:

```
papaya$ indent -troff importrtf.c | groff -mindent
```

создает код для *troff* и форматирует его с помощью *groff*.

Наконец, *indent* можно использовать в качестве простого средства отладки. Если вы поместили } в неправильном месте, то пропустив программу через *indent*, вы увидите, как компьютер представляет себе структуру блоков в вашей программе.

Интегрированные среды разработки

Хотя разработка программного обеспечения в Unix (а следовательно, и в Linux) традиционно основана на командной строке, на других платформах разработчики привыкли к так называемым «интегрированным средам разработки» (Integrated Development Environment – IDE), в которых в виде одного приложения объединены редактор, компилятор, отладчик и, возможно, другие инструменты разработчика. Разработчики, имевшие опыт работы с такой средой, обычно бывают ошеломлены, когда сталкиваются с командной строкой Linux, в которой нужно ввести команду *gcc*.[1]

Чтобы угодить таким разработчикам-мигрантам, а также потому, что разработчики Linux требуют все больших и больших удобств, для Linux тоже были разработаны интегрированные среды. Их существует немало, но широко распространена только одна, KDevelop.

KDevelop входит в состав проекта KDE, но может работать независимо от рабочего стола KDE. Она ведет учет всех файлов, входящих в ваш проект, генерирует файлы проектов (make-файлы), производит синтаксический анализ классов C++, в нее входят интегрированный отладчик и мастер, с помощью которого можно начать разработку нового приложения. KDevelop первоначально предназначалась в помощь при разработке приложений KDE, но может использоваться для разработки любых других программ, например, обычных программ командной строки и даже приложений GNOME.

---

[1] Авторам непонятно, почему ввести команду *gcc* труднее, чем выбрать пункт в меню, но это может быть связано с нашим социальным статусом.

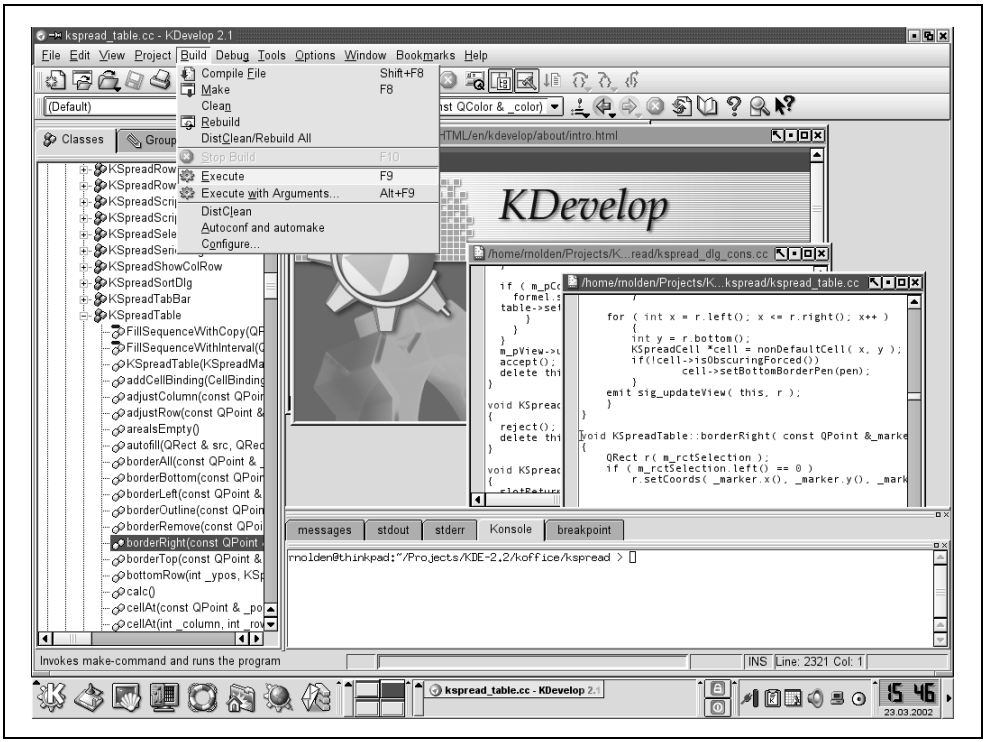
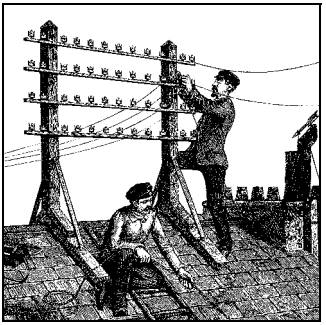


Рис. 14.1. *Интегрированная среда разработки KDevelop*

KDevelop слишком велика и насыщена функциями, чтобы знакомиться с ней в этой книге. Возбудим ваш аппетит хотя бы снимком экрана (рис. 14.1) и направим на *http://www.kdevelop.org*, откуда можно выполнить загрузку и получить всю информацию, включая полную документацию.

Между прочим, Emacs и XEmacs представляют собой отличную IDE, в которую интегрированы многие дополнительные средства, например *gdb*, как показано выше в этой главе.

15



Протоколы TCP/IP и PPP

Итак, вы застолбили свой участок во владениях Linux – установили и сконфигурировали вашу систему. Что дальше? В конечном счете, вы захотите связаться с другими системами – Linux и прочими, и голубиная почта для этого не очень подходит.

К счастью, Linux поддерживает множество методов обмена данными и работы в сети. Среди них последовательное соединение, TCP/IP и UUCP. В этой и следующей за ней главах мы рассмотрим способы конфигурации вашей системы для связи с внешним миром.

Полным руководством по конфигурации Linux для работы в сетях TCP/IP и UUCP является «Linux Network Administrator's Guide», входящее в Проект документирования Linux и опубликованное издательством O'Reilly.

Сетевое взаимодействие по протоколу TCP/IP

Linux обеспечивает полную реализацию TCP/IP (Transmission Control Protocol/Internet Protocol, сетевой протокол управления передачей/межсетевой протокол). TCP/IP оказался наиболее удачным механизмом работы в компьютерных сетях по всему миру. Располагая Linux и сетевой картой, можно подключить свой компьютер к локальной сети (LAN) или к Интернету – Всемирной сети, работающей на TCP/IP (при соответствующем сетевом подключении).

Построить небольшую локальную сеть из Unix-машин достаточно просто. Для этого необходимы сетевая карта Ethernet, установленная на каждой машине, соответствующие кабели Ethernet и другое оборудование. И если в вашей организации или университете есть доступ к Интернету, вы с легкостью можете присоединить вашу систему Linux к этой сети.

Поддержка TCP/IP в Linux имела свои подъемы и спады. В конце концов, реализация всего стека протоколов с нуля – это не то, что можно забавы ради сделать на выходных. С другой стороны, код стека TCP/IP в Linux значительно выиграл от того, что многочисленные бета-тестеры и разработчики

прошлись по нему, и со временем было исправлено много ошибок и проблем с настройками.

Текущая реализация TCP/IP и связанных с ним протоколов в Linux называется NET-4. Это не имеет никакого отношения к релизу BSD Unix, который называется NET-2. В данном контексте NET-4 означает четвертый вариант реализации TCP/IP в Linux. До NET-4 были (что не удивительно) NET-3, NET-2 и NET-1, последний из которых прекратил существование в версии ядра 0.99.pl10. NET-4 поддерживает большое количество сетевого оборудования и обладает практически всеми возможностями, которые вы можете ожидать от реализации TCP/IP в Unix.

Linux NET-4 поддерживает также SLIP (Serial Line Internet Protocol – интернет-протокол последовательных линий) и PPP (Point-to-Point Protocol – протокол типа «точка-точка»). SLIP и PPP позволяют получить доступ к Интернету по коммутируемым телефонным линиям с помощью модема. Если ваша организация или университет предоставляют доступ по SLIP или PPP, можно дозвониться до сервера и соединить вашу машину с Интернетом по телефонной линии. И наоборот, если ваша Linux-система имеет доступ к Интернету через локальную сеть, можно сконфигурировать ее как сервер SLIP или PPP.

Мы не будем больше касаться SLIP, т. к. большинство людей в наши дни использует PPP. Если вы намерены использовать SLIP в вашей системе, всю необходимую информацию можно найти в «Linux Network Administrator's Guide» Олафа Кирха (Olaf Kirch) и Терри Доусона (Terry Dawsan), O'Reilly.

Помимо этого руководства более или менее полную информацию по конфигурации TCP/IP и PPP в Linux можно найти в NET-4 HOWTO. В документе Linux Ethernet HOWTO описана настройка многочисленных драйверов сетевых карт Ethernet.

Представляет интерес и книга Крэйга Ханта (Craig Hunt) «TCP/IP. Network Administration»[1], содержащая всю информацию по конфигурированию и использованию TCP/IP в Unix-системах. Если вы собираетесь построить сеть на Linux-машинах или выполнять любую другую серьезную работу с TCP/IP, следует иметь в запасе знания по сетевому администрированию, представленные в этой книге.

Если вы всерьез решили заняться настройкой и обслуживанием сетей, возможно, вам нужно прочесть книгу «DNS and BIND»[2] Пола Альбитца (Paul Albitz) и Крикета Ли (Cricket Liu), O'Reilly. В живой и увлекательной манере в ней рассказывается обо всем, что связано с серверами имен.

---

[1] Крэйг Хант «TCP/IP. Сетевое администрирование», 3-е издание. – Пер. с англ. – СПб: Символ-Плюс, 2004.

[2] Пол Альбитц и Крикет Ли «DNS и BIND», 4-е издание. – Пер. с англ. – СПб: Символ-Плюс, 2002.

Основные понятия TCP/IP

Для того чтобы полностью оценить (и использовать) мощные возможности TCP/IP, необходимо поближе познакомиться с его основными принципами. Transmission Control Protocol/Internet Protocol – это набор *протоколов*, (это слово будет назойливо звучать в данной главе), определяющий способ взаимодействия различных машин по сети, а также взаимодействие между различными уровнями набора протоколов. Лучшими источниками прочных теоретических знаний в области протоколов Интернета являются книги Дугласа Комера (Douglas Comer) «Internetworking with TCP/IP», том 1 (Prentice Hall) и Ричарда Стивенса (W. Richard Stevens) «TCP/IP Illustrated», том 1 (Addison-Wesley).

TCP/IP был изначально разработан для использования в сети ARPAnet для поддержки военных и информационных разработок. Поэтому иногда можно услышать, как TCP/IP называют «интернет-протоколами DARPA». Вслед за этим первым Интернетом возникли другие сети TCP/IP, такие как National Science Foundation's NSFNET, а также тысячи других региональных и локальных сетей по всему миру. Все эти сети соединены между собой в единый конгломерат, известный как Интернет.

В сети TCP/IP каждой машине присвоен *IP-адрес*, который является 32-битным числом, идентифицирующим уникальным образом данную машину. Чтобы организовать сеть и присваивать адреса хостам, нужно немного знать об IP-адресах. IP-адрес обычно представляется в виде четырех десятичных чисел, разделенных точками. Например, IP-адрес 0x80114b14 (в шестнадцатеричном формате) можно записать как 128.17.75.20.

Здесь нужно сказать о двух особых случаях: динамических IP-адресах и маскируемых IP-адресах. Те и другие изобретены, чтобы справиться с возникшей нехваткой IP-адресов (что перестанет быть проблемой, когда все примут новый стандарт IPv6, отводящий шесть байт для IP-адреса – достаточно, чтобы снабдить адресом IP каждую амебу во вселенной).

Динамические IP-адреса часто используются в соединениях по коммутируемым линиям: вы дозваниваетесь до провайдера Интернет, который назначает вам IP-адрес из пула, выделенного им для этого вида сервиса. При следующем соединении вы можете получить другой номер IP. Смысл этого в том, что из всех клиентов провайдера лишь небольшая часть подключается к сети одновременно, поэтому требуется меньшее количество IP-адресов. Тем не менее, пока компьютер подключен к Интернету, его IP-адрес уникален – никакой другой компьютер не использует его в то же самое время.

Сокрытие IP-адресов (masquerading) позволяет нескольким компьютерам пользоваться IP-адресом совместно. Все машины в маскируемой сети пользуются частными IP-адресами, принадлежащими диапазону, который выделен для внутреннего употребления и не может служить реальным адресом в Интернете. Одни и те же частные IP-адреса могут быть использованы в любом количестве таких сетей, поскольку за пределами локальной сети они никому не видны. Одна машина, «сервер маскировки», будет отображать эти

закрытые IP-номера в один открытый IP-номер (который может быть динамическим или статическим) и с помощью хитроумного механизма отображения направлять входящие пакеты данных на нужную машину.

IP-адрес делится на две части: адрес сети и адрес хоста. Старшие биты адреса указывают на адрес сети, а оставшиеся биты определяют адрес хоста. (Обычно каждый *хост* является отдельной машиной в сети.) Размер этих двух полей адреса зависит от типа рассматриваемой сети. Например, в сети класса B (где первый байт IP-адреса имеет значение от 128 до 191) два первых байта идентифицируют сеть, а два оставшихся байта идентифицируют хост (рис. 15.1). В приведенном выше примере IP-адреса сеть имеет адрес 128.17, а адрес хоста – 75.20. Другими словами, машина с IP-адресом 128.17.75.20 является хостом с номером 75.20 в сети 128.17.

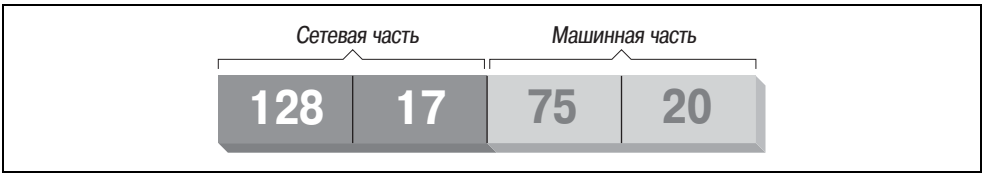


Рис. 15.1. *IP-адрес*

Кроме того, часть IP-адреса, идентифицирующая хост, может быть подразделена далее для выделения *адреса подсети*. Такое разбиение на подсети позволяет разделять большие сети на более мелкие подсети, каждой из которых можно управлять независимо от других сетей. Например, если организации выделена одна сеть класса B, в которой на информацию о хосте используется два байта, то в такой сети может быть до 65 534 хостов.[1] При желании сеть может быть разделена на подсети, управляемые разными отделами данной организации. При разделении на подсети первый байт из адреса хоста (который является третьим байтом во всем IP-адресе) может использоваться в качестве адреса подсети, а второй байт будет считаться адресом хоста для этой подсети (рис. 15.2). В этом случае IP-адрес 128.17.75.20 идентифицирует хост номер 20 в подсети 75 сети 128.17.

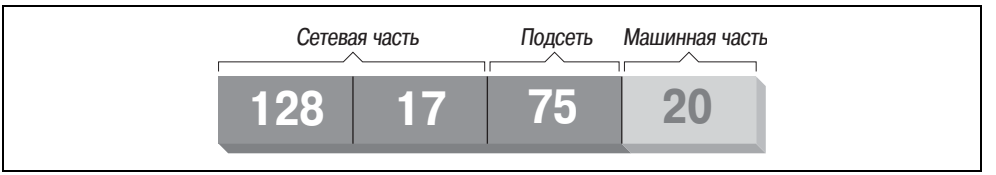


Рис. 15.2. *IP-адрес с делением на подсети*

Процессы (на одной или на разных машинах), которым необходима связь по TCP/IP, обычно указывают IP-адрес нужной машины, а также *адрес порта*.

---

[1] Почему не 65 536? По причинам, которые мы обсудим позже, адреса 0 и 255 недопустимы.

IP-адрес используется, конечно, для определения маршрута для передачи данных вызываемой машине. Адрес порта является 16-битным числом, определяющим приложение или сервис на вызываемой машине, которому предназначены данные. Номера портов можно представить как номера офисов в большом офисном здании: все здание имеет один IP-адрес, но у каждой фирмы в этом здании есть отдельный офис.

Рассмотрим пример реального использования IP-адресов и номеров порта. Программа *ssh* предоставляет возможность пользователю с одной машины начать сеанс регистрации на другой машине, причем все данные, которыми обмениваются эти две машины, будут зашифрованы. На удаленной машине работает «демон» *ssh*, *sshd*, который прослушивает определенный порт, ожидая входящих соединений (в данном случае порт имеет номер 22).[1]

При запуске *ssh* пользователь указывает адрес системы, к которой он хочет подключиться, и программа *ssh* пытается открыть соединение с портом 22 на удаленной машине. В случае успеха *ssh* и *sshd* получают возможность общаться между собой, чтобы предоставить вход данному пользователю.

Обратите внимание, что *ssh*-клиент на локальной машине тоже имеет свой адрес порта. Этот адрес динамически присваивается клиенту при его запуске, поскольку удаленному серверу *sshd* нет необходимости заранее знать номер порта входящего клиента *ssh*. Когда клиент инициирует соединение, номер его порта является частью информации, посылаемой демону *sshd*. Сервер *sshd* можно представить как фирму с известным всем адресом. Любой, кто хочет связаться с *sshd*, запущенным на определенной машине, должен знать не только IP-адрес машины (адрес офисного здания), но и номер порта, на котором можно найти *sshd* (номер офиса в здании). Адрес и номер порта *ssh*-клиента включены как часть «обратного адреса» на конверте.

Семейство TCP/IP содержит несколько протоколов. Transmission Control Protocol (TCP), или протокол управления передачей, отвечает за надежную связь с установкой соединения между двумя процессами, которые могут быть запущены на разных машинах в сети. User Datagram Protocol (UDP), или протокол пользовательских дейтаграмм, похож на TCP за исключением того, что он не устанавливает соединение и не гарантирует доставку пакетов. Процессы, использующие UDP, должны иметь собственные процедуры подтверждения и синхронизации, если это необходимо.

TCP и UDP передают и получают данные блоками, которые называют пакетами. Каждый пакет содержит фрагмент информации, которую нужно передать другой машине, а также заголовок, определяющий адреса портов отправителя и получателя.

Internet Protocol (IP), или межсетевой протокол, находится в иерархии протоколов ниже TCP и UDP. Он отвечает за передачу и маршрутизацию паке-

---

[1] На многих системах *sshd* не всегда прослушивает порт 22, а вместо него этот порт прослушивает демон служб Интернета – *inetd*. Но сейчас мы не будем отвлекаться на такие детали.

тов TCP или UDP по сети. Для этого протокол IP заключает каждый пакет TCP или UDP в другой пакет (известный как *дейтаграмма* IP), который содержит заголовок с информацией о маршруте и получателе. В заголовке дейтаграммы IP содержатся IP-адреса машин отправителя и получателя.

Следует отметить, что IP ничего не знает о номерах портов; за них отвечают TCP и UDP. Аналогично TCP и UDP не обрабатывают IP-адреса, которые находятся (как видно из названия) исключительно в ведении IP. Как видите, метафора с почтой, конвертами и адресами подходит достаточно точно: каждый пакет можно представить в виде письма, вложенного в конверт. TCP и UDP запечатывают письмо в конверт, на котором указывают номера портов отправителя и адресата (номер офиса).

IP выступает в роли почтовой службы офисного здания. IP получает конверт и запечатывает его в другой конверт с IP-адресами получателя и отправителя (адрес офисного здания). Служба доставки (о которой мы еще не говорили) доставляет письмо в соответствующее офисное здание. Здесь почтовая служба распечатывает наружный конверт и передает письмо TCP или UDP, которые доставляют его в соответствующий офис, основываясь на номере порта (он написан на внутреннем конверте). Каждый конверт имеет обратный адрес, который IP и TCP/ UDP используют для ответа на письмо.

Чтобы указывать машины в Интернете более понятным человеку способом, хостам, наряду с IP-адресами, даются имена. Служба доменных имен (Domain Name Service, DNS) заботится о преобразовании имен хостов в IP-адреса и наоборот, а также управляет распределением в рамках всего Интернета базы данных с соответствиями имен и IP-адресов. Использование имен хостов также позволяет изменять IP-адрес, связанный с машиной (например, если компьютер перевели в другую сеть), не беспокоясь о том, что после смены адреса машину нельзя будет найти. Необходимо лишь обновить запись в DNS для этого хоста, и все ссылки на эту машину по имени будут работать и дальше.

DNS является огромной, распределенной по всему миру базой данных. Каждая организация управляет частью этой базы, в которой перечислены имеющиеся в ней машины. Если в вашей организации эта работа поручена вам, помощь можно получить из «Linux Network Administrator's Guide» или «TCP/IP. Network Administration» (O'Reilly). Если этого недостаточно, наиболее полную информацию вы найдете в книге «DNS and BIND», O'Reilly.

Для целей обычного администрирования достаточно знать, что в вашей системе должен быть запущен демон *named* (произносится как «name-dee»). Этот демон является вашим окном в DNS.

Может возникнуть вопрос, как пакет от одной машины (офисное здание) добирается до другой. Это главная задача IP, а также некоторых других протоколов, которые помогают IP в его работе. Кроме обработки IP-дейтаграмм на каждом хосте (как на почте) IP отвечает за маршрутизацию пакетов между хостами.

Прежде чем обсуждать, как работает маршрутизация, мы должны описать модель, по которой построены сети TCP/IP. Сеть – это просто набор машин,

которые физически некоторым образом соединены между собой, например, при помощи Ethernet или последовательных соединений. В терминах TCP/IP каждая сеть имеет свои собственные методы для внутренней маршрутизации и передачи пакетов.

Сети соединены между собой *шлюзами* (*gateway*), которые также известны как маршрутизаторы (*routers*). Шлюз – это хост, имеющий прямое соединение с двумя или более сетями; он может передавать информацию между сетями и направлять пакеты из одной сети в другую. В частности, шлюзом может быть рабочая станция, имеющая более одного интерфейса Ethernet. Каждый из интерфейсов соединен со своей сетью, и операционная система использует эти соединения для того, чтобы машина могла работать в качестве шлюза.

Чтобы конкретизировать наши рассуждения, представим себе воображаемую сеть, состоящую из машин *eggplant*, *papaya*, *apricot* и *zucchini*. На рис. 15.3 представлена конфигурация этих машин в сети.

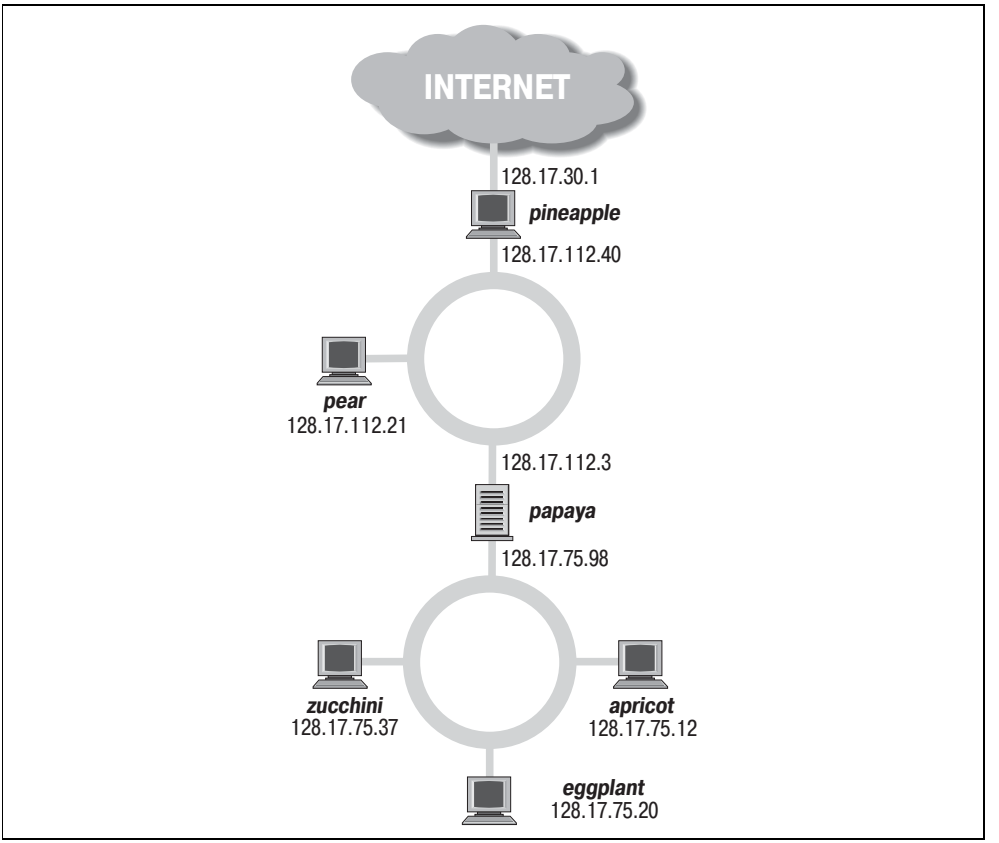


Рис. 15.3. *Сеть с двумя шлюзами*

Обратите внимание, что *papaya* соединена и с другой сетью, где находятся машины *pineapple* и *pear*. Машины имеют соответствующие IP-адреса:

Имя хоста	IP-адрес
eggplant	128.17.75.20
apricot	128.17.75.12
zucchini	128.17.75.37
papaya	128.17.75.98, 128.17.112.3
pear	128.17.112.21
pineapple	128.17.112.40, 128.17.30.1

Как можно видеть, у *papaya* два IP-адреса: один в подсети 128.17.75 и другой в подсети 128.17.112. Два IP-адреса есть и у *pineapple*: один для подсети 128.17.112, а другой для подсети 128.17.30.

Чтобы определить маршрут передачи пакетов между машинами, IP использует сетевую часть IP-адреса. Для этого на каждой машине в сети имеется *таблица маршрутизации* (*routing table*), в которой хранится список сетей и шлюзов для этих сетей. Для доставки пакета на определенную машину IP просматривает сетевую часть адреса назначения. Если для этой сети есть запись в таблице маршрутизации, IP отправляет пакет через соответствующий шлюз. В противном случае пакет отправляется через шлюз «по умолчанию», заданный в таблице.

Таблицы маршрутизации могут содержать записи как для сетей, так и для отдельных машин. Кроме того, на каждой машине есть запись о маршруте для себя самой.

Рассмотрим таблицу маршрутизации для машины *eggplant*. Выполнив команду *netstat -rn*, мы увидим следующее:

```
eggplant:$ netstat -rn
Kernel IP routing table
Destination   Gateway       Genmask        Flags  MSS  Window irtt Iface
128.17.75.0   128.17.75.20  255.255.255.0  UN     1500 0         0 eth0
default       128.17.75.98  0.0.0.0        UGN    1500 0         0 eth0
127.0.0.1     127.0.0.1     255.0.0.0      UH     3584 0         0 lo
128.17.75.20  127.0.0.1     255.255.255.0  UH     3584 0         0 lo
```

Первая колонка содержит адреса сетей (и хостов) назначения, которые включены в таблицу. Первой указана запись для сети 128.17.75 (обратите внимание, что в записях для сетей адрес хоста равен 0), где расположен хост *eggplant*. Все пакеты, посылаемые в эту сеть, будут отправлены через шлюз 128.17.75.20, который является IP-адресом *eggplant*. Обычно маршрут к собственной сети идет через интерфейс самой машины.

Колонка `Flags` таблицы маршрутизации дает информацию об адресе назначения для этой записи. `U` указывает на то, что маршрут работает («up»), `N` указывает на то, что это маршрут к сети, `H` – на то, что это маршрут к хосту и т. д. Поле `MSS` показывает число байт, отправляемое соответствующему соединению за один раз (максимальный размер сегмента), `Window` указывает

количество фреймов, которое может быть отправлено до получения подтверждения о приеме, `irtt` выдает статистику использования маршрута, а `Iface` указывает сетевое устройство, используемое для этого маршрута. В системах Linux интерфейсы Ethernet именуются *eth0*, *eth1* и т. д. *lo* – это кольцевое устройство (loopback), о котором мы вскоре расскажем.

Вторая запись в таблице является маршрутом по умолчанию, который применяется ко всем пакетам, посылаемым сетям или отдельным хостам, для которых нет записи в таблице. В данном случае маршрут по умолчанию лежит через машину *papaya*, которую можно считать дверью во внешний мир. Все машины в подсети 128.17.75 для связи с машинами любой другой подсети должны пройти через машину *papaya*.

Третья запись в таблице сделана для адреса 127.0.0.1, который является адресом *кольцевого интерфейса* (*loopback*). Этот адрес используется, если машине необходимо выполнить подключение по TCP/IP к себе самой. Устройство *lo* используется в качестве интерфейса с целью предотвращения использования сети Ethernet (через интерфейс *eth0*) при loopback-подключениях. В этом случае ресурсы сети не тратятся впустую, когда компьютер хочет поговорить сам с собой.

Последняя запись в таблице маршрутизации сделана для IP-адреса 128.17.75.20, который является собственным адресом хоста *eggplant*. Как можно заметить, в качестве шлюза выступает адрес 127.0.0.1. Таким образом, каждый раз, когда машина *eggplant* создает TCP/IP подключение к себе, она это делает через сетевое «устройство» *lo*, а адрес закольцовывающегося интерфейса используется как шлюз.

Представим себе, что *eggplant* хочет послать пакет *zucchini*. Дейтаграмма IP содержит адрес источника 128.17.75.20 и адрес назначения 128.17.75.37. IP определяет, что сетевая часть адреса назначения равна 128.17.75 и соответственно использует запись для 128.17.75.0 в таблице маршрутизации. Пакет посылается прямо в сеть, где *zucchini* может получить его и обработать.

Что произойдет, если *eggplant* захочет послать пакеты машине, находящейся вне локальной сети, например *pear*? В этом случае адрес назначения будет 128.17.112.21. IP попытается найти в таблице маршрутизации маршрут для сети 128.17.112, но поскольку такого маршрута нет, будет выбран шлюз по умолчанию, которым является *papaya*. Получив пакет, *papaya* ищет адрес назначения в своих собственных таблицах маршрутизации. Таблица маршрутизации на машине *papaya* может выглядеть следующим образом:

```
Destination   Gateway       Genmask        Flags  MSS  Window irtt Iface
128.17.75.0   128.17.75.98  255.255.255.0  UN     1500 0         0 eth0
128.17.112.0  128.17.112.3  255.255.255.0  UN     1500 0         0 eth1
default       128.17.112.40 0.0.0.0        UGN    1500 0         0 eth1
127.0.0.1     127.0.0.1     255.0.0.0      UH     3584 0         0 lo
128.17.75.98  127.0.0.1     255.255.255.0  UH     3584 0         0 lo
```

Мы видим, что *papaya* соединена с сетью 128.17.75 через устройство *eth0* и с сетью 128.17.112 – через устройство *eth1*. Маршрут по умолчанию идет че-

рез *pineapple*, которая является шлюзом в Большое Нечто (насколько это известно машине *papaya*).

Получив пакет, предназначенный для *pear*, маршрутизатор *papaya* видит, что адрес назначения находится в сети 128.17.112, и отправляет этот пакет, используя вторую запись в таблице маршрутизации.

Аналогично, если *eggplant* хочет связаться с машинами, находящимися за пределами локальной сети организации, она отправляет пакет через шлюз *papaya*. Оттуда, в свою очередь, исходящие пакеты будут отправлены через *pineapple* и т. д. Пакеты передаются от одного шлюза другому до тех пор, пока они не достигнут сети назначения. Такова структура, на которой основан Интернет: кажущаяся бесконечной цепь сетей, связанных между собой шлюзами.

Требования к аппаратуре

Протоколами TCP/IP можно пользоваться в Linux без какого-либо сетевого оборудования вообще, сконфигурировав режим «loopback», который позволяет машине подключаться к себе самой. Это необходимо для некоторых приложений и игр, использующих сетевое «устройство» кольцевого интерфейса.

Однако если вы хотите использовать Linux в сети Ethernet TCP/IP, очевидно, вам потребуется сетевой адаптер Ethernet. Linux поддерживает множество адаптеров для шин ISA, EISA, PCI, а также карманные и PCMCIA. В главе 1 приводится частичный список поддерживаемых карт Ethernet. Полную информацию по аппаратной совместимости с устройствами Ethernet вы найдете в Linux Ethernet HOWTO.

В последние годы в Linux была добавлена поддержка высокоскоростных сетевых соединений, отличных от Ethernet, таких как HIPPI. Эта тема выходит за рамки нашей книги, но если она вас интересует, некоторую информацию можно получить в каталоге *Documentation/networking* исходного кода ядра.

Если у вас есть соединение ADSL и используется маршрутизатор ADSL, то для Linux оно представляется обычным соединением Ethernet, поэтому не требуется ни специального оборудования (кроме карты Ethernet, разумеется), ни особых драйверов помимо драйвера самой карты Ethernet. Если вы хотите соединить машину Linux непосредственно с модемом ADSL, то вам также не потребуется специального оборудования или драйвера, но нужно будет запустить протокол под названием PPPoE (PPP over Ethernet), о чем будет сказано подробнее ниже.

Linux также поддерживает протоколы SLIP и PPP, которые позволяют использовать модемы для связи с Интернетом по телефонным линиям. В этом случае вам потребуется модем, совместимый с сервером SLIP или PPP, к которому вы подключаетесь. Например, для многих серверов требуются модемы 56kbps V.90 (большинство также поддерживает K56flex). В этой книге мы описываем настройку PPP, т. к. именно этот протокол используется большинством провайдеров Интернета. При необходимости использования более старого протокола SLIP обращайтесь к «Linux Network Administrator's Guide» (O'Reilly).

Наконец, существует протокол PLIP, позволяющий соединить два компьютера через параллельные порты, для чего требуется специальный соединительный кабель.

Настройка TCP/IP для использования в сети Ethernet

В этом разделе мы обсудим, как конфигурировать TCP/IP в системах Linux для подключения к Ethernet. Возможно, эта система будет частью локальной сети, в которой уже используется TCP/IP. В этом случае уже будут сконфигурированы и доступны шлюз, сервер имен и т. д.

Следующая информация касается в первую очередь подключения к Ethernet. Если вы собираетесь использовать PPP, прочтите этот раздел, чтобы познакомиться с основными понятиями, а специальные инструкции по настройке PPP вы найдете в разделе «Соединение по коммутируемой линии по PPP» далее в этой главе.

С другой стороны, вы, возможно, хотите установить целую сеть из машин Linux (или из машин Linux и других систем). В этом случае вам придется позаботиться о ряде вопросов, которые здесь не обсуждаются. Сюда входит установка своего сервера имен, а также настройка шлюза, если вашу сеть необходимо соединить с другими сетями. Если вашу сеть необходимо соединить с Интернетом, нужно также получить IP-адреса и сопутствующую информацию у вашего провайдера.

Описанный здесь метод должен работать во многих системах Linux, сконфигурированных для уже настроенных сетей, но наверняка не для всех. За более подробной информацией мы направляем вас к книгам по сетевому администрированию TCP/IP, которые были упомянуты в начале этой главы.

Прежде всего, мы предполагаем, что на вашей системе Linux установлено все необходимое программное обеспечение TCP/IP. В него входят основные клиенты, такие как *ssh* и FTP, команды системного администрирования, такие как *ifconfig* и *route* (обычно расположенные в */etc* или */sbin*), и файлы конфигурации сети (такие как */etc/hosts*). Документы по сетям в Linux, о которых говорилось ранее, помогут разобраться с установкой поддержки сети в Linux, если у вас ее еще нет.

Мы также предполагаем, что ядро было скомпилировано и сконфигурировано с включенной поддержкой TCP/IP. Информация по компилированию ядра системы приводится в разделе «Сборка нового ядра» главы 7. Для того чтобы включить работу с сетью, необходимо утвердительно ответить на соответствующие вопросы на шаге *make config* или *make menuconfig*, затем перекомпилировать ядро и перезагрузить машину.

Затем необходимо изменить ряд файлов конфигурации, используемых NET-4. По большей части это простая процедура. Однако, к сожалению, между различными дистрибьюторами Linux имеются значительные разногласия по поводу того, где следует находиться многочисленным программам и файлам конфигурации TCP/IP. В большинстве случаев их можно найти в */etc*, но иногда их можно обнаружить в */usr/etc*, */usr/etc/inet* или в других необыч-

ных местах. В худшем случае вам придется прибегнуть к команде *find*, чтобы найти эти файлы. Также имейте в виду, что не во всех дистрибутивах файлы конфигурации NET-4 и программное обеспечение находятся в одном месте; они могут быть разнесены по разным каталогам.

Мы расскажем о том, как установить и настроить поддержку сети в Linux вручную. В результате вы должны получить некоторое представление о том, что в действительности происходит, и самостоятельно справиться с проблемами, которые могут возникнуть при автоматической установке, осуществляемой вашим дистрибутивом. Однако разумно попробовать сначала установить поддержку сети с помощью тех программ настройки, которые есть в вашем дистрибутиве. Многие из них стали достаточно развитыми и могут автоматически определять ряд необходимых настроек.

В этом разделе предполагается, что в системе используется одно устройство Ethernet. Если же в вашей системе имеется более одного сетевого соединения (следовательно, система выступает в роли шлюза), все указания могут быть с легкостью экстраполированы.

Здесь мы также обсудим конфигурирование систем только с одним кольцевым интерфейсом (без Ethernet или PPP-соединений). Если у вас нет доступа к сети, вы, возможно, захотите сконфигурировать TCP/IP на использование лишь кольцевого интерфейса (loopback). Тогда вы сможете без реального подключения к сети использовать приложения, которые требуют TCP/IP.

Конфигурация сети

Прежде чем вы начнете конфигурировать TCP/IP, необходимо выяснить следующую информацию о настройке вашей сети. В большинстве случаев эту информацию может предоставить администратор вашей локальной сети или провайдер доступа к сети.

Ваш IP-адрес

Это уникальный адрес машины в формате десятичных чисел, разделенных точками, например 128.17.75.98. Этот номер предоставляется вам сетевым администратором.

Если вы конфигурируете закольцовывающийся интерфейс (в этом случае нет PPP и сетевой карты, а имеется только TCP/IP-подключение к собственной машине), вашим IP-адресом будет 127.0.0.1.

Маска подсети

Это четыре числа, разделенные точками, как IP-адрес, определяющие, какая часть IP-адреса является адресом подсети, а какая – адресом хоста в этой подсети.

Маска подсети является числом, которое складывается с IP-адресом при помощи операции логическое И, в результате чего выясняется, к какой подсети принадлежит ваш адрес. Например, ваша маска подсети может быть такой: 255.255.255.0. Если ваш IP-адрес – 128.17.75.20, то часть адреса, указывающая на подсеть, будет 128.17.75.

Следует различать «адрес сети» и «адрес подсети». Напоминаем, что для адресов сетей класса B первые два байта (в данном случае 128.17) определяют сеть, а вторые два байта указывают на хост. Однако с маской подсети 255.255.255.0 адрес 128.17.75 считается адресом целой подсети (т. е. подсеть 75 сети 128.17), а 20 будет адресом хоста.

Сетевую маску выбирает сетевой администратор, и поэтому он может предоставить вам эту информацию.

Это касается также закольцовывающегося интерфейса. Поскольку его адрес всегда равен 127.0.0.1, сетевая маска для этого адреса всегда будет 255.0.0.0.

Адрес вашей подсети

Это часть вашего IP-адреса, указывающая на подсеть, как это определено маской подсети. Например, при маске подсети 255.255.255.0 и IP-адресе 128.17.75.20 адрес подсети будет 128.17.75.0.

Системы только с закольцовывающимся интерфейсом не имеют адреса подсети.

Широковещательный адрес

Этот адрес используется для широковещательной отправки пакетов всем машинам вашей подсети. Обычно он равен адресу подсети (см. выше), где на месте хоста стоит 255. Для подсети с адресом 128.17.75.0 широковещательный адрес будет 128.17.75.255. Аналогично, для подсети с адресом 128.17.0.0 широковещательный адрес будет 128.17.255.255.

Обратите внимание, что некоторые системы используют сам адрес подсети для отправки широковещательных сообщений. Если у вас есть какие-нибудь сомнения, поделитесь ими с администратором вашей сети.

Системы только с закольцовывающимся интерфейсом не имеют адреса широковещания.

IP-адрес вашего шлюза

Это адрес машины, которая выступает шлюзом по умолчанию для связи с внешним миром. На самом деле у вас может быть несколько адресов шлюзов, например, если ваша сеть напрямую соединена с несколькими другими сетями. Однако только один из шлюзов будет выступать в качестве маршрута *по умолчанию*. (Вспомните пример из предыдущего раздела, где сеть 128.17.112.0 соединена через *papaya* с сетью 128.17.75.0 и выходит в остальной мир через *pineapple*.)

Ваши сетевые администраторы предоставят информацию об IP-адресах всех шлюзов вашей сети, а также о сетях, с которыми они соединяют вашу сеть. Позже вы используете эти данные, когда с помощью команды *route* будете вводить в таблицу маршрутизации записи для каждого шлюза.

Системы только с закольцовывающимся интерфейсом не имеют адресов шлюзов. Это же верно и для изолированных сетей.

IP-адрес вашего сервера имен

Это адрес машины, выполняющей преобразование имен хостов в адреса для вашей системы. Этот адрес вам предоставят администраторы сети.

Возможно, вы захотите запустить собственный сервер имен (сконфигурировав и запустив named). Однако если нет крайней необходимости в собственном сервере имен (например, если другого в вашей локальной сети нет), мы рекомендуем использовать адрес сервера имен, который вам предоставил администратор. Информация по управлению сервисом *named* содержится в большинстве книг по конфигурированию TCP/IP.

Естественно, системы только с закольцовывающимся интерфейсом не имеют адреса сервера имен.

Сетевые файлы rc

Файлы *rc* – это системные сценарии конфигурации, выполняемые *init* во время начальной загрузки. Они запускают всех основных системных демонов (таких как *sendmail*, *crond* и т. д.) и используются для настройки параметров сетей. Файлы *rc* обычно находятся в каталоге */etc/rc.d*.

Обратите внимание на то, что существует *много* способов управлять настройками сети, описанными здесь. Каждый дистрибутив Linux использует слегка отличающийся от других механизм автоматизации этих настроек. Описываемый нами метод является общим и позволяет создать два *rc*-файла, которые выполняют соответствующие команды для выхода вашей машины в сеть. В большинстве дистрибутивов есть свои собственные сценарии, которые выполняют примерно то же самое. Если у вас есть сомнения, сначала попытайтесь сконфигурировать сеть так, как предлагается в документации к вашему дистрибутиву, и лишь в качестве последнего средства используйте методы, описываемые здесь. (Например, дистрибутив Red Hat использует сценарий */etc/rc.d/init.d/network*, который получает информацию о сети из файлов в */etc/sysconfig*. Программа *control-panel,* поставляемая вместе с Red Hat, конфигурирует сеть автоматически без редактирования этих файлов. Дистрибутив SuSE, с другой стороны, распределяет настройки по нескольким файлам, таким как */sbin/init.d/network*, */sbin/init.d/route* и другим, и позволяет сконфигурировать большинство аспектов сетевой конфигурации с помощью утилиты *yast2*.)

Далее мы опишем файлы *rc*, используемые для конфигурирования TCP/IP в наиболее распространенных дистрибутивах:

Red Hat

Сетевые настройки разбросаны по файлам для каждого уровня *init*, который предполагает работу с сетью. В частности, каталог */etc/rc.d/rc1.d* управляет загрузкой уровня 1 (однопользовательский уровень), и поэтому там нет никаких сетевых команд, а */etc/rc.d/rc3.d* управляет загрузкой уровня 3 и имеет специальные файлы для запуска сети.

SuSE

Все файлы запуска для всех системных сервисов сгруппированы вместе в каталоге */sbin/init.d*. Они весьма универсальны и получают фактические значения из системного файла конфигурации */etc/rc.config*. Самыми важными файлами здесь являются: */sbin/init.d/network*, который запускает и останавливает сетевые интерфейсы, */sbin/init.d/route*, который настраивает маршрутизацию, и */sbin/init.d/serial*, который конфигурирует последовательные порты. Если у вас есть оборудование ISDN, к нему также применяются файлы */sbin/init.d/i4l* и */sbin/init.d/i4l_hardware*. Имейте в виду, что обычно не требуется (да и не следует) редактировать эти файлы, а вместо них редактируйте файл */etc/rc.config*.

Debian

Конфигурация сети (сетевые карты, IP-адреса и маршрутизация) настраивается в файле */etc/init.d/network*. Основные сетевые демоны (*portmap* и *inetd*) сами инициализируются сценарием запуска-остановки */etc/init.d/netbase*.

Для иллюстрации мы возьмем два файла: */etc/rc.d/rc.inet1* и */etc/rc.d/rc.inet2*. *rc.inet1* используется для настройки оборудования и основных сетевых параметров, тогда как *rc.inet2* настраивает сетевые сервисы. Такое разделение используется во многих дистрибутивах, хотя имена файлов могут различаться.

init использует файл */etc/inittab* для того, чтобы выбрать процесс, который надо запустить во время загрузки. Для запуска файлов */etc/rc.d/rc.inet1* и */etc/rc.d/rc.inet2* из *init* в */etc/inittab* могут быть следующие записи:

```
n1:34:wait:/etc/rc.d/rc.inet1
n2:34:wait:/etc/rc.d/rc.inet2
```

Файл *inittab* описан в разделе «Файлы init, inittab и rc файлы» главы 5. Первое поле задает уникальный двухсимвольный идентификатор для каждой записи. Во втором поле перечислены уровни исполнения, на которых запускаются сценарии; в данной системе сеть инициализируется на уровнях исполнения 3 и 4. Слово `wait` в третьем поле сообщает *init*, что продолжать работу можно только после завершения выполнения сценария. Последнее поле является именем сценария, который следует выполнить.

Пока вы занимаетесь настройкой сети, возможно, вы захотите запустить *rc.inet1 и rc.inet2* вручную (в качестве `root`) для отладки каких-либо проблем. Позднее можно добавить записи для них в другой *rc*-файл или в */etc/inittab*.

Как упоминалось ранее, *rc.inet1* конфигурирует основной сетевой интерфейс, устанавливая для вашей системы IP-адрес и адрес сети, а также данные в таблице маршрутизации. Для настройки этих параметров используются две программы: *ifconfig* и *route*, которые обычно находятся в */sbin*.

ifconfig используется для настройки определенных параметров интерфейса сетевых устройств, таких как IP-адрес, маска подсети, широковещательный

адрес и т. д. Для создания и модификации таблицы маршрутизации используется *route*.

В большинстве случаев подойдет файл *rc.inet1*, подобный приведенному ниже. Разумеется, придется отредактировать его для использования в своей системе. Не используйте IP-адрес и адрес сети, приведенные здесь в качестве примера; они могут соответствовать реальной машине в Интернете:

```
#!/bin/sh
# файл /etc/rc.d/rc.inet1 - Настройка TCP/IP интерфейсов.
# Сначала конфигурируем закольцовывающийся интерфейс (loopback)

HOSTNAME=`hostname`

/sbin/ifconfig lo 127.0.0.1  # по умолчанию использует маску сети 255.0.0.0
/sbin/route add 127.0.0.1    # маршрут к устройству loopback

# Затем конфигурируем устройства Ethernet. Если вы используете
# только loopback или SLIP, закомментируйте остальные строки.

# Отредактируйте для вашей установки.
IPADDR="128.17.75.20"       # ЗАМЕНИТЬ вашим IP-адресом
NETMASK="255.255.255.0"     # ЗАМЕНИТЬ вашей маской подсети
NETWORK="128.17.75.0"       # ЗАМЕНИТЬ вашим адресом сети
BROADCAST="128.17.75.255"   # ЗАМЕНИТЬ вашим широковещательным адресом
GATEWAY="128.17.75.98"      # ЗАМЕНИТЬ адресом вашего шлюза по умолчанию

# Сконфигурируйте устройство eth0 на использование этих данных.
/sbin/ifconfig eth0 ${IPADDR} netmask ${NETMASK} broadcast ${BROADCAST}

# Добавить маршрут для собственной подсети
/sbin/route add ${NETWORK}

# Добавить маршрут для шлюза по умолчанию
/sbin/route add default gw ${GATEWAY} metric 1

# Конец конфигурации Ethernet
```

Как вы видели, команда *ifconfig* имеет следующий формат:

```
ifconfig interface device options…
```

Например:

```
ifconfig lo 127.0.0.1
```

присваивает устройству *lo* (loopback) IP-адрес 127.0.0.1, а

```
ifconfig eth0 127.17.75.20
```

присваивает устройству *eth0* (первая карта Ethernet) адрес 127.17.75.20.

Помимо указания IP-адреса устройства Ethernet обычно требуют, чтобы были установлены маска подсети с помощью параметра *netmask* и широковещательный адрес с помощью параметра *broadcast*.

Команда *route* имеет здесь следующий формат:

```
route add [ -net | -host ] destination [ gw gateway ]
[ metric metric ] options
```

где *destination* – адрес назначения для этого маршрута (либо ключевое слово default), *gateway* – IP-адрес шлюза для этого маршрута и *metric* – метрика для маршрута (которую мы обсудим позже).

Для добавления записей в таблицу маршрутизации используется *route*. Необходимо добавить маршрут для кольцевого интерфейса (как говорилось ранее), для локальной сети и для шлюза по умолчанию. Например, если шлюз по умолчанию имеет адрес 128.17.75.98, можно использовать команду:

```
route add default gw 128.17.75.98
```

route имеет несколько параметров. Задание *–net* или *–host* перед адресом назначения (*destination*) указывает *route*, что адрес является адресом сети или отдельного хоста, соответственно. (В большинстве случаев маршруты указывают маршруты к сетям, но иногда может понадобиться указать маршрут к отдельной машине. Для такой записи в таблице используется *–host*.)

Параметр *metric* устанавливает *значение метрики* для данного маршрута. Значения метрики используются в тех случаях, когда есть несколько маршрутов к месту назначения и система должна принять решение, какой из них выбрать. Выбираются маршруты с наименьшим значением метрики. В нашем примере установим значение метрики для маршрута по умолчанию равным 1, и ему будет отдаваться предпочтение перед другими маршрутами.

Каким образом могут появиться несколько маршрутов к одному определенному адресу назначения? Прежде всего, можно использовать несколько команд *route* в *rc.inet1* для одного адреса, например, когда есть несколько шлюзов в некоторую сеть. Однако таблицы маршрутизации могут динамически пополняться новыми записями, если работает *routed* (описываемый позднее). Когда работает *routed*, в таблице маршрутизации на вашей машине могут создаваться новые записи, если другие системы широковещательно распространяют информацию о маршрутах. Установив значение метрики в 1, вы гарантируете, что новые записи в таблице маршрутизации не получат приоритета над вашим шлюзом по умолчанию.

Вам стоит прочитать страницы руководства для команд *ifconfig* и *route* с подробным описанием их синтаксиса. Возможно, вашей конфигурации подойдут другие параметры этих команд.

Пойдем дальше. *rc.inet2* применяется для запуска различных демонов, используемых TCP/IP. Они не обязательны для работы с сетью и поэтому выделены в отдельный *rc*-файл. Как правило, нужно попытаться сконфигурировать *rc.inet1* и убедиться, что ваша система способна принимать и посылать пакеты в сеть, прежде чем начинать конфигурировать файл *rc.inet2*.

rc.inet2 запускает следующие демоны: *inetd*, *syslogd* и *routed*. Файл *rc.inet2* на вашей системе может запускать и другие серверы, но мы рекомендуем закомментировать их на время отладки сетевой конфигурации.

Наиболее важным среди этих серверов является *inetd*, который выступает оператором для других системных демонов. Он работает в фоновом режиме и прослушивает определенные сетевые порты, ожидая входящие соединения.

Когда выполняется подключение, *inetd* создает новую копию соответствующего демона для этого порта. Например, когда выполняется входящее соединение FTP, *inetd* запускает *in.ftpd*, который затем управляет FTP-соединением. Это проще и эффективней, чем запускать много индивидуальных копий каждого демона. В этом случае демоны запускаются только при необходимости.

syslogd – это демон регистрации системных событий. Он собирает сообщения от различных приложений и записывает их в файл регистрации, основываясь на информации о конфигурации из */etc/syslogd.conf*.

routed – это сервер, используемый для управления динамической информацией о маршрутизации. Когда ваша система пытается послать пакет в другую сеть, это может потребовать дополнительных записей в таблице маршрутизации. *routed* заботится об управлении таблицей маршрутизации без необходимости вмешательства пользователя.

Вот пример *rc.inet2*, который запускает *syslogd*, *inetd* и *routed*:

```
#! /bin/sh
# Пример /etc/rc.d/rc.inet2

# Запуск syslogd
if [ -f /usr/sbin/syslogd ]
then
      /usr/sbin/syslogd
fi

# Запуск inetd
if [ -f /usr/sbin/inetd ]
then
      /usr/sbin/inetd
fi

# Запуск routed
if [ -f /usr/sbin/routed ]
then
      /usr/sbin/routed -q
fi
```

named является одним из многих дополнительных серверов, которые вы, возможно, захотите запустить в *rc.inet2*. *named* – это сервер имен, который отвечает за преобразование IP-адресов (локальных) в имена и наоборот. Использование *named* может понадобиться, если в вашей сети нет сервера имен или вы сами хотите снабжать локальными именами хостов другие машины вашего домена. Настройка *named* достаточно сложна и требует предварительного планирования. Заинтересованных читателей мы отсылаем к книге «DNS and BIND», O'Reilly.

/etc/hosts

/etc/hosts содержит перечень IP-адресов и имен хостов, которым они соответствуют. Обычно */etc/hosts* содержит только записи для вашей локальной

машины и, возможно, других «важных» машин (таких как сервер имен или шлюз). Преобразование адресов в имена для других машин в сети выполняет локальный сервер имен.

Например, если ваша машина называется *eggplant.veggie.com* и ее IP-адрес – 128.17.75.20, то */etc/hosts* может выглядеть так:

```
127.0.0.1              localhost
128.17.75.20           eggplant.veggie.com eggplant
```

Если вы используете только кольцевой интерфейс, единственной строкой в файле */etc/hosts* будет адрес 127.0.0.1.

/etc/networks

Файл */etc/networks* содержит имена и адреса вашей собственной и других сетей. Он используется командой *route* и позволяет задавать сеть именем вместо адреса.

Любая сеть, маршрут для которой вы хотите добавить при помощи команды *route* (обычно вызываемой из *rc.inet1*), должна для удобства иметь запись в */etc/networks*. В противном случае вместо имени вам придется указывать IP-адрес сети.

Например:

```
default       0.0.0.0       # маршрут по умолчанию    - обязательно
loopnet       127.0.0.0     # кольцевая сеть          - обязательно
veggie-net    128.17.75.0   # Изменить на ваш собственный адрес сети
```

Теперь вместо команды:

```
route add 128.17.75.20
```

можно использовать:

```
route add veggie-net
```

/etc/host.conf

Файл */etc/hosts.conf* указывает, как ваша система разрешает имена хостов. В нем должно быть две строки:

```
order hosts,bind
multi on
```

Эти строки указывают библиотекам распознавания при поиске любых имен сначала проверить файл */etc/hosts*, а затем (если адрес не найден) запросить сервер имен (в случае наличия такового). Запись `multi` позволяет указать несколько IP-адресов для одного имени хоста в */etc/hosts*.

В системах, использующих новую библиотеку *glibc2* (что относится к большинству новых дистрибутивов), вместо */etc/host.conf* используется */etc/nsswitch.conf*. В этом случае файл должен содержать строки: `hosts: files dns` и `networks: files dns`.

/etc/resolv.conf

Файл */etc/resolv.conf* конфигурирует распознаватель имен, указывая адрес вашего сервера имен (если таковой есть) и домены, которые подставляются по умолчанию, если указанное имя не является полным доменным именем. Например, если этот файл содержит строку:

```
search vpizza.com vpasta.com
```

то при использовании имени хоста *blurb* будет сделана попытка найти имена *blurb.vpizza.com* и *blurb.vpasta.com* (в этом порядке). Это удобно, т. к. не надо набирать полные имена часто используемых доменов. С другой стороны, чем больше доменов вы здесь укажете, тем дольше будет выполняться просмотр DNS.

Например, машина *eggplant.veggie.com* с сервером имен по адресу 128.17.75.55 будет иметь следующие строки в файле */etc/resolv.conf*:

```
domain      veggie.com
nameserver  128.17.75.55
```

Можно указать более одного сервера имен, для каждого из которых должна иметься строка `nameserver` в *resolv.conf*.

Установка имени хоста

Необходимо установить имя хоста для вашей системы (hostname) с помощью команды *hostname*. Обычно она вызывается из */etc/rc.d/rc.sysinit* (*/sbin/init.d/boot* в системах SuSE). Чтобы выяснить, откуда она вызывается, просто просмотрите свои системные *rc*-файлы. Например, если имя вашего хоста (полное) – *eggplant.veggie.com*, отредактируйте соответствующий *rc*-файл, чтобы он выполнял команду */bin/hostname eggplant.veggie.com*. Обратите внимание, что исполняемый файл *hostname* на вашей системе может оказаться не в */bin*, а в другом каталоге.

Проверяем сеть

После того как вы изменили все это множество файлов конфигурации, вы должны быть готовы перезагрузить систему (используя ядро с поддержкой TCP/IP) и попытаться использовать сеть.

При первой загрузке системы вы можете запретить выполнение *rc.inet1* и *rc.inet2* и запустить их вручную после загрузки. Это позволит отслеживать сообщения об ошибках, изменять сценарии и делать повторные попытки. Когда вы будете удовлетворены устойчивой работой этих настроек, можно разрешить выполнение сценариев из */etc/inittab*.

Хорошим способом проверки сетевого соединения является простое *ssh*-подключение к другому хосту. Сначала попытайтесь подключиться к хосту в вашей локальной сети, и если это удастся, попробуйте соединения с хостами из других сетей. В первом случае проверяется соединение с локальной подсетью, а во втором – с остальным миром через ваш маршрутизатор.

Может оказаться, что вы подключаетесь к удаленным хостам через шлюз, в то время как соединиться с локальными машинами не удается. Это означает ошибку в маске подсети или записи для локальной сети в таблице маршрутизации.

При проверке соединения с другими машинами сначала попытайтесь использовать только IP-адрес удаленного хоста. Если это удалось, а соединение по имени хоста не устанавливается, возможно, что-то не так с конфигурацией вашего сервера имен (например, */etc/resolv.conf* и */etc/host.conf*) или с маршрутом к серверу имен.

Самым частым источником проблем с сетью является неправильно сконфигурированная таблица маршрутизации. Просмотреть таблицу маршрутизации можно с помощью команды

```
netstat -rn
```

В предыдущем разделе мы рассмотрели формат таблицы маршрутизации, выводимой этой командой. Просмотр страниц руководства для *netstat*(8) будет также полезным. Команда *netstat* без параметра *-n* показывает имена хостов и сетей вместо их IP-адресов.

Для отладки таблиц маршрутизации можно либо редактировать *rc.inet1* и затем перегружаться, либо вручную добавлять и удалять записи, используя команду *route*. Полный синтаксис команды *route* вы найдете на странице руководства *route*(8). Обратите внимание, что просто отредактировав и перезапустив *rc.inet1*, вы не сотрете старые записи в таблице маршрутизации; необходимо либо перезагрузить систему, либо использовать *route del* для удаления этих записей.

Если вам кажется, что совсем ничего не работает, возможно, есть проблемы с настройкой вашего устройства Ethernet. Для начала проверьте, была ли обнаружена ваша сетевая карта по соответствующему адресу и/или IRQ во время загрузки. Эту информацию можно получить из сообщений при загрузке ядра, а если вы используете *syslogd*, загрузочные сообщения ядра сохраняются также в файле, например в */var/log/messages*.

Хороший способ определить, действительно ли проблемы вызваны картой Ethernet – воспользоваться командой *ifconfig имя_интерфейса*, например:

```
ifconfig eth0
```

В результате будут показаны статистические сведения об интерфейсе. Если видно, что интерфейс принимал и посылал пакеты, значит, ядро его видит и существенной проблемы с аппаратурой нет. Если после выполнения команды

```
ifconfig
```

вы не видите своей карты в списке, значит, ядро даже не увидело ее.

Если сетевая карта обнаруживается неправильно, может потребоваться изменить параметры ядра и исправить этот недостаток. Много информации по настройке сетевых карт содержится в Linux Ethernet HOWTO. В большинстве случаев исправить ошибку просто. Для этого достаточно установить соот-

ветствующее прерывание и адрес порта в загрузочной строке LILO. Например, при загрузке через LILO с командой:

```
lilo: linux ether=9,0x300,0,1,eth0
```

будут выбраны IRQ 9, базовый адрес 0×300 и внешний трансивер (четвертое значение, установленное в 1) для устройства *eth0*. Если вы используете внутренний трансивер (когда карта поддерживает оба типа), измените четвертое значение параметра `ether` на 0.

Кроме того, не стоит недооценивать возможность того, что сетевая карта неисправна или неправильно подсоединена к машине или к сети. Плохая сетевая карта или кабель могут быть причиной бесконечных неприятностей, включая неустойчивую работу сети, отказы системы и т. д. Когда вы перепробовали уже все, замените сетевую карту и/или кабель и выясните, не это ли было источником проблемы.[1]

Если ваша сетевая карта распознается, но в системе по-прежнему имеются проблемы, связанные с сетью, причиной этого может быть неверная конфигурация устройства с помощью *ifconfig*. Убедитесь, что вы указали правильный IP-адрес, широковещательный адрес и маску подсети. Вызов *ifconfig* без аргументов выдает информацию по конфигурации вашего устройства Ethernet.

Коммутируемые линии и протокол PPP

Для связи по TCP/IP с помощью модема (например, через подключение по коммутируемому каналу к провайдеру Интернета) или с помощью другого последовательного устройства (такого как «нуль-модемный» кабель между двумя машинами) Linux имеет программную поддержку протокола PPP. PPP – это протокол, который преобразует пакеты, передаваемые по сети (например, TCP/IP), в формат, легко передаваемый через модем или последовательный кабель. Скорее всего, сервер вашего провайдера Интернета использует PPP для обслуживания подключений по коммутируемым линиям. Сконфигурировав PPP под Linux, вы сможете напрямую соединиться с провайдером.

SLIP (Serial Line Internet Protocol) – это более старый протокол, в котором есть все основные функции, выполняемые PPP. Однако ему не хватает некоторых важных возможностей, таких как управление IP-адресами и размерами пакета. В наши дни SLIP практически полностью вытеснен PPP. Однако некоторые провайдеры все еще могут использовать SLIP вместо PPP. В этом случае отсылаем вас к другим источникам информации, таким как «Linux Network Administrator's Guide».

---

[1] Однажды один из авторов три часа подряд пытался выяснить, почему ядро не распознает сетевую карту. Как оказалось, 16-битная карта была вставлена в 8-битный разъем – «сам виноват!».

В этом разделе мы рассмотрим настройку PPP-*клиента*, т. е. системы, которая подключается к провайдеру Интернета (или к другому PPP-серверу) для того, чтобы выйти во внешний мир. Можно настроить Linux-машину в качестве PPP-сервера, но это более сложная задача, о которой рассказывается в «Linux Network Administrator's Guide».

Основные настройки PPP для модемов

В большинстве частей света люди используют обычные модемы с дозвонкой для передачи цифровых данных по коммутируемым телефонным линиям. Поэтому в первую очередь мы рассмотрим конфигурацию для модемов. Затем покажем, как сконфигурировать PPP для более быстрого и удобного типа линий, называемого ISDN (Integrated Services Digital Network – цифровая сеть с предоставлением комплексных услуг) и весьма популярного в Европе, а также имеющегося в США, но недостаточно там рекламируемого.

Требования

Большинство систем Linux поставляются с предустановленным программным обеспечением, необходимым для работы с PPP. Главное, что вам нужно, – это ядро, скомпилированное с поддержкой PPP, демон pppd и различные средства, имеющие отношение к PPP, включая программу *chat*.

В большинстве дистрибутивов Linux поддержка PPP включена в предварительно скомпилированное ядро или в модуль ядра, подключаемый по требованию. Однако может оказаться необходимым самостоятельно скомпилировать поддержку PPP. Для этого требуется лишь включить относящиеся к PPP опции при конфигурировании ядра и перекомпилировать его. Обычно PPP компилируется в отдельный модуль, поэтому в этом случае достаточно скомпилировать только модули ядра. Информацию по компилированию ядра и модулей см. в разделе «Сборка нового ядра» главы 7.

Утилиты *pppd* и *chat* являются приложениями пользовательского уровня, контролирующими использование PPP на вашей системе. Они входят практически во все дистрибутивы Linux. В системах Red Hat эти утилиты устанавливаются в */usr/sbin* и находятся в RPM-пакете *ppp*.

Кроме того, для использования PPP необходим модем, совместимый с Linux и типом модемов, используемых на сервере вашего провайдера. В эту категорию попадает большинство стандартных модемов 14.4, 28.8, 56K и другие. Есть очень немного типов модемов, не поддерживаемых Linux, а что касается провайдеров, то они обычно не используют ничего настолько эзотерического, что заставило бы вас купить особый модем.

Единственный тип модемов, которого надо остерегаться, – это так называемый «Winmodem». Первоначально этот модем продавался US Robotics, но теперь его выпускают и другие производители. Winmodem использует центральный процессор для преобразования цифровых сигналов в аналоговые, чтобы их можно было посылать по обычным телефонным линиям, в отличие от обычных модемов, где есть специальный чип, который выполняет эту

функцию. Проблема с Winmodem заключается в том, что информация о программировании этих устройств закрыта их изготовителями, а потому написание драйверов под Linux для этого класса устройств затруднительно. (Кроме того, многие издеваются над идеей тратить драгоценные циклы CPU на производство модемных сигналов, т. е. на работу, которую лучше оставить специализированному оборудованию. С другой стороны, обновление так называемых «программных модемов» состоит в замене драйвера операционной системы, который управляет их работой. Это существенное преимущество по сравнению с альтернативой покупки нового оборудования.)

Имена последовательных устройств

Под Windows 95/98/ME и MS-DOS модемы и другие последовательные устройства называются COM1 (первое устройство), COM2 (второе) и так далее до COM4. (Большинство систем поддерживает до четырех последовательных устройств, но существуют многопортовые карты, которые могут увеличить это число.) Под Linux эти же устройства называются */dev/ttyS0*, */dev/ttyS1* и так далее до */dev/ttyS3*.[1] В большинстве систем во время установки создается символическая ссылка */dev/modem*, которая указывает на последовательное устройство, где находится модем, как это показано в следующем листинге:

```
% ls -l /dev/modem
lrwxrwxrwx  1 root   root   10 May 4 12:41 /dev/modem -> /dev/ttyS0
```

Если для вашей системы эта ссылка не подходит (допустим, потому, что вы знаете, что ваш модем находится не в */dev/ttyS0*, а в */dev/ ttyS2*), вы легко можете изменить ее, зарегистрировавшись как `root` и введя:

```
# ln -sf /dev/ttyS2 /dev/modem
```

Установка PPP

Настройка PPP состоит из нескольких шагов. Первый из них – это написание так называемого «сценария chat», осуществляющего обмен информацией (так называемый «handshaking»), которая необходима для установки PPP-соединения между вашей машиной и провайдером. В процессе «handshaking» передается различная информация, такая как регистрационное имя и пароль. Второй шаг – это написание сценария, который запускает демон *pppd*. Этот сценарий заставляет модем звонить провайдеру и запускает PPP. На последнем шаге редактируется системный файл */etc/resolv.conf* для того, чтобы машина знала, где находится сервер имен доменов (DNS). Мы последовательно рассмотрим эти шаги.

Прежде всего, необходимо узнать следующую информацию:

- Номер дозвона до ISP (провайдера интернет-услуг).

---

[1] Старые версии Linux использовали также специальные устройства «callout», к которым обращались по именам от */dev/cua0* до */dev/cua3*. Начиная с ядра версии 2.2, они больше не используются.

- Регистрационное имя и пароль для связи с провайдером.
- IP-адрес DNS вашего провайдера.

Эта информация предоставляется провайдером на этапе открытия счета.

Кроме того, может понадобиться следующая информация:

- IP-адрес сервера провайдера.
- IP-адрес вашей системы (если он не выделяется провайдером динамически).
- Маска подсети, которую вам следует использовать.

Последние три пункта обычно автоматически определяются при установке PPP-соединения, однако иногда этого не происходит. Так что вам не повредит иметь эти данные на случай, если они понадобятся.

Создание сценария для chat

chat – это программа, которая осуществляет обмен некоторой информацией (например, передачу регистрационного имени и пароля) между клиентом и сервером PPP при установке соединения. Кроме того, *chat* отвечает за набор номера провайдера модемом и некоторые другие простые задачи.

Программа *chat* автоматически вызывается *pppd* при запуске (мы обсудим это позже). Вам необходимо лишь написать простой сценарий оболочки, который вызывает *chat* для управления подключением. Следующий пример является простым сценарием *chat*. Отредактируйте (как `root`) файл */etc/ppp/my-chat-script* и введите туда следующие строки:

```
#!/bin/sh
# my-chat-script: программа дозвона до провайдера
exec chat -v          \
     `` ATZ           \
     OK ATDT555-1212  \
     CONNECT ``       \
     ogin: mdw        \
     assword: my-password
```

Задание ogin и assword (без начальных букв) позволяет ждать приглашения, начинающиеся как с прописных, так и строчных букв.

Убедитесь, что файл *my-chat-script* является выполняемым. Сделать его выполняемым можно командой `chmod 755 /etc/ppp/my-chat-script`.

Обратите внимание на то, что если строка заканчивается обратной косой чертой, то после черты не должно быть никаких символов. Обратная косая черта в сценариях оболочки означает продолжение на следующей строке.

Третья строка сценария запускает сам *chat* с параметрами, указанными в следующих строках. Каждая строка содержит два поля, разделенные пробелами, – «ожидаемая» строка и «посылаемая» строка. Идея состоит в том, что сценарий *chat* должен отвечать «посылаемой» строкой на получаемую от модемного соединения «ожидаемую» строку. Например, последняя

строка сценария информирует *chat* о том, что ему следует ответить строкой `my-password` при запросе `assword`, полученном от сервера провайдера.

Первая строка сценария дает указание программе *chat* послать модему команду `ATZ`, что должно вызвать повторную инициализацию модема. (В «ожидаемой» строке стоит «`''`». Это означает, что до команды `ATZ` ничего не ожидается.) Вторая строка ожидает от модема ответ `OK`, после чего набирается номер с использованием строки `ATDT555-1212`. (Если ваша АТС использует импульсный набор номера, измените эту строку на `ATDP555-1212`. Номер телефона, конечно, нужно указать тот, где находится модемный пул провайдера.)

Когда модем отвечает сигналом `CONNECT`, посылается перевод строки (указанный как `''`). После этого *chat* ожидает запрос `ogin:`, прежде чем послать регистрационное имя, а затем ожидает строку `assword:` перед отправкой пароля.

Строки рассматриваемого примера, начинающиеся с `AT`, являются строками управления (AT-командами) для стандартных модемов типа Hayes. В руководстве к вашему модему должно объясняться их использование; оно не зависит от операционной системы (Linux или другой). Например, запятая, поставленная в набираемом номере, заставит модем сделать паузу перед набором остальных цифр. Если для доступа к внешней линии надо набрать специальную цифру (в данном примере 9), можно послать модему строку `ATDT9,,,555-1212`.

Обратите внимание, что это очень простой сценарий *chat*, не обрабатывающий тайм-ауты, ошибки или другие неординарные ситуации, которые могут возникнуть при попытке дозвониться до провайдера. Поинтересуйтесь в страницах руководства для *chat*, как учесть такие ситуации в вашем сценарии. Также нужно заранее выяснить, какие строки запросов использует ваш провайдер (мы предполагаем, что это `login` и `password`). Существует несколько способов выяснить это. Возможно, эту информацию вам заранее даст провайдер, либо предоставит сценарий для входа для другой системы, такой как Windows 95 (которая использует механизм, очень схожий с *chat*). В противном случае вы можете вручную дозвониться до сервера провайдера, используя простой эмулятор терминала, такой как *minicom* или *seyon*. Страницы руководства для этих команд помогут вам в этом.

Запуск демона pppd

Теперь мы готовы настроить демон *pppd* для инициирования PPP-соединения с помощью сценария *chat*, который только что написали. Обычно для этого пишут еще один сценарий оболочки, который вызывает *pppd* с набором параметров.

Команда *pppd* имеет следующий формат:

```
pppd device-name baudrate options
```

В табл. 15.1 представлены параметры, поддерживаемые *pppd*. Некоторые из них, скорее всего, вам никогда не понадобятся.

Таблица 15.1. Основные параметры pppd

Параметр	Эффект
lock	Блокирует последовательное устройство, запрещая доступ к *pppd*
crtscts	Использует аппаратный контроль потока
noipdefault	Не пытается выяснить локальный IP-адрес. IP-адрес назначается удаленной системой
user *username*	Указывает hostname или username для идентификации по методу PAP или CHAP
netmask *mask*	Указывает маску подсети для подключения
defaultroute	Добавляет маршрут по умолчанию в таблицу маршрутизации локальной системы, используя IP-адрес удаленной системы в качестве шлюза
connect *command*	Использует заданную команду *command* для инициализации соединения. *pppd* предполагает, что этот сценарий находится в */etc/ppp*. Если это не так, следует указать полный путь к сценарию
local_IP_address: remote_IP_address	Указывает локальный и/или удаленный IP-адрес. Один или оба могут быть указаны как 0.0.0.0, тогда адрес будет назначен удаленной системой
debug	Протоколирует информацию о соединении через демон *syslog*

Команда *pppd* обычно вызывается из сценария на языке оболочки. Отредактируйте файл */etc/ppp/ppp-on*, добавив следующие строки:

```
#!/bin/sh
# the ppp-on script

exec /usr/sbin/pppd /dev/modem 38400 lock crtscts noipdefault \
     defaultroute 0.0.0.0:0.0.0.0 connect my-chat-script
```

Как и в предыдущем примере с файлом *my-chat-script*, следует убедиться, что файл является исполняемым, и остерегаться символов после обратной косой черты в конце строки.

Когда этот сценарий будет готов, вы сможете подключиться к провайдеру, используя команду:

```
% /etc/ppp/ppp-on
```

Для выполнения этой команды не нужны привилегии суперпользователя. При запуске сценария вы должны услышать, как модем набирает номер, и если все в порядке, через минуту PPP-соединение будет успешно установлено. Если PPP работает, команда *ifconfig* должна выдать запись для ppp0:

```
# ifconfig
lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Bcast:127.255.255.255  Mask:255.0.0.0
          UP BROADCAST LOOPBACK RUNNING  MTU:3584  Metric:1
```

```
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0

ppp0      Link encap:Point-to-Point Protocol
          inet addr:207.25.97.248  P-t-P:207.25.97.154  Mask:255.255.255.0
          UP POINTOPOINT RUNNING  MTU:1500  Metric:1
          RX packets:1862 errors:0 dropped:0 overruns:0 frame:0
          TX packets:1288 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0
          Memory:73038-73c04
```

Мы видим, что PPP работает, локальный IP-адрес – 207.25.97.248, а IP-адрес удаленного сервера – 207.25.97.154.

При желании иметь уведомление об установлении PPP-соединения (сценарий *ppp-on* возвращается немедленно) добавьте следующую строку в */etc/ppp/ip-up*:

```
/usr/bin/wall "PPP is up!"
```

/etc/ppp/ip-up выполняется, когда PPP устанавливает IP-соединение, поэтому можно использовать этот сценарий для вызова команды *wall* после подключения.

Другой несложный сценарий оболочки можно использовать для завершения PPP-сессии. Отредактируйте файл */etc/ppp/ppp-off* следующим образом:

```
#!/bin/sh
# Простой сценарий ppp-off

kill `cat /var/run/ppp0.pid`
```

Запуск */etc/ppp/ppp-off* завершит работу демона PPP и закроет модемное соединение.

Настройка DNS

Само по себе использование *pppd* вместе с *chat* лишь устанавливает PPP-соединение и назначает вам IP-адрес. Для использования доменных имен нужно сообщить системе IP-адрес сервера имен, установленного у вашего провайдера. Это можно сделать, отредактировав файл */etc/resolv.conf*, который в деталях описан на страницах руководства для *resolver*. Однако для большинства задач достаточно прописать строки двух типов. Один из них служит для указания списка доменов, в котором ведется поиск всякий раз, когда используется доменное имя, а другой тип строк содержит адреса серверов DNS.

Пример файла */etc/resolv.conf* может выглядеть так:

```
# Пример /etc/resolv.conf
search cs.nowhere.edu nowhere.edu
nameserver 207.25.97.8
nameserver 204.148.41.1
```

Первая строка указывает на то, что каждый раз при использовании доменного имени (такого как *orange* или *papaya*) его следует искать в списке указанных доменов. В этом случае программа, отвечающая за распознавание имен хостов, будет сначала расширять имена типа *papaya* до *papaya.cs.nowhere.edu* и пытаться найти систему по этому имени, а затем, при необходимости, пробовать вариант *papaya.nowhere.edu* и пытаться найти систему снова.

Строки, начинающиеся с `nameserver`, указывают IP-адреса серверов DNS (они должны быть предоставлены провайдером), которые используются вашей системой для распознавания доменных имен. Если вы укажете более одной строки `nameserver`, указанные сервера будут вызываться по очереди до тех пор, пока один из них не выдаст результат. В этом случае один DNS-сервер считается первичным, а остальные запасными.

Ошибки при настройке PPP

Настройка PPP, описанная здесь, является очень простой и, скорее всего, не подойдет на все случаи жизни. Лучшим источником дополнительной информации являются страницы руководства для *pppd* и *chat*, а также Linux PPP HOWTO и относящиеся к ним документы.

К счастью, и *chat* и *pppd* протоколируют свою работу, а также все возникающие ошибки, используя стандартные возможности демона *syslog*. Отредактировав */etc/syslog.conf*, вы можете указать файл, куда эти программы будут записывать сообщения. Для этого добавьте следующие строки:

```
# Сохранять сообщения от chat
local2.*                                        /var/log/chat-log

# Сохранять сообщения от pppd
daemon.*                                        /var/log/pppd-log
```

Теперь сообщения от *chat* будут протоколироваться в */var/log/chat-log*, а сообщения от *pppd* – в */var/log/pppd-log*.

Обратите внимание на то, что эти записи будут содержать конфиденциальную информацию, такую как используемые при подключении к провайдеру регистрационные имена и пароли! Следует разрешать такую запись сообщений только на время отладки конфигурации PPP. Когда же все будет работать, удалите оба файла, а также удалите введенные строки из */etc/syslog.conf*.

chat сохраняет некоторые сообщения об ошибках и в файле */etc/ppp/connect-errors*, который не управляется демоном *syslog*. (Этот файл можно спокойно оставить.)

PAP и CHAP

Некоторые провайдеры требуют использовать специальные протоколы аутентификации, такие как PAP (Password Authentication Protocol) или CHAP (Challenge Handshake Authentication Protocol). Эти протоколы основаны на некоторых «общих секретах», известных и клиенту, и серверу. В большинстве случаев это просто пароль на доступ к провайдеру.

Если провайдер требует использовать PAP или CHAP, то настроить эти протоколы можно, добавив некоторые данные в файлы */etc/ppp/pap-secrets* или */etc/ppp/chap-secrets*, соответственно. В каждом файле есть четыре поля, разделенных пробелами или табуляцией. Вот пример файла *pap-secrets*:

```
# Secrets for authentication using PAP
# client      server       secret           IP or Domain
mdw          *             my-password
```

Первое поле содержит имя вашей системы, ожидаемое удаленным сервером. Обычно это ваше регистрационное имя для доступа к провайдеру. Второе поле указывает имя сервера провайдера; звездочка «*» позволяет использовать эту запись для всех серверов, с которыми вы хотите связаться. Третье поле содержит «общий секрет», предоставленный провайдером. Как указывалось ранее, обычно это ваш пароль. Четвертое поле используется провайдерами для того, чтобы ограничить IP-адреса или имена доменов, к которым могут иметь доступ подключающиеся по модему пользователи. Но для большинства конфигураций PPP-клиентов это поле не требуется.

Файл *chap-secrets* содержит те же четыре поля, но в нем нельзя использовать «*» вместо имени сервера провайдера. Это секрет, который провайдер передает вам при открытии у него бюджета.

Если используются PAP или CHAP, не обязательно включать в сценарий *chat* команды для обмена паролями и регистрационными именами после получения сообщения `CONNECT` – обо всем дальнейшем позаботится *pppd*. Поэтому вы можете отредактировать */etc/ppp/my-chat-script* так, чтобы в нем были только строки:

```
#!/bin/sh
# my-chat-script: программа дозвона до провайдера
exec chat -v            \
     '' ATZ             \
     OK ATDP555-1212    \
     CONNECT ''
```

Кроме того, нужно добавить параметр `user` в строку вызова *pppd* в */etc/ ppp/ ppp-on*:

```
#!/bin/sh
# the ppp-on script

exec /usr/sbin/pppd /dev/modem 38400 lock crtscts noipdefault \
     user mdw defaultroute 0.0.0.0:0.0.0.0 connect my-chat-script
```

PPP поверх ISDN

Уже многие годы ISDN предлагает удобную, высокоскоростную цифровую связь; она особенно популярна в Европе, где цены и маркетинговая политика сделали ее более привлекательной, чем в США. ISDN, где в одной линии интегрированы передача данных и голоса, предлагает более быструю установку соединения и значительно более высокие скорости передачи данных по сравнению с традиционными модемами.

Линии ISDN могут передавать данные со скоростью 64 Кбит/с. В отличие от аналоговых линий, эта скорость постоянна, т. к. не зависит от капризов аналоговой передачи, подверженной влиянию разнообразных помех. Новый протокол, названный ADSL (Asymmetric Digital Subscriber Line), поднимает планку скорости доступа по телефонным линиям, но ISDN по-прежнему сохраняет большую долю рынка.

В этом разделе мы опишем, как настроить доступ к провайдеру по линии ISDN. Мы рассмотрим только самый обычный способ соединения – «синхронный PPP», а не особый режим, называемый *Raw IP* по ISDN. Более того, в этом разделе обсуждаются только внутренние платы ISDN, которые настраиваются не так, как соединения по телефонным линиям, рассматривавшиеся в предыдущем разделе. Для установки внешних устройств ISDN, или так называемых ISDN-модемов (термин, конечно, является оксюмороном, т. к. нет модуляции-демодуляции), можно пользоваться командами, подобными тем, которые были описаны в предыдущем разделе, поскольку такое устройство выглядит для операционной системы как обычный модем, но предлагающий некоторые дополнительные команды, быструю установку соединения и высокую скорость.

Возможно, вам потребуется дополнительная информация помимо той, которую мы здесь представили. Все информация, касающаяся ISDN для Linux есть на *http://www.isdn4linux.de* (хотя домен зарегистрирован в Германии, вся информация там на английском языке).

В некотором смысле установка ISDN-соединения значительно проще установки соединения по аналоговым линиям, т. к. у цифровых линий просто отсутствуют некоторые проблемы (плохие линии связи, длительное время соединения и т. д.). После набора номера соединение устанавливается в течение миллисекунд. Но это может вызвать другие проблемы. Поскольку соединение устанавливается и закрывается так быстро, неправильно сконфигурированная система будет звонить снова и снова, что может дорого обойтись. Это особенно опасно с внутренними адаптерами ISDN, т. к. вы не услышите щелчков и прочего шума, как у обычных модемов, и нет никаких индикаторов, показывающих состояние соединения. Однако статус линии ISDN можно проверить с помощью простых программ.

Настройка коммутируемого PPP-соединения по ISDN выполняется в два шага:

1. Конфигурирование оборудования ISDN.
2. Настройка и запуск демона PPP и модификация таблицы маршрутизации для использования линии ISDN.

Мы расскажем об этих шагах в следующих разделах.

Конфигурирование оборудования ISDN

Сначала надо предоставить ядру доступ к карте ISDN. Как и для любого другого оборудования, для вашей платы потребуется драйвер устройства, сконфигурированный с правильными параметрами.

Linux поддерживает большое число плат ISDN. Мы не можем рассказать о каждой из них, но процедура настройки для всех карт весьма похожа. Если ваша плата здесь не рассматривается, документацию к вашей карте можно найти в каталоге *Documentation/isdn* в исходных текстах ядра Linux.

Мы будем рассматривать платы, использующие так называемый драйвер *HiSax*. Этот драйвер работает почти со всеми картами, которые построены на чипсете HSCX от Siemens, а это большинство имеющихся на сегодня в продаже пассивных карт. Сюда входят, в частности, USR Sportster internal TA и хорошо известные платы Teles, ELSA и Fritz. Другие платы используют схожие настройки. Linux поддерживает даже некоторые активные карты, включая широко известные AVM B1 и IBM Active 2000 ISDN.

Первым делом нужно включить в ядро поддержку ISDN. Рекомендуется скомпилировать все, связанное с ISDN, в виде модулей, особенно на период экспериментов с настройкой. Вам потребуются следующие модули:

- Поддержка ISDN.
- Поддержка синхронного PPP.
- Один модуль драйвера устройства для вашего оборудования. Если вы используете драйвер HiSax, необходимо указать, какую марку платы ISDN вы приобрели, и какой протокол ISDN хотите использовать. Практически всегда используется протокол EURO/DSS1, но если вы живете в Германии и линия ISDN у вас уже давно, протокол может называться 1TR6, а в США – US NI1. Если сомневаетесь, спросите вашу телефонную компанию.

Откомпилируйте и установите модули, как описано в главе 7. Теперь вы готовы к настройке оборудования ISDN. В некоторых дистрибутивах, таких как SuSE, настройка линии ISDN отличается простотой и удобством. Мы же пойдем другим путем, предположив, что ваш дистрибутив не столь дружелюбен к пользователям, не работает автоматическая конфигурация или вы просто хотите знать, что происходит за кулисами.

Теперь необходимо загрузить модуль драйвера устройства, используя *modprobe*. Это автоматически загрузит все остальные модули. Все драйверы устройств используют некоторое число параметров. Модуль `hisax` среди прочих принимает следующие параметры:

`id=`*`boardid`*

Устанавливает идентификатор для платы ISDN. Вы можете выбрать любое имя, но оно не должно совпадать с идентификаторами других устройств.

`type=`*`boardtype`*

Указывает точную марку и тип платы. Например, значение 16 для *`boardtype`* выбирает поддержку для USR Sportster internal TA. Полный список типов плат вы найдете в *Documentation/isdn/README.hisax*.

`protocol=`*`protocoltype`*

Выбирает подпротокол ISDN. Возможными значениями для *`protocoltype`* могут быть: 1 – для старого немецкого протокола `1TR6`, 2 – для обычного `EDSS1` (так называемого Euro ISDN) и 3 – для выделенных линий.

`irq=`*`irqno`*

Указывает используемое прерывание. Не все платы требуют этот параметр.

`io=`*`addr`*

Указывает используемый адрес I/O. Требуется не всеми платами. Некоторым платам требуется два адреса I/O. В этом случае используются параметры `io0` и `io1`.

Например, следующая команда загрузит драйвер HiSax, который будет использоваться с платой Teles 16.3, протоколом Euro ISDN, адресом I/O, равным 0×280, и прерыванием IRQ, равным 10 (весьма часто встречающийся случай):

```
tigger # modprobe hisax type=3 protocol=2 io=0x280 irq=10
```

За дополнительной информацией по вашему оборудованию обращайтесь к *Documentation/isdn/README.HiSax* или другому эквивалентному файлу.

Этот модуль не очень разговорчив. Если команда *modprobe* ничего не выдала, возможно, все прошло успешно. Можно также проверить системный журнал в */var/log/messages*. Там должно быть несколько строк, начинающихся с `HiSax:` (или с имени используемого драйвера) и заканчивающихся строкой:

```
HiSax: module installed
```

Если модуль не загрузился, причину, скорее всего, можно также найти в */var/log/messages*. Чаще всего проблемы возникают из-за неправильно указанных адреса I/O или прерывания IRQ, либо неправильно выбранного типа карты. Если у вас на этой же машине установлена Windows, загрузите эту ОС и посмотрите, какие она выставляет IRQ и адрес I/O.

Прежде чем переходить к следующему разделу, можно выполнить еще одну проверку – позвонить самому себе. Это возможно, т. к. благодаря ISDN в вашем распоряжении всегда две телефонные линии. Поэтому одна из них может использоваться для исходящего звонка, а вторая – для входящего.

Для того чтобы ISDN-подсистема сообщала о том, что происходит с телефонными линиями, надо сделать ее более разговорчивой, чем это предусмотрено по умолчанию. Это можно сделать средствами трех утилит, входящих в пакет isdn4k-utils, который вы легко сможете найти на любимом FTP-сервере, посвященном Linux.

В пакете isdn4k-utils среди прочего есть три утилиты: *hisaxctrl* предназначена для настройки драйвера устройства, *isdnctrl* – для настройки высокого уровня подсистемы ISDN, и *isdnlog* – очень полезное средство, протоколирующее все, что происходит с вашими линиями ISDN. В то время как *hisactrl* и *isdnctrl* можно использовать без каких-либо настроек, для *isdnlog* требуется небольшой файл настройки. На данный момент мы удовлетворимся быстрым решением, но когда ваше ISDN-соединение будет работать, вы, возможно, захотите настроить *isdnlog* так, чтобы видеть, куда уходят деньги.

А сейчас скопируйте в */etc/isdn/isdn.conf* один из примеров конфигурационных файлов, расположенных в пакете *isdnlog*. Вам потребуется изменить, по крайней мере, следующие строки:

COUNTRYCODE=

Добавьте сюда телефонный код вашей страны, например, 1 для Соединенных Штатов и Канады, 44 для Великобритании, 46 для Швеции и т. д.

AREAPREFIX=

Если код области в вашей стране имеет фиксированный префикс, поместите его сюда. Для большинства европейских стран префикс равен 0, для Финляндии – 9, а для Соединенных Штатов, Дании и Норвегии он отсутствует.

AREACODE=

Введите сюда код области. Если вы указали AREAPREFIX в предыдущем пункте, не вводите его сюда. Например, Стокгольм (столица Швеции) имеет код области 08. В этом случае вы вводите 0 в AREAPREFIX и 8 в AREACODE.

Когда вы это сделаете, выполните следующие команды, чтобы ваша система ISDN стала более разговорчивой:

```
tigger # /sbin/hisaxctrl boardid 1 4
tigger # /sbin/isdnctrl verbose 3
tigger # /sbin/lsdnlog /dev/isdnctrl0 &
```

Если вы используете драйвер, отличный от HiSax, может потребоваться другая команда. Например, для драйвера PCBit в пакете isdn4k-utils есть команда *pcbitctl*.

Теперь можно позвонить самому себе. Попробуйте все ваши MSN (multiple subscriber numbers – множественные номера абонента, которые являются номерами вашего телефона ISDN), для того чтобы посмотреть, может ли плата обнаружить их все. В течение или после звонка проверьте */var/log/messages*. Там должны быть строки следующего вида:

```
Mar 16 18:34:22 tigger kernel: isdn_net: call from 4107123455,1,0 -> 123456
Mar 16 18:34:33 tigger kernel: isdn_net: Service-Indicator not 7, ignored
```

Это говорит о том, что ядро обнаружило голосовой звонок (индикатор сервиса равен 0) на номер MSN 123456 с телефона 123455 и с кодом области (0)4107.

Обратите внимание на то, как указан набранный номер, т. к. эта информация понадобится в дальнейшем. В одних сетях номер посылается с кодом области, а в других без него. В любом случае поздравляем вас с тем, что ваше оборудование ISDN теперь настроено правильно.

Настройка синхронного PPP

Настройка демона PPP вновь состоит из нескольких шагов. В Linux плата ISDN рассматривается как сетевой интерфейс, который нужно настраивать особыми командами. Кроме того, необходимо указать регистрационное имя

и пароль, выданные провайдером. По окончании настройки запускается демон *ipppd*, который до запроса на установку соединения скрывается в потемках фонового режима.

Сначала настроим «сетевой интерфейс». Для этого требуется несколько команд, которые системные администраторы обычно помещают в сценарий, хранящийся, например, в файле */sbin/pppon*. Вот пример файла, который можно откорректировать для ваших нужд:

```
/sbin/isdnctrl addif ippp0
/sbin/isdnctrl addphone ippp0 out 0123456789
/sbin/isdnctrl dialmax ippp0 2
/sbin/isdnctrl eaz ippp0 123456
/sbin/isdnctrl huptimeout ippp0 100
/sbin/isdnctrl l2_prot ippp0 hdlc
/sbin/isdnctrl l3_prot ippp0 trans
/sbin/isdnctrl encap ippp0 syncppp
/sbin/ifconfig ippp0 1.1.1.1 pointopoint 123.45.67.89 metric 1
```

Давайте рассмотрим каждую команду:

`isdnctrl addif ippp0`

Сообщает ядру, что будет использоваться новый ISDN-интерфейс с именем `ippp0`. Всегда используйте имена, начинающиеся с `ippp`.

`isdnctrl addphone ippp0 out 0123456789`

Сообщает интерфейсу ISDN, какой телефонный номер следует использовать. Это номер дозвона до провайдера. Если ранее вы использовали доступ по аналоговой линии, узнайте у провайдера, какой номер надо использовать теперь, т. к. номер для доступа по ISDN может быть другим.

`isdnctrl dialmax ippp0 2`

Указывает, сколько раз следует набирать номер при неудачном соединении, прежде чем прекратить попытки.

`isdnctrl eaz ippp0 123456`

Указывает один из ваших собственных номеров MSN. Это очень важно, т. к. без него работа будет невозможна. Если провайдер проверяет доступ по номеру, убедитесь, что вы указали тот MSN, который зарегистрировали у провайдера.

`isdnctrl huptimeout ippp0 100`

Указывает время в секундах (последнее число в этой команде), в течение которого линия может бездействовать, перед тем как ядро закроет соединение. Команда не обязательная, но она может сохранить много денег, если у вас нет неограниченного доступа. Если вы забудете закрыть соединение сами, ядро сделает это за вас.

`isdnctrl l2_prot ippp0 hdlc`

Указывает используемый протокол второго уровня. Возможны следующие значения: `hdlc`, `x75i`, `x75ui` и `x75bui`. Большинство провайдеров использует `hdlc`. Если есть сомнения, спросите у провайдера.

`isdnctrl l3_prot ippp0 trans`

Указывает используемый протокол третьего уровня. («`l`» в тексте параметра – это буква «L».) В настоящее время доступен только `trans`.

`isdnctrl encap ippp0 syncppp`

Указывает используемую инкапсуляцию. Есть несколько возможных значений, но если вы хотите использовать синхронный PPP (или этого требует провайдер), необходимо указать `syncppp`. Другим, не столь частым значением, является `rawip`. Но поскольку он обеспечивает очень слабые возможности аутентификации, лишь немногие провайдеры все еще используют его, несмотря на то что у него немного лучше производительность благодаря меньшей избыточности информации.

`ifconfig ippp0 1.1.1.1 pointopoint 123.45.67.89 metric 1`

Создает новый сетевой интерфейс. Если IP-адрес не присваивается автоматически (как это обычно делается при телефонных подключениях), вместо 1.1.1.1 необходимо указать ваш IP-адрес. Также надо изменить 123.45.67.89 на IP-адрес сервера провайдера.

При желании можно перевернуть всю настройку, написав сценарий, который закрывает интерфейсы и т. д. Например, в нем будет использоваться команда `isdnctrl delif`. Но строгой необходимости в таком сценарии нет, если только вы не хотите отключить все телефонные соединения на этапе выполнения.

Необходимо еще настроить сам демон *ipppd*, что делается в файле */etc/ ppp/ ioptions*. Кроме того, можно создать отдельные файлы конфигурации для каждого демона *ipppd*, но это необходимо только в случае, если вы хотите использовать разные соединения ISDN, т. е. если у вас есть несколько учетных записей для доступа по линии.

Вот пример файла *ioptions*, который является достаточно универсальным для работы с большинством провайдеров. Он не дает максимальной производительности, но зато весьма стабилен. Если вы хотите его оптимизировать, спросите вашего провайдера о возможных параметрах и прочтите страницы руководства для *ipppd*(8):

```
debug
/dev/ippp0
user yourusername
name yourusername
mru 1500
mtu 1500
ipcp-accept-local
ipcp-accept-remote
noipdefault
-vj -vjccomp -ac -pc -bsdcomp
defaultroute
```

Здесь необходимо изменить только две вещи, а именно: нужно заменить `yourusername` в третьей и четвертой строках на имя пользователя, предоставлен-

ное провайдером для связи с его системой. Мы не будем объяснять значение всех параметров; почитайте страницы руководства в случае сомнений.

Доступ по ISDN требует тех же параметров безопасности, что и модемное подключение. С инструкциями по настройке файла *pap-secrets* или *chap-secrets* можно познакомиться в разделе «PAP и CHAP» ранее в этой главе.

Теперь все собрано воедино. Сначала запустим демон *ipppd*:

```
tigger # /sbin/ipppd pidfile /var/run/ipppd.ippp0.pid file /etc/ppp/ioptions &
```

Демон *ipppd* будет теперь ожидать запрос на соединение. Так как мы еще не настроили автоматическое подключение, запустим его вручную с помощью следующей команды:

```
tigger # isdnctrl dial ippp0
```

Теперь следует проверить */var/log/messages*. Там должно быть много сообщений, начинающихся с *ipppd*. Последнее из них должно содержать слова `local IP address` и `remote IP address` вместе с IP-адресами. Если вы нашли эти сообщения, значит, все готово. Так как ранее был использован параметр `defaultroute`, ядро настроено на использование ISDN-соединения как маршрута по умолчанию, и поэтому теперь у вас есть доступ к огромному миру Интернета. Начните с проверки IP-адреса провайдера командой *ping*. При желании закрыть соединение введите:

```
tigger # isdnctrl hangup ippp0
```

А если оно не работает?

Если соединение не работает, несмотря на то что оборудование успешно распознано и вы выполнили все описанные настройки, на помощь снова придет */var/log/messages*. Весьма возможно, что там вы найдете причину ошибки.

Самая обычная ошибка – это неправильно указанное имя пользователя или пароль. Если у вас есть проблемы с аутентификацией, в файле журнала вы найдете строку вида:

```
PAP authentification failed
```

или:

```
CHAP authentification failed
```

Тщательно проверьте *chap-secrets* или *pap-secrets*. Провайдер по своим файлам журналов тоже может определить, где именно произошла ошибка аутентификации.

Конечно, возможен вариант, что ваш провайдер не поддерживает синхронный PPP, хотя в наши дни большинство провайдеров это делает. В этом случае за точными настройками обращайтесь к провайдеру.

Если соединение по-прежнему не работает, обратитесь за помощью к провайдеру. У хорошего провайдера должна быть служба поддержки, которая поможет подключить вашу Linux-систему. Если провайдер заявляет, что он

поддерживает только Windows, значит, пора сменить провайдера. Сейчас много провайдеров, нормально относящихся к Linux. Часто сам персонал службы поддержки использует Linux и может помочь вам, даже если официальная политика провайдера не предполагает поддержки Linux.

Если по какой-либо причине приходится работать с недружелюбным провайдером, попробуйте найти других клиентов этого же провайдера, использующих Linux. Настройка соединения в нестандартных случаях – это сложная игра с ключами и параметрами демона *ipppd* и подсистемы ISDN в ядре, и если кто-нибудь уже знает, как это делать, следует этим воспользоваться.

Что дальше?

Когда ISDN-соединение работает и есть доступ в Интернет, можно выполнить некоторые настройки для удобства или внести усовершенствования в работу соединения. Вот некоторые предложения:

- Заставьте *ipppd* автоматически набирать номер провайдера. Это можно сделать, установив маршрут по умолчанию на устройство *ippp0*:

  ```
  /sbin/route add default netmask 0.0.0.0 ippp0
  ```

  Теперь, когда ядру потребуется отправить IP-пакет на IP-адрес, для которого нет специального маршрута, он заставит демон *ipppd* установить соединение. Используйте эту возможность только в том случае, если вы установили также параметр `huptimeout` в подсистеме ISDN, иначе придется отдать много денег телефонной компании (если у вас нет неограниченного доступа).

  Такая конфигурация может быть опасной для вашего кошелька, т. к. есть программы, пытающиеся время от времени установить соединение с Интернетом (Netscape – один из первых кандидатов). Если вы все же используете это, чаще проверяйте состояние соединения (см. далее в этом разделе).

- Попробуйте воспользоваться утилитами мониторинга ISDN-соединения. Многие из таких инструментов входят в пакет *isdn4k-utils*, включая средства командной строки *imon* и *imontty*, а также средства с графическим интерфейсом X.

- Настройте *isdnlog* для протоколирования именно того, что вам нужно, и получайте с помощью *isdnrep* подробный отчет об использовании линии ISDN. Отчет включает не только звонки с компьютерной системы и на нее, но и звонки на другие устройства ISDN, такие как телефоны и факсы. Есть только одно предостережение: плата ISDN не может фиксировать исходящие телефонные соединения, сделанные с других устройств. Но этот сервис предоставляют многие телефонные компании, отправляя сигнал обратно для того, чтобы ваша система могла зарегистрировать номер. Этот сервис часто предоставляется бесплатно или за символическую плату. Поинтересуйтесь об этом у вашей телефонной компании.

- Для истинных искателей приключений: эксперимент с Multilink-PPP. Как известно, с ISDN у вас есть как минимум две линии. Если вам нужен

канал повышенной емкости, почему бы не использовать обе линии? Это то, что делает Multilink-PPP. Для его использования надо включить параметр `Support generic MP` во время конфигурирования ядра, а также посмотреть файлы *Documentation/isdn/README.syncppp* и *Documentation/isdn/syncppp.FAQ* в исходных текстах ядра, чтобы воспользоваться советами, как это лучше сделать. Конечно, эту возможность также должен поддерживать и ваш провайдер.

ADSL

Скорость в 64 Кбит/сек., предоставляемая ISDN, это прекрасно, но если вам нужен доступ к мультимедийным файлам, или вы просто интенсивно используете Интернет, то может потребоваться еще более широкополосный доступ. Для этого необязательно тянуть новые кабели в квартиру или офис – ADSL (Asynchronous Digital Subscriber Line) служит удобной альтернативой, дающей пропускную способность, в восемь раз превышающую ту, которую дает стандартный доступ по коммутируемой линии, и достигаемую на обычной телефонной линии. Недостаток ADSL в том, что до ближайшей телефонной станции должно быть не более 3-4 километров (2-3 мили), поэтому в сельской местности такой сервис недоступен. Существуют другие технологии, названия которых звучат очень похоже, например, SDSL. На физическом уровне они существенно отличаются, но настройка их для работы с Linux-машиной не должна отличаться от ADSL.

ADSL не требует дозвона: подключившись к своей учетной записи, вы соединены постоянно. Некоторые провайдеры периодически обрывают соединение (обычно раз в сутки), и тогда вы снова должны зарегистрироваться, чтобы получить доступ к сети.

Как уже говорилось, не существует каких-либо карт или драйверов ADSL. Что касается оборудования, то ADSL представляет собой обычное соединение Ethernet с теми же кабелями.

Способ подключения Linux-машины к линии ADSL определяется вашим провайдером. Некоторые провайдеры предоставляют все необходимое оборудование, например, ADSL-модем и ADSL-маршрутизатор, в аренду. Другие провайдеры требуют, чтобы вы купили оборудование – самостоятельно или у того же провайдера. Всю необходимую информацию предоставит вам сам провайдер.

Есть два способа подключения по ADSL – непосредственно к модему ADSL или к промежуточному маршрутизатору ADSL. Если у вас стоит маршрутизатор ADSL (со встроенным модемом ADSL или без него), вы просто подсоединяете к нему свой кабель Ethernet и можете работать. Если вы хотите подключить Linux-машину непосредственно к модему ADSL, то также соединяете компьютер с модемом при помощи кабеля Ethernet, но вам потребуется запустить на машине специальный протокол – PPPoE (PPP over Ethernet). Этот протокол поддерживается особым демоном с именем *pppoed*. Способ его установки зависит от вашего дистрибутива, где должна быть со-

ответствующая документация. Некоторые дистрибутивы дают возможность установить PPPoE в своей программе конфигурации.

Наконец, следует отметить, что существуют, хотя их немного, извращенные модемы ADSL, которые подключаются не через Ethernet, а кабелем USB. Технически эта идея слаба, и таких модемов нужно избегать, но если уж вы связались с одним из них, то дополнительные сведения и драйвер, осуществляющий PPPoE через USB (что следовало бы назвать PPP over Ethernet over USB!), можно найти на *http://www.prout.be/ECI/*.

Каким бы способом вы ни подключались к ADSL (с маршрутизатором ADSL или без него), необходимо задать правильный IP-адрес. Он может быть статическим (тогда вы должны найти его в информации, полученной от провайдера) или динамическим, назначаемым через протокол DHCP (Dynamic Host Communication Protocol), и тогда вы можете запросить динамический IP-адрес с помощью команды:

```
dhclient eth0
```

Конечно, если у вашей Ethernet-карты другое имя, нужно заменить им `eth0`. Вместо *dhclient* можно воспользоваться утилитой *pump*. Существует также демон DHCP с именем *dhcpcd*, который выполняется в фоновом режиме и при необходимости назначает динамические IP-адреса.

Наконец, многие провайдеры требуют, чтобы время от времени вы активизировали свою линию. Способ осуществления этого определяется вашим провайдером Интернета. Часто для этого требуется зайти на некоторый веб-сайт и ввести идентификационные данные, предоставленные вам провайдером. Как отмечалось, может потребоваться периодически повторять эту процедуру.

Настройка NFS и NIS

Установив TCP/IP, можно настроить машину на использование NFS (Network File System – сетевая файловая система) или NIS (Network Information Service – сетевой информационный сервис). NFS позволяет вашей системе напрямую совместно использовать файлы с некоторой сетью машин. Доступ к файлам в NFS прозрачен: вы можете использовать их так, как если бы они находились на локальном диске. В терминах сетевого администрирования это выглядит так: одна система монтирует удаленную файловую систему в локальный каталог так же, как может быть монтирована локальная файловая система. NFS также позволяет вам экспортировать файловые системы, давая возможность другим машинам в сети напрямую монтировать ваши диски.

NIS (ранее известная как Yellow Pages, или YP) – это система, позволяющая вашему хосту автоматически получать от сетевых серверов информацию об учетных записях пользователей, групп, точках монтирования файловой системы и других системных базах данных. Допустим, к примеру, что у вас есть много машин, которые должны иметь одинаковые учетные записи пользователей и групп (информация, обычно находящаяся в */etc/passwd* и */etc/group*). Пользователи должны иметь возможность заходить на любую

из этих машин и напрямую получать доступ к их файлам (скажем, смонтировав файловую систему своего исходного каталога с центрального сервера, используя NFS). Очевидно, управлять учетными записями на множестве машин не так просто. Например, для того чтобы добавить нового пользователя, необходимо зайти по очереди на все машины и создать на каждой из них учетную запись. Однако при использовании NIS система автоматически запрашивает такую информацию по сети у централизованно управляемой базы данных, а не только ищет ее в локальных файлах *etc/passwd*. NIS+ – это расширенная служба NIS, которую уже начинают использовать.

Если вашей системе Linux предстоит взаимодействовать с другими системами в локальной сети, вполне возможно, что NFS и NIS уже широко используются в локальной сети. В этом разделе мы покажем, как сконфигурировать систему в качестве клиента NFS и NIS, т. е. монтировать удаленные файловые системы и участвовать в существующем домене NIS. Можно превратить вашу систему и в сервер NFS и в сервер NIS, но в конфигурировании Unix или Linux-систем в качестве NFS/NIS-сервера существует множество тонких моментов. Вместо того чтобы давать здесь неполные, и потому опасные, советы по настройке сервера, мы отсылаем вас к книге «Managing NFS и NIS» Хала Штерна (Hal Stern), изданной O'Reilly. Если вы уже знакомы с настройкой NFS/NIS на других системах Unix, то в Linux вы найдете не много отличий. Все особенности вы найдете в страницах руководства и Linux HOWTO.

Настройка NFS

Настроить вашу систему для монтирования удаленных файловых систем с помощью NFS легко. Предположим, что TCP/IP сконфигурирован и распознавание имен хостов работает правильно. Тогда можно просто добавить в файл */etc/fstab* строку следующего вида:

```
# device          directory           type    options
allison:/usr          /fsys/allison/usr   NFS     defaults
```

Как и при обычном монтировании файловой системы, сначала создайте каталог точки монтирования (в данном примере */fsys/allison/usr*). Строка файла */etc/fstab* позволяет локальной системе смонтировать каталог */usr*, находящийся в сети на машине *allison*.

Перед тем как этот пример монтирования с помощью NFS заработает, системный администратор сервера NFS (здесь *allison*) должен сконфигурировать систему для *экспорта* данного каталога (здесь */usr*) в вашу систему. На большинстве Unix-систем для этого надо просто отредактировать файл, такой как */etc/exports*, или выполнить несложную команду. Экспортирование каталога позволяет монтировать его на других машинах, использующих NFS. Сам экспортируемый каталог не обязательно должен быть корневым каталогом файловой системы, т. е. вы можете экспортировать */usr*, даже если */usr* не имеет собственной файловой системы.

Экспортируя каталог, администратор может сделать каталог доступным *только для чтения* (*read-only*). В этом случае вы не сможете осуществлять

запись в эту файловую систему. Поле `options` в строке файла */etc/fstab* из предыдущего примера нужно изменить на `ro` вместо `defaults`.

Несколько слов предостережения по поводу NFS. Прежде всего, NFS не любит, когда серверы с удаленными файловыми системами выключаются или нарушается сетевое соединение. Когда по какой-то причине сервер NFS не доступен, ваша система периодически выводит предупреждающие сообщения на консоль (или в системные журналы). Если такое происходит, можно попробовать размонтировать удаленную файловую систему недоступного сервера.

Другое, на что следует обратить внимание при монтировании файловых систем NFS, – это идентификаторы владельца (UID) и группы (GID) файлов на удаленной файловой системе. Для доступа через NFS к собственным файлам UID и GID вашей учетной записи должны совпадать с идентификаторами на сервере NFS. Простой способ проверить это – выполнить *ls -l*. Если UID или GID не совпадают с каким-либо локальным пользователем, *ls* выведет UID/GID файлов в виде чисел, а в случае совпадения будет показано имя пользователя или группа.

Если ID не совпадают, есть несколько способов решить эту проблему. Можно просто изменить UID вашей учетной записи (и GID первичной группы) так, чтобы они совпадали с записями на сервере NFS (отредактировав ваш локальный файл */etc/passwd*). Тогда после внесения изменений потребуется выполнить *chown* и *chgrp* для всех локальных файлов. Другое решение – создать отдельную учетную запись с подходящим UID/GID. Однако лучше всего, вероятно, воспользоваться NIS для управления базами данных пользователей и групп. При таком подходе вам не надо создавать локальные учетные записи, вместо этого они предоставляются сервером NIS. Далее мы расскажем об этом подробнее.

Другим тонким местом NFS является ограничение разрешений для `root` на файловых системах, монтируемых через NFS. Если только NFS-сервер явно не предоставил доступа в качестве `root` к файловым системам NFS, у вас не будет полного доступа к файлам при регистрации в качестве суперпользователя на вашей локальной системе. Это сделано в целях безопасности: предоставление суперпользователю неограниченного доступа к файлам на удаленных файловых системах NFS открывает возможности для злоупотреблений, особенно, когда клиент и сервер NFS принадлежат или управляются разными людьми. Поэтому при регистрации в качестве `root` на локальной системе вы не будете обладать всей полнотой власти над удаленными файловыми системами.

Наконец, следует помнить, что NFS никоим образом не использует шифрования. Если монтировать файловые системы через Интернет, то в любой момент их могут перехватить или подменить (кое-кто шутливо раскрывает NFS как «No File Security»). Кроме того, доступ к монтированным через NFS каталогам, находящимся вне локальной сети, окажется слишком медленным, если только у вас нет действительно высокоскоростного соединения.

Настройка NIS

NIS является сложной системой просто потому, что она очень гибкая. Это сетевая база данных общего назначения, дающая вашей системе прозрачный доступ к информации об учетных записях пользователей, группах, файловых системах и т. д. из баз данных, находящихся в сети. Цель NIS состоит в облегчении управления сетями. Например, если информация об учетных записях пользователей (такая, как в */etc/passwd*) хранится на одном сервере, этими записями можно пользоваться на нескольких машинах. В предыдущем разделе, посвященном NFS, мы показали, что для эффективного удаленного доступа к файлам UID и GID на клиентах и серверах NFS должны совпадать. Использование NIS позволяет определять UID и GID не локально, а удаленно.

Если в сети, куда подключена ваша машина, уже используется NIS, скорее всего, вы сможете добавить вашу машину в качестве клиента NIS, дав ей таким образом возможность напрямую получать из сети базы данных пользователей, групп и другие. До некоторой степени это вообще делает ненужным создание локальных учетных записей пользователей или групп. Кроме пользователей, создаваемых локально, таких как `root`, `bin` и т. д., все другие пользователи создаются на сервере NIS. Если совместить использование NIS с монтированием личных каталогов пользователей на сервере NFS, нет также необходимости выделять пользователям место на локальном диске. NIS может значительно сократить объем работы системного администратора.

В системах NIS могут быть *серверы*, *подчиненные серверы* и *клиенты*. Как вы могли догадаться, серверы – это системы, где создаются и откуда управляются базы NIS. Подчиненные серверы NIS – это системы, на которые сервер копирует свои базы данных. Они могут предоставлять информацию другим системам, но изменения в самих базах должны выполняться на основном сервере. Клиенты NIS – это системы, запрашивающие информацию из баз серверов. Подчиненные серверы используются просто для уменьшения нагрузки на основной сервер NIS, иначе все запросы NIS придется обрабатывать одной машине.

Чтобы полностью разобраться, как работает NIS и как управлять сервером NIS, потребовалось бы прочесть целую книгу (снова отсылаем к «Managing NFS and NIS»). Однако при чтении материала о NIS вы можете столкнуться с различными терминами. Первоначально NIS называлась YP, однако сейчас это название не используется, поскольку Yellow Pages является зарегистрированной в Великобритании торговой маркой (это телефонная книга, в конце концов).

Есть по крайней мере две реализации NIS под Linux: «традиционная» реализация NIS и отдельная реализация, известная как «NYS» (сокращение от NIS+, YP и Switch). Код клиента NIS для «традиционной» реализации содержится в стандартной библиотеке C и уже установлен на большинстве систем Linux. (Это необходимо, чтобы такие программы, как *login*, могли иметь прозрачный доступ как к базам NIS, так и к локальным системным файлам.)

Код клиента NYS содержится в Network Services Library (библиотеке сетевых сервисов) – *libnsl*. В системах Linux, использующих NYS, такие программы, как *login*, должны быть скомпилированы с использованием этой библиотеки. С другой стороны, стандартная библиотека C *glibc2*, устанавливаемая в настоящее время большинством дистрибутивов, содержит поддержку NIS+. Различные дистрибутивы Linux используют разные версии клиентского кода NIS или NYS, а некоторые используют смесь из обоих. Для уверенности мы опишем, как настроить систему и для традиционной реализации NIS, и для NYS. Поэтому, что бы ни было установлено на вашей системе, оно должно работать как клиент. Еще более осложняет ситуацию система подключаемых модулей аутентификации PAM (Pluggable Authentication Modules), применяемая рядом дистрибутивов. В этом случае программы типа *login* компонуются с библиотекой PAM, которая, в свою очередь, загружает модуль библиотеки PAM, реализующий принятую в системе схему аутентификации или делегирующий задачу другим библиотекам.

Мы предполагаем, что вы установили и запустили все необходимые демоны NIS (такие как *ypbind*), используемые обычным NIS для связи с NIS-сервером. Если ваша Linux-система не поддерживает NIS, обратитесь к документации, например, Linux NIS HOWTO, чтобы настроить ее с нуля. Практически все дистрибутивы Linux сейчас поставляются в комплекте с поддержкой NIS-клиента (и сервера), и все, что от вас потребуется, – это отредактировать несколько файлов конфигурации.

Первым шагом является установка домена NIS, в котором будет работать ваша система. Эту информацию вам могут предоставить сетевые администраторы. Обратите внимание, что имя домена NIS не обязательно должно совпадать с именем домена DNS, которое устанавливается командой *hostname*. Например, если полное имя системы – *loomer.vpizza.com*, то имя домена DNS будет *vpizza.com*. Однако имя домена NIS может быть абсолютно другим, например *vpizzas*. Имя домена NIS выбирается администратором сервера NIS и не связано с ранее описанным именем домена DNS.

Имя домена обычно устанавливается командой *domainname* во время загрузки, возможно, в одном из системных *rc*-файлов (например, в */etc/rc.d/rc.inet1*, описанном ранее). Сначала нужно проверить, что *domainname* не выполняется в одном из существующих *rc*-файлов. Команда использует следующий формат:

```
domainname domain-name
```

например `domainname vpizzas`. Сама команда обычно находится в */sbin/domainname* и может иметь слегка отличающееся имя, такое как *domainname-yp*.

Установка имени домена под NYS происходит несколько иначе. Вы должны создать (или отредактировать) файл */etc/yp.conf*. Этот файл должен содержать две строки: одну, указывающую имя вашего домена NIS, и другую, указывающую имя хоста сервера NIS. Например:

```
domain vpizzas
ypserver allison.vpizza.com
```

устанавливает, что именем домена NIS является `vpizzas`, и указывает, что в качестве сервера NIS следует использовать `allison.vpizza.com`. Если в этом файле нет строки `ypserver`, система рассылает широковещательное сообщение во время загрузки для определения имени сервера NIS. Имя хоста нужного сервера NIS вам могут предоставить сетевые администраторы.

После выполнения этих двух шагов ваша система должна получить возможность прозрачного доступа к базам NIS. Простым способом проверки этого может быть запрос к базе данных паролей на сервере NIS. Команда *ypwhich* выполняет запрос к специальным базам NIS, например:

```
ypwhich username passwd
```

Если будет возвращена строка из NIS-базы *passwd* для заданного пользователя, значит, запрос к базе NIS был выполнен успешно. (Можно проверить корректность возвращаемой информации, выполнив ту же команду с другой системы в вашем домене NIS, где настройка NIS гарантированно работает.) База данных NIS *passwd* не идентична файлу */etc/passwd* на вашей системе, хотя и использует тот же формат. Дополнительная информация по устранению неполадок в вашей конфигурации NIS содержится в документах Linux HOWTO.

16

World Wide Web и электронная почта

В предыдущей главе мы подключились к сети. Возможно, это была тяжелая работа, но результат того стоил: теперь ваша система является частью сообщества. Если вы подключены к Интернету, следующим шагом будет получение доступа ко всем его богатствам. По общему мнению, самыми полезными вещами в Интернете являются World Wide Web и электронная почта. Именно они и являются предметом внимания данной главы.

World Wide Web

Несомненно, каждый, кто имеет хоть какое-то отношение к компьютерам, пользовался в какой-то мере World Wide Web. Как когда-то текстовые процессоры и электронные таблицы заставляли людей обращаться к компьютерам, так сегодня это делает Web. Мы расскажем о некоторых средствах, с помощью которых можно получить доступ к Web из Linux.

С самого своего возникновения Linux была тесно связана с Интернетом в целом и с Web в частности. Например, Linux Documentation Project (LDP – проект документирования Linux) предлагает доступ к множеству относящихся к Linux документов через Web. Домашняя страница LDP (рис. 16.1), которую можно найти по адресу *http://www.tldp.org*, содержит ссылки на множество других связанных с Linux страниц по всему миру.

Веб-броузеры Linux обычно могут показывать информацию, предоставляемую различными типами серверов, не только серверами HTTP, посылающими клиентам страницы HTML. Например, обращаясь к документу по HTTP, вы, возможно, увидите страницу, подобную изображенной на рис. 16.1, т. е. с вложенными рисунками, ссылками на другие страницы и т. д. При обращении к документу по FTP вы увидите листинг каталогов FTP-сервера (рис. 16.2). Щелкнув по ссылке в FTP-документе, вы либо получите указанный файл, либо увидите содержимое другого каталога.

Ссылка на документ или другой ресурс Web осуществляется с помощью *унифицированного идентификатора ресурса (Uniform Resource Locator*), или

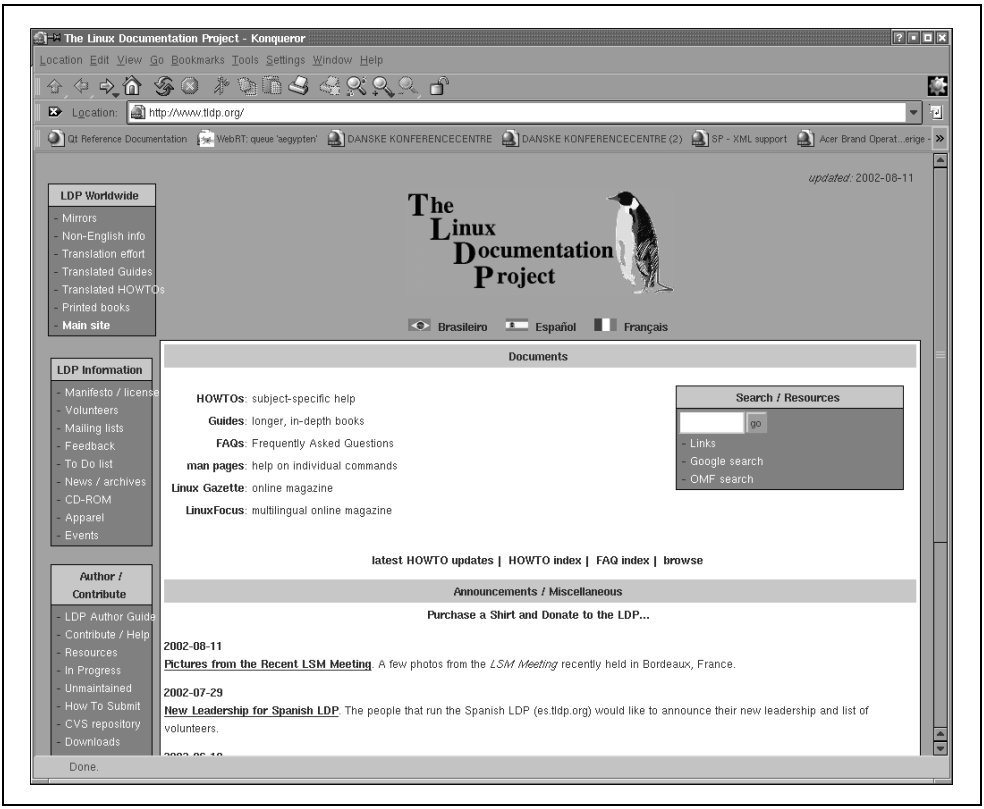


Рис. 16.1. *Домашняя страница Linux Documentation Project (LDP) в Web*

URL. URL – это имя, уникальным образом идентифицирующее веб-документ, включая имя машины, на которой находится документ, имя файла и используемый для доступа к нему протокол (FTP, HTTP и т. д.). Например, у Linux-газеты, периодически выходящей на сайте LDP, адрес следующий:

http://www.tldp.org/LDP/LG/current/index.html

Рассмотрим этот URL. Его первая часть – *http:* – идентифицирует протокол, используемый для документа, которым в данном случае является HTTP. Вторая часть URL – *//www.tldp.org* – идентифицирует машину, на которой находится документ. Последняя часть URL – *LDP/LG/current/index.html* – это логический путь к документу на *www.tldp.org*. Он похож на путь к файлу в Unix и указывает на файл *index.html* в каталоге *LDP/LG/current*. Поэтому для доступа к свежему номеру Linux-газеты надо запустить броузер, указав ему адрес *http://www.tldp.org/LDP/LG/current/index.html*. Что может быть проще?

Оказывается, благодаря некоторым соглашениям веб-серверов это действительно можно сделать проще. Если в качестве последнего элемента URL указан каталог, то сервер понимает, что вам нужен файл *index.html* в этом ката-

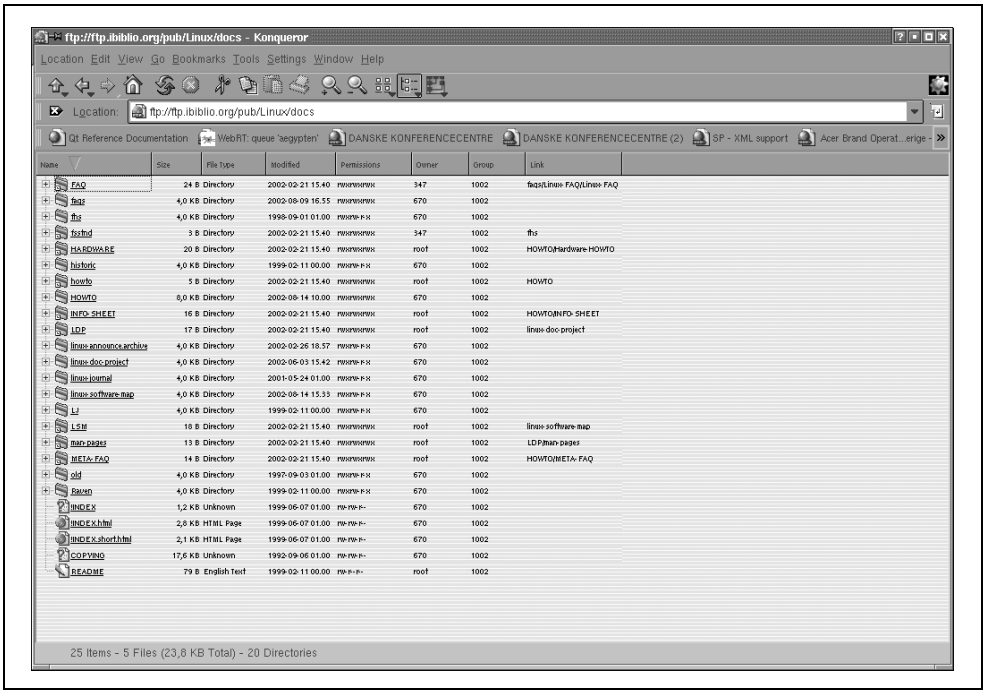


Рис. 16.2. *Каталог на FTP-сервере, отображаемый в веб-броузере Konqueror*

логе. Поэтому добраться до свежего номера Linux-газеты можно по более короткому URL:

http://www.tldp.org/LDP/LG/current

Для доступа к файлу по анонимному FTP можно использовать URL:

ftp://ftp.ibiblio.org/pub/linux/docs/INFO-SHEET/

Этот URL возвращает вводную информацию по Linux с *ftp.ibiblio.org*. Использование этого URL через броузер идентично использованию *ftp* для получения файла вручную.

Чтобы лучше понять Web, нужно просто начать им пользоваться. В следующем разделе мы объясним, как работать с некоторыми броузерами, а затем расскажем, как настроить свою машину в качестве веб-сервера, предоставляющего документы в Интернете.

Конечно, для того чтобы войти в Web, нужна система с прямым соединением с Интернетом (по Ethernet или PPP). В следующих разделах мы предполагаем, что вы уже настроили TCP/IP на своей системе и можете успешно пользоваться такими клиентами, как *ssh* и *ftp*.

Использование Konqueror и других броузеров

Konqueror – это один из самых популярных броузеров для Linux. Он поддерживает JavaScript и Java, может выполнять подключаемые модули Netscape (что позволяет добавлять такие функции, как просмотр презентаций Flash) и хорошо интегрирован с рабочим столом KDE, описанным в разделе «K Desktop Environment» главы 11. На самом деле при установке KDE в качестве составной части системы устанавливается Konqueror. В разделе, посвященном KDE, уже говорилось, как с помощью Konqueror читать локальные справочные файлы. Теперь мы отправимся с ним в Web.

Работа с Konqueror обычно самоочевидна, но если вы хотите познакомиться с ним глубже, зайдите с помощью Konqueror на сайт *http://www.konqueror.org*.

Ниже предполагается наличие Linux-машины с установленной поддержкой сети, системы X и броузера Konqueror. Как отмечалось, машина должна быть настроена на работу с TCP/IP и такими клиентами, как *ssh* и *ftp*.

Запустить Konqueror просто. Выполните команду:

```
eggplant$ konqueror url
```

где *url* – это полный веб-адрес, или URL, документа, который вы хотите увидеть. Если вы не укажете URL, Konqueror по умолчанию должен показать рекламную страницу, как на рис. 16.3.

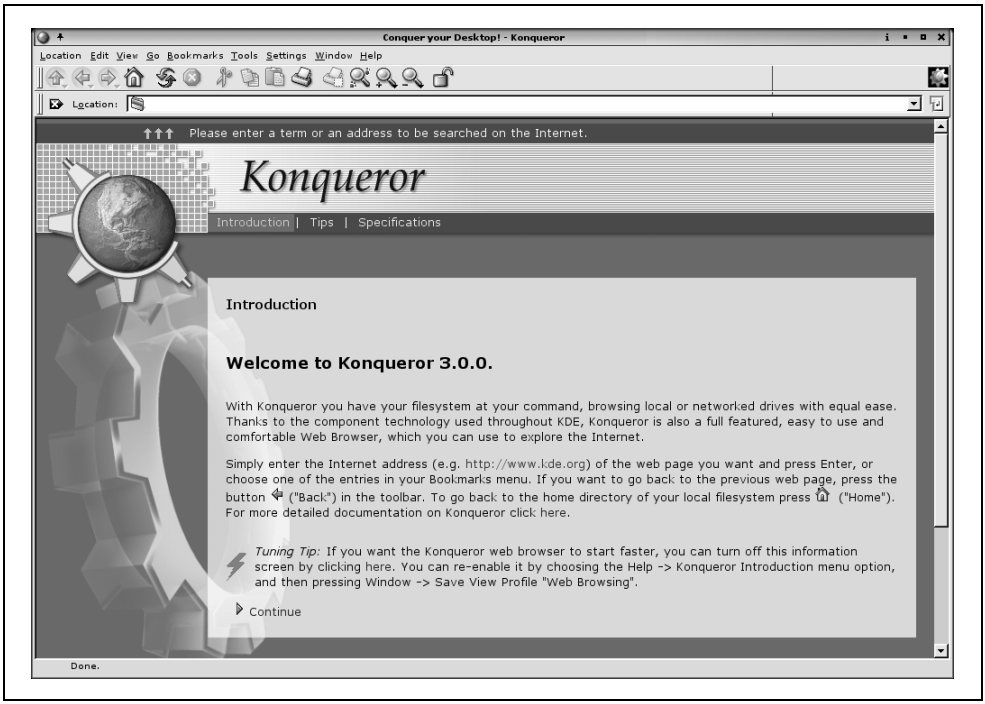


Рис. 16.3. *Рекламная страница Konqueror*

При запуске Konqueror из KDE можно просто нажать <Alt>+<F2>, чтобы открыть окно ввода команд *minicli* и ввести URL. В результате запустится Konqueror, который направится прямо на указанный вами URL.

Мы предполагаем, что у читателя есть некоторый опыт работы с веб-броузером на какой-либо компьютерной системе, поэтому не станем останавливаться на элементарных вещах, а отметим некоторые детали, специфичные для Linux.

Имейте в виду, что доступ к документам в Web иногда может быть медленным. Это зависит от скорости сетевого соединения между вами и сервером, а также от сетевого трафика. Иногда при большой загруженности веб-сайты просто отклоняют новые соединения. В этом случае Konqueror выводит соответствующее сообщение об ошибке. В нижней части окна Konqueror выводится состояние соединения, и когда осуществляется передача данных, эмблема KDE в правом верхнем углу оживает. Щелчок по эмблеме возвращает вас к домашней странице Konqueror.

При переходах по ссылкам в Konqueror каждый документ сохраняется в *окне history*, и к нему можно вернуться через меню Go. Нажатие кнопки Back (со стрелкой, направленной влево) в верхней части окна Netscape возвращает к документам, которые просматривались ранее. Аналогично кнопка Forward перемещает вперед по уже посещенным ссылкам.

Можно сделать «закладки» для часто посещаемых сайтов (или URL). При просмотре документа, к которому вы, возможно, захотите вернуться в будущем, выберите в меню пункт Bookmarks|Add Bookmark или просто <Ctrl>+<B>. Для того чтобы просмотреть список закладок, выберите меню Bookmarks. Щелчок по любой из закладок приведет вас к соответствующему документу в Web. Наконец, закладки могут показываться постоянно в отдельном окне слева от основного окна броузера, если выбрать Window→ Show Navigation Panel и щелкнуть по первой из появившихся кнопок (она похожа на настоящую закладку). С помощью панели навигации можно перейти в свой домашний каталог, открыть список установленного оборудования, историю сеанса работы и многое другое. Попробуйте поработать с Konqueror, и вы найдете в нем много полезных функций.

Как упоминалось ранее, можно обратиться к новому URL, указав его в качестве аргумента при запуске *konqueror*. Однако можно просто ввести URL в окне адреса в верхней части окна Konqueror. Окно адреса поддерживает функцию автозавершения: если начать ввод адреса, который вы уже посещали, то Konqueror автоматически покажет его и даст возможность выбрать. Закончив ввод URL (с помощью автозавершения или без него), нужно нажать <Enter>, и вы попадете в соответствующий документ.

Konqueror – это мощное приложение с множеством параметров. Для настройки его поведения нужно выбрать Settings→Configure Konqueror... Разделы Konqueror Browser и Enhanced Browsing содержат особенно интересные настройки. В разделе Cookies можно установить для каждого домена, будут ли от него приниматься cookies, и посмотреть те cookies, которые уже есть на компьютере. Сравните это с другими броузерами, которые прячут cookies где-то глу-

боко, так что их трудно посмотреть (или даже невозможно без специальных программ!).

Наконец, стоит отметить одну особую функцию. Веб-броузеры регистрируются на серверах с использованием так называемой строки «User Agent», которая может содержать любой текст, но обычно хранит название и версию веб-броузера, а также название и версию операционной системы хоста. Некоторые весьма неумные вебмастеры выводят разные веб-страницы (или вообще ничего не выводят!), если веб-броузер – не Internet Explorer, поскольку думают, что Internet Explorer – единственный веб-броузер, который в состоянии показывать их веб-сайт.[1] Но если перейти в раздел User Agent, можно обмануть веб-сервер, прикинувшись другим броузером, которому этот веб-сервер не стыдится выдавать документы. Нужно щелкнуть New, выбрать имя домена, к которому вы хотите получить доступ, и ввести свою строку User Agent или выбрать одну из заготовленных.

Konqueror – не единственный броузер, который может читать документы Web. Есть еще броузер Mozilla – версия open source броузера Netscape Navigator, благодаря которому очень многие и познакомились с Web. Если в вашем дистрибутиве нет Mozilla, его можно загрузить с *http://www.mozilla.org*. Функции Mozilla вполне аналогичны возможностям Konqueror, и все, что можно делать в одном броузере, можно делать и в другом. В Mozilla все же есть возможность, отсутствующая в Konqueror: он может хранить несколько веб-страниц на разных «закладках» в главном окне броузера. Чтобы одновременно просматривать несколько страниц в Konqueror, нужно открыть несколько окон Konqueror (в Mozilla это также можно сделать).

Еще один многосторонний броузер – Lynx. Это текстовый броузер, поэтому, работая с ним, вы не увидите графику. Однако это делает его быстрым, и вы можете найти его удобным. Lynx можно использовать и без X Window System. И наконец, для тех, кто не хочет расставаться с Emacs, существует Emacs/W3 – полноценный броузер, используемый из Emacs или XEmacs.

Настройка собственного веб-сервера

Теперь, когда вы увидели, что предлагает Web, вы готовы построить свою автозаправку на информационном автобане. Запустить собственный веб-сервер легко; для этого необходимо выполнить две вещи: настроить демон *httpd* и написать документы, которые вы собираетесь предоставлять на сервере.

httpd – это демон, обслуживающий HTTP-запросы на вашей машине. Обращение к любому документу по URL *http* обслуживается *httpd*. Точно так же URL, начинающийся с *ftp*, обслуживается демоном *ftpd* и т. д. Нет единого веб-демона: каждый тип URL использует свой демон для доступа к информации на сервере.

---

[1] Если веб-сайт можно просматривать единственным броузером или на нем стоит пометка «оптимизирован для X-броузера», то с него нужно бежать, сдерживая возмущение такой некомпетентностью вебмастера.

Существует несколько серверов HTTP. Здесь мы рассмотрим сервер Apache *httpd*, отличающийся гибкостью и простотой настройки. В этом разделе обсуждаются основные аспекты установки и настройки этой версии *httpd*. Далее мы поговорим о том, как написать собственные документы на HTML (язык разметки, используемый в веб-страницах), а также коснемся более тонких моментов конфигурирования сервера, таких как предоставление интерактивных форм.

Во всех современных версиях Linux сервером *httpd* по умолчанию должен быть Apache. Но если вы выбрали «минимальную» или «настольную» конфигурацию во время установки системы, может потребоваться установить Apache вручную. Возможно, вы захотите установить более новую версию, чем та, что входит в ваш дистрибутив (на момент написания книги текущей стабильной версией была 2.0.40). Тогда можно загрузить двоичные файлы или исходные тексты с *http://www.apache.org* и собрать Apache самостоятельно. Веб-сайт *apache.org* содержит полную документацию программы.

Книга «Apache – The Definitive Guide» Бена Лурье (Ben Laurie) и Питера Лурье (Peter Laurie) охватывает все тонкости работы с Apache.

Местонахождение разных файлов Apache зависит от дистрибутива или от установленного вами пакета. Далее мы рассмотрим обычную установку. Для начала найдите различные части вашей системы:

/usr/sbin/httpd

Это выполняемый файл, который является собственно сервером. В дистрибутиве Debian он находится в */usr/sbin/apache*.

/etc/httpd

Содержит файлы конфигурации для *httpd*, главный из которых – *httpd.conf*. Позже мы обсудим, как вносить изменения в эти файлы. В системах Debian это */etc/apache* вместо */etc/httpd*.

/usr/local/httpd

Содержит сценарии HTML для обслуживания клиентов сайта. Этот каталог доступен всем пользователям Web вместе со всем, что в нем находится, и поэтому представляет серьезную угрозу безопасности для всего, что не является публичными данными.

/var/log/httpd

Содержит файлы журналов, ведущихся сервером.

Теперь наша задача – изменить файлы конфигурации в соответствующем каталоге. В нем должны быть, по крайней мере, следующие четыре файла: *access.conf-dist*, *httpd.conf-dist*, *mime.types* и *srm.conf-dist*. Скопируйте файлы с именами, заканчивающимися на *-dist*, и измените их для вашей системы. Например, *access.conf-dist* надо скопировать в *access.conf* и отредактировать.

Последние версии Apache выполняют много настроек автоматически, но мы расскажем, как вручную произвести настройку, если возникнут неполадки.

На *http://www.apache.org* вы найдете полную документацию по конфигурированию *httpd*. Здесь мы предлагаем примеры файлов конфигурации, соответствующие реально работающему *httpd*.

httpd.conf

Файл *httpd.conf* является основным файлом конфигурации сервера. Сначала скопируйте *httpd.conf-dist* в *httpd.conf* и отредактируйте его. Мы остановимся только на некоторых наиболее важных параметрах. Файл *httpd.conf-dist* снабжен пространными комментариями.

Директива `ServerType` задает режим, в котором запускается сервер – как автономный демон (в данном примере) или из *inetd*. По многим причинам обычно лучше запускать *httpd* в автономном режиме. В противном случае *inetd* должен будет запускать новый экземпляр *httpd* для каждого входящего соединения.

Хитрым моментом здесь является указание номера порта. Вы можете захотеть запустить *httpd* без привилегий суперпользователя (т. е. у вас может и не быть привилегий `root` на нужной машине, а вы хотите запустить *httpd* от своего имени). В этом случае необходимо использовать номер порта от 1024 и выше. Например, если указать:

```
Port 2112
```

то вы можете запустить *httpd* как обычный пользователь. В этом случае HTTP URL к этой машине должен указываться как:

*http://www.*ecoveggie.*org:2112/*...

Если номер порта в URL не указан (обычно так и делается), используется порт с номером 80.

Директива

```
DocumentRoot /usr/local/httpd/htdocs
```

задает каталог, в котором хранятся документы, предоставляемые по протоколу HTTP. Эти документы написаны на языке HTML

Например, если кто-либо обратится к URL:

http://www.ecoveggie.org/fruits.html

то реальным файлом, к которому будет осуществляться доступ, окажется */usr/local/httpd/htdocs/fruits.html*.

Директива `UserDir` определяет каталог, который может быть создан каждым пользователем в своем домашнем каталоге для хранения публично доступных HTML-файлов. Например, если бы мы использовали URL:

http://www.ecoveggie.org/~mdw/linux-info.html

реально мы обратились бы к файлу *~mdw/public_html/linux-info.html*.

В следующих строках включаются функции указателя *httpd*:

```
# Если URL указан без имени файла, считать, что запрашивается файл
# index (если таковой существует).
DirectoryIndex index.html

# Включить 'забавный' индекс каталогов
FancyIndexing on
```

В данном примере, если броузер попытается обратиться к серверу, указав в URL только каталог (без имени файла), ему будет возвращен файл *index.html* из данного каталога, если таковой существует. В противном случае *httpd* генерирует причудливый индекс со значками, представляющими разные типы файлов. На рис. 16.2 показан пример такого индекса.

Значки назначаются директивой `AddIcon`, как показано ниже:

```
# Устанавливает разные значки для необычных индексов по имени файлов
# Например, DocumentRoot/icons/movie.xbm используется для файлов,
# заканчивающихся на .mpg и .qt
AddIcon /icons/movie.xbm .mpg
AddIcon /icons/back.xbm ..
AddIcon /icons/menu.xbm ^^DIRECTORY^^
AddIcon /icons/blank.xbm ^^BLANKICON^^
DefaultIcon /icons/unknown.xbm
```

Имена файлов значков (такие как */icons/movie.xbm*) по умолчанию указываются относительно `DocumentRoot`. (Существуют и другие способы указать путь к документам и значкам, например при помощи псевдонимов. Это обсуждается позже.) Имеется также директива `AddIconByType`, позволяющая указать значок для документа, основываясь на его типе MIME, и директива `AddIconByEncoding`, позволяющая указать значок для документа, основываясь на его кодировке (например в зависимости от того, сжат ли документ).

Кроме того, можно указать значок, который используется, если ничто из вышеперечисленного не подходит. Это делается директивой `DefaultIcon`.

Необязательные директивы `ReadmeName` и `HeaderName` указывают имена файлов, включаемых в индекс, генерируемый *httpd*:

```
ReadmeName README
HeaderName HEADER
```

Здесь, если в текущем каталоге есть файл *README.html*, он будет добавлен в индекс. Файл *README* будет добавлен, если *README.html* не существует. Аналогично *HEADER.html* или *HEADER* добавляются в начало индекса, генерируемого *httpd*. Вы можете использовать их для описания содержимого конкретного каталога, когда индекс запрашивается броузером.

```
# Имя локального файла доступа
AccessFileName .htaccess

# Тип MIME по умолчанию
DefaultType text/plain
```

Директива `AccessFileName` указывает имя *локального файла доступа* для каждого каталога. (Об этих файлах будет сказано ниже.) Директива `DefaultType` указывает тип MIME для документов, не перечисленных в *mime.types*.

Следующие строки задают каталоги для полезных файлов:

```
# Устанавливает место расположения значков.
Alias /icons/ /usr/local/html/icons/

# Устанавливает место расположения CGI-программ
ScriptAlias /cgi-bin/ /usr/local/httpd/cgi-bin/
```

Директива `Alias` указывает псевдоним для путей к любым документам, к которым обращаются по URL или которые перечислены в *srm.conf*. Ранее мы применяли директиву `AddIcon` для того, чтобы установить имена значков, используя при этом такие имена, как */icons/movie.xbm*. Здесь мы указываем, что имя */icons/* нужно преобразовать в */usr/local/html/icons/*. Там и должны храниться все файлы значков. Вы можете использовать `Alias` для создания и других псевдонимов.

Директива `ScriptAlias` похожа на `Alias`, но устанавливает реальное расположение CGI-сценариев в системе. В данном примере мы хотим хранить сценарии в каталоге */usr/local/httpd/cgi-bin/*. Каждый раз когда в URL первым каталогом указан */cgi-bin/*, он преобразуется в настоящее имя каталога. Подробная информация по CGI и сценариям есть в книге Скотта Гулича (Scott Guelich), Шишира Гундаварама (Shishir Gundavaram) и Гюнтера Бирзнекса (Gunther Birznieks) «CGI Programming with Perl»[1] (O'Reilly).

Элементы `<Directory>` определяют параметры и атрибуты для конкретного каталога:

```
# Установить параметры для каталога сценариев cgi-bin.
<Directory /usr/local/html/cgi-bin>
Options Indexes FollowSymLinks
</Directory>
```

Здесь мы указали, что каталог сценариев CGI должен иметь параметры доступа `Indexes` и `FollowSymLinks`. В число возможных параметров доступа входят:

`FollowSymLinks`

Необходимо следовать по символьным ссылкам в данном каталоге для доступа к документам, на которые они указывают.

`ExecCGI`

Позволить выполнение сценариев CGI из этого каталога.

`Indexes`

Разрешить создание индексов из этого каталога.

`None`

Запретить все параметры для этого каталога.

`All`

Разрешить все параметры для данного каталога.

---

[1] С. Гулич, Ш. Гундаварам, Г. Бирзнекс «CGI программирование на Perl», 2-е издание. – Пер. с англ. – СПб: Символ-Плюс, 2001.

Существуют и другие параметры; подробнее см. в документации *httpd*.

Далее мы разрешаем некоторые параметры и другие атрибуты для */usr/local/httpd/htdocs* – каталога, содержащего наши документы HTML:

```
<Directory /usr/local/httpd/htdocs>

Options Indexes FollowSymLinks

# Позволить локальному файлу доступа, .htaccess, переопределять
# любые атрибуты, указанные здесь.
AllowOverride All

# Ограничение доступа к документам в данном каталоге
<Limit GET>
order allow,deny
allow from all
</Limit>

</Directory>
```

Мы включили параметры `Indexes` и `FollowSymLinks` для данного каталога. Директива `AllowOverride` позволяет локальному файлу доступа для каждого каталога (в данном случае *.htaccess*) переопределять любые атрибуты, указанные здесь. Файл *.htaccess* имеет тот же формат, что и глобальный файл *access.conf*, но применяется только к каталогу, в котором он находится. Таким образом, можно указать атрибуты для отдельных каталогов, включив в каждый из них файл *.htaccess* вместо перечисления их в глобальном файле.

Локальные файлы доступа в основном применяются для того, чтобы дать возможность пользователям устанавливать разрешения на доступ к личным каталогам HTML (таким как *~/public_html*), не обращаясь к администратору с просьбой изменить глобальный файл доступа. Однако при этом возникает проблема безопасности. Например, пользователь может позволить в своем каталоге выполнение CGI-сценариев на стороне сервера. Если вы отключите `AllowOverride`, пользователи не смогут обойти параметры доступа, указанные в глобальном файле *access.conf*. Это можно сделать, указав:

```
AllowOverride None
```

что отключит локальные файлы доступа *.htaccess*.

Поле `<Limit GET>` используется для задания правил доступа для броузеров, пытающихся получить документы с данного сервера. В нашем случае мы указали `order allow,deny`. Это означает, что правила, разрешающие (`allow`) доступ, рассматриваются перед запрещающими (`deny`) правилами. Затем мы пишем правило `allow from all`, которое означает, что любой хост может обращаться к документам на этом сервере. Если вы хотите запретить доступ из конкретных доменов или систем, можно добавить строку:

```
deny from .nuts.com biffnet.biffs-house.us
```

Первая запись запрещает доступ всем машинам из домена *nuts.com*. Вторая запись запрещает доступ с сайта *biffnet.biffs-house.us*.

srm.conf и access.conf

Эти файлы следует оставить пустыми. В прежних версиях Apache в файле *srm.conf* (от Server Resource Map – карта ресурсов сервера) перечислялись возможности, предоставляемые сервером, а *access.conf* управлял доступом к файлам Apache. Все, что раньше находилось в этих файлах, теперь содержится в файле *httpd.conf*.

Запуск httpd

Теперь мы готовы запустить *httpd* и позволить нашей машине обслуживать запросы HTTP. Как указывалось ранее, можно запустить *httpd* из *inetd* или как автономный сервер. Здесь мы опишем, как запустить *httpd* в автономном режиме.

Все, что необходимо для запуска *httpd*, – это выполнить команду:

```
httpd -f configuration-file
```

где *configuration-file* – это путь к *httpd.conf*. Например

```
/usr/sbin/httpd -f /etc/httpd/httpd.conf
```

запустит *httpd* с файлами конфигурации, находящимися в */etc/httpd*.

Ошибки при запуске сервера или при обращении к документам регистрируются в журналах *httpd* (их местонахождение задано в *httpd.conf*). Помните, что вы должны запускать *httpd* как `root`, если используется порт с номером 1023 и меньше. Когда вы отладите работу *httpd*, его можно будет запускать автоматически, включив соответствующую строку с командой *httpd* в один из ваших системных *rc*-файлов, например */etc/rc.d/rc.local*.

В некоторые релизы Apache входит утилита *apachectl*, которая контролирует процесс *httpd* (запуск, остановку, перезагрузку и т. д.). Например директива:

```
apachectl configtest
```

позволяет проверить корректность файла конфигурации без запуска сервера. Наконец, запускать, перезапускать и останавливать Apache можно командой */etc/init.d/apache* с одним из параметров `start`, `restart` или `stop`.

Конечно, для запроса документов по HTTP из броузера их надо еще написать. Эта тема выходит за рамки данной книги. Рекомендуем книги по HTML, изданные O'Reilly: «HTML & XHTML. The Definitive Guide»[1] Чака Муссиано (Chuck Musciano) и Билла Кеннеди (Bill Kennedy) и «HTML Pocket Reference» Дженнифер Нидерст (Jennifer Niederst). О веб-странице особого рода, заполняемой данными из базы, рассказывается в главе 18 данной книги.

---

[1] Чак Муссиано, Билл Кеннеди «HTML и XHTML», 4-е издание. – Пер. с англ. – СПб: Символ-Плюс, 2002.

Электронная почта

Электронная почта (email) – это одна из самых желанных функций компьютерной системы. Обмениваться сообщениями могут локальные пользователи системы Linux или пользователи с различных хостов сети. Чтобы организовать электронную почту, нужны программы трех типов: *mail user agent*, или почтовый клиент, *mail transport agent* (MTA), или агент передачи почты, и *transport protocol*, или транспортный протокол.

Почтовый клиент предоставляет интерфейс для чтения почты, создания новых сообщений и хранения сообщений. Linux предлагает большой выбор почтовых клиентов, которые постоянно совершенствуются. Некоторые почтовые клиенты предоставляют особые возможности, например их можно использовать в качестве клиента новостей или как веб-броузер. Почтовые клиенты различаются по степени поддержки MIME. (MIME расшифровывается как Multimedia Internet Mail Extensions. Это скорее общий стандарт для описания содержимого электронных сообщений, чем ориентированный на мультимедиа.) Однако давать какие-либо рекомендации сложно, поскольку во всех почтовых клиентах поддержка MIME непрерывно улучшается.

Почтовый клиент для передачи почты между пользователями (как локальными, так и между системами) полагается на MTA. MTA, в свою очередь, использует для передачи почты транспортный протокол – обычно UUCP (Unix-to-Unix Copy) либо SMTP (Simple Mail Transport Protocol – простой протокол передачи почты).

Существует несколько возможных сценариев использования электронной почты на Linux-системах, и в зависимости от этих сценариев требуется установка разных наборов программного обеспечения. Однако почтовый клиент потребуется в любом случае.

Первый сценарий применяется при доступе к Интернету по коммутируемой линии через провайдера (ISP). Обычно в этом случае на Linux-машине есть только один пользователь, хотя это необязательно. Провайдер принимает вашу почту из Интернета и хранит ее на своих жестких дисках. Вы можете забрать свою почту в любое время, используя обычный протокол POP3 (Post Office Protocol) или более новый протокол IMAP. Исходящая почта в этом случае практически всегда посылается по протоколу SMTP, который повсеместно используется для передачи почты в Интернете.

В простейшем случае почтовый клиент используется как для получения почты по POP3 или IMAP, так и для отправки ее по SMTP. При этом даже не надо устанавливать MTA, т. к. почтовый клиент возьмет на себя всю работу. Этот вариант не отличается большой гибкостью, но он вполне может подойти, если нужно просто получить доступ к электронной почте. Эту схему работы поддерживают такие почтовые клиенты, как KMail из KDE и почтовая программа, встроенная в Mozilla (описываются далее).

Если нужна большая гибкость (за нее, конечно, придется заплатить трудом по настройке и управлению), можно установить MTA, например Postfix, который описан в следующем разделе. Необходимо установить программу, ко-

торая будет забирать почту с сервера POP3 или IMAP, находящегося у провайдера. Эта программа будет забирать почту по вашему требованию и передавать сообщения работающему на вашей системе MTA, который распределит почту по почтовым папкам получателей. Именно это делает программа *fetchmail*, которую мы рассмотрим в этой главе. Исходящая почта по-прежнему посылается по SMTP, но MTA на вашей системе позволяет отсылать почту не сразу на SMTP-сервер провайдера, а на собственный сервер, предоставляемый MTA. Затем MTA пересылает почту провайдеру. Такая конфигурация позволяет настроить MTA на периодическую отсылку почты, чтобы не подключаться каждый раз к Интернет-провайдеру.

Третий сценарий предназначен для машин с постоянным подключением к Интернету, либо через шлюз по сети Ethernet, либо по выделенной линии. В этом случае вы, возможно, захотите получать почту, как только она появится у провайдера, не сохраняя ее там. Это также требует настройки MTA. Входящая почта будет направлена вашему SMTP-серверу (т. е. MTA). Ваш провайдер должен также настроить свою систему соответствующим образом.

Существует, конечно, еще много сценариев для работы с почтой, и возможны также комбинации из трех вышеперечисленных. Если вы собираетесь настраивать почтовый сервис для целой сети, необходимо прочитать «Linux Network Administrator's Guide», а также книгу о вашем MTA.

У вас есть большой выбор между программами для настройки электронной почты на Linux. Мы не можем рассказать обо всех доступных решениях, но опишем некоторые, наиболее часто используемые пакеты, неплохо подходящие для своих задач. Самым популярным решением для Linux в настоящее время мы считаем следующее: почтовые клиенты Kmail и Mozilla, агент передачи почты Postfix и реализация протоколов POP3 и IMAP в программе *fetchmail*. Эти пакеты относительно просты в настройке, но предоставляют все возможности, необходимые большинству пользователей. Эти средства охватывают все три описанные выше сценария.

Postfix MTA – агент передачи почты

Есть несколько MTA, работающих под Linux. Исторически самым распространенным пакетом под Unix был *sendmail*, появившийся на сцене довольно давно. Обычно его считают более сложным в использовании, чем остальные, но он тщательно документирован в книге «Sendmail» Брайана Костелеса (Bryan Costales) и Эрика Олмена (Eric Allman), O'Reilly.

Postfix – более новый MTA, разработанный гуру в области безопасности Wietse Venema в качестве замены для Sendmail. Он спроектирован как совместимый с Sendmail, но обеспечивающий более высокий уровень защиты и более простой в настройке.

Postfix – очень гибкий и надежный программный продукт, имеющий несколько уровней защиты от потенциальных злоумышленников. Postfix также создавался с расчетом на достижение высокой производительности и использует технологии, ограничивающие такие замедляющие работу операции, как создание новых процессов и обращение к файловой системе. Его проще

настраивать и администрировать, чем другие почтовые пакеты, поскольку в нем используются простые файлы конфигурации и справочные таблицы для перезаписи адресов. Он отличается простотой в использовании в качестве базового MTA при сохранении возможности применения в более сложной среде.

Postfix встраивается во многие дистрибутивы Linux, поэтому возможно, что он уже установлен на вашей машине. Если это не так, можно найти готовый пакет или скомпилировать его самостоятельно из исходного кода. На домашней странице Postfix (*http://www.postfix.org/*) есть ссылки для загрузки исходного кода («Download») и пакетов для различных дистрибутивов Linux («Packages and Ports»).

Postfix выпускается в двух сериях версий: официальной и экспериментальной. Экспериментальные версии содержат все свежие патчи и новые функции, которые могут измениться перед включением в официальный релиз. Термин «экспериментальный» не должен вас отпугивать: эти версии очень стабильны и тщательно протестированы. Если вам нужна функция, имеющаяся только в экспериментальной версии, вы вполне можете воспользоваться ею. Прочтите комментарии к версиям в обеих сериях, которые объяснят, в чем различия между ними.

Некоторые сведения о DNS

Прежде чем устанавливать Postfix, нужно понять, что если ваша машина будет получать почту с других машин через Интернет, необходимо правильно настроить DNS для вашего домена. О DNS рассказывается в главе 15.

Предположим для целей нашего рассказа, что вы конфигурируете хост с именем *halo* в домене *example.org*, и что на вашей машине есть учетная запись пользователя *michael*. Независимо от того, каким способом вы собираетесь получать почту, у вашего хоста *halo.example.org* должна быть A-запись DNS, отображающая это имя хоста в его IP-адрес.

В данном примере вашим почтовым адресом будет *michael@halo.example.org* или *michael@example.org*. Если вы хотите использовать адрес первого вида, то для получения сообщений достаточно настроить A-запись DNS.

Если ваша машина должна получать всю почту для *example.org* (*michael@example.org*), домен должен обладать MX-записью DNS, указывающей на ваш хост *halo.example.org*. Если вы самостоятельно конфигурируете DNS для своего домена, обязательно прочтите документацию, чтобы понять, как это работает. Либо обсудите со своим администратором DNS или интернет-провайдером маршрутизацию почты на вашу машину.

Postfix часто использует DNS в обычной работе и пользуется соответствующими библиотеками Linux для выполнения запросов DNS. Вы должны правильно настроить свою систему для выполнения поиска в DNS (см. «Настройка DNS» в главе 15). Postfix обычно должен найти MX-запись, чтобы осуществить доставку. Не следует рассчитывать, что когда Postfix сообщает о проблеме DNS с адресом, а вы обнаруживаете, что распознавание домена происходит правильно, то доставка почты пройдет успешно. Если Postfix со-

общает о проблеме, можно быть почти уверенным, что проблема действительно есть.

Установка Postfix

Несмотря на наличие готовых дистрибутивов может потребоваться самостоятельно скомпилировать пакет, чтобы использовать дополнительные библиотеки или функции, отсутствующие в вашем дистрибутиве. Возможно также, что вам потребуется последняя версия, чтобы получить доступ к новой функции, еще не включенной в ваш дистрибутив.

Прежде чем устанавливать Postfix, учтите, что в этот пакет входят три команды – */usr/bin/newaliases*, */usr/bin/mailq* и */usr/sbin/sendmail*, которые обычно используются Sendmail. Postfix заменяет эти команды другими, которые работают с системой Postfix, а не с Sendmail. Следует переименовать существующие команды Sendmail, чтобы они не были уничтожены при установке Postfix и вы могли при желании снова воспользоваться первоначальными исполняемыми модулями Sendmail:

```
# mv /usr/bin/newaliases /usr/bin/newaliases.orig
# mv /usr/bin/mailq /usr/bin/mailq.orig
# mv /usr/sbin/sendmail /usr/sbin/sendmail.orig
```

Postfix использует файлы базы данных Unix для хранения своей информации о псевдонимах и справочной таблице, поэтому перед компиляцией Postfix необходимо установить библиотеки *db3*. Эти библиотеки есть в RPM-пакете db3-devel или пакете Debian libdb3. Если вы не пользуетесь администратором пакетов, то можете получить библиотеки непосредственно у Sleepycat Software (*http://www.sleepycat.com/*). Если вы пользуетесь RPM, выполните следующую команду и проверьте, установлены ли в вашей системе необходимые библиотеки:

```
# rpm -qa | grep db3-devel
db3-devel-3.2.9-5
```

Вы должны увидеть нечто похожее на вторую строку, в которой показан пакет db3-devel и номер его версии. Если *rpm* ничего не возвратит, нужно сначала установить эти библиотеки и только потом устанавливать Postfix.

В Debian узнать, установлены ли библиотеки, можно с помощью *dpkg*:

```
# dpkg -l libdb3
```

Если вы загрузили двоичный дистрибутив Postfix, воспользуйтесь своим менеджером пакетов (описанным в главе 7) для его установки. Если вы загрузили файл с исходным кодом *postfix-1.1.11.tar.gz*, переместите его в подходящий каталог (например, в свой личный) и распакуйте. Цифры в имени файла указывают версию пакета. У вас могут оказаться другие цифры в зависимости от того, какую версию вы загрузите.

Для компиляции Postfix следуйте описанной ниже базовой процедуре. Учтите, что вам потребуются права `root`, чтобы создать пользователя и группу и установить пакет.

1. Переименуйте свои исполняемые файлы Sendmail, как описано ранее.
2. Создайте пользователя с именем *postfix* и группу с именем *postdrop*. Процедура создания учетных записей пользователей и групп описана в разделе «Управление учетными записями пользователей» главы 5.
3. Выполните *gunzip* над сжатым файлом, получив файл с именем *postfix-1.1.11.tar*.
4. Выполните:

   ```
   tar -xvf postfix-1.1.11.tar
   ```

   чтобы распаковать исходный код в каталог *postfix-1.1.11*.
5. Перейдите в каталог, созданный при распаковке файла. Вы обнаружите в нем файл *INSTALL*, содержащий подробные инструкции по компиляции системы Postfix. Обычно для этого требуется лишь ввести команду `make`, находясь в этом каталоге.
6. Если компиляция выполнится без ошибок, введите `make install`, чтобы установить Postfix на вашей машине. В ответ на приглашения сценария установки следует принимать все значения по умолчанию.

После установки файлы Postfix будут размещены в следующих каталогах:

/usr/libexec/postfix

В этом каталоге содержатся различные демоны Postfix. Postfix использует расщепленную архитектуру, в которой несколько отдельных программ выполняют раздельные задачи. Первым запускается демон *master*. Он занимается запуском других программ по мере необходимости. По большей части вас не должно беспокоить, какие программы здесь находятся. Запуск и останов Postfix осуществляется командой *postfix*, расположенной в каталоге */usr/sbin*.

/etc/postfix

Обычно в этом каталоге находятся десятки файлов конфигурации Postfix, но Postfix использует только *master.cf* и *main.cf*, а также несколько справочных таблиц. Остальные файлы служат примерами, документирующими различные параметры, используемые для настройки.

Файл *master.cf* управляет различными процессами Postfix. В нем есть строка для каждой компоненты Postfix. Структура файла описана в комментариях, находящихся в самом файле. Обычно для простой установки Postfix не требуется делать никаких изменений.

Файл *main.cf* является глобальным файлом конфигурации SMTP. Он содержит список параметров, для которых установлено одно или несколько значений в формате:

```
parameter = value
```

Комментарии обозначаются символом решетки (#) в начале строки. Комментарии нельзя помещать на одной строке с параметрами. Строки ком-

ментариев могут начинаться с пробелов или табуляции, но должны быть на отдельных строках.

Если для параметра устанавливается несколько значений, они разделяются запятыми или пробельными символами (включая перевод строки), но если параметр описывается в нескольких строках, вторая и последующие строки должны начинаться с пробельного символа. Значения могут ссылаться на другие параметры, при этом имя параметра предваряется знаком доллара ($).

Вот пример записи, содержащей комментарии, несколько строк и ссылку на параметр:

```
# Вот все машины, с которых я жду почту.
mynetworks = $myhostname
192.168.75.0/24
10.110.12.15
```

/usr/sbin

Все команды Postfix находятся в */usr/sbin*, и их имена начинаются с `post`. Существуют команды для создания файлов индексов, управления почтовой очередью и других задач администрирования системы Postfix. Здесь находится команда *postfix*, используемая для останова и запуска Postfix (описывается ниже).

/var/spool/postfix

Администратор очередей Postfix – важная компонента системы Postfix, которая принимает входящие почтовые сообщения и взаимодействует с другими компонентами Postfix для их доставки. Он хранит свои файлы в каталоге */var/spool/postfix*. Очереди, которыми он управляет, показаны ниже. Postfix предоставляет несколько утилит для управления очередями, например *postcat*, *postsuper* и *mailq*, но для просмотра очереди можно воспользоваться обычными командами Linux, такими как *find* и *cat*.

/var/spool/postfix/incoming

Все входящие сообщения, полученные через сеть или локально.

/var/spool/postfix/active

Сообщения, которые администратор очередей доставляет в данное время или готовится доставить.

/var/spool/postfix/deferred

Сообщения, которые не удалось доставить немедленно. Postfix будет пытаться доставить их снова.

/var/spool/postfix/corrupt

Сообщения, которые совершенно невозможно прочесть, либо они повреждены другим образом, и доставить их невозможно. Они хранятся здесь для того, чтобы при необходимости можно было посмотреть на них и выяснить, в чем проблема. Эта очередь используется редко.

/usr/local/man

Postfix устанавливает документацию в виде страниц руководства. В документации описываются утилиты командной строки, демоны и файлы конфигурации.

Как уже отмечалось, Postfix устанавливает новые файлы */usr/bin/newaliases*, */usr/bin/mailq* и */usr/sbin/sendmail*.

Конфигурирование Postfix

Перед тем как впервые запустить Postfix, нужно проверить правильность формата таблицы псевдонимов и задания некоторых критически важных параметров конфигурации.

В Sendmail исторически использовался файл */etc/aliases*, который описывал отображение одних локальных имен пользователей в другие. Postfix сохраняет эту традицию. Файл */etc/aliases* – это обычный текстовый файл, используемый в качестве входных данных при создании файла индексированной базы данных для быстрого поиска псевдонимов в системе. Есть, по крайней мере, два важных псевдонима, которые должны быть установлены в файле */etc/aliases*. Если раньше у вас на машине работала Sendmail, то эти псевдонимы, возможно, уже правильно установлены, но проверьте, чтобы в вашем файле были записи для `root` и `postmaster`, указывающие на реального пользователя, получающего почту в вашей системе. После проверки псевдонимов выполните команду *newaliases*, которая перестроит файл индекса в формате, необходимом для Postfix.

Файл */etc/postfix/main.cf* содержит много параметров, но среди них есть лишь несколько важных, которые нужно проверить, прежде чем запускать Postfix. Мы расскажем о них в этом разделе. Если вы установили Postfix из готового дистрибутива, то эти параметры, возможно, уже правильно установлены. Не исключено, что значения Postfix по умолчанию окажутся действенными в вашей системе, но для верности отредактируйте файл */etc/postfix/main.cf*.

myhostname

Это полное имя хоста для вашей системы. По умолчанию Postfix использует имя, которое возвращает функция `gethostname`. Если это значение не полностью квалифицировано и вы не установили этот параметр, Postfix не запустится. Проверить это имя можно, выполнив команду *hostname*. Будет правильным задать здесь явно полное имя своего хоста:

```
myhostname = halo.example.org
```

mydomain

Задает имя домена для этой системы. Это значение используется затем по умолчанию в других местах. Если не задать его явно, Postfix использует доменную часть `myhostname`. Если вы установили `myhostname`, как показано выше, и *example.org* соответствует вашей системе, задавать этот параметр необязательно.

mydestination

Задает список доменных имен, для которых данная система должна принимать почту. Иными словами, значением этого параметра должна быть доменная часть почтовых адресов, для которых вы хотите получать почту. По умолчанию Postfix использует значение, указанное в `myhostname`. Если вы настраиваете свою систему для приема почты для всего своего домена, задайте само имя домена. В качестве значения этого параметра можно использовать переменные `$myhostname` и `$mydomain`:

```
mydestination = $myhostname $mydomain
```

myorigin

Этот параметр используется для добавления имени домена в сообщения, отправляемые локально, если оно в них отсутствует. Например, если пользователь вашей системы отправляет сообщение, содержащее в поле `From:` только локальное имя пользователя, Postfix дописывает это значение к локальному имени. По умолчанию Postfix использует `myhostname`, но если ваша машина обрабатывает почту всего домена, вместо этого нужно задать `$mydomain`:

```
myorigin = $mydomain
```

Некоторые дистрибутивы Linux, содержащие Postfix, настраивают его на использование по умолчанию *procmail*. Программа *procmail* представляет собой отдельного агента доставки почты (MDA), который может фильтровать и сортировать почту при доставке индивидуальным пользователям вашей системы. Если вам нужны возможности, предоставляемые *procmail*, следует тщательно изучить документацию по этой программе, чтобы разобраться, как она взаимодействует с Postfix. Во многих системах, которые не осуществляют фильтрацию почты на уровне MTA, *procmail* образует ненужный дополнительный уровень сложности, поскольку Postfix тоже может осуществлять локальную доставку и обеспечивает часть таких же функций. В вашем дистрибутиве использование *procmail* может быть задано в параметрах `mailbox_command` или `mailbox_transport`. Если вы хотите, чтобы Postfix непосредственно выполнял локальную доставку, можете смело закомментировать любой из этих параметров в своем файле */etc/postfix/main.cf*.

Запуск Postfix

После проверки важных параметров конфигурации, описанных выше, и перестройки индексного файла псевдонимов можно запускать Postfix. Зарегистрировавшись как суперпользователь, выполните команду:

```
postfix start
```

Остановить Postfix можно командой:

```
postfix stop
```

После модификации каких-либо файлов конфигурации Postfix необходимо перезагрузить выполняющийся образ Postfix, выполнив команду:

```
postfix reload
```

Когда работает Postfix, все пользователи на вашей системе должны быть в состоянии отправлять и принимать сообщения электронной почты.

Если у вас есть приложения, зависящие от Sendmail, они должны сохранить работоспособность, и пользоваться командой *sendmail* можно, как обычно. Можно направлять ей сообщения из сценариев через канал и выполнять `sendmail -q` для очистки очереди. Эквивалентной командой Postfix для очистки очереди служит `postfix flush`. Параметры Sendmail, относящиеся к выполнению ее в качестве демона и установке задержки очереди, не работают, поскольку эти функции не поддерживаются командой *sendmail* в Postfix. Все параметры Postfix устанавливаются в двух ее файлах конфигурации. Многие параметры относятся к очередям Postfix. Их описание можно найти на странице руководства для *qmgr*(8).

Ведение журналов в Postfix

После запуска или перезагрузки Postfix нужно проверить по журналу, нет ли сообщений Postfix о каких-либо проблемах. (В большинстве дистрибутивов Linux используется файл */var/log/maillog*, но можно проверить это по файлу */etc/syslog.conf*.) Последние сообщения Postfix можно вывести командой *tail /var/log/maillog*. Поскольку процесс Postfix выполняется длительное время, неплохо периодически просматривать журнал, даже если вы не перезагружали программу. Узнать, не было ли во время работы Postfix каких-нибудь важных сообщений, можно с помощью команды:

```
egrep '(reject|warning|error|fatal|panic):' /var/log/maillog
```

Обычно Postfix информирует о том, что происходит в системе, передавая массу полезных сведений *syslogd*. В Linux *syslogd* по умолчанию осуществляет запись синхронно, т. е. после каждой записи в файл журнала выполняется синхронизация, переносящая все содержимое памяти на диск. В результате может снижаться производительность Postfix (и других процессов). Изменить этот режим по умолчанию можно, если указать дефис перед именем файла журнала в */etc/syslog.conf*. Запись в *syslog.conf* для ведения журнала работы с почтой должна выглядеть так:

```
mail.*                  -/var/log/maillog
```

После внесения изменений в файл конфигурации необходимо, чтобы *syslogd* заново прочел его. Повторную инициализацию можно осуществить командой `killall -HUP syslogd`.

Запуск Postfix во время загрузки системы

Благодаря совместимости Postfix с Sendmail, если система настроена для запуска Sendmail во время инициализации, то, скорее всего, Postfix правильно запустится при начальной загрузке системы. Однако останов системы может не произойти корректно. Большинство дистрибутивов Linux останавливают Sendmail, находя процесс с именем *sendmail* и убивая его. Однако процессы Postfix, будучи во многом совместимыми с Sendmail, не выполняются под именем *sendmail*, поэтому такой останов не срабатывает.

Если вы хотите, чтобы система завершала работу корректно, следует создать свой *rc*-сценарий для Postfix, как описано в главе 5. Для запуска и останова Postfix нужно включить в этот сценарий те же самые команды, которые вы выполняете в командной строке, т. е. *postfix start* и *postfix stop*. Ниже приведен пример сценария, который можно принять за основу. Может потребоваться посмотреть другие *rc*-сценарии в вашей системе, чтобы выяснить, не нужны ли дополнительные системные проверки или выполнение других соглашений, а потом внести в этот пример необходимые изменения:

```
#!/bin/sh
PATH=""
RETVAL=0

if [ ! -f /usr/sbin/postfix ] ; then
    echo "Unable to locate Postfix"
    exit 1
fi
if [ ! -f /etc/postfix/main.cf ] ; then
    echo "Unable to locate Postfix configuration"
    exit 1
fi

case "$1" in
    start)
        echo -n "Starting Postfix: "
        /usr/sbin/postfix start > /dev/null 2>1
        RETVAL=$?
        echo
        ;;

    stop)
        echo -n "Stopping Postfix: "
        /usr/sbin/postfix stop > /dev/null 2>1
        RETVAL=$?
        echo
        ;;

    restart)
        echo -n "Restarting Postfix: "
        /usr/bin/postfix reload > /dev/null 2>1
        RETVAL=$?
        echo
        ;;

    *)
        echo "Usage: $0 {start|stop|restart}"
        RETVAL=1

esac
exit $RETVAL
```

Поместите этот сценарий в */etc/rc.d/init.d* или */etc/init.d*, в зависимости от того, каким дистрибутивом Linux вы пользуетесь. Затем создайте необходи-

мые символические ссылки в каждом из каталогов *rcN.d* для каждого уровня исполнения, на котором нужно запускать Postfix (см. раздел «init, inittab и rc-файлы» в главе 5). Например, если вы хотите запускать Postfix на уровнях исполнения 3 и 5 и останавливать на уровнях исполнения 0 и 6, создайте символические ссылки, как нижеуказанные, для RedHat. В Debian каталоги *rcN.d* находятся непосредственно в */etc*:

```
# cd /etc/rc.d/rc3.d
# ln -s ../init.d/postfix S97postfix
# cd /etc/rc.d/rc5.d
# ln -s ../init.d/postfix S97postfix
# cd /etc/rc.d/rc0.d
# ln -s ../init.d/postfix K97postfix
# cd /etc/rc.d/rc6.d
# ln -s ../init.d/postfix K97postfix
```

Если вы создаете *rc*-сценарий для Postfix, измените настройки системы, чтобы Sendmail не запускалась при инициализации.

Управление ретрансляцией в Postfix

Установка по умолчанию позволяет любой системе в вашей подсети ретранслировать почту через ваш почтовый сервер. Если вы хотите изменить эту установку по умолчанию, то можете указать в параметре `mynetworks` список хостов или сетей, которым вы разрешаете ретранслировать почту через вашу машину. Вы можете задать список IP-адресов или шаблоны сети/маски сети, чтобы соответствующие им клиенты SMTP могли ретранслировать почту. Перечисленные сети или IP-адреса могут быть любыми. Например, если вы хотите ретранслировать почту со своей машины на работе через систему Postfix, установленную у вас дома, можете задать IP-адрес своей машины на работе в конфигурации Postfix дома.

Вот пример разрешения ретрансляции почты из локальной подсети (192.168.75.0/28) и еще одного хоста, расположенного где-то в другом месте:

```
mynetworks = 192.168.75.0/28 10.150.134.15
```

Если вы хотите разрешить ретрансляцию для мобильных пользователей, у которых нет статических IP-адресов, нужно воспользоваться каким-то из механизмов аутентификации SMTP. Postfix может работать с аутентификацией SASL (для чего требуется компиляция Postfix с дополнительными библиотеками и специальная настройка программного обеспечения клиентов) и с pop-before-smtp (что требует работы сервера POP на той же машине для первоначальной аутентификации пользователей).

Важно, чтобы ретрансляция была разрешена только тем пользователям, которым вы доверяете. На заре развития Интернета открытая ретрансляция встречалась сплошь и рядом. К сожалению, широкое распространение спама послужило причиной ликвидации таких свобод. Если ваш MTA не защищен, вы подвергаете себя и другие системы в Интернете опасности злоупотреблений. Распространители спама постоянно разыскивают открытые ре-

трансляторы, и если вы допустите, что он появится в сети, то вскоре его обнаружат. К счастью, установка Postfix по умолчанию настроена на правильное поведение. Однако, когда в конфигурацию Postfix вносится много изменений (особенно, как ни забавно, при настройке для борьбы со спамом), можно случайно открыть систему для ретрансляции.

Если вы хотите, чтобы ваша собственная система Postfix могла ретранслировать почту через другой MTA, задайте IP-адрес сервера ретрансляции с помощью параметра `relayhost`. Postfix обычно самостоятельно определяет, куда доставлять сообщения, основываясь на адресе получателя. Однако если, например, ваша система защищена брандмауэром, может потребоваться, чтобы Postfix передавал все сообщения другому почтовому серверу, осуществляющему фактическую доставку. Когда вы указываете сервер-ретранслятор, Postfix обычно выполняет запрос DNS для получения адреса почтового ретранслятора (MX) для этой системы. Можно отменить этот поиск в DNS, заключив имя хоста в квадратные скобки:

```
relayhost = [mail.example.org]
```

Дополнительные настройки

Описанная выше настройка создает простую установку Postfix для отправки и получения почты пользователями вашей системы. Но Postfix является чрезвычайно гибким MTA с большим количеством параметров конфигурации, в число которых входят поддержка множественных виртуальных доменов, почтовых списков рассылки, блокировка спама и поиск вирусов. Информацию о более сложных видах конфигурации можно найти в страницах руководства, файлах HTML и примерах файлов конфигурации, поставляемых с Postfix.

Получение почты с помощью Fetchmail

Если провайдер хранит вашу почту, пока вы ее не заберете, и вы не хотите использовать свою почтовую программу для загрузки почты, вам потребуется программа, которая забирает почту с компьютера провайдера. Существует множество таких программ. Мы коротко расскажем о *fetchmail*, т. к. она устойчива и гибка и может работать и с POP3 и с IMAP.

Можно загрузить *fetchmail* из своего любимого архива Linux, и, скорее всего, она также входит в состав вашего дистрибутива. Если вы скачали исходный код *fetchmail*, распакуйте, соберите и установите программу в соответствии с инструкциями. Текущей версией на момент написания книги была 5.9.13.

Поведением *fetchmail* можно управлять через параметры командной строки или через файл конфигурации. Неплохо сначала попробовать забрать свою почту, передавая всю необходимую информацию через параметры командной строки, и в случае успеха написать файл конфигурации.

Для примера предположим, что провайдер использует протокол POP3, имя пользователя – `joeuser`, а его пароль – `secret`. Имя машины, на которой работает POP3-сервер, – *mail.isp.com*. Тогда пользователь может забрать свою почту с помощью следующей команды:

```
fetchmail --protocol POP3 --username joeuser mail.isp.com
```

Затем *fetchmail* запросит пароль, и после того как пользователь укажет его правильно, заберет почту и передаст ее MTA для дальнейшей доставки. Здесь мы предположили, что SMTP-сервер использует на локальной машине порт 25. Так и должно быть, если MTA установлен корректно.

Экспериментируя с *fetchmail*, неплохо указать параметр *--keep*. В этом случае *fetchmail* не удалит почту из вашего POP3-ящика. Обычно все сообщения удаляются с жесткого диска провайдера, после того как они доставлены на локальную машину пользователя. Большинство провайдеров ограничивают количество почты, которое можно хранить на их машинах, и если вы не будете удалять почту с сервера провайдера, вы можете дойти до пределов установленных квот. Однако на время тестирования стоит использовать *--keep*, чтобы не потерять ни одного письма.

Используя рассмотренные параметры *fetchmail*, вы сможете в большинстве случаев забрать свою почту. Например, если провайдер использует новый протокол IMAP, просто укажите в командной строке `IMAP` вместо `POP3`. Если провайдер использует необычные установки, вам может понадобиться один из параметров, о которых вы узнаете на страницах руководства для *fetchmail*(1).

Если вы удовлетворены процессом загрузки, можно написать файл конфигурации *fetchmail*, чтобы не вводить все параметры вручную. Этот файл называется *.fetchmailrc* и должен храниться в исходном каталоге. Закончив его редактирование, убедитесь, что он имеет значение разрешений, равное 0600, чтобы никто кроме вас не мог прочитать его, т. к. в нем может храниться ваш пароль:

```
chmod 0600 ~/.fetchmailrc
```

Полный синтаксис конфигурационного файла описан на страницах руководства по *fetchmail*, но обычно вам нужны только очень простые строки, начинающиеся с `poll`. Для того чтобы указать те же данные, что и в командной строке предыдущего примера, но на этот раз вместе с паролем, надо внести следующую строку в файл конфигурации:

```
poll mail.isp.com protocol pop3 username joeuser password secret
```

Теперь можно запускать *fetchmail* без параметров. *fetchmail* не будет запрашивать пароль, т. к. уже знает его из файла конфигурации. Для безопасности можно добавить слово `keep` в строку `poll`.

Использование *fetchmail* с файлом конфигурации имеет еще одно преимущество: можно забирать почту из любого количества почтовых ящиков. Для этого нужно просто добавить другие строки `poll` в файл *.fetchmailrc*, и *fetchmail* по очереди заберет почту с разных серверов.

Как и когда запускать *fetchmail*, зависит от способа соединения с Интернетом. При постоянном подключении или неограниченном доступе можно вызывать *fetchmail* с помощью *cron* с приемлемым интервалом (например, один раз в час). Но при непостоянном и дорогостоящем подключении к Интернету стоит запускать *fetchmail* вручную, когда вам действительно нужно

получить и прочесть почту, чтобы минимизировать время соединения. Наконец, используя PPP для доступа к провайдеру Интернета, можно добавить вызов *fetchmail* из сценария *ip-up*, который запускается сразу после установки соединения с Интернетом. При такой настройке почта будет автоматически забираться, когда вы просматриваете веб-страницу и ваш компьютер дозванивается до провайдера.

Другие вопросы администрирования электронной почты

В этом разделе мы опишем задачи, сервисы и некоторые дополнительные утилиты, участвующие в управлении системой электронной почты.

Обычно следует использовать только один хост для получения всей вашей почты. Можно использовать и более сложные решения, но это не одобряется, поскольку могут образовываться циклы – виртуальное болото из потерянной информации. Почта может петлять через одни и те же машины, пока не наступит «тайм-аут» – будет превышено предельное количество машин, через которые она может пройти.

Регистрация адреса

Чтобы получать почту прямо из сети Интернет, необходимо зарегистрировать доменное имя для вашей системы. Подробнее об этом читайте выше в разделе «Некоторые сведения о DNS».

Обслуживание почтовой системы

Нужно создать задание для *cron*, при выполнении которого будет регулярно проверяться очередь почты (обычно */var/spool/mqueue*) и делаться попытка доставить почту, которая не была ранее доставлена из-за временной недоступности хоста, переполнения файловой системы или по тысяче других причин. Работа с утилитой *cron* обсуждается в разделе «Запуск задач по расписанию с помощью cron» главы 8.

Администратор почты также должен периодически проверять очередь на предмет наличия «застрявших» сообщений:

```
$ mailq -v
```

Эта команда генерирует отчет о сообщениях в очереди. Фактическое имя команды и ее синтаксис могут отличаться в зависимости от выбранного MTA.

Работа с KMail

KMail – очень дружественная пользователю богатая функциями почтовая программа, поставляемая с KDE и хорошо интегрирующая почту с другими утилитами. Например, если в полученном сообщении есть ссылка на веб-страницу, можно щелкнуть по ней, и тогда появится веб-броузер KDE Konqueror, который покажет эту страницу. Если же сообщение содержит вложенный файл MP3, можно щелкнуть по нему, и он будет воспроизведен проигрывателем MP3, входящим в KDE. На рис. 16.4 приведен снимок экрана KMail.

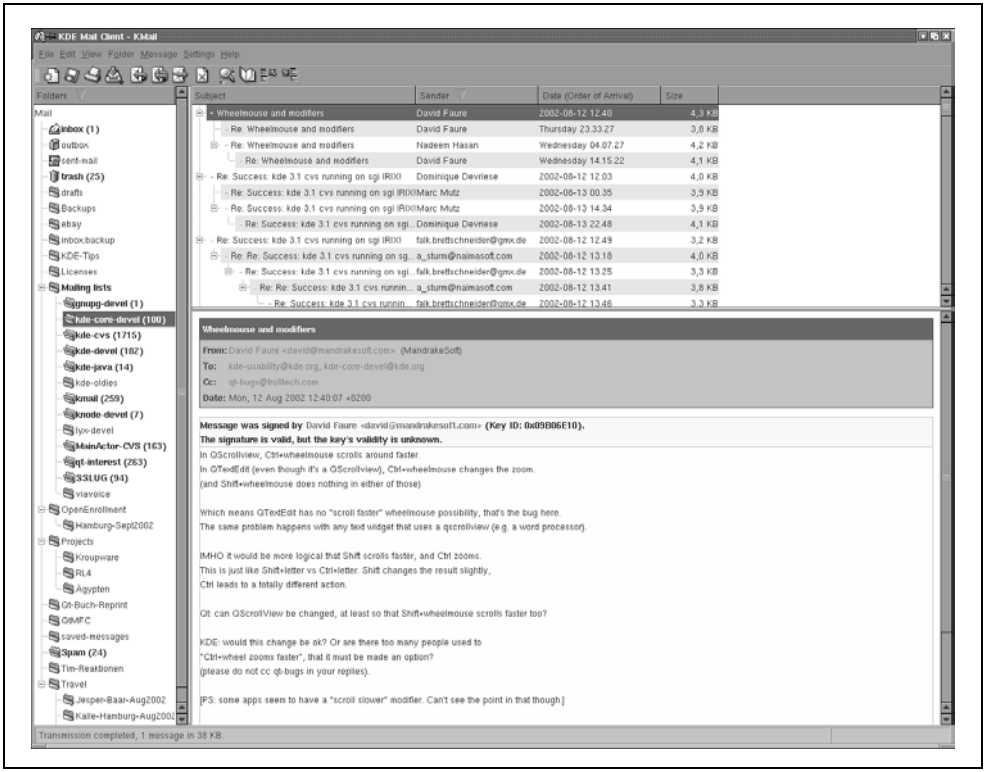


Рис. 16.4. *Почтовая программа KMail*

В KMail есть масса функций и настроек, но мы расскажем только о тех из них, которые позволят вам быстро начать работу, а глубже изучить KMail вы можете самостоятельно. Как видно из рис. 16.4, окно KMail делится по умолчанию на три части. В левой части показано дерево ваших папок (при первом запуске будут, конечно, видны только папки, устанавливаемые по умолчанию). Справа в верхней части показан список сообщений в текущей выделенной папке, а справа в нижней части показано текущее выделенное сообщение. Распределить пространство между этими частями можно иначе, если перетащить разделительную линию между ними. В последних версиях KMail есть еще четвертая часть, с помощью которой можно заглянуть глубже в структуру отдельного сообщения и показать части MIME, из которых оно состоит. Однако по умолчанию такое представление выключено, поскольку не требуется большинству пользователей.

Чтобы начать работать с KMail, необходимо сообщить программе некоторую информацию. Выберите пункт Configure KMail в меню Settings, после чего откройте группу настроек Identity, щелкнув по ее значку. Здесь можно задать различные идентифицирующие данные, например, если вы хотите указывать разный обратный адрес в зависимости от того, пишете ли вы письмо в

качестве служащего компании или как частное лицо. Для начала достаточно заполнить поля Name и Email Address на вкладке General (рис. 16.5).

Затем перейдите в группу настройки сети Network. Здесь нужно создать хотя бы одну учетную запись для исходящей почты и одну – для входящей.

Начнем с исходящей почты, которая настраивается на вкладке Sending (рис. 16.6). Щелкните по кнопке Add. Последует вопрос, будете ли вы использовать SMTP или работать непосредственно с установленной *sendmail*. В большинстве случаев, когда имеется локально установленный MTA, требуется выбрать SMTP. Затем на вкладке General настройки транспорта SMTP задайте имя транспорта (его можно выбрать произвольно, поскольку оно нужно только вам, чтобы в дальнейшем различать настройки, а в сетевых коммуникациях оно не участвует). В любом случае нужно указать имя хоста и порт. Порт почти всегда имеет номер 25, а имя хоста вам должен сообщить провайдер. Если установлен локальный MTA и вы хотите работать с ним,

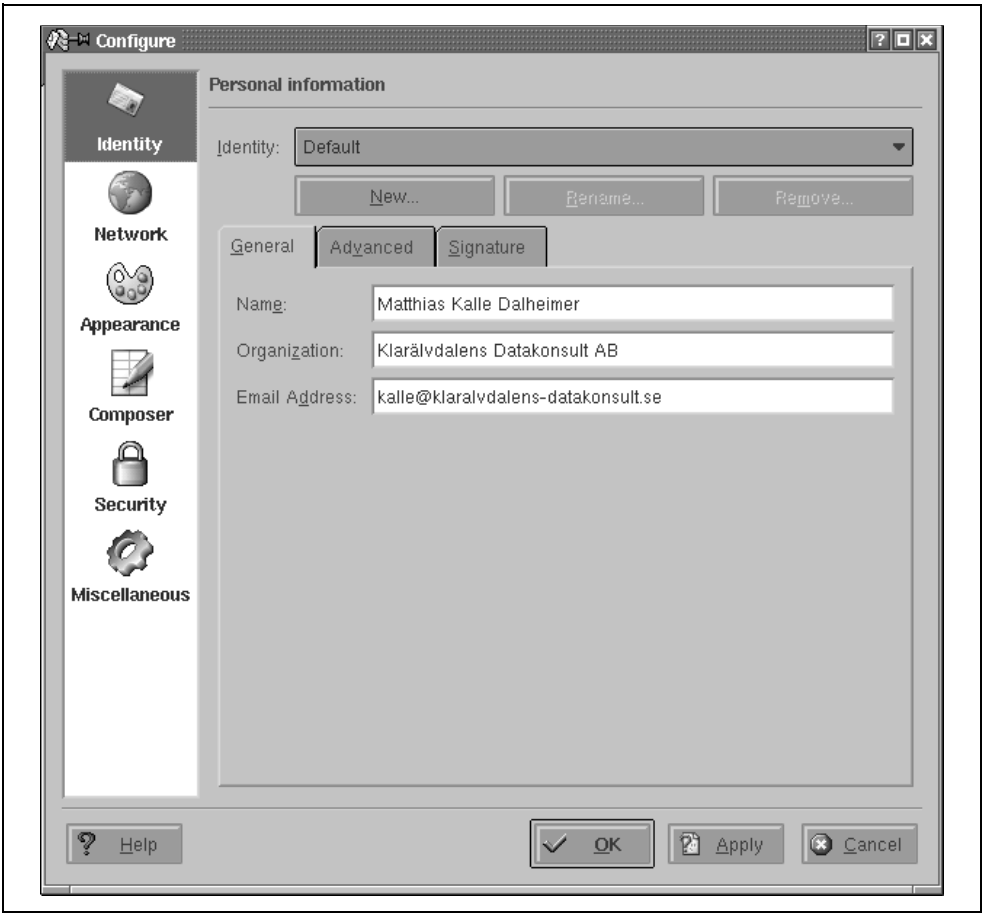


Рис. 16.5. *Настройка идентифицирующих данных в KMail*

просто укажите localhost. Если ваш почтовый сервер требует аутентификации (если не уверены, узнайте об этом у провайдера), установите соответствующий флажок и введите имя регистрации и пароль.

Этих настроек должно хватить для отправки исходящей почты, но мы рекомендуем выполнить некоторые дополнительные действия, чтобы установить максимальную степень защиты. KMail облегчает эту задачу, поскольку автоматически определяет настройки системы безопасности используемого сервера SMTP. Перейдите на вкладку Security и щелкните по кнопке с надписью «Check what the server supports» (узнать, какие настройки поддерживает сервер). KMail проверит соединение с сервером и выберет установки с наибольшей поддерживаемой степенью защищенности. К сожалению, у многих провайдеров почтовые серверы работают вообще без всякой защиты.

Перейдем к настройке приемной части. Закройте все диалоговые окна, чтобы вернуться к группе настройки Network, и выберите закладку Receiving.

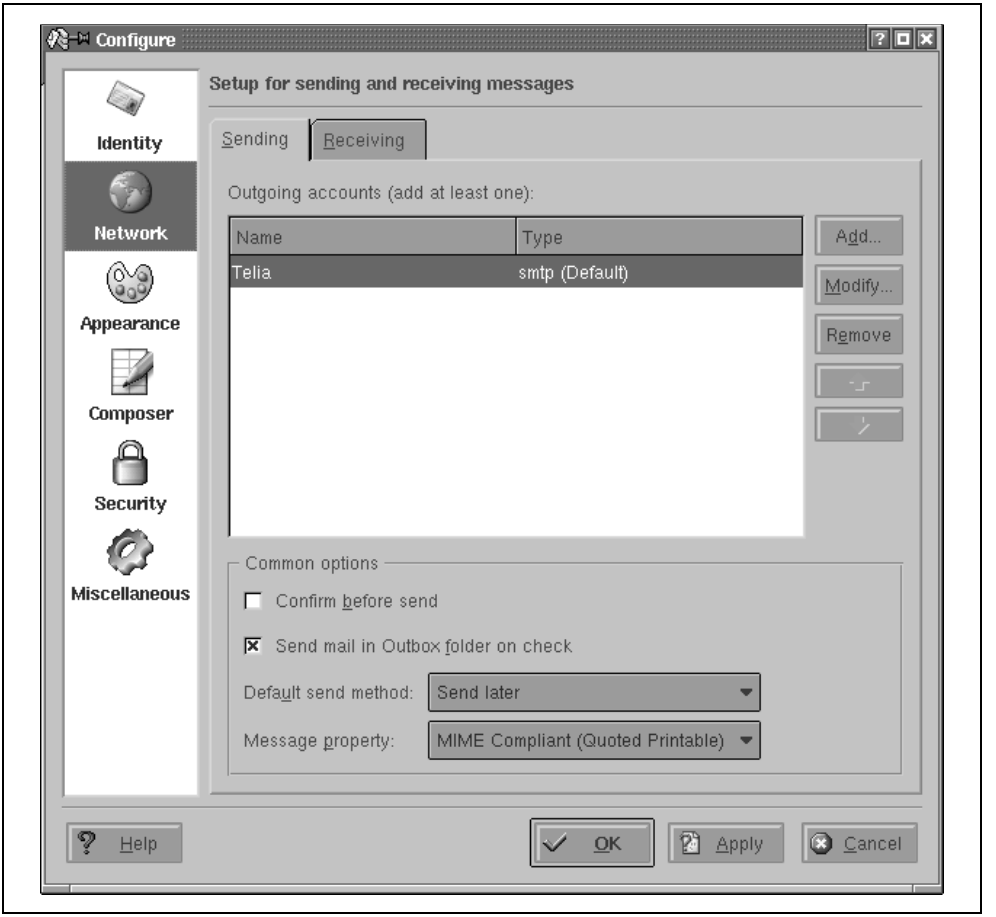


Рис. 16.6. *Настройка в KMail идентификационных данных для исходящей почты*

Здесь можно настроить несколько учетных записей, которые нужно опрашивать. Это удобно, если есть несколько провайдеров, которые хранят вашу электронную почту. Щелкните по кнопке Add и выберите тип почтового сервера. Если у вас работает собственный локальный MTA, нужно выбрать Local Mailbox. Обычно можно принять настройки по умолчанию, предлагаемые на следующей странице (при этом нужно изменить имя «Default» на что-либо более подходящее).

Если вы получаете сообщения непосредственно с сервера провайдера, нужно выбрать POP3 или IMAP, в зависимости от того, какой протокол поддерживает провайдер. В появившемся окне диалога снова введите выбранное вами имя, затем укажите имя регистрации, свой пароль, имя хоста, на котором хранится ваша почта, и порт (обычно 110 для POP3 и 143 для IMAP). Всю эту информацию вам должен сообщить провайдер или системный администратор. Все остальные параметры можно пока оставить, как есть, и поэкспериментировать с ними позднее.

Закройте все диалоги кнопкой OK. Теперь все должно быть готово для получения почты. Для этого выберите пункт меню File→Check Mail. В результате должны быть загружены все сообщения из всех заданных ящиков входящей почты. Если этого не произойдет или будут получены сообщения об ошибках, снова проверьте все значения, которые вы ввели на разных страницах настройки и сравните их с информацией, полученной от вашего провайдера или системного администратора. Типичными ошибками являются опечатки в имени хоста, имени пользователя или пароле.

Чтобы отправить сообщение, нажмите <Ctrl>+<N> или выберите Message→New Message. Откроется окно редактора сообщений, в котором можно ввести адрес получателя, тему и фактическое тело сообщения. Если вы установили несколько идентифицирующих вас наборов данных, можно выбрать тот, который будет использован в данном сообщении. Завершив составление письма, нажмите <Ctrl>+<N>. В зависимости от настроек транспорта исходящей почты сообщение помещается в папку для исходящих сообщений (и будет ожидать там дальнейшей обработки – режим по умолчанию) или сразу передается. Если вы хотите отменить эту настройку для конкретного сообщения, выберите в меню редактора сообщений Message→Queue (поместить в очередь) или Message→Send Now (отправить немедленно).

Сообщения, помещенные в папку для исходящих, по умолчанию не отсылаются автоматически (можно, однако, настроить KMail так, чтобы исходящие сообщения всегда отправлялись при проверке входящей почты). Чтобы отправить все сообщения из папки для исходящих, выберите пункт File→Send Queued в главном меню KMail.

Если возникнут проблемы при отправке сообщений, снова проверьте, не сделаны ли опечатки в настройках. Кроме того, для предотвращения пересылки нежелательной коммерческой почты (так называемого «спама») через свои серверы некоторые провайдеры требуют, чтобы вы проверили свой почтовый ящик на сервере (с предоставлением своего имени пользователя и пароля) и таким образом идентифицировали себя, прежде чем вам будет разре-

шено отправить электронную почту через этот сервер. После проверки входящей почты вам дается некоторое время (обычно 15 минут) на отправку исходящей почты.

Теперь у вас достаточно знаний о KMail, чтобы продолжить его освоение самостоятельно. Возможно, первое, что вам понадобится (особенно если вы каждый день получаете много почты!), – это создать папки с помощью Folder→Create и настроить фильтры с помощью Settings→Configure Filters. В результате вы сможете направлять сообщения с некоторыми характеристиками (например, с определенными адресатами или темами) в отдельные папки. Например, можно помещать все сообщения из почтового списка рассылки в специально выделенную папку.

Работа с Mozilla Mail & News

Mozilla Mail & News – это почтовый клиент, устанавливаемый вместе с веб-броузером Mozilla, если он устанавливается не в самой минимальной конфигурации (включающей только броузер и редактор сообщений). Вполне возможно, что в вашем дистрибутиве уже есть Mozilla. Если нет или вы хотите установить более новую версию, можно загрузить его с *http://www.mozilla.org*.

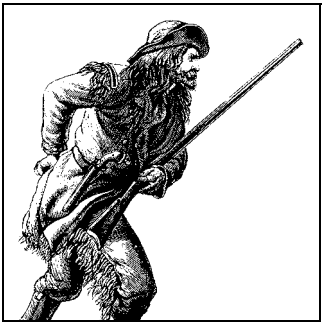
Основные идеи установки и работы Mozilla Mail почти такие же, как для KMail, поэтому мы осветим только различия. Чтобы открыть почтовый клиент, запустите Mozilla и выберите в меню Windows→Mail and Newsgroups. При первом запуске почтовой программы появляется мастер настройки электронной почты. Установите учетную запись электронной почты на первой странице и свои идентификационные данные на второй (работа Mozilla с учетными записями не столь гибкая, как в KMail, поскольку идентификационные данные здесь привязаны к учетным записям, тогда как в KMail идентифицирующие данные можно менять произвольно).

На третьей странице выберите POP или IMAP в качестве протокола для получения входящей почты (локально получать почту с помощью Mozilla Mail & News нельзя, что является большим недостатком) и укажите имена серверов входящей и исходящей почты (укажите `localhost` в обоих случаях, если вы работаете с собственным MTA). Введите остальные сведения на следующих страницах, и вы готовы к запуску Mozilla Mail & News. Размещение информации на экране по умолчанию в точности такое же, как в KMail.

Как и при работе в KMail, первое, что вам, возможно, потребуется в Mozilla Mail & News, – это организовать дополнительные папки и фильтры для сортировки входящих сообщений. Создать новую папку можно, щелкнув правой кнопкой по списку папок и выбрав в появившемся контекстном меню New Folder. Правила для фильтров можно настроить с помощью Tools→Message Filters.

На этом наше обсуждение использования электронной почты в Linux завершается. Как вы видели, существует много параметров, от самых простых до сложных, с помощью которых можно администрировать и обрабатывать ежедневный поток сообщений электронной почты.

17



Основы безопасности

В этой главе мы обсудим основы системы безопасности Linux. К сожалению, тема безопасности приобретает все большее значение, особенно с ростом постоянно подключенных к сети систем, уязвимых для удаленных атак даже тогда, когда на них не работают.

По большей части безопасность является просто практическим проявлением здравого смысла. Многие хорошие приемы являются также и простейшей, но часто игнорируемой практикой, о чем мы и расскажем в первую очередь. Затем мы перейдем к некоторым менее очевидным правилам и в завершение обсудим сложную тему сетевой безопасности. Мы также приведем ряд способов использования сетевых экранов (брандмауэров), с помощью которых простые конфигурации могут быть защищены от сетевых атак.

Общий взгляд на систему безопасности

Не всегда легко получить взвешенное представление о проблемах безопасности компьютерных систем. Средства массовой информации склонны создавать сенсации из историй с взломами системы защиты, особенно если они касаются известных компаний или учреждений. С другой стороны, управление системой безопасности часто оказывается технически сложной и трудоемкой задачей. Многие пользователи Интернета придерживаются точки зрения, что их машины не хранят ценных данных, а потому обеспечение безопасности не является для них проблемой. Другие тратят огромные силы на то, чтобы укрепить защиту своих систем от несанкционированного использования. Независимо от того, к какой крайней позиции в этом спектре вы ближе, следует иметь в виду, что для вашей системы защиты всегда существует риск стать объектом атаки. Есть тьма причин, по которым злоумышленники могут заинтересоваться взломом защиты вашей системы. Ценность имеющихся в системе данных – лишь одна из них. Далее в этой главе мы обсудим и другие причины. Вы сами должны решить, какие затраты можете позволить себе на обеспечение безопасности, но, по нашему мнению, лучше перестраховаться.

Традиционно предметом забот была безопасность систем, доступ к которым осуществлялся с подключенных проводами терминалов или системной консоли. В этой сфере наибольшая опасность обычно исходила изнутри самой организации – владельца системы, и лучшим видом защиты была физическая безопасность, т. е. размещение системных консолей, терминалов и машин в закрытых для доступа помещениях. Даже когда компьютерные системы стали объединяться в сети, доступ к ним оставался очень ограниченным. Получение доступа к имевшимся сетям обходилось очень дорого, либо это были закрытые сети, не допускавшие подключений из произвольной точки.

Распространение Интернета послужило толчком к возникновению проблем сетевой безопасности. Компьютер, подключенный к Интернету, потенциально подвержен угрозе со стороны десятков миллионов машин, разбросанных по всему свету. С облегчением доступа к сети растет и число антиобщественно настроенных людей, сосредоточенных на том, чтобы чинить другим неприятности. Для системного администратора представляет интерес ряд видов асоциального поведения в Интернете. В этой главе нас будут интересовать следующие из них:

Отказ в обслуживании (Denial of service, или DoS)

Этот вид атаки снижает эффективность сервиса, предоставляемого системой, или вообще срывает его работу.

Незаконное проникновение (Intrusion)

При этом виде атаки осуществляется доступ к системе путем подбора паролей или компрометации некоторого сервиса. Получив доступ к системе, нарушитель может искалечить или украсть данные либо воспользоваться системой для организации атак на другие машины.

Подслушивание (Snooping)

Этот вид атаки состоит в перехвате данных других пользователей, паролей или другой конфиденциальной информации. Иногда при этом виде атаки осуществляется и подмена данных. Обычно осуществляется путем перехвата данных, передаваемых в сети, но может осуществляться и путем компрометации системы с целью перехвата библиотечных или системных вызовов, несущих секретную информацию (например, пароли).

Вирусы, черви и трояны (Viruses, worms и Trojan Horses)

Любая такая атака состоит в том, чтобы заставить пользователя выполнить программу, созданную атакующим. Эти программы можно получить по электронной почте, с веб-сайта или даже внутри некой другой, по виду безвредной программы, загруженной из Интернета и установленной локально.

Атака типа DoS обычно заключается в генерировании огромного количества запросов к сервису, предоставляемому системой. Такая повышенная активность может привести к нехватке системной памяти, мощности процессорной обработки или пропускной способности сети. В результате система отказывается выполнять последующие запросы либо скорость ее работы стано-

вится ниже допустимой. Для действенности данной атаки злоумышленник должен воспользоваться ошибками в проектировании сервиса или суметь сгенерировать запросы в количестве, значительно превышающем мощность сервиса.

Более коварной формой DoS-атаки является распределенная атака на отказ в обслуживании (DDoS). В этом варианте генерация запросов к сервису организуется с большого числа машин. Это увеличивает разрушительное действие DoS-атаки по двум направлениям: цель оказывается поражена огромным объемом трафика, а злоумышленник может скрыться среди тысяч ничего не подозревающих участников атаки. Использование большого числа машин, с которых ведется атака, особенно осложняет взятие ведущихся DDoS-атак под контроль и исправление последствий. Даже те, кого не заботит состояние собственных данных, должны принимать меры защиты против таких атак, чтобы снизить риск стать невольным соучастником в какой-либо DDoS-атаке.

Второй вид атаки, который иногда называют взломом, или крекингом, (*cracking*[1]), чаще всего ассоциируется в сознании людей с безопасностью. Многие компании и учреждения хранят секретные данные в компьютерных системах, доступных через сеть. Обычный предмет беспокойства среднего пользователя Интернета – это хранение данных кредитных карточек веб-сайтами. Когда дело касается денег, у нечестных людей возникает побуждение получить доступ к таким конфиденциальным данным и украсть их или незаконно использовать.

Иногда для получения неавторизованного доступа или срыва работы сервисов используются очень изобретательные методы, хотя и не являющиеся этичными. Для разработки алгоритма вторжения часто требуется хорошее знание взламываемой системы, позволяющее использовать ее слабость. Часто обнаруженный способ вторжения оформляется в виде так называемого *rootkit*, набора программ или сценариев, которыми каждый, кто обладает лишь элементарными знаниями, может воспользоваться для проникновения через пробел в системе защиты. Подавляющее большинство атак незаконного проникновения осуществляется так называемыми «script kiddies», пользующимися такими готовыми инструментальными наборами, не имея никакого представления о системах, которые они атакуют. К счастью, системному администратору обычно легко защитить систему от таких хорошо известных атак. В этой главе мы обсудим различные способы защиты своей системы.

---

[1] Термины *cracking* и *hacking* (крекинг и хакинг) часто путают в обиходе. В то время как *cracking* осуществляет аморальное или незаконное поведение (такое как компрометация системы защиты), *hacking* – общий термин, относящийся к программированию, ковырянию в чем-то, проявлению большого интереса. В популярных изданиях часто «хакерством» называют то, что в действительности является «крекерством». Сообщество Linux стремится восстановить положительный подтекст «хакерства».

Первые шаги в организации защищенной системы

Есть ряд самых элементарных вещей, которые можно сделать, чтобы защитить Linux-систему от самых элементарных угроз безопасности. Конечно, в зависимости от конкретной конфигурации системы, способов ее применения и т. д. эти действия могут оказаться сложнее, чем описываемые здесь простые настройки. В этом разделе мы кратко расскажем об основных механизмах защиты Linux от большинства стандартных атак – это базовый подход, используемый одним из авторов при установке новой системы.

Выключение ненужных сетевых демонов

Первым шагом в защите Linux-машины является останов или отключение всех сетевых демонов, которые вам не требуются. В целом, любой сетевой порт, на котором система ожидает соединений, представляет собой риск, поскольку может существовать эксплойт (способ использования слабости в системе защиты) против демона, использующего этот порт. Быстро узнать, какие порты открыты, можно с помощью *netstat -an*, как показано ниже (некоторые строки мы укоротили):

```
# netstat -an
Active Internet connections (servers and established)
Proto Recv-Q Send-Q Local Address           Foreign Address         State
tcp        0      0 0.0.0.0:7120            0.0.0.0:*               LISTEN
tcp        0      0 0.0.0.0:6000            0.0.0.0:*               LISTEN
tcp        0      0 0.0.0.0:22              0.0.0.0:*               LISTEN
```

Мы видим, что система ожидает соединений на портах 7120, 6000 и 22. Из файла */etc/services* или с помощью параметра *-p* команды *netstat* часто можно узнать, какие демоны связаны с этими портами. В данном случае это сервер шрифтов X, сервер X Window System и демон *ssh*.

Если вы обнаружите много других открытых портов для таких демонов, как *telnetd*, *sendmail* и т. д., спросите себя, действительно ли вам нужно, чтобы работали эти демоны. Время от времени объявляется о появлении эксплойтов для различных демонов, и если вам не удается старательно следить за этими обновлениями, касающимися безопасности, ваша система может оказаться уязвимой для атаки. Кроме того, *telnetd*, *ftpd* и *rshd* осуществляют передачу паролей для аутентификации открытым текстом через Интернет. Гораздо лучше применять *sshd*, который шифрует передаваемые данные и использует более сильный алгоритм аутентификации. Даже если вы не применяете *telnetd*, оставлять его работающим на вашей машине не очень разумно, поскольку не исключено, что кто-нибудь найдет способ проникновения в систему с его помощью.

Для останова сервисов обычно достаточно отредактировать соответствующие файлы конфигурации, зависящие от вашего дистрибутива, и перезагрузить систему (чтобы гарантировать, что их не стало). В системах Red Hat, на-

пример, многие демоны запускаются сценариями, расположенными в каталоге */etc/rc.d/init.d*. Помешать запуску соответствующих демонов можно, переименовав или удалив эти сценарии. Другие демоны запускаются демонами *inetd* или *xinetd* в ответ на входящие сетевые соединения. Модифицировав их файлы конфигурации, можно ограничить количество демонов, выполняющихся системой.

Если работа некоторого сервиса на вашей машине абсолютно необходима (например, X-сервера!), постарайтесь запретить соединения с этим сервисом непрошенным хостам. Например, надежнее всего разрешить соединения *ssh* только определенным доверенным хостам, например машинам из вашей локальной сети. В случае сервера X и сервера шрифтов X, которые работают на многих настольных машинах Linux, обычно нет оснований разрешать соединения с ними кому-либо, кроме самого локального хоста. Ограничение соединений с этими демонами можно выполнить с помощью TCP-оболочек или IP-фильтрации, которые будут описаны в этой главе.

«Горячая десятка» ошибок, которые делать нельзя

Мы сделали заявление, что безопасность – это, в основном, просто здравый смысл, так в чем он заключается в данном случае? В этом разделе мы приводим наиболее частые ошибки в организации защиты. (В действительности в этом списке нет 10 ошибок, но их достаточно, чтобы воспользоваться стандартной фразой «top 10».) Систематически избегать их все несколько труднее, чем может показаться вначале.

Не используйте простые или легко угадываемые пароли

Никогда не используйте пароль, который совпадает или похож на ваш ID пользователя, имя, дату рождения, название компании или кличку вашей собаки. Если вы радиолюбитель, не используйте свой позывной; если вы любите машины, не используйте название фирмы/модели или регистрационный номер своей машины – идея должна быть понятна. Ваши пароли не должны быть простыми словами, которые можно найти в словаре. Лучшие пароли – бессмысленные строки. Есть одно хорошее правило: использовать пароль, основанный на простом правиле и фразе, которую вы можете запомнить. Например, можно выбрать такое правило: выбрать последнюю букву каждого слова во фразе «Mary had a little lamb, its fleece was white as snow». Получится пароль `ydaebsesesw`, который не так легко угадать, но легко запомнить. Другой распространенный прием – использование в пароле цифр и символов пунктуации; на практике некоторые программы *passwd* этого требуют. Еще лучше комбинировать оба эти приема.

Не пользуйтесь учетной записью root без необходимости

Одна из причин, по которым многие распространенные настольные операционные системы, например Windows, столь уязвимы для атак «вирусов», распространяемых через электронную почту, и им подобных, заключается в отсутствии всесторонней системы прав доступа. В таких сис-

темах любому пользователю разрешены доступ к любому файлу, выполнение любой программы и перенастройка системы любым образом. Из-за этого легко вынудить пользователя выполнить программу, которая может нанести системе реальный ущерб. Напротив, модель безопасности Linux предоставляет право выполнения большого числа привилегированных задач, таких как установка нового программного обеспечения или модификация файлов конфигурации, одному пользователю — `root`. Не поддавайтесь соблазну всегда использовать учетную запись `root`! В таком случае вы отказываетесь от одного из самых мощных средств защиты против вирусов и троянов (не говоря уже о случайных командах *rm -rf* *!). Всегда регистрируйтесь как обычный пользователь и применяйте команды *su* или *sudo*, чтобы временно получить права `root` при необходимости выполнить привилегированную задачу. Есть и еще одно дополнительное преимущество в таком ограниченном использовании учетной записи `root`: регистрация в журнале. Команды *su* и *sudo* при своем вызове помещают в файл системного журнала записи с указанием ID пользователя, выполнившего *su* или *sudo*, а также дату и время выполнения этой команды. Это очень помогает проследить, кто и когда пользовался правами `root`.

Не раскрывайте свои пароли

Никогда никому не сообщайте свой пароль. Это также означает, что вы не должны записывать свой пароль на «стикерах», приклеиваемых к монитору, или в тетради, хранящейся в верхнем ящике стола. Если вам нужно предоставить кому-либо временный доступ к своей системе, создайте для него отдельную учетную запись. Это даст вам некоторую возможность проследить за тем, что делали на вашей машине, и затем легко убрать последствия. Если вам действительно необходимо разрешить кому-то войти на вашу машину как `root`, воспользуйтесь командой *sudo*, позволяющей предоставлять пользователям права `root` без раскрытия пароля суперпользователя.

Не доверяйте безоглядно загружаемым исполняемым модулям

Очень удобно устанавливать на машине программы, получаемые в двоичном виде, но прежде чем запускать исполняемые модули, следует задуматься о том, насколько вы им доверяете. Если вы устанавливаете программные пакеты, загруженные прямо с официального сайта вашего дистрибутива или серьезного сайта разработчиков, то можно быть достаточно уверенным в безопасности программ. Если вы получаете их с неофициального зеркального сайта, следует подумать о том, насколько вы доверяете его администраторам. Не исключено, что кто-то распространяет модифицированное программное обеспечение с «черным входом», который позволит получить несанкционированный доступ к вашей машине. Такой подход несколько параноидален, но он принят многими организациями, распространяющими Linux. Например, Debian разрабатывает средства для проверки отсутствия модификации программных пакетов. Для других дистрибутивов готовятся аналогичные меры защиты целостности пакетов программного обеспечения.

Если вы хотите установить и выполнить программу, которую получили в двоичном виде, то должны принять некоторые меры, снижающие риск. К сожалению, этими приемами непросто воспользоваться новичкам в среде Linux. Во-первых, никогда не выполняйте программы, которым не вполне доверяете, в качестве `root`, если только права `root` не требуются программе специально. В результате ущерб, возможно наносимый программой, будет ограничен только файлами и каталогами конкретного пользователя. Если вы хотите получить некоторое представление о том, что делает программа, не выполняя ее, можно применить к исполняемым модулям команду *strings*. Она покажет все жестко закодированные строки, которые есть в коде. Следует поискать в них обращения к важным файлам или каталогам, таким как */etc/passwd*, */bin/login* и т. д. Обнаружив ссылку на важный файл, следует подумать о том, согласуется ли она с назначением рассматриваемой программы. Если нет, будьте осторожны. При достаточной технической подготовке можно сначала запустить программу и посмотреть, что она делает, с помощью таких программ, как *strace* или *ltrace*, которые показывают системные и библиотечные вызовы, осуществляемые программой. Поищите в трассировке следы необычных операций с файловой системой или сетью.

Не упускайте из виду файлы журналов

Файлы системных журналов – ваши друзья, которые могут многое рассказать о происходящем в системе. Вы можете найти в них сведения о том, когда происходили сетевые соединения с вашей системой, кто пользовался учетной записью `root`, были ли неудачные попытки входа в систему. Файлы журналов нужно периодически проверять и научиться определять, где нормальная работа, а где, что еще важнее, ненормальная. Обнаружив что-то необычное, проведите расследование.

Не допускайте слишком сильного устаревания системы

Необходимо поддерживать программное обеспечение на машине в достаточно обновленном состоянии. Та машина Linux с ядром версии 1.2, которая стоит у вас в углу и годами отлично обслуживает принтеры, может послужить хорошей темой разговора на вечеринке, но в любой момент может стать причиной инцидента в системе защиты. Своевременное обновление программного обеспечения гарантирует, что в нем будут устранены все выявленные программные ошибки и бреши в системе безопасности. Большинство дистрибутивов Linux предоставляет ряд пакетов, являющихся исключительно исправлениями в системе защиты, поэтому поддержание защищенности системы не связано с такими проблемами, как изменение файлов конфигурации и функций. Необходимо по крайней мере следить за появлением таких обновлений.

Не забывайте о физической безопасности

Проникновение через систему защиты компьютерных систем чаще всего осуществляется сотрудниками тех организаций, в которых они установлены. Самая тщательная настройка программной защиты не поможет, если кто-то может подойти к машине и запустить с дискеты код эксплой-

та. Если в вашей машине установлены BIOS или системный PROM, позволяющие изменять порядок устройств для начальной загрузки, сделайте так, чтобы загрузка с дискет и CD-ROM происходила после загрузки с жесткого диска. Если настройку BIOS можно защитить паролем, воспользуйтесь этой функцией. Если можно повесить замок на корпус машины, сделайте это. Если можно поставить машину в физически защищенном месте, например в запертом помещении, еще лучше.

Настройка TCP wrappers

Ранее мы говорили о том, что подключение машины к сети существенно увеличивает угрозу атаки на нее. Разобравшись с тем, что в отношении защиты советует здравый смысл, можно более пристально взглянуть на проблемы сетевой безопасности. Сейчас мы опишем простой, но действенный метод уменьшения риска нежелательного доступа к системе через сеть с помощью средства, называемого *TCP-оболочкой* (TCP wrapper). Этот механизм создает «оболочку», в которую помещается имеющийся сервис, например почтовый сервер, чтобы осуществлять фильтрацию подключений к нему из сети, отказывая в соединении неавторизованным сайтам. Это простой способ добавить управление доступом к сервисам, которые разрабатывались без его учета. Чаще всего этот метод применяется вместе с демонами *inetd* или *xinetd*.

TCP wrappers – некий эквивалент охранников, или «вышибал», которых можно встретить у входа на крупные собрания людей или в ночные клубы. Когда вы приходите в такое место, то первым делом встречаете охранника, который может поинтересоваться вашим именем и адресом. Затем он сверяется со списком приглашенных, и если находит вас в нем, то отодвигается и пропускает вас внутрь.

Когда осуществляется соединение с сервисом, защищенным TCP wrapper, вы прежде всего сталкиваетесь с этой оболочкой. Она интересуется именем хоста или адресом, откуда исходит соединение, и проверяет его по списку, ограничивающему доступ. Если источник указан в списке, оболочка разрешает сетевому соединению доступ к фактической программе-демону.

Использовать TCP wrappers можно двумя способами в зависимости от дистрибутива Linux и конфигурации. Если управление сервисами осуществляется демоном *inetd* (проверьте, существует ли файл */etc/inetd.conf*), то TCP wrappers реализуются с помощью специального демона под названием *tcpd*. Если же используется демон *xinetd* (проверьте, существует ли каталог */etc/xinetd.d*), то обычно настраивают *xinetd* для непосредственного использования TCP wrappers. Оба случая будут рассмотрены в последующих разделах.

Использование TCP wrappers с inetd

Если ваша система запускает сетевые сервисы с помощью демона *inetd*, то для использования TCP wrappers может оказаться необходимым отредактировать файл */etc/inetd.conf*. Возьмем в качестве примера демон сервиса finger, *in.fingerd*. Идея состоит в том, чтобы вместо демона *in.fingerd* посредст-

вом *inetd* запускался демон *tcpd*. Демон *tcpd* выполняет операцию оболочки TCP, а затем запускает вместо нее *in.fingerd*, если соединение разрешено.

Настройка для оболочек TCP требует очень простой модификации */etc/inetd.conf*. Для демона finger в этом файле может существовать такая запись:

```
# /etc/in.fingerd демон finger
finger    stream  tcp  nowait  root  /usr/sbin/in.fingerd  in.fingerd
```

Чтобы защитить демон finger с помощью *tcpd*, модифицируйте запись в */etc/inetd.conf* следующим образом:

```
# /etc/in.fingerd демон finger
finger    stream  tcp  nowait  root  /usr/sbin/tcpd /usr/sbin/in.fingerd
```

Мы организовали выполнение команды *tcpd* вместо фактической команды *in.fingerd*. Полный путь к демону finger передается *tcpd* в качестве аргумента, и *tcpd* использует этот аргумент для запуска реального демона после проверки того, что доступ разрешен.

Такую модификацию нужно осуществить для каждой программы-демона, которую вы хотите защитить. Большинство систем Linux уже настроено на запуск *tcpd*, поэтому необходимость в этих изменениях может отсутствовать.

Использование TCP wrappers с xinetd

xinetd служит заменой *inetd*, принятой в некоторых дистрибутивах (например, Red Hat). В большинстве случаев *xinetd* содержит встроенную поддержку TCP wrappers, поэтому вам потребуется только модифицировать файлы конфигурации TCP-оболочки (*/etc/hosts.allow* и */etc/hosts.deny*), что будет описано в следующем разделе. Если вы устанавливаете *xinetd* сами, необходимо скомпилировать поддержку оболочек TCP, как описано в документации по *xinetd*.

/etc/hosts.allow и /etc/hosts.deny

TCP wrappers используют два файла конфигурации, */etc/hosts.allow* и */etc/hosts.deny*. Эти файлы используются для задания правил доступа к каждому сетевому демону, защищенному TCP wrappers. Эти файлы подробно описаны на странице руководства *hosts_access*, но мы покажем здесь их механизм, поскольку в стандартном случае они довольно просты.

Когда вызывается оболочка TCP, она получает IP-адрес подключающегося хоста и пытается определить его имя с помощью обратного поиска DNS. Затем она справляется в файле */etc/hosts.allow*, разрешен ли именно этому хосту доступ к запрашиваемому сервису. Если хост найден в списке, ему разрешается доступ, и вызывается фактический сетевой демон. Если соответствия в файле */etc/hosts.allow* не найдено, по файлу */etc/hosts.deny* проверяется, запрещен ли доступ конкретно этому хосту. Если найдено соответствие в этом файле, соединение закрывается. Если хост не найден ни в том, ни в дру-

гом файле, то доступ разрешается. Эта простая технология достаточно мощна, чтобы удовлетворить большинству требований ограничения доступа.

Синтаксис *hosts.allow* и *hosts.deny* довольно прост. Каждый файл содержит набор правил. Каждое правило обычно занимает одну строку, но может быть разбито на несколько строк с помощью обратной косой черты в конце строки. Общий вид правила следующий:

```
daemon_list : client_list : shell_command
```

daemon_list представляет собой разделенный запятыми список демонов, к которым относится данное правило. Демоны указываются с помощью основного имени их команды, т. е. фактически выполняемой программы демона, осуществляющей обработку запрошенного сервиса. client_ list представляет собой разделенный запятыми список имен хостов или IP-адресов, к которым применяется правило. Мы продемонстрируем это ниже на примере. Задание shell_command необязательно и определяет команду, которая будет выполнена, если правило применимо. С помощью такой команды можно, например, регистрировать в журнале входящие соединения.

daemon_list и client_list могут содержать шаблоны, соответствующие группе демонов или хостов, что позволяет не перечислять каждый из них отдельно. Кроме того, есть ряд лексем, упрощающих чтение и составление правил. Правила составления шаблонов довольно сложны, поэтому мы не станем их подробно обсуждать, а отошлем читателя к странице руководства *hosts_access*.

Начнем с простого файла *hosts.deny*, который выглядит так:

```
# /etc/hosts.deny
ALL: ALL
```

Первая строка – комментарий. Следующая строка является правилом, которое интерпретируется так: «запретить доступ ко всем (ALL) сервисам со всех (ALL) хостов». Если при этом файл */etc/hosts.allow* пуст, то результатом такого правила будет запрещение доступа к чему бы то ни было со всех машин в Интернете – включая локальный хост! Чтобы решить эту проблему, можно внести в файл простое изменение:

```
# /etc/hosts.deny
ALL: ALL EXCEPT localhost
```

Таким правилом можно надежно пользоваться почти во всех случаях: это обеспечивающее безопасность значение по умолчанию. Напомним, что к правилам */etc/hosts.allow* система обращается *раньше*, чем к правилам */etc/hosts.deny*, поэтому, добавляя правила в *hosts.allow*, можно отменить это значение по умолчанию в *hosts.deny*. Допустим, например, что мы хотим предоставить любой системе в Интернете доступ к демону finger. Для этого мы добавим в */etc/hosts.allow* правило, которое выглядит так:

```
# /etc/hosts.allow
in.fingerd: ALL
```

Чаще всего оболочки TCP применяют для того, чтобы ограничить группу хостов, которым разрешен доступ к некоторому сервису. Хосты могут указы-

ваться с помощью IP-адреса, имени хоста или некоторого шаблона адреса или имени (например, чтобы задать группу хостов). Допустим, что мы хотим сделать демон finger доступным только небольшой группе доверенных хостов. В таком случае модифицируем наш файл *hosts.allow* следующим образом:

```
# /etc/hosts.allow
in.fingerd: spaghetti.vpasta.com, .vpizza.com, 192.168.1.
```

В этом запросе мы разрешили FTP-запросы с хоста с именем *spaghetti.vpasta.com*, а также со всех хостов в домене *vpizza.com*, а также с любой системы, IP-адрес которой начинается с `192.168.1`.

Важно разобраться с правилами соответствия хостов и IP-адресов в *hosts.allow* и *hosts.deny*, в которых присутствие и положение точки является критическим. Если шаблон начинается с точки, предполагается, что это имя домена, к которому должна принадлежать система, посылающая запрос. Если шаблон оканчивается точкой, предполагается, что это шаблон IP-адреса. Есть и другие способы задания групп хостов, в том числе с помощью сетевых групп NIS и явных масок IP-адресов. Полное описание синтаксиса этих шаблонов есть на странице руководства *hosts_access*.

Брандмауэры: фильтрация IP-пакетов

TCP wrappers можно использовать для того, чтобы разрешить доступ к определенным сервисам на машине определенной группе хостов, но часто желательно организовать более тонкий контроль над пакетами, поступающими в данную систему (или посылаемыми из нее!). Нужно также учесть, что TCP wrappers действуют только в отношении сервисов, настроенных на работу с *inetd* или *xinetd*, тогда как некоторые сервисы (например, *sshd* в некоторых системах) являются «автономными» и обладают собственными функциями управления доступом. Есть и другие сервисы, в которых не реализовано никакого контроля доступа, поэтому необходим другой уровень защиты, если мы хотим управлять соединениями с этими сервисами.

В настоящее время для пользователей Интернета стало обычным делом защищать себя от сетевых атак с помощью технологии, называемой *IP-фильтрацией*. При IP-фильтрации ядро изучает каждый передаваемый или принимаемый пакет и прежде чем пропустить его, решает, разрешить ли его прохождение, отбросить или каким-то образом модифицировать. IP-фильтрацию часто называют «брандмауэром» (firewalling), поскольку путем тщательной фильтрации пакетов, приходящих в систему или покидающих ее, создается некая «стена» между машиной и остальной частью Интернета. IP-фильтрация не защитит вас от вирусов или троянов, как и от дефектов в программах, но она создает защиту против многих видов сетевых атак, например некоторых видов DoS-атак и IP-фальсификации («спуфинга», пометки пакетов как пришедших не от той системы, которая их в действительности отправила). IP-фильтрация также предоставляет дополнительный уровень контроля, предотвращающий доступ к системе нежелательных пользователей.

Для того чтобы заставить систему IP-фильтрации действовать, нужно знать, каким пакетам разрешать движение, а какие отвергать. Обычно решение о фильтрации пакета принимается на основании его заголовка, в котором содержатся такие данные, как IP-адреса отправителя и получателя, тип протокола (TCP, UDP и т. д.) и номера портов отправителя и получателя (указывающие конкретный сервис, для которого предназначен пакет). Различные сетевые сервисы используют различные протоколы и номера портов. Например, большинство веб-серверов принимают запросы TCP на порт 80. Если мы хотим отфильтровать весь входящий трафик HTTP от нашей системы, нужно организовать IP-фильтр, который отклоняет все пакеты TCP, адресованные порту 80.

Для решения некоторых задач фильтрации недостаточно изучить только заголовок пакета, а требуется проанализировать фактические данные, которые несет пакет. Эту технологию иногда называют «stateful inspection», или проверкой пакетов с учетом состояния протокола, поскольку пакет рассматривается не изолированно, а в контексте текущего сетевого соединения. Пусть, например, нужно разрешить пользователям нашей локальной сети работать с FTP-серверами, находящимися за ее пределами. FTP – сложный протокол, который использует одно соединение TCP для отправки серверу команд, а другое – для передачи фактических данных. К сожалению, в спецификации FTP не определен конкретный номер порта для передачи данных, поэтому клиент и сервер должны договориться о номерах портов с помощью ряда команд. Если не пользоваться технологией stateful inspection, то для осуществления передачи данных по FTP потребуется разрешить подключения TCP к произвольным портам. Stateful inspection решает эту проблему путем анализа переговоров между клиентом и сервером относительно номеров портов и разрешения прохождения пакетов TCP через выбранный ими порт.

IP-фильтры реализованы в ядре Linux, в котором есть код, проверяющий каждый принимаемый и передаваемый пакет и применяющий правила фильтрации, которые решают судьбу этого пакета. Правила настраиваются с помощью утилиты, которая в командной строке принимает от пользователя аргументы и преобразовывает их в спецификации фильтра, сохраняемые ядром и используемые им в качестве правил.

Включенная в ядро Linux IP-фильтрация прошла в своем развитии три поколения, у каждого из которых свой механизм настройки. Первое поколение носило название *ipfw* (от «IP firewall») и обеспечивало основные возможности фильтрации, но было недостаточно гибким и эффективным в сложных конфигурациях. Сейчас *ipfw* используется редко. Второе поколение IP-фильтров, под названием *IP-цепочек* (*IP chains*), было значительно эффективнее *ipfw* и до сих пор широко используется. Последнее поколение фильтров называется *netfilter/iptables*. *netfilter* – это компонент ядра, а *iptables* – утилита настройки, выполняемая в пространстве пользователя. Оба названия часто используют взаимозаменяемым образом. *netfilter* не только обладает большей гибкостью в настройке, но и допускает расширение. В следующих разделах мы опишем *netfilter* и приведем примеры некоторых простых конфигураций.

Основы netfilter

netfilter входит в ядро Linux версий 2.4.0 и выше. Основная утилита для настройки и показа таблиц фильтрации называется *iptables* и включена во все современные дистрибутивы Linux. Команда *iptables* дает возможность настроить большой и сложный набор правил фильтрации пакетов, и потому у нее много параметров командной строки. Мы расскажем здесь о самых употребимых из них. Страница руководства *iptables* содержит полное их описание.

Для начала заглянем вперед и покажем, к чему мы должны придти:

```
iptables -A INPUT -m state --state NEW -m tcp -p tcp --dport 22 -j ACCEPT
```

Эта команда задает правило фильтрации IP-пакетов для приема новых входящих соединений TCP на порт 22 (сервис *ssh*) нашей локальной системы. Она также использует модуль расширения *state* для отслеживания соединений. Ниже будет рассказано, как все это работает.

Важным понятием *netfilter* является *цепочка* (*chain*), состоящая из списка правил, применяемых к пакетам, когда они приходят в систему, исходят из нее или проходят насквозь. По умолчанию в ядре определены три цепочки, но администратор может создавать новые цепочки и присоединять их к уже имеющимся. Три готовые цепочки следующие:

`INPUT`

Эта цепочка применяется к полученным пакетам, адресованным локальной системе.

`OUTPUT`

Эта цепочка применяется к пакетам, передаваемым локальной системой.

`FORWARD`

Эта цепочка применяется, когда пакет должен быть маршрутизирован данной системой с одного сетевого интерфейса на другой. Она используется, когда система выступает как маршрутизатор пакетов или шлюз и не является ни источником, ни получателем пакета.

Каждое правило в цепочке связано с группой критериев, определяющих, к каким пакетам применяется правило и какие действия должны быть совершены над пакетами, подпадающими под действие этого правила. Над пакетами могут совершаться такие действия, как принятие пакета (разрешение на прием или передачу), отбрасывание пакета (просто отказ принять или передать его) или передача пакета в другую цепочку. (Последнее полезно при создании пользовательских цепочек, допускающих иерархическое построение сложных правил фильтрации пакетов.) Пакет проходит через все правила цепочки, пока не будет принят, отброшен или не достигнет конца цепочки; если он достигает конца, к нему применяется операция, установленная для цепочки по умолчанию. В качестве действия цепочки по умолчанию можно назначить принятие или отбрасывание всех пакетов.

В Linux *netfilter* поддерживает ряд других интересных операций, допустимых в правилах фильтрации. Одним из важных достоинств *netfilter* является

возможность расширения. Можно разрабатывать расширения, усиливающие функции *netfilter*. Вот некоторые примеры более сложной обработки пакетов:

Регистрация пакетов

Можно создать правила, лишь регистрирующие описание пакета, соответствующего правилу, которое можно впоследствии подвергнуть анализу. Это очень полезно для обнаружения атак и тестирования настроек фильтрации.

Stateful inspection

В *netfilter* есть ряд вспомогательных модулей, поддерживающих технологию stateful inspection, например для управления соединениями FTP, как описано выше.

Преобразование сетевых адресов

Преобразование сетевых адресов (Network Address Translation – NAT), называемое также *сокрытием IP-адресов* (*IP masquerading*), дает возможность изменять IP-адреса и номера портов при прохождении пакетов через цепочку. Чаще всего с помощью NAT осуществляют доступ к Интернету нескольких систем в закрытой сети через один IP-адрес. NAT – сложная тема, которую мы не будем подробно обсуждать, но приведем один простой пример в конце данной главы. Подробнее о NAT можно узнать из *NAT HOWTO* или в книге «TCP/IP. Network Administration»[1] (O'Reilly).

Подсчет пакетов и байтов

В *netfilter* есть счетчики, позволяющие получать статистику применения каждого правила к сетевому трафику, и несколько основанных на ней систем учета. Счетчики можно увидеть при выводе наборов правил командой *iptables* в режиме подробного отчета; мы покажем это в примере 17.3.

Использование команды iptables

С помощью команды *iptables* вносятся изменения в цепочки и наборы правил *netfilter*. Она позволяет создавать новые цепочки, удалять цепочки, выводить правила цепочки, очищать цепочки (т. е. удалять все имеющиеся в цепочке правила) и устанавливать действие по умолчанию для цепочки. *iptables* также дает возможность вставлять, добавлять, удалять и заменять правила в цепочке.

У команды *iptables* большое число аргументов и параметров командной строки, но достаточно использовать ее несколько раз, и синтаксис становится достаточно очевиден. В этом разделе мы расскажем только о самых частых способах применения *iptables*, поэтому некоторые аргументы и параметры в последующем изложении опущены. В частности, мы не обсуждаем здесь цепочки, определяемые пользователем. В табл. 17.1 перечислены аргументы *iptables*, действующие над цепочками, а в табл. 17.2 перечислены аргументы *iptables*, действующие над отдельными правилами.

---

[1] Крэйг Хант «TCP/IP. Сетевое администрирование», 3-е издание. – Пер. с англ. – СПб: Символ-Плюс, 2004.

Таблица 17.1. Операции iptables над цепочками

Аргумент	Описание
-L *цепочка*	Показывает правила в конкретной цепочке или во всех цепочках
-F *цепочка*	Удалить правила из заданной цепочки или всех цепочек
-Z *цепочка*	Обнулить счетчик байтов для заданной цепочки или всех цепочек
-P *цепочка действие*	Установить *действие* действием по умолчанию для заданной цепочки

Таблица 17.2. Операции iptables над правилами

Аргумент	Описание
-A *цепочка спецификация-правила*	Добавить правило в *цепочку*
-D *цепочка номер-правила*	Удалить правило номер *номер-правила* из *цепочки*
-R *цепочка номер-правила спецификация-правила*	Заменить спецификацию для правила *номер-правила*
-I *цепочка номер-правила спецификация-правила*	Вставить в цепочку правило с заданными номером и спецификацией. Если номер не задан, предполагается «1»

Каждое правило фильтрации содержит параметры, описывающие пакеты, соответствующие правилу. Наиболее употребительные параметры правил приведены в табл. 17.3. Восклицательный знак (!) перед параметром обращает его. Например, параметр -dport 80 означает «соответствие порту назначения 80», а параметр -dport ! 80 означает «соответствует любому порту назначения, кроме 80».

Таблица 17.3. Параметры правил iptables

Параметр	Соответствие
-p ! *протокол*	Протокол пакета. Допускаются значения tcp, udp, icmp или all
-s ! *источник/маска*	Адрес источника пакета, указанный как имя хоста или IP-адрес. *маска* задает необязательную маску сети литералом или количеством бит. Например, /255.255.255.0 задает маску сети литералом, /24 задает количество бит в маске
-d ! *источник/маска*	Адрес получателя пакета. Синтаксис такой же, как в адресе источника
--sport ! *порт*	Порт источника пакета. Задается как номер порта или имя сервиса из */etc/services*
--dport ! *порт*	Порт получателя пакета. Синтаксис такой же, как в порте источника
-i ! *интерфейс*	Сетевой интерфейс, через который получен пакет
-o ! *интерфейс*	Сетевой интерфейс, через который пакет будет отправлен

Ряд важных параметров, используемых при создании наборов правил, перечислен в табл. 17.4.

Таблица 17.4. Другие важные параметры iptables

Параметр	Описание
`-v`	Подробный вывод. Полезнее всего при выводе правил по *-L*
`-n`	Показ IP-адресов в числовой форме (т. е. без поиска в DNS)
`-m` *модуль*	Загрузить расширение *iptables* с указанным именем `module`

Помимо параметров соответствия в каждом правиле *netfilter* должно быть указано действие, совершаемое над пакетом, соответствующим правилу. Обычно в правиле указывается, что пакет должен быть принят или отброшен, как описывается ниже. Если для правила не задано действие, счетчики пакетов и байтов для этого правила наращиваются, а пакет передается следующему правилу в цепочке. Благодаря этому можно создавать правила, используемые только для статистики. При задании действия для правила применяется синтаксис:

```
-j target
```

Здесь *-j* происходит от «jump» и означает, что если пакет соответствует этому правилу, происходит переход к действию с именем *target*. Значение *target* может быть следующим:

`ACCEPT`

Разрешить передачу или прием пакета.

`DROP`

Отбросить пакет.

`QUEUE`

Передать пакет на обработку программе в пространстве пользователя.

`RETURN`

При использовании в цепочке, определенной пользователем, осуществляется возврат пакета в «вызвавшую» цепочку. При использовании во встроенной цепочке пакет перебрасывается в ее конец, где над ним выполняется действие, установленное для цепочки по умолчанию.

При использовании параметра *-j* можно задать в *target* имя цепочки, определенной пользователем, что дает возможность определять «вложенную цепочку» правил для обработки этого пакета. Как уже говорилось, *target* со значением `RETURN` используется для возврата пакета из цепочки, определенной пользователем, в «вызвавшую» цепочку.

Создание наборов правил для IP-фильтрации

Часто при реализации фильтрации IP труднее всего решить, чего вы действительно от нее хотите. Можно ли разрешить беспрепятственное установление

исходящих соединений? Следует ли разрешить перемещение пакетов ICMP? Какие сервисы UDP требуются? Что нужно регистрировать в журналах?

Большая проблема при создании правил фильтрации связана с тем, что большинству пользователей непривычно думать на языке адресов, протоколов и номеров портов. Мы чаще думаем в терминах приложений и конечных пользователей. Чтобы строить правила фильтрации, нужно научиться переводить наши требования на высоком уровне в детали низкого уровня, на котором действует фильтрация.

Некоторое понимание того, как работают сервисы, которыми вы управляете с помощью IP-фильтрации, просто необходимо. В первую очередь нужно знать, какой из протоколов TCP или UDP использует сервис и на каком порту. В файле */etc/services* можно часто найти много нужной информации. Например, при поиске в этом файле `smtp` вы обнаружите `tcp/25`, что говорит об использовании протоколом SMTP порта TCP с номером 25. Аналогично при поиске DNS можно найти две записи, в одной из которых указано `udp/53`, а в другой – `tcp/53`; это означает, что сервис использует порт 53, но либо с протоколом TCP, либо с протоколом UDP.

У некоторых протоколов, например FTP, есть две различных записи в */etc/services*. Как говорилось выше, FTP использует один порт для передачи команд (`tcp/21`) и другой – для передачи данных (`tcp/20`). К сожалению, клиенты и серверы FTP могут выбирать любые порты для передачи данных, поэтому FTP представляет собой некоторую проблему для правил фильтрации. К счастью, *netfilter* обеспечивает некоторую помощь благодаря функции, называемой *отслеживанием соединения* (*connection tracking*), и вспомогательному модулю, ориентированному на сервис FTP. Благодаря этому необходимо создать правило только для передачи команд FTP, а *netfilter* автоматически найдет и разрешит работу соединения, осуществляющего передачу данных. Мы продемонстрируем это позднее в примере 17.2.

Если информации в */etc/services* окажется недостаточно, можно почитать соответствующий документ RFC, в котором описывается протокол, используемый службой. Обычно о сервисе требуется знать лишь то, какие протоколы и порты он использует, что можно легко выяснить в RFC.

Управление IP-фильтрацией и файлы сценариев

Правила фильтрации хранятся и используются ядром аналогично записям маршрутизации: при перезагрузке системы правила IP-фильтрации настраиваются заново. Чтобы гарантировать восстановление настройки брандмауэра при перезапуске системы, необходимо поместить соответствующие команды *iptables* в файл сценария, который автоматически выполняется системой во время загрузки. В связке с программным пакетом *iptables* поставляются две программы с именами *iptables-save* и *iptables-restore*, которые записывают текущую конфигурацию *netfilter* в файл и восстанавливают ее из этого файла, соответственно. Эти средства значительно облегчают задачу управления настройкой брандмауэра.

В дистрибутивах Linux принят слегка отличающийся подход к управлению конфигурацией брандмауэра:

Red Hat (версии 7.0 и выше)

Сначала установите правила IP-фильтрации с помощью соответствующих команд *iptables*. Затем выполните команду:

```
/sbin/service iptables save
```

В результате правила фильтрации будут записаны в файл */etc/sysconfig/iptables*, автоматически считываемый во время начальной загрузки.

Debian

Задайте правила *iptables* следующим образом:

1. Отредактируйте */etc/default/iptables* и установите `enable_iptables_initd=true`.
2. Настройте *iptables* вручную с помощью команд *iptables*.
3. Вызовите */etc/init.d/iptables save_active*, чтобы сохранить настройки.

Во время начальной загрузки системы сохраненная конфигурация будет автоматически восстановлена.

SuSE Linux

Простую, но не слишком гибкую конфигурацию можно создать, запустив *yast2* и выбрав модуль настройки брандмауэра Security&Users→Firewall. Другой способ:

1. Отредактируйте */etc/sysconfig/SuSEfirewall2*. Этот файл подробно документирован.
2. При необходимости определите пользовательские правила фильтрации в */etc/sysconfig/scripts/SuSEfirewall2-custom*. Для этого нужно углубленное понимание того, как правила фильтрации действуют в Linux.
3. Запустите фильтрацию командой */sbin/SuSEfirewall2 start*.

Примеры конфигураций netfilter

В этом разделе мы приведем некоторые простые, но полезные конфигурации IP-фильтрации. Цель состоит не в том, чтобы дать набор решений, которыми можно безоговорочно воспользоваться. Мы хотим просто показать, как выглядит полезный набор правил фильтрации IP, и дать основу для создания ваших собственных конфигураций.

Простой пример IP-фильтрации

Сейчас мы продемонстрируем базовое использование IP-фильтрации, аналогичное тому, как мы использовали TCP-оболочки выше в этой главе. Мы хотим отфильтровать пакеты от любых хостов в Интернете, за исключением пакетов, предназначенных демону finger от небольшой группы хостов. Хотя для выполнения той же функции могут быть использованы TCP wrappers, IP-фильтрация позволяет проверять многие различные типы пакетов

(например, пакеты ICMP «ping») и часто необходима, чтобы защитить сервисы, не управляемые оболочками TCP.

В отличие от TCP wrappers, правила *iptables* не могут использовать имена хостов для указания источника или получателя пакетов: при задании правил нужно пользоваться IP-адресами. Однако это даже хорошо, потому что обратный поиск имени хоста не вполне надежен для идентификации пакета (можно фальсифицировать DNS, представив, будто у некоторого IP-адреса другое имя хоста). В примерах 17.1 и 17.2 вместо имен хостов использованы IP-адреса, которые можно получить с помощью такой утилиты, как *nslookup*.

Пример 17.1. Простой пример ipchains

```
# Загрузить модули отслеживания соединений, если они не скомпилированы в ядре.
modprobe ip_conntrack
modprobe ip_conntrack_ftp

# Установить DROP политикой по умолчанию для цепочки INPUT.
iptables -P INPUT DROP

# Разрешать пакеты, принадлежащие существующему соединению.
# '-A INPUT' для дописывания в цепочку INPUT.
# '-m state' использует модуль stateful inspection.
iptables -A INPUT -m state --state ESTABLISHED,RELATED -j ACCEPT

# Разрешать все пакеты, поступившие от кольцевого интерфейса,
# т. е. от локального хоста. '-i lo' задает кольцевой интерфейс.
iptables -A INPUT -i lo -j ACCEPT

# Разрешать новые входящие соединения и пакеты, принадлежащие
# существующим соединениям, на порт 22 (ssh).
iptables -A INPUT -m state --state NEW -m tcp -p tcp \
         --dport 22 -j ACCEPT

# Разрешать новые входящие соединения FTP от 192.168.1/24.
iptables -A INPUT -m state --state NEW -m tcp -p tcp -s 192.168.1/24 \
         --dport 21 -j ACCEPT

# Разрешать новые входящие соединения FTP от spaghetti.vpizza.com,
# у которого IP-адрес 10.21.2.4.
iptables -A INPUT -m state --state NEW -m tcp -p tcp -s 10.21.2.4 \
         --dport 21 -j ACCEPT

# Разрешать новые входящие соединения FTP от *.vpizza.com.
# Там две сети: 172.18.1.0 and 172.25.3.0.
iptables -A INPUT -m state --state NEW -m tcp -p tcp -s 172.18.1/24 \
         --dport 21 -j ACCEPT
iptables -A INPUT -m state --state NEW -m tcp -p tcp -s 172.25.3/24 \
         --dport 21 -j ACCEPT
```

Этот набор правил принимает все пакеты, принадлежащие существующему соединению. Это требуется в случае FTP, когда клиент и сервер могут договориться об альтернативном порте для соединения, в котором передаются данные. Модуль отслеживания соединений (заданный в правилах с помо-

щью -m state) удостоверяет, что соединение передачи данных может быть принято.

IP-фильтрация для защиты целой сети

В предыдущем примере продемонстрирована IP-фильтрация для отдельного хоста. В этом разделе мы рассмотрим случай, когда некоторая сеть машин (например, все машины в доме или небольшом офисе) подключены к Интернету через машину-шлюз. Мы можем написать правила *netfilter* для фильтрации трафика между Интернетом и внутренней сетью. В данном случае мы помещаем правила в две цепочки – INPUT и FORWARD. Напомним, что INPUT используется для фильтрации входящих пакетов, адресованных данному хосту, а FORWARD применяется к пакетам, пересылаемым шлюзом (т. е. адресованным внутренней сети или Интернету). Мы предполагаем, что машина-шлюз использует для связи с Интернетом интерфейс ppp0.

Пример 17.2. Использование netfilter для защиты сети IP

```
# Загрузить модули отслеживания соединений, если они не скомпилированы в ядре.
modprobe ip_conntrack
modprobe ip_conntrack_ftp

# Установить DROP политикой по умолчанию для цепочки INPUT и FORWARD.
iptables -P INPUT DROP
iptables -P FORWARD DROP

# Разрешать все пакеты, поступившие от кольцевого интерфейса.
iptables -A INPUT -i lo -j ACCEPT

# Создать новую цепочку, определенную пользователем. Она будет содержать
# правила для INPUT и FORWARD, поэтому, объединив их в одной цепочке,
# мы избегаем их дублирования.
iptables -N allowfwdin

# Разрешать пакеты, принадлежащие существующему соединению.
# Обратите внимание, что это (и последующие) правила помещены
# в цепочку, определенную пользователем.
iptables -A allowfwdin -m state --state ESTABLISHED,RELATED -j ACCEPT

# Разрешить новые запросы соединений от машин во внутренней сети.
# Это позволит машинам внутренней сети устанавливать соединения
# с Интернетом, но не наоборот. Обратите внимание на использование
# '-i ! ppp0' для задания пакетов, поступающих с интерфейсов, отличных от ppp0.
iptables -A allowfwdin -m state --state NEW -i ! ppp0 -j ACCEPT

# Разрешать новые входящие соединения на порт 22 (ssh).
iptables -A allowfwdin -m state --state NEW -m tcp -p tcp \
         --dport 22 -j ACCEPT

# Разрешать новые входящие соединения FTP от 192.168.1/24.
iptables -A allowfwdin -m state --state NEW -m tcp -p tcp -s 192.168.1/24 \
         --dport 21 -j ACCEPT

# Разрешать новые входящие соединения FTP от spaghetti.vpizza.com.
```

```
iptables -A allowfwdin -m state --state NEW -m tcp -p tcp -s 10.21.2.4 \
         --dport 21 -j ACCEPT

# Разрешать новые входящие соединения FTP от *.vpizza.com.
iptables -A allowfwdin -m state --state NEW -m tcp -p tcp -s 172.18.1/24 \
         --dport 21 -j ACCEPT
iptables -A allowfwdin -m state --state NEW -m tcp -p tcp -s 172.25.3/24 \
         fs

# Ко всем пакетам, прошедшим цепочку, определенную пользователем, теперь
# применяется действие LOG, которое их регистрирует.
# Используется модуль 'limit', чтобы заблокированные пакеты
# не регистрировались слишком быстро.
iptables -A allowfwdin -m limit --limit 2/sec -j LOG

# Установить DROP действием по умолчанию для цепочки, определенной пользователем.
iptables -A allowfwdin -j DROP

# Направлять все пакеты, полученные для INPUT или FORWARD, в нашу цепочку,
# определенную пользователем.
iptables -A INPUT -j allowfwdin
iptables -A FORWARD -j allowfwdin

# Включить IP-маршрутизацию (требуется для всех маршрутизаторов IP,
# независимо от использования IP-фильтрации).
echo 1 >/proc/sys/net/ipv4/ip_forward
```

Чтобы проследить все попытки проникнуть через систему защиты, мы добавили правило, регистрирующее пакеты, которые будут отброшены. Однако при поступлении большого количества плохих пакетов это правило могло бы вызвать переполнение диска записями в журнале или недопустимое замедление работы шлюза (поскольку регистрация пакета отнимает гораздо больше времени, чем пересылка или фильтрация). Поэтому мы используем модуль `limit`, контролирующий частоту применения действия правила. В предшествующем примере мы разрешили регистрировать в среднем два плохих пакета в секунду. Все остальные пакеты проскочат правило и будут просто отброшены.

Чтобы посмотреть на правила, которые мы создали (см. пример 17.3), воспользуемся *iptables* с параметром *-L*. Режим подробного вывода (*-v*) показывает больше информации, чем базовая команда вывода.

Пример 17.3. Вывод правил iptables для примера 17.2

```
# iptables -L -v
Chain INPUT (policy DROP 0 packets, 0 bytes)
 pkts bytes target     prot opt in     out     source               destination
   16  1328 ACCEPT     all  --  lo     any     anywhere             anywhere
    0     0 allowfwdin all  --  any    any     anywhere             anywhere

Chain FORWARD (policy DROP 0 packets, 0 bytes)
 pkts bytes target     prot opt in     out     source               destination
    0     0 allowfwdin all  --  any    any     anywhere             anywhere

Chain OUTPUT (policy ACCEPT 9756 packets, 819K bytes)
```

```
 pkts bytes target     prot opt in     out     source               destination

Chain allowfwdin (2 references)
 pkts bytes target     prot opt in     out     source               destination
    0     0 ACCEPT     all  --  any    any     anywhere             anywhere  \
    state RELATED,ESTABLISHED
    0     0 ACCEPT     all  --  !ppp0  any     anywhere             anywhere  \
    state NEW
    0     0 ACCEPT     tcp  --  any    any     anywhere             anywhere  \
    state NEW tcp dpt:ssh
    0     0 ACCEPT     tcp  --  any    any     192.168.0.0/24       anywhere  \
    state NEW tcp dpt:ftp
    0     0 ACCEPT     tcp  --  any    any     10.21.2.4            anywhere  \
    state NEW tcp dpt:ftp
    0     0 ACCEPT     tcp  --  any    any     172.18.0.0/24        anywhere  \
    state NEW tcp dpt:ftp
    0     0 ACCEPT     tcp  --  any    any     172.25.0.0/24        anywhere  \
    state NEW tcp dpt:ftp
    0     0 LOG        all  --  any    any     anywhere             anywhere  \
    limit: avg 2/sec burst 5 LOG level warning
    0     0 DROP       all  --  any    any     anywhere             anywhere
```

Пример сокрытия IP-адресов

С помощью правил *netfilter* можно осуществлять *сокрытие IP-адресов* (*IP masquerading*) – особый вид NAT, при котором пакеты из внутренней сети изменяются таким образом, что выглядят, как исходящие из одного IP-адреса. Это часто используется, когда есть ряд машин, объединенных в локальную сеть, и одна из машин подключена к Интернету с единственным IP-адресом. Такая ситуация встречается в домашних сетях, когда ISP выделяет один IP-адрес. С помощью сокрытия IP-адресов целая сеть машин может совместно использовать этот адрес. В результате того, что шлюз осуществляет сокрытие IP-адресов, пакеты из внутренней сети будут выглядеть так, будто исходят от машины-шлюза, а пакеты из Интернета будут пересылаться обратно нужному хосту во внутренней сети. Добиться этого можно с помощью определенной хитрой перезаписи пакетов с помощью *netfilter*.

Настроить *netfilter* для поддержки сокрытия IP-адресов значительно проще, чем объяснить, как оно работает! Более полная информация о том, как осуществляются сокрытие IP-адресов и NAT, имеется в *NAT HOWTO*. Самая простая конфигурация приводится в примере 17.4.

Здесь предполагается, что есть Linux-система, действующая как шлюз для внутренней сети. У шлюза имеется PPP-соединение с Интернетом через интерфейс `ppp0` и соединение с локальной сетью через интерфейс `eth0`. Предлагаемая конфигурация разрешает исходящие соединения из внутренней сети с Интернетом, но блокирует входящие соединения из Интернета с машинами внутренней сети, за исключением шлюза. Оказывается, для такого режима не нужны специальные команды, поскольку он действует по умолчанию, когда NAT используется подобным образом.

Пример 17.4. Базовая конфигурация для сокрытия IP-адресов

```
# Загрузить модуль поддержки NAT, если он не скомпилирован с ядром.
modprobe iptables_nat

# Скрыть IP-адреса для всех маршрутизируемых соединений,
# поддерживаемых устройством ppp0.
iptables -t nat -A POSTROUTING -p ppp0 -j MASQUERADE

# Включить IP-маршрутизацию.
echo 1 >/proc/sys/net/ipv4/ip_forward
```

В этой конфигурации нужно отметить некоторые важные детали. Функциональность NAT обеспечивается отдельным модулем, который нужно загрузить, если он не встроен в ядро. Модуль NAT использует новую цепочку с именем `POSTROUTING`, которая обрабатывает пакеты после того, как ядро выполнит над ними операции маршрутизации (т. е. определит, предназначен пакет для Интернета или для машин локальной сети). Параметр `MASQUERADE` определяет все операции по трансляции адресов и отслеживанию пакетов.

Обратите внимание, что в этой конфигурации не происходит фильтрации исходящих соединений. Все хосты частной сети смогут устанавливать исходящие соединения с любыми хостами и любыми портами. В «*packet filtering HOWTO*» есть полезная информация о том, как объединить IP-фильтрацию с преобразованием адресов.

18

LAMP

Для большинства веб-сайтов недостаточно просто написать пару строк кода HTML; динамический контент – вот что сегодня востребовано. Как правило, посетителям веб-сайтов не требуется все то, что жаждут предложить им веб-дизайнеры, но распоряжаются здесь последние, поэтому о динамическом контенте мы и поговорим в этой главе.

Linux, как можно догадаться, – отличная платформа для генерации динамического контента. Несметное число веб-сайтов, предоставляющих динамический контент, уже работает под Linux; это одна из главных областей приложений, в которых отличается Linux.

Динамический контент можно получить, применяя два совершенно различных способа программирования: программирование на стороне сервера и программирование на стороне клиента. В программировании на стороне клиента интерактивные HTML-страницы чаще всего получают с помощью JavaScript и апплетов Java.

Из-за ограничений, присущих JavaScript и Java, в большинстве случаев применяется программирование на стороне сервера. Есть много способов использования программ, выполняемых на сервере, и много различных программных пакетов для поддержки этой технологии, но существует одна комбинация таких пакетов, которая распространена повсеместно. Она стала настолько популярной, что для нее был придуман «акроним» LAMP, за которым скрывается Linux-Apache-MySQL-PHP. О веб-сервере Apache мы уже рассказывали, Linux посвящена книга в целом, поэтому нам осталось обсудить два последних пакета – MySQL и PHP, а также совместное применение всех четырех составляющих.

Для того чтобы получить действующую установку LAMP, нужно настроить Apache, что подробно описано в разделе «Настройка собственного веб-сервера» главы 16, а также установить MySQL и PHP. О том, как запустить в работу последние два пакета, будет рассказано в этой главе.

Но прежде чем перейти к техническим деталям, хочется обсудить, стоит ли труда установка и освоение системы LAMP.

LAMP позволяет легко обеспечить большой объем контента и навигацию по нему пользователей вашего веб-сайта.

Допустим, у вас есть сайт с множеством JPEG-файлов фотографий, которые вы сняли в разное время. Для посетителей сайта желательно иметь возможность выбрать различный размер просматриваемых фотографий. Одни захотят посмотреть фотографии архитектурных памятников, когда бы они ни были сделаны; другие пожелают увидеть фотографии, сделанные во время вашего последнего путешествия, когда бы оно ни происходило.

Чтобы упростить навигацию и извлечение материала, вы сначала помещаете свои JPEG-файлы в базу данных MySQL,[1] один тип данных которой специально предназначен для больших двоичных объектов (BLOB), таких как изображения JPEG. Вы организуете их по любому принципу (по теме, по путешествию и т. д.) и записываете всю информацию в таблицы базы данных.

Теперь вы создаете форму, в которой посетители веб-сайта могут указать желаемые размеры фото. Это может быть простая форма, как на рис. 18.1.

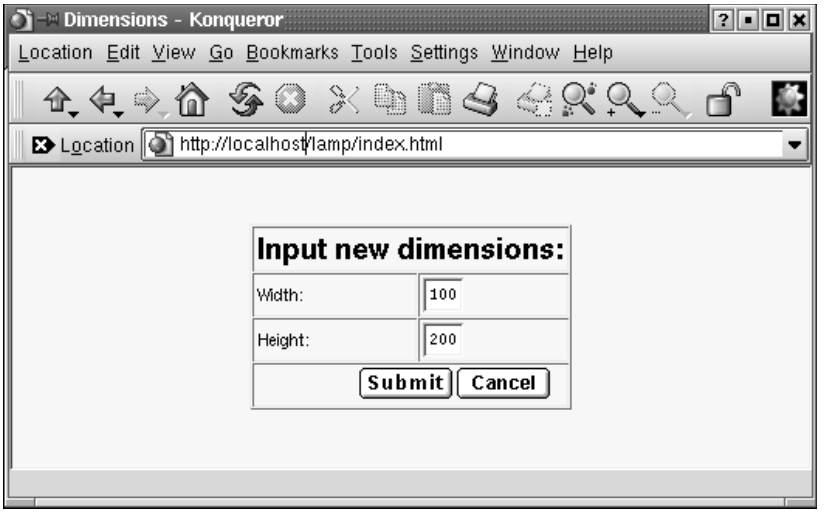


Рис. 18.1. *Простая форма для ввода*

Следующая ваша страница – динамическая. Короткий PHP-код получает запрос посетителя и определяет, что нужно показать на этой странице. Она может выглядеть, как на рис. 18.2.

А где же тут участвует MySQL? Ее участие не сразу видно, но она играет ключевую роль, поскольку код PHP запрашивает у нее данные. Сочетание встроенного кода PHP и быстрой базы данных оставляет у посетителя довольно приятное впечатление.

---

[1] Вообще-то, так лучше не делать. – *Примеч. науч. ред.*

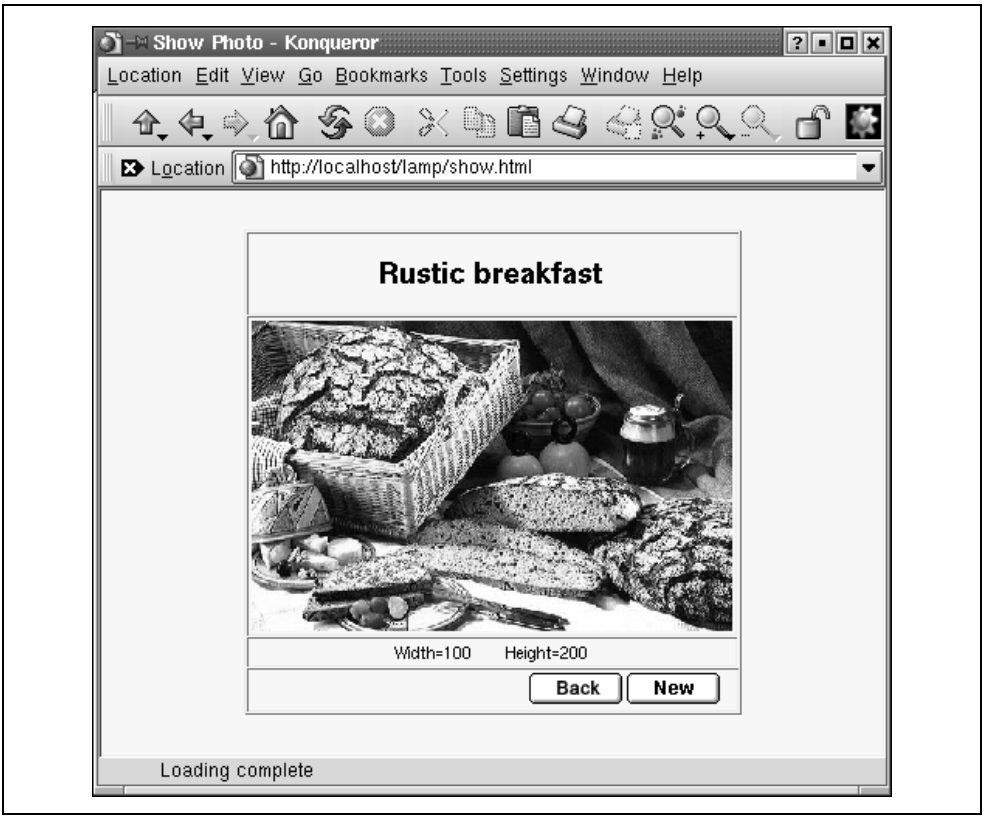


Рис. 18.2. *Динамическая веб-страница, сгенерированная PHP*

MySQL

MySQL – база данных с открытым исходным кодом (open source), очень быстрая и относительно просто управляемая. Если вам требуются более сложные возможности баз данных, такие как репликации или распределенные базы данных, либо вы предполагаете хранение гигабайтов данных, то нечто крупное типа Oracle может оказаться лучшим выбором, но для большинства задач MySQL прекрасно подойдет.

Скорее всего в вашем дистрибутиве есть система MySQL, которую можно установить, но если вам нужно все самое лучшее и свежее, можно зайти на *http://www.mysql.com/downloads/* и загрузить весь пакет. На данный момент последняя стабильная версия имеет номер 3.23.47. Учтите, что некоторые пользователи MySQL сообщали о повреждении таблиц базы данных под ядром 2.2.14 при большой нагрузке, поэтому если вы рассчитываете использовать MySQL в практических приложениях, может потребоваться обновление ядра.

О выборе баз данных

Наверное, следует отметить, что MySQL – не единственный вариант выбора базы данных в качестве сервера для динамического веб-сайта. Например, Postgres, устанавливаемая по умолчанию на системах Red Hat, – тоже очень хорошая база данных open source. Поэтому при желании развернуть систему «LAPP», выбрав Postgres, вполне можно сделать это. Большая часть приводимой в данной главе информации будет применима и в такой конфигурации. Однако MySQL можно рассматривать как стандартную базу данных для динамических веб-сайтов на системах Linux. Это обусловлено простотой ее использования, скоростью и универсальностью. Кроме того, если вам потребуется дополнительная документация по развертыванию динамических веб-сайтов под Linux (к которой мы весьма рекомендуем обратиться), то более вероятно найти такую информацию (например, в виде специальных книг по этой теме) о MySQL, чем о Postgres.[1]

[1] Р. Стоунз, Н. Мэттью «Основы PostgreSQL». – Пер. с англ. – СПб: Символ-Плюс, 2002.

В версиях MySQL, скомпилированных с помощью gcc 2.96, могут происходить неожиданные аварии. Данная версия gcc не является официальной стабильной версией, но есть по крайней мере один дистрибутив (Red Hat), в который, к сожалению, она включена в качестве компилятора по умолчанию. Поэтому если у вас происходят непонятные аварии сервера базы данных и вы пользуетесь gcc 2.96, попробуйте установить готовый двоичный дистрибутив или воспользоваться более стабильной версией компилятора, например 2.95.3.

Если вы хотите скомпилировать MySQL самостоятельно, нужно загрузить пакет с исходным кодом, распаковать его и установить с помощью команд:

```
owl$ ./configure --prefix=/usr/local/mysql
owl$ make
owl# make install
```

Обратите внимание, что в вашей системе путь для установки может оказаться иным. Кроме того, для третьей команды потребуется, вероятно, зарегистрироваться как `root`. Запомните путь установки, потому что позднее он потребуется для настройки PHP.

Следующим шагом рекомендуем создать пользователя и группу с именем `mysql`, как описано в разделе «Создание учетных записей» главы 5. Получите привилегии этого пользователя с помощью команды *su - mysql* и выполните:

```
owl$ scripts/mysql_install_db
```

По соображениям безопасности полезно запретить пользователю `mysql` вход в систему. Это легко сделать, зарегистрировавшись как `root` и поместив звездочку во вторую (password) колонку в файле */etc/password* и/или */etc/shadow*.

После этого действия нужно выполнить еще одну команду как root, после чего можно вернуться к своей обычной учетной записи. Следующая команда запускает сервер MySQL:

```
owl# /usr/local/mysql/bin/safe_mysqld &
```

Можно также добавить параметр *--log* или *--log-long-format*, чтобы регистрировать в файле журнала все, что происходит с сервером базы данных.

Чтобы проверить, правильно ли запустился сервер, можно выполнить (в качестве обычного пользователя) следующую команду (если вы установили MySQL в другое место, нужно, конечно, изменить путь):

```
owl$ /usr/local/mysql/bin/mysqladmin version
mysqladmin  Ver 8.19 Distrib 3.23.37, for suse-linux on i686
Copyright (C) 2000 MySQL AB & MySQL Finland AB & TCX DataKonsult AB
This software comes with ABSOLUTELY NO WARRANTY. This is free software,
and you are welcome to modify and redistribute it under the GPL license.

Server version                3.23.37-log
Protocol version        10
Connection                Localhost via UNIX socket
UNIX socket                /var/lib/mysql/mysql.sock
Uptime:                        43 days 11 hours 39 min 17 sec

Threads: 1  Questions: 142  Slow queries: 0  Opens: 160  Flush tables: 1  Open
tables: 13 Queries per second avg: 0.000
```

Это должно работать без ввода пароля. Хотим отметить, однако, что работать с базой данных без пароля не рекомендуется, поскольку это увеличивает шансы потенциального нарушителя получить доступ к ценным данным, которые могут находиться в вашей базе. Можно оставить базу данных без пароля на время тестирования, но завершив все проверки, нужно обязательно установить пароль и еще раз проверить, что все работает так же хорошо, когда пароль установлен. Если вы создали пароль для пользователя базы данных с именем root (либо это сделал за вас ваш дистрибутив; обращайтесь к документации в случае затруднений), нужно задать параметр *-p*, чтобы *mysqladmin* запросил ваш пароль.

Следует добавить, что в большинстве дистрибутивов есть сценарий для запуска сервера MySQL, которым можно воспользоваться вместо запуска сервера вручную (особенно если MySQL установлен с инсталляционного носителя). Часто этот сценарий помещается в */etc/init.d/mysql*.

Если сервер базы данных запущен и работает, можно приступить к созданию пользователей базы данных и новых баз данных. Следует отметить, что в исходные коды MySQL включен вполне приемлемый учебник, кроме того, массу документации можно найти на *http://www.mysql.com*, поэтому мы расскажем лишь самые основные вещи, которые позволят вам начать работу. Если вы действительно хотите стать экспертом по MySQL, советуем прочесть книгу «Managing & Using MySQL» издательства O'Reilly, которую написали

Рэнди Джей Яргер (Randy Jay Yarger), Джордж Риз (George Reese) и Тим Кинг (Tim King).[1]

Есть два способа общения с ядром MySQL: можно пользоваться консольным клиентом либо писать так называемые SQL-сценарии и посылать их базе данных, чтобы выполнять сразу несколько команд SQL. SQL (Structured Query Language) – это структурированный язык запросов, используемый для работы с реляционными базами данных. Далее в этой главе мы расскажем о его применении. Оба способа выполнения команд SQL предполагают, что вы правильно ввели комбинацию имя пользователя-пароль.

Для работы с MySQL важно знать, что учетные записи пользователей Linux и пользователей MySQL различны. Иными словами, в MySQL есть собственная система управления бюджетами пользователей. Однако чаще всего пользователи дают своим учетным записям в MySQL такие же имена, как в Linux, чтобы избежать путаницы.

По умолчанию в MySQL есть одна учетная запись с именем `root`, для которой не установлен пароль (вот вам пример «защиты по умолчанию»...). Это означает, что можно получить доступ к серверу базы данных через интерактивную утилиту командной строки *mysql* следующим образом:

```
owl$ mysql -u root

Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 13 to server version: 3.23.39.

Type 'help;' or '\h' for help. Type '\c' to clear the buffer.

mysql:
```

Параметр *-u* позволяет указать пользователя базы данных. Если эта команда не работает, то, возможно, в вашем дистрибутиве MySQL для пользователя `root` установлен пароль. Найдите этот пароль в документации и запустите программу *mysql* снова:

```
owl$ mysql -u root -p
```

Вы должны получить приглашение для ввода пароля.

Допустим, что вы смогли подключиться к серверу базы данных. Тогда выполните команду:[2]

```
mysql> show databases;
+-------------+
| Database    |
+-------------+
| mysql       |
| test        |
+-------------+
2 rows in set (0.11 sec)
```

[1] Рэнди Джей Яргер, Джордж Риз и Тим Кинг «MySQL и mSQL». – Пер. с англ. – СПб: Символ-Плюс, 2000.

[2] Это не команда SQL, а команда администрирования MySQL.

Она сообщает, что этот сервер управляет двумя базами данных. Одна из них называется *mysql* и содержит внутреннюю информацию MySQL, включая имена пользователей, а другая называется *test* и может использоваться для экспериментов. Создать новые базы данных очень просто, и мы покажем, как это делается, чуть позже. Заметьте, что все команды SQL должны оканчиваться точкой с запятой – вероятно, это должно доставить удовольствие программистам C.

Теперь нужно задать пароль для пользователя `root` (если его еще нет). Это делают две команды SQL:

```
mysql> SET PASSWORD FOR root=PASSWORD('new_topsecret_passwd');
mysql> FLUSH PRIVILEGES;
```

Снова обратите внимание на точку с запятой в конце этих команд: если вы забудете ввести этот символ и нажмете <Enter>, MySQL молча уставится на вас и будет ждать, когда вы введете что-нибудь еще.

Кстати, в командах SQL не учитывается регистр символов. Мы написали их заглавными буквами, потому что при этом легче определять в сценариях SQL, где находятся ключевые слова и переменные параметры.

Обратите также внимание на команду *FLUSH PRIVILEGES*. Она важна, поскольку только после ее выполнения MySQL обновляет свою базу данных пользователей.

Теперь мы хотим создать нового пользователя с именем `olof` с такими же правами доступа, как у `root`, за исключением права создания новых пользователей. За исключением этого права, у `olof` будут все привилегии для работы со всеми базами данных на этом сервере MySQL:

```
mysql> GRANT ALL PRIVILEGES ON *.* TO olof@localhost IDENTIFIED BY 'olof_passwd';
mysql> FLUSH PRIVILEGES;
```

Пользователь `olof` сможет подключаться к базе данных только с локальной машины. Это хорошее решение, не создающее лишних проблем с безопасностью. Мы советуем разрешать доступ только с локальной машины, если в обязательном порядке не требуется иного. Даже в связке LAMP достаточно локального доступа, потому что процесс веб-сервера выполняется на локальной машине и с базой данных соединяется этот процесс, а не процесс веб-броузера пользователя.

Но если вам действительно необходим доступ к базе данных по сети, можно воспользоваться такими командами:

```
mysql> GRANT ALL PRIVILEGES ON *.* TO username@"%"  IDENTIFIED BY 'user_passwd';
mysql> FLUSH PRIVILEGES;
```

Если вам кажется чрезмерным предоставление всех прав доступа, за исключением создания новых пользователей, создадим еще одного пользователя, который сможет выполнять операции `SELECT`, `INSERT`, `UPDATE`, `DELETE` и `DROP`, но только в базе данных *test* (и только подключившись с локальной машины):

```
mysql> GRANT SELECT, INSERT, UPDATE, DELETE, DROP ON test.* TO gonzo@localhost
       IDENTIFIED BY 'gonzo_passwd';
mysql> FLUSH PRIVILEGES;
```

Если вам не доводилось работать с базами данных SQL прежде, эти операции могут показаться непонятными. Поскольку они все равно понадобятся вам при организации своей системы LAMP, мы кратко опишем их здесь:

SELECT

Наиболее часто используемая команда SQL. Она запрашивает в базе данных информацию с определенными свойствами. Например, можно запросить данные по всем клиентам в указанном городе. SELECT никак не модифицирует базу данных.

INSERT

Эта команда SQL вводит новые записи в таблицу базы данных. С ее помощью (в интерактивном режиме или чаще внутри некоторой программы) можно, например, ввести в таблицу клиентов базы данных запись о новом клиенте.

UPDATE

Эта команда SQL модифицирует записи, имеющиеся в базе данных. С ее помощью можно, например, увеличить розничные цены всех товаров, имеющихся в базе данных, на 15% (вот вам и инфляция!).

DELETE

Эта команда SQL удаляет из базы данных целые записи. Будьте осторожны с ней, поскольку восстановить данные будет нельзя, кроме как с резервной копии (если она есть!).

Есть и другие команды SQL и соответствующие привилегии (например, DROP, позволяющая удалять целые таблицы и даже целые базы данных), но они используются реже, чем приведенная «большая четверка».

Теперь мы хотим создать новую базу данных, в которую потом будем помещать таблицы и данные. Это делается командой SQL CREATE DATABASE:

```
mysql> create database test_database;
Query OK; 1 row affected (0.03 sec)
```

MySQL показала, что все прошло удачно, но для пущей уверенности еще раз спросим, какими базами данных управляет сервер:

```
mysql> show databases;
+-----------------+
| Database        |
+-----------------+
| mysql           |
| test            |
| test_database   |
+-----------------+
6 rows in set (0.00 sec)
```

Теперь мы хотим определить таблицу в нашей новой базе данных, но сначала необходимо сообщить серверу MySQL о том, что мы действительно хотим работать с этой базой данных:

```
mysql> use test_database
Database changed
```

Как видите, здесь мы не поставили в конце точки с запятой, поскольку это не команда SQL а управляющий оператор для консольного клиента MySQL. Но не было бы ошибкой поставить и здесь точку с запятой.

Определение таблицы, где, в конечном счете, будут храниться ваши данные, происходит с помощью команды SQL `CREATE TABLE`. Например:

```
mysql> CREATE TABLE comment_table(
    ->   id INT NOT NULL auto_increment,
    ->   comment TEXT,
    ->   PRIMARY KEY(id));
Query OK, 0 rows affected (0.10 sec)
```

Здесь мы определили таблицу `comment_table`, в которой есть две колонки, т. е. два поля данных в каждой записи. Одна колонка называется `id` и содержит уникальный идентификатор каждой записи, который не может повторяться в различных записях, а потому помечен как *первичный ключ* (*primary key*), что на жаргоне баз данных означает «уникальный идентификатор». Другая колонка – это переменная типа `TEXT`, которая может содержать до 65 535 символов.

Теперь мы можем проверить, какие таблицы есть в нашей базе данных `test_database`:

```
mysql> show tables;
+-------------------------+
| Tables_in_test_database |
+-------------------------+
| comment_table           |
+-------------------------+
1 row in set (0.00 sec)
```

Как видим, все в порядке, и можно начать добавлять в нашу таблицу записи с данными. Это происходит с помощью SQL-команды `INSERT`:

```
mysql> INSERT INTO comment_table VALUES ('0','comment');
Query OK, 1 row affected (0.06 sec)
```

Наконец, можно проверить, какие данные содержатся в нашей таблице:

```
mysql> SELECT * FROM comment_table;
+----+---------+
| id | comment |
+----+---------+
|  1 | comment |
+----+---------+
1 row in set (0.01 sec)
```

Мы запросили все (*) колонки таблицы comment_table. Возможно, вы заметили нечто странное: мы попросили MySQL вставить в первую колонку 0, а вместо этого там оказалась 1. Это вызвано тем, что мы определили для этой колонки тип INT NOT NULL auto_increment; это означает, что значением колонки не может быть NULL, и MySQL должна автоматически выбрать следующее доступное значение. Это удобно, потому что можно вводить в таблицу новые записи, не беспокоясь о выборе для первой колонки уникальных значений:

```
mysql>  INSERT INTO comment_table VALUES ('0','comment1');
Query OK, 1 row affected (0.00 sec)

mysql> SELECT * FROM comment_table;
+----+----------+
| id | comment  |
+----+----------+
|  1 | comment  |
|  2 | comment1 |
+----+----------+
2 rows in set (0.00 sec)
```

Как видите, мы снова задали 0 в качестве значения первой колонки, но MySQL автоматически выбрала очередное допустимое значение.

Теперь вы достаточно знаете о MySQL, чтобы приступить к самостоятельным экспериментам или начать читать какую-либо другую книгу о базах данных, мечтая создать очередной успешный веб-сайт электронной коммерции.[1]

А можно вместо мечтаний продолжить чтение этой книги и познакомиться со сценариями SQL.

Необязательно вводить все команды в приглашение командной строки клиента MySQL, можно выполнять пакетные файлы с командами SQL, направляя их на стандартный ввод программы *mysql*. Например, сохраним следующий код SQL как файл с именем *create_db.sql*:

```
DROP DATABASE IF EXISTS test_database;
CREATE DATABASE test_database;
USE test_database;
CREATE TABLE comment_table( id INT NOT NULL auto_increment,\
 comment TEXT,PRIMARY KEY(id));

INSERT INTO comment_table VALUES ('0','comment');
INSERT INTO comment_table VALUES ('0','comment1');
```

Выполнить этот сценарий можно из обычной командной строки Linux:

```
mysql -u root -p < create_db.sql
```

Строка

```
DROP DATABASE IF EXISTS test_database;
```

весьма опасна; ее можно использовать, только если в вашей базе данных нет важной информации.

---

[1] Или первый успешный веб-сайт электронной коммерции, как поправит кое-кто.

По правде говоря, нет абсолютной необходимости создавать новую базу данных для каждого нового проекта, хотя это весьма рекомендуется. Теоретически можно сваливать все данные в базу данных *test*, которая устанавливается вместе с MySQL, если только все имена таблиц будут различными. На практике сопровождение такой базы данных превратится в кошмар, как только количество таблиц в ней станет заметным.

PHP

Для укомплектования нашего набора из Linux, Apache, PHP и MySQL нужен еще интерпретатор языка PHP. PHP – это рекурсивный акроним, раскрываемый как PHP:Hypertext Preprocessor. Он разрабатывается уже несколько лет. Чаще всего используются версии 3 и 4. В этой главе мы используем PHP4, поскольку на момент написания книги это самая свежая стабильная версия.[1]

Примеры кода PHP

Приятной особенностью PHP является возможность ввода кода PHP непосредственно в код HTML. Веб-сервер передает все, что находится между тегами `<?php` и `?>`, модулю PHP, который интерпретирует и выполняет команды. Приведем очень простой пример кода PHP, помещенного на HTML-страницу; если вы уже установили PHP, можете выполнить его прямо из своего веб-сервера (если нет, ниже мы покажем, как устанавливать PHP):

```
<html>
<body>
<?php
echo "Hi,";
?>
LAMP enthusiasts.
</body>
</html>
```

Как вы, возможно, и предполагали, броузер покажет следующий текст:

```
Hi, LAMP enthusiasts.
```

Этот крайне простой пример показывает, как сервер Apache взаимодействует с интерпретатором PHP: код между `<?php` и `?>` передается интерпретатору PHP, который выполняет команду `echo`, а та выводит свои параметры в веб-броузер. Кроме того, строка `LAMP enthusiasts` просто добавляется в виде обычного текста HTML (и поскольку в ней нет разметки, она не похожа на HTML).

Конечно, возможности PHP значительно шире. Как и большинство языков программирования, он может использовать переменные и принимать решения, как в следующем сценарии (для краткости HTML-окружение опущено):

---

[1] Л. Аргерих и др. «Профессиональное PHP программирование», 2-е издание. – Пер. с англ. – СПб: Символ-Плюс, 2003.

```
<?php
echo "Dear friends, today's date is: ";
echo date("F d, Y")."\n";
echo "<br>";
echo "We are in the ";

if ( date ("m") <= 6 ) {
        echo "first ";
        } else {
        echo "second ";
}
echo "half of the year ".date("Y");
?>
```

Возможно, вы уже разобрались, что этот сценарий основывает свое решение в операторе `if` на том, какой сейчас месяц. Обратите внимание на тег HTML `<br>` в выводе PHP. Это совершенно допустимо и служит распространенным приемом при работе с PHP. Ваш веб-броузер получит следующие данные (конечно, с другими датами, если только в вашем компьютере исправны часы и у вас не случилась деформация времени):

```
Dear friends, today's date is: May 04, 2002
<br>We are in the first half of the year of 2002
```

Веб-броузер сделает перенос в том месте, где стоит тег `<br>`.

Как и большинство языков программирования, PHP поддерживает функции в целях обеспечения модульности кода. Функции позволяют выполнять фрагмент кода в различных местах без многократного его дублирования.

PHP поставляется с весьма обширными библиотеками функций, которые можно дополнительно загружать из Интернета. Подключить библиотеку функций можно с помощью операторов `include()` и `require()`, имеющих между собой незначительные различия.

Если вы хотите программировать на PHP, следует ознакомиться с документацией по библиотекам функций, которые поставляются вместе с интерпретатором PHP, поскольку благодаря этим функциям вам не придется изобретать велосипед при решении стандартных задач.

Вот определение двух простых функций: `show_date`, которая выводит текущую дату в жестком формате и добавляет перевод строки, и `show_halfyear`, которая выводит `first` или `second` в зависимости от текущего месяца:

```
<?php

function show_date() {

echo  date("F d, Y")."\n <br>";
}

function show_semester() {
if (date ("m")<=6){
```

```
        echo "first ";
        } else {
        echo "second ";
        }
}

?>
```

Назовем этот сценарий *functions.php* и перепишем наш первоначальный сценарий с использованием этих функций:

```
<?php

require(functions.php);
echo "Dear friends, the date today is: ";

show_date();

echo "<br>";
echo "We are in the ";

show_semester();

echo "semester of ".date("Y");
?>
```

Оператор `require()` сообщает интерпретатору PHP о необходимости загрузить наш сценарий с функциями и сделать содержащиеся в нем функции доступными текущему сценарию.

Конечно, мы очень поверхностно коснулись возможностей PHP. Если у вас появился к нему интерес, можете прочесть книгу «Programming PHP» Расмуса Лердорфа (Rasmus Lerdorf), создателя PHP, и Кевина Тэтроу (Kevin Tatroe), издательство O'Reilly.

До выхода PHP3 язык PHP был интерпретируемым и его код хранился в буфере. Циклы и другие фрагменты кода, выполнение которых повторялось, многократно подвергались синтаксическому анализу – перед каждым выполнением кода. Конечно, достичь оптимальной производительности при этом было нельзя.

PHP4 был написан полностью заново и состоит из ядра языка (под названием «Zend») и функциональных модулей (которые очень гибки и расширяемы). В отличие от PHP3, PHP4 допускает работу в многопоточной среде, что позволяет использование PHP в качестве модуля различными веб-серверами.

Выполнять PHP4 можно не только как модуль, но и как программу CGI, запускаемую веб-сервером, что связано с некоторыми дополнительными издержками. При выполнении PHP в качестве CGI-программы каждая новая страница, содержащая код PHP, требует запуска нового экземпляра интерпретатора PHP, что, в свою очередь, требует создания нового процесса и загрузки в него интерпретатора PHP. Когда интерпретатор заканчивает создание страницы, его процесс завершается, память освобождается, закрываются все дескрипторы файлов и соединения с базами данных.

В качестве модуля веб-сервера интерпретатор PHP становится частью веб-сервера и постоянно загружен в память. Кроме того, он может сохранять такие ресурсы, как соединения с базами данных, при переходе к другим страницам, что значительно повышает производительность.

На всех интенсивно посещаемых сайтах, использующих PHP, он загружен именно в виде модуля, поскольку это обеспечивает более высокую производительность.

PHP4 как модуль Apache

Как мы уже отмечали, для достижения лучшей производительности следует запускать интерпретатор PHP в качестве модуля веб-сервера. В настоящее время большинство дистрибутивов, в том числе Slackware, Debian, SuSE и RedHat, включают в себя как Apache, так и модуль PHP4 для Apache, поэтому обычно нет необходимости самостоятельно компилировать модуль PHP4. Однако иногда такое решение может оказаться правильным.

Для поддержки обширной функциональности модулю PHP4 требуется большое количество дополнительных библиотек или модулей. Если устанавливать модуль PHP4 с инсталляционного компакт-диска, то программа установки автоматически установит необходимые модули. Однако модули, поставляемые с дистрибутивами, обычно содержат функции, способные удовлетворить все вкусы и потребности. В результате система может оказаться более тяжеловесной и медлительной, чем вызывается необходимостью.

Поэтому преимущество самостоятельной компиляции PHP4 состоит в том, что вы можете решить, какую функциональность включить в этот модуль. Проверьте по документации, какие дополнительные библиотеки вам может потребоваться установить.

Поскольку мы твердо убеждены в необходимости знать о том, что творится за кулисами, даже если используются более удобные готовые решения, то дадим несколько советов относительно того, как начать работу с чистого листа и как различные части взаимодействуют друг с другом.

Для загрузки модуля PHP4 в веб-сервер Apache во время его работы необходимо присутствие модуля Apache *mod_so*. Проверить его наличие можно с помощью команды:

```
owl$ httpd -l
Compiled-in modules:
        http_core.c
        mod_so.c
```

Если этот модуль недоступен, проверьте, не пропустили ли вы установку каких-либо дополнительных пакетов Apache из вашего дистрибутива. Если вы компилировали Apache самостоятельно, прочтите в документации, как получить этот модуль.

Можно также скомпилировать модуль PHP4 непосредственно в Apache, но это требует тесной связи компиляции Apache и PHP4 и не дает особых преимуществ, поэтому мы не станем освещать здесь эту тему.

Теперь нужно скомпилировать PHP, чтобы сделать из него динамический разделяемый объект (Dynamic Shared Object, DSO). К счастью, это не так сложно, как может показаться. Загрузите PHP4 с *http://www.php.net/download.php*. В итоге вы получите пакет с именем *php-4.1.1.tar.gz* (фактический номер версии может быть несколько иным). Распакуйте *tar*-файл и сконфигурируйте PHP командой:

```
owl ./configure \
                --with-mysql=/usr/lib/mysql\
                --with-ldap=yes\
                --with-gd=yes\
                --with-zlib=yes\
                --with-config-file-path=/etc/\
                --with-apxs=/usr/lib/apache/apxs\
                --enable-versioning\
                --enable-track-vars\
                --enable-thread-safety
```

В пространной документации PHP можно прочесть о многочисленных дополнительных опциях, но для начала достаточно этих. Учтите, что может потребоваться заменить некоторые из указанных путей теми, которые соответствуют фактическим адресам в вашей системе. После *configure* выполните *make*, а затем *make install*, чтобы установить PHP (может потребоваться выполнить *make install* как `root`).

Теперь отредактируйте *httpd.conf*, файл конфигурации Apache. Если вы установили Apache с инсталляционного диска, возможно, что следующие строки там уже есть и нужно лишь раскомментировать их. Во всяком случае в *httpd.conf* должны быть следующие строки:

```
LoadModule php4_module libexec/libphp4.so
AddModule mod_php4.c

AddType application/x-httpd-php .php
```

Теперь перезапустите Apache:

```
owl$ apachectl restart
```

После того как сервер перезапущен, нужно проверить, правильно ли загружается модуль PHP4. Для этого можно написать небольшую программку PHP, например:

```
<?php
phpinfo();
?>
```

Сохраните этот файл как *phpinfo.php* в каталоге *htdocs* своего установочного каталога Apache (обычно */usr/local/httpd/htdocs*) и сделайте файл исполняемым с помощью команды *chmod*.[1] Теперь вы должны загрузить эту страницу

[1] На самом деле прав на выполнение этого файла в данном случае не требуется. Достаточно прав на чтение пользователем, под которым запущен веб-сервер. – *Примеч. науч. ред.*

в свой веб-броузер, обратившись по адресу *http://localhost/phpinfo.php*. Если все в порядке, на странице будет показана конфигурация модуля PHP4.

Сервер LAMP в действии

Теперь все компоненты сервера LAMP установлены и пора запустить несколько примеров.

Если вы еще не сделали этого, читая предыдущий раздел, советуем проверить сейчас свою установку с помощью очень простого файла PHP. Сохраните следующий код PHP в файле с именем *info.php*:

```
<?
phpinfo();
?>
```

Поместите этот файл в каталог, в котором ваш сервер Apache ищет файлы контента. Часто им оказывается */usr/local/httpd/htdocs*, и в нем могут уже находиться файлы, которые ваш дистрибутив поместил туда во время установки (во всяком случае, если вы устанавливали Apache с инсталляционного носителя). Если что-то не так, найдите файл конфигурации Apache *httpd.conf*. Часто этот файл находится в каталоге */etc/httpd/*, но если в вашей системе это не так, поищите его командой:

```
locate httpd.conf
```

Найдите в этом файле строку, начинающуюся с `DocumentRoot`. Нужно найти указанный в ней каталог, в котором должен находиться подкаталог *htdocs*, и поместить туда файл *info.php*. Теперь, используя любой броузер, обратитесь к URL *http://localhost/info.php*. В результате вы получите некоторую информацию о настройке вашего модуля PHP.

В PHP есть ряд встроенных функций для работы с данными, хранящимися в MySQL (и других базах данных).

Реляционная база данных состоит из ряда таблиц. Если у вас есть необходимые права доступа, PHP может посылать запросы и обрабатывать данные в этих таблицах. Теперь мы можем написать несколько сценариев PHP для работы с таблицами базы данных. Мы предполагаем, что вы создали базу данных `test_database` и таблицу `comment_table`, а также пользователя `olof`, как было описано выше.

С помощью любого текстового редактора введите следующий код, создающий небольшую HTML-страницу, с помощью которой можно помещать данные в эту таблицу посредством HTML-формы:

```
<html>
<?php
if ($comment){
$conn=mysql_connect("localhost","olof","secret" )
 or die("Could not connect to MySQL as olof");

mysql_select_db("test_database", $conn)
```

```
  or die("could not select the test_database");

 $string="INSERT INTO comment_table VALUES ('0','$comment')";
 mysql_query($string)
  or die(mysql_error());}
?>

<form action=<? echo $PHP_SELF ?> method="get">
 <input  type="text" name="comment" size="80"> <br>
<input type="submit">
</form>

</html>
```

Чтобы выполнить этот сценарий, сохраните его как файл с расширением *.php*, скопируйте в каталог документов вашего веб-сервера и обратитесь к сценарию через веб-броузер. Например, если вы сохранили сценарий как файл *edit.php*, можно перейти на URL *http://localhost/edit.php*, и сценарий будет выполнен. Веб-сервер знает, что все, находящееся между тегами <? и ?>, должно быть пропущено через модуль PHP. Таким образом код PHP можно непосредственно встраивать в страницу HTML.

Теперь, когда мы можем вводить комментарии в нашу базу данных, мы хотим посмотреть на них. Поэтому приведем сценарий для чтения из базы данных:

```
<html>
<?php
$conn=mysql_connect("localhost","olof","secret" )
 or die("Could not connect to MySQL as olof");

mysql_select_db("test_database", $conn)
  or die("could not select the test_database");

 $string="SELECT * FROM comment_table";
 $result= mysql_query($string)
  or die(mysql_error());

 $numbers_cols= mysql_num_fields($result);

 print "<b>query: $string</b>";
 print "<table border =1>\n";
 print "<tr>";
 print "<td> ID</td>";
 print "<td> Comment </td>";
 print "</tr>";
while (list($id,$comment) = mysql_fetch_array($result)){
        print "<tr>";
        print "<td>$id</td>";
        print "<td>$comment</td>";
        print "</tr>";}
 print "</table>";


?>
</html>
```

Как видите, мы использовали теги HTML для разметки таблиц, в которых показывается содержимое базы данных, что вполне естественно и очевидно.

Мы стремились привести по возможности более простые примеры, чтобы не перегружать вас большим объемом информации. Если вы хотите глубже окунуться в удивительный мир LAMP, советуем прочесть хорошую книгу «Web Database Applications with PHP & MySQL» Хью Вильямса (Hugh E. Williams) и Дэвида Лэйна (David Lane), изданную O'Reilly, или «MySQL/PHP Database Applications» Джея Гринспэна (Jay Greenspan) и Брэда Бергера (Brad Burger), изданную John Wiley & Sons.

A

Источники информации по Linux

В этом приложении перечислены различные сетевые ресурсы информации по Linux. Все эти документы доступны в электронном виде через Интернет, но многие из них имеются в печатной форме.

Дистрибутивы Linux часто содержат некоторую часть этой документации и обеспечивают ее доступность в работающей системе. Как уже отмечалось ранее, документацию в системе Linux можно найти в ряде мест, в том числе на страницах руководства Unix, info-страницах GNU и справочной документации в формате HTML (например, ее выводит KDE Help Center).

В большинстве дистрибутивов Linux документация по отдельным программам, такая как файлы README и примечания к выпуску, хранятся в каталоге */usr/share/doc*, а если у вас установлен исходный код ядра, то поставляемая с ним документация обычно находится в каталоге */usr/src/linux/Documentation*.

Информацию более интерактивного характера пользователи Linux обычно получают из следующих источников:

Телеконференции

Большинство телеконференций, относящихся к Linux, находится в иерархии *comp.os.linux*, но есть много региональных, относящихся к конкретному дистрибутиву или посвященных проектам open source.

IRC

Internet Relay Chat (IRC) – традиционная чат-система Unix, которую часто используют для получения немедленных ответов на вопросы, задаваемые пользователями.

Почтовые списки рассылки

Большинство проектов Linux и open source, от ядра до KDE, пользуется почтовыми списками рассылки в качестве главного средства связи между разработчиками проекта. Во многих группах пользователей есть почтовые списки с локальным уклоном.

Проект документирования Linux (LDP)

Основным источником свободно распространяемой документации по Linux служит Проект документирования Linux (Linux Documentation Project, LDP). Главный веб-сайт LDP находится на *http://www.tldp.org*, но есть много зеркальных сайтов, раскиданных по всему свету, и какой-нибудь из них может оказаться ближе к вам или быть менее загруженным.

Документация в LDP организована по нескольким типам. *Guide* – это длинные, часто объемом с книгу, руководства, детально описывающие такие большие темы, как, скажем, работу в сети. *HOWTO* – среднего объема документы, освещающие различные конкретные задачи, например настройку звуковых карт. Для более мелких задач по специальным темам, не заслуживающим полного HOWTO, есть *mini-HOWTO*. Наконец, существует целый ряд *FAQ* с ответами на часто возникающие вопросы по Linux.

Документы LDP предоставляются в ряде различных форматов, в том числе HTML, простой текст, PDF и PostScript. Многие документы переведены на другие языки группой переводчиков-добровольцев.

В библиографии более подробно перечислен ряд документов LDP.

FTP-сайты

Хотя в вашем дистрибутиве Linux содержатся компилированные двоичные пакеты многих приложений Linux, но часто приходится компилировать приложения из исходного кода, если приложение не доступно в двоичном виде, если вы хотите посмотреть на исходный код или вы просто предпочитаете самостоятельно собирать приложения из исходного кода. Ниже приводится несколько популярных сайтов, где можно взять исходный код программного обеспечения для Linux.

Многие из этих сайтов пользуются исключительной популярностью и очень заняты. Очень рекомендуется пользоваться зеркальными сайтами (другими компьютерами, которые регулярно загружают программное обеспечение с главного сайта), которые ближе к вам расположены. Обычно с зеркальным сайтом легче соединиться, и работает он быстрее.

Помимо перечисляемых здесь FTP-сайтов, у многих веб-сайтов, приводимых в этом приложении, есть соответствующие FTP-сайты для загрузки.

FTP-сайт	Описание
ftp://ftp.gnu.org	Основной сайт для загрузки проекта GNU
ftp://ftp.ibiblio.org	Большой архивный сайт Linux, ранее называвшийся *metalab.unc.edu* и *sunsite.unc.edu*, один из первых появившихся архивных сайтов Linux
ftp://ftp.x.org	Архив программного обеспечения для X Window System

Сайты World Wide Web

В этом разделе перечислены лишь некоторые из тысяч веб-сайтов Интернета, посвященных Linux. Они довольно произвольно разбиты на ряд категорий. Из-за динамической природы Web некоторые из них уже не будут действовать, и, несомненно, появится много новых, когда вы будете читать эти строки.

Общая документация

Эти сайты предлагают электронную документацию, статьи о Linux или информацию, относящуюся к отдельным областям Linux.

Веб-сайт	Описание
http://www.andamooka.org	Веб-сайт, содержащий ряд бесплатных книг, некоторые из которых относятся к Linux, например «KDE 2.0 Development»
http://www.linas.org/linux	Сайт Linux Enterprise Computing
http://www.linux-laptop.net	Сайт Linux on Laptops
http://www.linuxfocus.org	Linux Focus, бесплатный сетевой журнал
http://www.linuxgazette.com	Linux Gazette, ежемесячный электронный журнал
http://www.linuxjournal.com	Веб-сайт журнала Linux Journal
http://www.linuxmagazine.com	Веб-сайт журнала Linux Magazine
http://www.linuxnewbie.org	Информационный сайт для новичков в Linux
http://www.tldp.org	Главный сайт Linux Documentation Project

Проекты Open Source

Здесь перечислены веб-сайты некоторых наиболее популярных программных проектов open source и бесплатного программного обеспечения.

Веб-сайт	Описание
http://koffice.kde.org	Проект KDE Office Suite
http://www.abisource.com	Текстовый процессор AbiWord
http://www.alsa-project.org	Проект звуковых драйверов Alternative Linux Sound Architecture (ALSA)
http://www.apache.org	Проект веб-сервера Apache
http://www.cups.org	Common UNIX Printing System (CUPS)
http://www.dosemu.org	Проект Linux DOS Emulator
http://www.gnome.org	Проект GNOME Desktop
http://www.gnu.org	Проект GNU

Веб-сайт	Описание
http://www.isdn4linux.de	ISDN4Linux, поддержка ISDN в Linux
http://www.kde.org	K Desktop Environment (KDE)
http://www.kernel.org	Официальный сайт ядра Linux
http://www.linux-usb.org	Проект Linux USB
http://www.mozilla.org	Проект веб-броузера Mozilla
http://www.mysql.com	База данных MySQL
http://www.openoffice.org	Проект офисного пакета OpenOffice, версии open source для StarOffice
http://www.postfix.org	Проект почтовой программы Postfix
http://www.povray.org	Трассировщик лучей Persistence Of Vision
http://www.winehq.com	Проект WINE
http://www.xfree86.org	Проект XFree86

Языки и инструменты программирования

Это сайты популярных языков программирования для Linux и программных проектов Linux.

Веб-сайт	Описание
http://www.sourceforge.net	Головной сайт для многих программных проектов Linux; предоставляет место для документации, хранилище исходного кода, отслеживание ошибок и компиляцию программ
http://savannah.gnu.org	GNU Savannah, предлагает функции, аналогичные SourceForge, но официально признан Free Software Foundation, поскольку все ПО сайта лицензировано по GPL
http://www.blackdown.org	Домашняя страница проекта переноса Java на Linux
http://www.perl.com	Официальный сайт языка программирования Perl
http://www.php.net	Веб-сайт языка программирования PHP
http://www.python.org	Домашняя страница языка программирования Pyhon
http://www.tcl.tk	Один из главных сайтов языка программирования TCL/Tk
http://sal.kachinatech.com	SAL, сайт, посвященный научным приложениям под Linux (Scientific Applications on Linux)

Новостные и информационные сайты

Эти сайты предлагают новости, интересные пользователям Linux.

Веб-сайт	Описание
http://www.desktoplinux.com	Сайт, посвященный Linux на рабочем столе
http://www.linux.com	Сайт с общей информацией и новостями Linux (с очень приятным URL)
http://www.linuxdailynews.net	Linux Daily News
http://www.linuxtoday.com	Linux Today
http://www.lwn.net	Linux Weekly News
http://linuxsecurity.com	Новости и общая информация по проблемам безопасности в Linux
http://www.newsforge.com	NewsForge
http://www.slashdot.org	Новостной и дискуссионный сайт, рекламирующий себя как «Новости для компьютерных фанатов. Все существенное»
http://www.theregister.co.uk	The Register, британский сайт для IT-индустрии с уклоном в Linux
http://www.theregus.com	Американское издание The Register
http://www.varlinux.org	Новостной сайт VarLinux, для Value Added Resellers (VARs)

Сайты каталогов и загрузки программного обеспечения Linux

Ниже перечислены некоторые сайты, хранящие большие библиотеки программного обеспечения Linux со средствами поиска и ссылки на сайты загрузки.

Веб-сайт	Описание
http://www.freshmeat.net	Большой каталог программного обеспечения Linux и open source
http://www.icewalkers.com	Еще один большой каталог программного обеспечения Linux
http://www.linuxapps.com	Большой каталог программного обеспечения Linux
http://www.portalux.com	Linux Center, портал с каталогом приложений и ссылками на новости и статьи

Дистрибутивы Linux

Вот длинный, но отнюдь не исчерпывающий список различных дистрибутивов Linux. Сюда входят как дистрибутивы, созданные большими компаниями, например, Red Hat, так и небольшие специализированные дистрибутивы, разработанные отдельными людьми или небольшими группами.

Веб-сайт	Описание
http://www.archlinux.org	Arch Linux
http://www.asp-linux.com	ASPLinux
http://www.bearops.com	BearOps Linux
http://www.beehive.nu	Beehive Linux
http://www.connectiva.com	Connectiva
http://www.crux.nu	CRUX Linux
http://www.debian.org	Debian GNU/Linux
http://www.elxlinux.com	ELX Linux
http://www.eridani.co.uk	Eridani Linux
http://www.icepack-linux. com	Icepack Linux
http://www.libranet.com	Libranet Linux
http://www.lindows.com	Lindows.com
http://www.lycoris.com	Lycoris
http://www.linuxmandrake.com	Mandrake Linux
http://murix.sourceforge.net	Murix
http://www.ibiblio.org/peanut	Peanut Linux
http://www.probatus.com	Probatus
http://www.redhat.com	Red Hat Linux
http://www.rocklinux.org	ROCK Linux
http://www.rootlinux.org	ROOT Linux
http://www.sco.com	SCO Linux (бывший Caldera)
http://www.slackware.com	Slackware Linux
http://www.suse.com	SuSE Linux
http://www.turbolinux.com	Turbolinux
http://www.unitedlinux.com	United Linux
http://www.ibiblio.org/ vectorlinux	Vector Linux
http://www.voodoolinux.com	Voodoo Linux
http://www.xandros.com	Xandros Desktop Linux
http://www.yellowdoglinux.com	Yellow Dog Linux

Коммерческие разработчики программного обеспечения Linux

Ниже перечислены некоторые компании, предлагающие коммерческое программное обеспечение и услуги для платформы Linux, но не дистрибутивы.

Веб-сайт	Описание
http://www.codeweavers.com	CodeWeavers, разработчик CrossOver Office и CrossOver Plugin, продуктов, основанных на программном обеспечении проекта Wine, предлагающие возможность выполнения приложений Windows под Linux.
http://www.oeone.com	OEone, разработчик основанной на Linux операционной среды.
http://www.opensound.com	4Front Technologies, разработчик улучшенных звуковых драйверов ядра для Linux и некоторых других операционных систем.
http://www.trolltech.com	TrollTech, разработчик Qt, межплатформенного графического инструментария. Qt лежит в основе KDE.
http://www.vistasource.com	VistaSource, бывшая Applix, разработчик Applixware Office Suite.
http://www.vmware.com	VmWare, продает программное обеспечение виртуальных машин, позволяющих выполнять одну операционную систему поверх другой, например, Windows поверх Linux и наоборот.
http://www.ximian.com	Ximian, разрабатывает и поддерживает улучшенную версию настольной рабочей среды GNOME.

RFC и другие стандарты

Вот некоторые из многочисленных сайтов, на которых размещены стандарты, используемые Linux и Интернет.

Веб-сайт	Описание
http://www.faqs.org/rfcs	Архивный сайт Requests For Comments (RFC), технических документов, описывающих многие из протоколов, на которых основывается Интернет. Содержит много других документов стандартов и FAQ.
http://www.freestandards.org	The Free Standards Group, некоммерческая организация, занимающаяся распространением технологий open source путем разработки, применения и продвижения стандартов.
http://www.linuxbase.org	The Linux Standard Base, проект Free Standards Group для разработки и распространения ряда стандартов, которые должны улучшить совместимость дистрибутивов Linux и позволить приложениям выполняться на любой совместимой со стандартами Linux-системе.

Веб-сайт	Описание
http://www.w3c.org	World Wide Web Consortium, организация, которая разрабатывает спецификации, руководящие принципы, программное обеспечение и инструментарий для World Wide Web.

Разное

Наконец, вот несколько сайтов, которые не удалось отнести к какой-либо из выделенных категорий.

Веб-сайт	Описание
http://counter.li.org	The Linux Counter, уникальный сайт, пытающийся собрать данные и оценить количество пользователей Linux в мире.
http://www.li.org	Linux International, некоммерческая организация, занимающаяся пропагандой Linux и Linux-сообщества.
http://www.google.com	Вероятно, самый популярный поисковый механизм в Web, который обслуживается «фермой серверов», состоящей из нескольких тысяч Linux-машин.

B

Установка Linux на системы Digital/Compaq Alpha

В 1992 году Digital Equipment Corporation (позднее приобретенная Compaq) представила 64-битную суперскалярную, основанную на RISC-ядре, архитектуру под названием Alpha, которая получила восторженные отзывы специалистов за свою скорость. Linux является заманчивой альтернативой традиционным операционным системам, распространяемым вместе с Alpha, особенно для более старого оборудования Alpha, реализуемого сейчас на рынке. Однако процесс установки варьируется от системы к системе, поскольку Alpha быстро развивалась и продавалась в течение ряда лет с широким спектром аппаратных средств и встроенных программ («firmware», программы начального запуска, хранящиеся в ROM). Данное приложение является введением в основные аспекты установки Linux. Помимо этого необходимо прочесть все документы по установке Linux и по имеющимся техническим средствам, а также проявить определенную настойчивость и желание экспериментировать.

Полное описание систем Alpha должно было бы охватывать годы эволюции оборудования, начиная со старой системы UDB и заканчивая современными системами серий DS и ES, а также стандартными конфигурациями OEM. Ввиду многообразия конфигураций BIOS и загрузочных параметров невозможно предоставить детальные инструкции по установке для каждого типа систем Alpha. Мы надеемся, что данное приложение поможет новичкам в архитектуре Alpha понять, что надо делать при установке новой системы.

Настоящее описание не охватывает системы, основанные на VAX, MIPS, AMD или Intel CPU, или оборудование, использующее периферийные технологии с системами на основе Alpha. Мы сосредоточимся только на установке систем Linux на системы Alpha и компонентах, производимых Compaq или по лицензиям Alpha-технологии, таких как Samsung Semiconductor, API NetWorks Inc. или Mitsubishi Semiconductor.

История и современное состояние системы Alpha

В 1992 году Digital Equipment Corporation, известная также как DEC или Digital, представила систему Alpha с поддержкой семи аппаратных платформ, трех операционных систем, множества сетевых протоколов и разных компиляторов.

Alpha представляет собой крупнейший технический проект, когда-либо предпринятый Digital, к которому были подключены более 30 инженерных групп в 10 странах. Это был не первый основанный на RISC процессор, выпущенный Digital, но он был первым, который Digital решила пустить в свободную продажу. Фирма Digital Semiconductor (DS) была создана как внутренняя бизнес-группа для производства, продажи и распространения процессоров Digital на коммерческом рынке.

С целью соответствия уровню требований и развивающейся технологии Digital распределила производство процессора Alpha, заключив соглашения с Samsung Electronics и Mitsubishi Electric на производство текущих и будущих разработок процессора Alpha. Кроме того, по этим соглашениям Samsung и Mitsubishi получили лицензии на поиск рынков сбыта, продажу и распространение процессоров Alpha по всему миру, а также были включены в совместные проекты разработок, касающиеся семейства процессоров Alpha.

Относительно небольшая установочная база систем Alpha и тот факт, что большинство существующих систем являются «развивающимися платформами», позволяющими приспосабливать и настраивать их с помощью огромных архивов документации по аппаратным средствам, стимулировали дальнейшее развитие чипсетов Alpha. Однако это создает трудности для разработчиков Linux в плане охвата широкого диапазона систем рамками одной универсальной процедуры установки.

Compaq решилась на создание собственной архитектуры и операционной системы промышленных серверов как альтернативы Microsoft и Intel. 26 января 1998 года Digital и Compaq Computer Corporation объявили об объединении капитала в 9,6 миллиарда долларов, после чего Digital стала дочерним предприятием Compaq. DS вошла с многомиллионным пакетом, и имя Digital было включено в Compaq как название марки.

После слияния Compaq продолжила разработку чипа Alpha, и на 2002 год было назначено представление EV7. К сожалению, Compaq также объявила, что на EV7 эти разработки будут прекращены и в будущем в серверах Compaq верхнего ряда на смену Alpha придет архитектура Intel IA64.

Таким образом, архитектура Alpha является суперскалярной 64-битной RISC-архитектурой открытого промышленного стандарта, разрабатываемой фирмой Compaq и производимой Samsung и дочерними предприятиями.

Перенос Linux на Alpha и дистрибутивы

Порт Linux для систем Alpha возник не вчера. Он начинался как небольшое добавление к ядру Linux. Первая «заплата» для ядра была создана при фи-

нансировании и поддержке со стороны Digital. Даже при наличии всех «заплат» к ядру большая часть драйверов и «пользовательского кода» (userland), из которого состоит дистрибутив, не были «полностью 64-битными». (Иными словами, в программах для работы с 64-разрядными словами могли возникать отказы из-за неявных 32-разрядных вычислений или объявлений в программах более низкого уровня.) Само ядро не было чисто 64-разрядным вплоть до появления нестабильного ядра версии 2.1.x. Когда началась разработка ядер версии 2.1.x, порт для Alpha был включен непосредственно в систему исходного кода ядра Linux. На этом этапе Alpha-порт ядра Linux напрямую поддерживался в основных выпусках ядра.

Во время разработки ядра версии 2.1.x ядро Linux и его драйверы были сделаны полностью 64-разрядными. Большинство ошибок выравнивания в ядре было исправлено, и пользовательский код стал в большей степени 64-битным. В это же время фирма Red Hat Software быстро осознала, что 64-битная архитектура стала ориентиром рынка процессоров для серверов и рабочих станций, и создала полный Alpha-порт своего дистрибутива Linux – Red Hat Linux 2.1. Затем последовали и другие дистрибутивы, включая Debian и SuSE:

Debian

Дистрибутив Debian Linux является «официальным» дистрибутивом Free Software Foundation GNU/Linux, предоставляющим один из самых обширных наборов программного обеспечения. См. *http://www.debian.org*.

Red Hat

Фирма Red Hat Software несколько лет производит Alpha-порт своего дистрибутива. В последнее время, с целью получения прибыли, RedHat сосредоточилась на своем основном бизнесе, и Alpha-порт несколько отодвинулся на задний план. Compaq вступила в сотрудничество с RedHat с целью продолжить выход новых версий, но неизвестно, как долго Compaq будет придерживаться этой политики. См. *http://www.redhat.com*.

SuSE

Дистрибутивы SuSE Linux разработаны с учетом потребностей интернационализации и поддержки графического интерфейса. (SuSE учитывает требования европейского рынка.) SuSE также использует RPM для обновления пакетов и имеет полноценный порт Alpha. См. *http://www.suse.com*.

Другие дистрибутивы

Ряд более мелких дистрибутивов имеет Alpha-порты. В их число входят RockLinux (*http://www.rocklinux.org*), Polish Linux Distribution (*http://www.pld.org. pl*) и Kondara (*http://www.kondara.org*).

Чипсеты

Существует пять классов процессоров Alpha: 21064, 21066, 21164, 21264 и ожидаемый 21364. Классы 21064 и 21066 являются архитектурами Alpha первого поколения, но класс 21066 включает функции, обычно передаваемые чипсетам, в сам процессор, что создает особые характеристики и требования платформы. Классы 21164 и 21264 представляют второе и третье по-

коления архитектур, соответственно. Различные архитектуры были сопряжены с разными системными шинами и интерфейсами, которые подразделяют типы процессоров на различные семейства компьютеров Alpha от настольных систем до суперкомпьютерных кластеров.[1]

При установке Linux с дистрибутивов на CD-ROM нужно учитывать, что эти классы архитектур имеют различную поддержку, обеспечиваемую ядром Linux для функций чипсетов и системных узлов и интерфейсов, использующих эти функции. Если вы не можете загрузить ядро, которое кажется подходящим для вашей системы, попытайтесь использовать подобное ядро для родственной системы или попробуйте общее ядро. Если и оно не работает, обратитесь к FTP-сайту AlphaLinux и попробуйте использовать ядро более ранней или более поздней, разрабатываемой версии. Однако в большинстве случаев ваша установка пройдет нормально, если вы правильно выбрали файлы образов для загрузки и установки Linux.

Источники информации

Прежде чем вы выберете дистрибутив Linux, воспользуйтесь обширными сведениями о Linux и Alpha, имеющимися в Интернете. Масса подробной информации о процессорах Alpha и платформах, предназначенных для их поддержки, доступна для загрузки и печати, включая бесплатно поставляемые технические руководства, которые слишком дороги или отсутствуют в продаже. Примите во внимание, что много полезной документации можно получить у других поставщиков, не тех, кто изготовил ваш процессор или систему, а также из других источников, относящихся к другим дистрибутивам.

Вероятно, у вас будет хотя бы руководство по аппаратной части для вашей машины с описанием конфигурации системы и консолей ПЗУ, а также с кодами ошибок и диагностикой на случай возникновения проблем. В конце всех руководств по машинам и платам Digital есть приложение с перечислением близких документов, часть из которых тоже может оказаться полезной. Номера документов указывают на требуемые файлы из библиотечного архива Alpha.

Существуют веб-сайты, предоставляющие файлы FAQ, а также другую полезную информацию о некоторых конкретных системах Alpha.

Необходимо также иметь документы по установке и дополнения к ним для вашего инсталляционного CD-ROM и информацию, касающуюся ошибок и исправлений для выбранного вами дистрибутива. Если ваша документация неполна или описывает обычную процедуру установки Linux (вместо конкретной установки Linux на Alpha), в качестве руководства по установке можно использовать последовательность действий, описанную ниже.

Здесь представлен неполный список важных источников информации в Интернете об аппаратных средствах Alpha и об установке Linux на этих системах.

---

[1] Здесь мы не будем касаться применения процессоров Alpha во встроенных системах и дисплейных терминалах.

Сайты AlphaLinux

Организация AlphaLinux

Официальный сайт AlphaLinux в Интернете находится по адресу *http://www.linuxalpha.org*.

Сайт Compaq AlphaLinux

Этот сайт содержит интересную информацию об использовании Linux на оборудовании Alpha и находится по адресу *http://www.compaq.com/products/software/linux/index.html*.

Списки рассылки AlphaLinux

AXP-List от RedHat

Пошлите e-mail с темой сообщения `subscribe` на следующий адрес: *axp-list-request@redhat.com*.

Список рассылки Debian Alpha

Пошлите e-mail на адрес *debian-alpha-request@lists.debian.org* со строкой `subscribe` в теле сообщения.

FTP-сайты AlphaLinux

Официальный FTP-сайт AlphaLinux FTP

Этот сайт находится по адресу *ftp://ftp.linuxalpha.org*.

FTP-сайт Digital

Этот сайт находится по адресу *ftp://gatekeeper.dec.com*.

Минимальные требования к аппаратным средствам

Для работы Linux на системах Alpha требуется всего 8 Мбайт оперативной памяти (RAM), но большинство установочных программ, таких как программа установки Red Hat, требуют как минимум 32 Мбайт. Минимальное пространство на диске, необходимое для установки, составляет 170 Мбайт. Для запуска полноценной системы X Window и рабочего стола и обеспечения достаточной памяти для приложений необходимо иметь как минимум 16 Мбайт оперативной памяти и 500 Мбайт памяти на жестком диске.

Многие пользователи самых последних дистрибутивов Linux обнаружили, что для завершения установки необходимо 24 Мбайт или 32 Мбайт оперативной памяти. Хотя для запуска Linux на правильно сконфигурированной системе Alpha (с ядром, скомпилированным для поддержки свойств именно данной системы) нужно не более 16 Мбайт; для установки некоторых дистрибутивов вам может потребоваться больше памяти. Альтернативное решение состоит в установке более ранних версий ядра, имеющих меньшие размеры.

Во время установки требуется очень много оперативной памяти, поскольку объединяются требования памяти для начального загрузчика, для Milo (если он используется) и для загружаемого образа ядра Linux, а не потому, что велики требования к памяти со стороны самого ядра Linux. Если вы не мо-

жете завершить начальную установку, можно попытаться использовать для загрузки вашего дистрибутива Linux более старую компактную версию Milo или Milo, который хранится в неизменяемой оперативной памяти. Можно также выбрать более компактные приложения, чтобы сберечь память и место на жестком диске. Более детальная информация находится на *http://www.linuxalpha.org*.

Поддержка дисков IDE/ATAPI

Некоторые системы Alpha, особенно те, которые предназначены для использования в качестве сетевых серверов, не поддерживают интерфейсы IDE или ATAPI.[1] Мы рекомендуем быстрый SCSI-диск в качестве базового элемента системы, как при внутренней установке, так и при внешнем подключении. Однако если ваша система имеет свободный слот PCI, добавьте текущую карту контроллера PCIbus ATAPI/EIDE, поддерживаемого Linux, для того чтобы можно было также использовать более дешевые диски EIDE или быстрые приводы CD-ROM. Загружать систему с этого устройства вы, скорее всего, не сможете, поскольку у консоли SRM не будет для него драйвера.

Мышь и последовательные порты

Довольно часто при установке Linux возникают проблемы, связанные с подключением мыши через последовательный порт. Некоторые программы-установщики Linux неправильно отображают устройства на последовательные порты во время конфигурации ядра. Большинство жалоб касается автоматического определения мыши, установленной на первый последовательный порт. Мы рекомендуем устанавливать трехкнопочную мышь PS/2-типа, если в системе имеется аппаратный порт PS/2, а не использовать COM-порт. Более того, не подключайте к первому последовательному порту модем, т. к. Milo будет ретранслировать вывод модема на первый последовательный порт, что может привести к непредсказуемым результатам.

Подготовка к установке системы и общая процедура действий

Существует несколько различных систем Alpha, каждая из которых требует особого метода установки Linux. Поскольку ядро Linux компилируется поразному с учетом особенностей архитектуры CPU и системы, необходимо идентифицировать имеющееся у вас Alpha-оборудование, на случай если общее ядро работает плохо. Ядру Linux различные системы Alpha известны, в

[1] Первые системы Alpha поддерживали интерфейс SCSI-2 со скоростью 10 Мбит/с, а некоторые из последних плат системы Alpha поддерживают «fast and wide» SCSI-3 со скоростью работы 160 Мбит/с. Самые ранние системы (Jensen) поддерживали контроллер шины ISA Adaptec 1740, но большинство SCSI-контроллеров относятся к NCR (LSI Logic) или Adaptec, и в те времена считались высокопроизводительными.

основном, по кодовому техническому названию. Если вы знаете официальное название системы, то обычно можно найти кодовое название. Один из источников, где можно найти эти названия, это таблица типов систем на *http://www.linuxalpha.org/docs/systypes.txt*.

Прежде чем устанавливать операционную систему, нужно выяснить параметры графической подсистемы машины, аудиокомпонентов, системной памяти, класса процессора (CPU), интерфейсы и размеры дисков. Кроме того, нужно знать о наличии других операционных/файловых систем (если таковые имеются) и подключенных периферийных устройствах. Особенно следует обратить внимание на приводы для чтения CD-ROM и гибких дисков. Некоторые системы Alpha для полной установки Linux требуют изменить конфигурацию микропрограммного обеспечения и даже изменить оборудование.

Возможные проблемы аппаратной несовместимости

Выбор дистрибутива Linux может зависеть от того, каким оборудованием вы располагаете и какие другие операционные системы на нем работают. После того как вы соберете информацию об имеющемся оборудовании, вы сможете определить, какой дистрибутив наилучшим образом отвечает вашим потребностям. Получив систему, способную загружаться, вы всегда можете скомпилировать нужные пакеты исходного кода из других дистрибутивов. Вот несколько примеров того, как оборудование диктует выбор дистрибутива или метода установки:

- Некоторые системы Alpha требуют установить перемычки для загрузки Linux и установки ее в качестве «родной», загружаемой через микропрограммное обеспечение операционной системы.
- Если ваша система не имеет дисковода для гибких дисков, вам, возможно, не удастся установить на ней желаемый дистрибутив Linux из-за ограничений в поддержке файловых систем и устройств в прошивке систем, которые не имеют дисководов для гибких дисков. Мы рекомендуем установить или подключить дисковод в систему. Если дисковод просто «умер», его замена обойдется совсем недорого.[1]
- Если Linux устанавливается на компьютер параллельно с другой операционной системой (установленной в другом разделе жесткого диска или на другом жестком диске), разные версии Linux предоставляют разные уровни совместимости по файловым системам и взаимодействию с другими операционными системами. Например, если вы желаете работать на вашей системе с Tru64 Unix (Digital Unix, или DU) и с дистрибутивом Debian Linux Alpha, их основная (по умолчанию) файловая система, дисковые разделы и метки дисков будут конфликтовать. Аналогично, если вы создадите BSD-разметку дисков в Linux, она не будет восприниматься DU.

[1] Даже сверхтонкий дисковод для гибких дисков в 3,5 дюйма, использовавшийся в более старых системах Multia и Alphabook 1, может быть заменен на стандартный дисковод для ноутбука. В более новых конструкциях Alpha большинство аппаратных средств взаимозаменяемо со стандартными устройствами OEM PC.

Если вы хотите установить Linux на Alpha параллельно с другой операционной системой, надежнее всего создать разделы в другой операционной системе и определить дисковый раздел для установки Linux.

- Утилита *fdisk* от Linux не может разметить файловые системы DOSFS так, чтобы с ними могли работать SRM и другие утилиты. Поэтому необходимо форматировать дискеты MS-DOS для конфигурирования системы средствами MS-DOS или Windows NT (или покупать отформатированные в MS-DOS дискеты).
- Первая установка Linux на старые машины (особенно UDB/Multias) может закончиться неудачей, если разряжена встроенная батарейка в вашей системе. Жалобы на материнские платы, проданные с разряженными батарейками, – обычное дело. В некоторых системах стоит подзаряжаемый аккумулятор, который, однако, не будет заряжаться при включенном питании машины, если в момент включения он был разряжен. Например, монитор не будет работать, и изменения в микропрограммах, которые необходимы для завершения установки самозагружающейся системы, не будут сохранены.

Варианты установки

После того как вы собрали сведения о своем оборудовании и выбрали дистрибутив Linux, выберите способ установки. Системы Alpha имеют больше вариаций, чем типичные системы PC (т. к. они были разработаны для повсеместного использования – от простых дисплейных терминалов до суперкомпьютеров). Поэтому вам потребуется выяснить возможности вашего оборудования и выбрать метод загрузки, который вы будете использовать с дистрибутивом Linux. В противном случае вы можете обнаружить, что не сможете завершить установку, как того хотели, или что Linux будет трудно загрузить без повторной установки.

Если для установки Linux вы подключили привод CD-ROM, проверьте по документации к вашему оборудованию и к пакету Linux, что CD-ROM будет восприниматься BIOS или что загрузчик SRM будет его поддерживать. SRM распознает и поддерживает большинство стандартных SCSI CD-ROM и IDE-приводов, подключаемых с помощью интегрированных контроллеров. Но IDE CD-ROM, подключенный к дополнительному контроллеру PCI IDE, SRM, скорее всего, не увидит.

Хотя сетевая установка Linux (хранение файлов на одной машине и загрузка их по сети на другую машину, где вы хотите запустить Linux) возможна, в основных дистрибутивах Linux постоянно возникают проблемы с поддержкой такой возможности. Мы не будем описывать эту технологию. Мы предполагаем, что вы устанавливаете Linux на автономную Alpha-систему, а настройку для работы в сети можно выполнить позже.

Настройка начальной загрузки на некоторых старых системах Alpha требует установки перемычек на материнской плате или изменения системных данных в энергонезависимой RAM, или того и другого вместе. Возможно,

потребуется сначала переустановить перемычки перед инсталляцией, а потом снова переставить их для настройки системы.

Даже рутинная процедура установки AlphaLinux сложнее установки Linux на обычном Intel PC. Архитектура Intel PC предоставляет интерфейс к оборудованию в интегрированной системе BIOS, тогда как системы Alpha требуют, чтобы самозагрузка системы происходила путем определения и загрузки микропрограммного интерфейса перед загрузкой операционной системы. Различные способы этого обсуждаются в следующем разделе.

Типы встроенных программ

Каждая система Alpha содержит по крайней мере одну встроенную программу (firmware program), хранящуюся в системном ПЗУ или перезаписываемом ПЗУ, которая обеспечивает среду выполнения программы конфигурирования для установки аппаратной конфигурации, параметров начальной загрузки и других задач сопровождения системы. С этой встроенной программы обычно и начинается установка Linux. Есть следующие типы встроенных программ:

System Reference Maintenance (SRM)

Есть на системах, сконфигурированных для Tru64 Unix или Open VMS. Единственное исключение составляют системы, построенные на материнской плате UX, в которых есть собственная программа ARC BIOS, и серия Digtal XL, для которой не было выпущено SRM. SRM – лучшая среда для установки Linux, ей мы уделим основное внимание в этом приложении. SRM описана в электронном руководстве *http://ftp.digital.com/pub/Digital/info/semiconductor/literature/srmcons.pdf*.[1]

Alpha Reference Console (ARC)

Есть на старых системах, сконфигурированных для Windows NT. Она предлагает простой интерфейс консоли на основе меню для управления оборудованием системы.

AlphaBIOS

Замена ARC с целью сделать установку ОС на системах Alpha более универсальной и автоматизированной, как на PC с архитектурой Intel. Предлагает графический интерфейс пользователя.

Встроенные программы занимают мало места и весьма эффективны. В системной ROM обычно достаточно места для нескольких таких программ, а также других необходимых программ, например, отладчика и средств диагностики, которые не следует затирать.

Раньше для загрузки Linux во многих установках на Alpha использовалась Milo, в основном потому, что консоль SRM, используемая для загрузки

---

[1] Консольную программу SRM для Multia, AlphaStation, AlphaServer, AlphaPC164 и AXPpcisystems можно взять с *ftp://gatekeeper.dec.com* или *http//ftp.digital.com/pub/DEC/Alpha/firmware*.

Tru64 Unix и OpenVMS, для многих машин была труднодоступна. Стандартная встроенная программа первоначально загружалась через обращение к системной консоли при загрузке системы и указание загрузить Milo с дискеты. Затем текущий образ мини-загрузчика Milo загружался с дискеты и, в свою очередь, получал команду на загрузку ядра Linux с CD-ROM или с загрузочной дискеты Linux.

Сейчас Compaq сделала консоль SRM доступной на большинстве машин Alpha (исключение составляет серия XL). Поскольку SRM теперь гораздо легче доступна, она стала предпочтительным способом загрузки Linux. Milo попрежнему доступна и обновляется для тех, у кого нет SRM. Хотя можно обойтись несколько устаревшей Red Hat Milo для вашего оборудования в последнем Red Hat Linux, лучше всего перейти на консоль SRM, если она есть и поддерживается вашим оборудованием.

После установки Linux на многих системах работает утилита flash-RAM management utility (FMU), позволяющая «вбить» образ Milo в системную энергонезависимую RAM. Другие подобные утилиты могут поставляться с коммерческой версией ОС или с пакетом разработчика. Некоторые распространяются на чипе EPROM, который надо установить. Поскольку программы ARC и AlphaBIOS создают среду с графическим интерфейсом, они занимают больше места, и в стандартной системе вы встретите, в лучшем случае, только одну из них.

Загрузка Linux может быть сделана автоматической, подобно загрузке MS-DOS или Windows. Мы не рекомендуем использовать FMU, поскольку загрузка с консоли SRM столь же эффективна. В этом приложении мы отдали предпочтение SRM и *aboot*, поскольку они работают на всех важнейших платформах Alpha и дают наиболее последовательные и предсказуемые результаты. О других вариантах установки можно узнать из документации.

Часто не имеет значения, старая или новая версия встроенных программ у вас установлена. В действительности, ряд пользователей рекомендует не обновлять встроенных программ без реальной необходимости в этом. Иными словами, можно обновлять встроенные программы, если Linux не устанавливается нормально на вашу систему. На самом деле, некоторые дистрибутивы Linux требуют для иных систем вернуться к более старой версии встроенных программ.

Но в целом мы советуем при установке Linux использовать самую свежую версию вашей встроенной программы, особенно если это AlphaBIOS. Обновление встроенных программ выполняйте в соответствии с руководством по вашему аппаратному обеспечению. Обновление встроенных программ можно взять с *http://ftp.digital.com/pub/ DEC/Alpha/firmware/*.

Возможности и ограничения консоли SRM

Консоль SRM может загружать данные с дисков IDE/ATAPI, SCSI или с гибких носителей и использовать системный «родной» SCSI-контроллер жесткого диска для доступа к опознанному SCSI-устройству для загрузки. SRM может читать дискеты. SRM может осуществлять доступ к файловым систе-

мам формата MS-DOS (но не к тем, которые созданы при помощи *fdisk* под Linux), BSD-размеченным файловым системам UFS (но не файловым системам BSD-типа, размеченным Linux) и к файловым системам ISO9660.

Консоль SRM позволяет стартовать систему путем передачи контроля вторичному загрузчику, который загружает систему вслепую. SRM мало знает о дисковых разделах и файловых системах и воспринимает дисковые устройства как блочные устройства. Она считывает первый сектор из 512 байт устройства памяти. Этот сектор должен содержать адрес сектора и смещение, с которого SRM может начать чтение размера блока данных. SRM переходит к этому месту и загружает последующие данные в память. Данные должны быть файлом образа вторичного загрузчика, который загружает систему.

В системах Alpha обычно есть два вторичных загрузчика: обычный загрузчик из ядра Linux и отдельная утилита *aboot*. Утилита *aboot* – более гибкая, чем обычный загрузчик, и устанавливается по умолчанию большинством дистрибутивов. Однако можно использовать консоль SRM для того, чтобы загрузить загрузчик Milo. Для некоторых машин в настоящий момент нет MILO (DS20), поэтому в таких случаях использование SRM и *aboot* обязательно.

Если вы используете для загрузки Linux SRM и *aboot*, первый раздел диска должен начинаться с цилиндра с номером 2. В результате в начале диска остается место для установки *aboot*. Дополнительную информацию о SRM и *aboot* можно найти в SRM Howto на *http://www.linuxalpha.org/faq/srm.html*.

Возможности и ограничения встроенной программы ARC

Эта программа весьма ограничена. Она умеет обращаться только к файлам файловых систем MS-DOS, HPFS и ISO9660. Для MS-DOS или ISO9660 она распознает только имена файлов в формате 8.3.

Возможности и ограничения встроенной программы AlphaBIOS

Активной разработки AlphaBIOS больше не ведется (последняя версия 5.70). Если в вашей системе есть AlphaBIOS, перед установкой Linux установите самую последнюю модификацию AlphaBIOS. AlphaBIOS умеет обращаться к файлам только файловых систем MS-DOS и ISO9660. Как и ARC, система распознает только имена файлов в формате 8.3.

Возможности и ограничения мини–загрузчика Milo

Мини-загрузчик Milo не может загрузиться с диска самостоятельно. Чтобы Milo мог загрузить Linux, его самого нужно сначала загрузить из системной флэш-памяти, либо с диска с помощью загрузчика операционной системы (OSloader), разработанного для систем Alpha, такого как загрузочный менеджер Windows NT Alpha. Дистрибутивы Linux предоставляют для этих целей *linload.exe*.

Когда Milo находится на диске, он загружается с помощью ARC, AlphaBIOS или консоли SRM после запуска *linload.exe*. *linload.exe* распознает файловые системы MS-DOS (FAT), но не распознает файловые системы HPFS или VMS. Если Milo был загружен посредством *linload.exe*, он, очевидно, будет ограничен файловыми системами MS-DOS, ISO9660 и *ext2*.

Milo по умолчанию читает файловые системы *ext2* и может загружать образы операционных систем в форматах ISO9660 или MS-DOS (FAT), используя параметры командной строки. Milo загрузит сжатые файлы образа ядра, созданные с помощью *gzip*, если задано полное имя (например, *vmlinux.gz*), но мы обнаружили, что по крайней мере некоторые дистрибутивы Milo не могут обрабатывать такие файлы; во всяком случае загруженные из файловых систем ISO9660 или MS-DOS. Кроме того, MILO доступен не для всех платформ. Список этих систем есть на *http://www.linuxalpha.org/docs/systypes.txt*. За дополнительной информацией обратитесь на *http://www.linuxalpha.org/faq/milo.html*.

Установка Linux

Отлично, вы достали руководства по своему оборудованию и выбрали дистрибутив Linux, который будете устанавливать (тот, который соответствует вашему оборудованию). В этом разделе мы рассмотрим в самом общем виде, что нужно сделать для успешной установки Linux.

Общая последовательность действий

При обычной установке Linux с CD-ROM последовательность действий, как правило, будет следующей:

1. Соберите информацию о системном оборудовании для того, чтобы выбрать правильные установочные файлы и процедуры. Загляните в руководства по вашему оборудованию или руководства по системному администрированию. Получите сообщения об ошибках и просмотрите исправления к тому дистрибутиву, который будет использован для установки вашего пакета. Соберите текущую документацию по установке программ, если вы полагаете, что информация, предоставленная продавцом, устарела или неполна.
2. Оцените размер жестких дисков и решите, как они будут разбиты на разделы для Linux. В главе 3 представлены основные соображения по распределению дискового пространства и созданию дисковых разделов, но для Alpha цифры придется подправить. Средства установки, которые вы выбрали, должны поддерживать тот или иной способ деления диска на разделы, но не всегда могут удовлетворить всем требованиям.
3. Определите способ загрузки Linux после завершения установки. Это может повлиять на выбор способа установки.
4. Выберите способ установки Linux в соответствии с вашими аппаратными средствами и встроенными программами, требованиями к разбивке дис-

ка и загрузке Linux. Мы считаем SRM наилучшей встроенной программой для загрузки Linux практически во всех случаях.

5. Если загрузка осуществляется не с CD-ROM, создайте необходимые дискеты с данными для вашей системы (образ ядра и образ виртуального диска (ramdisk)).
6. Настройте системное оборудование, как это необходимо для установки Linux.
7. При использовании обычной процедуры установки Milo создайте для вашей системы дискету с образом Milo.
8. Включите питание системы, обратитесь к консоли. При использовании Milo нужно загрузить загрузчик Linux (обычно это программа LINLOAD.EXE, которая затем используется для загрузки Milo). Если на вашей системе установлена Windows NT, можно для загрузки Milo воспользоваться NT Osloader. В противном случае вы будете использовать системную встроенную программу для загрузки начального загрузчика, который затем будет применен для подготовки системы к установке Linux.
9. Если используется Milo, загрузите с его помощью ядро и запустите программу установки с CD-ROM.
10. Создайте дополнительные дисковые разделы и отформатируйте их в соответствии с требованиями загрузки вашей системы.
11. Загрузите дополнительные утилиты Linux, приложения, компиляторы и языки программирования или библиотеки. Настоятельно рекомендуется заново скомпилировать текущую стабильную версию ядра Linux, приспособленную к требованиям вашей системы.
12. Если вы удовлетворены установкой Linux, настройте ее для автоматической загрузки или загрузки из меню менеджера загрузки. Для некоторых старых систем также может потребоваться установка перемычек. Большинство систем требует внесения изменений через системную консоль.

Подготовка программного обеспечения к установке

Посмотрите на веб-сайте фирмы, выпустившей компакт-диск с вашим дистрибутивом, есть ли сообщения об ошибках, «заплаты» и более поздние версии программного продукта, который вы устанавливаете. Также поищите на веб-сайте AlphaLinux независимый анализ исправленных ошибок и заплаток.

Если загрузка выполняется не с CD-ROM, создайте загрузочные дискеты, необходимые для установки Linux (2 для загрузки с помощью SRM, 3 для загрузки с помощью AlphaBIOS или ARC через MILO). Чтобы узнать, какие дискеты могут потребоваться для начальной загрузки, посмотрите таблицу на веб-сайте AlphaLinux:

http://www.linuxalpha.org/docs/systypes.txt

Создать эти дискеты можно на машине IBM PC/MS-DOS, воспользовавшись RAWRITE. EXE для создания дискет с файлами образов, или с помощью *dd* под Unix.

Подготовка оборудования к установке

Ключевыми вопросами для оборудования Alpha являются дисковые разделы и файловые системы, а также поддерживаемые видеоадаптеры. При создании конфигурации диска мы рекомендуем для надежности использовать *fdisk*, а не утилиту, основанную на GUI.

Ваше руководство по оборудованию окажет помощь при возникновении проблем, например, даст расшифровку звуковых сигналов. В UDB есть светодиодный индикатор, который высвечивает число или букву, если при загрузке системы обнаружена ошибка в необходимой программе, находящейся в NVRAM или ROM.

Если ваша система находилась некоторое время в нерабочем состоянии, убедитесь в работе охлаждающих вентиляторов. Перегрев оборудования в момент переконфигурирования ПЗУ будет особенно неприятен.

Настройка встроенных программ для начала установки

Поскольку мы не можем рассмотреть в этой книге все конфигурации BIOS, мы дадим ссылки на подробную информацию об установке и конфигурировании каждого типа встроенных программ, используемых совместно с Milo.

Программное обеспечение	Местонахождение информации
Консоль ARC	*http://www.linuxalpha.org/faq/alphabios-howto.html*
AlphaBIOS	*http://www.linuxalpha.org/faq/alphabios-howto.html*
SRM	*http://www.linuxalpha.org/faq/srm.html*

Загрузка образа ядра Linux

Как отмечалось выше, есть два возможных способа загрузки Linux на системах Alpha. Сначала мы расскажем о способе, использующем SRM, затем – MILO. На загрузке MILO из SRM мы останавливаться не будем.

По окончании инициализации SRM выводится следующее приглашение (в более новых системах ему предшествует P00):

```
>>
```

Увидеть список устройств, распознанных SRM, можно с помощью команды *show dev*:

```
>> show dev
```

Необходимо проглядеть этот список, чтобы найти свое устройство начальной загрузки и выяснить, как оно называется в SRM.

Теперь можно загрузить ядро Linux и начать установку. Базовый синтаксис при загрузке из SRM следующий:

```
boot device -file file_name -flags "kernel flags"
```

Следующие указания демонстрируют, как начать установку с дискеты.

1. Введите в ответ на приглашение SRM команду такого типа:

   ```
   >> boot dva0 -file vmlinux.gz -flags "root=/dev/fd0 load_ramdisk=1"
   ```

   В этом примере предполагается, что ядро названо *vmlinux.gz*, но, в зависимости от дистрибутива, имя может быть иным. Необходимо помнить о том, что флаги должны быть заключены в двойные кавычки, иначе ядру будет передан только первый параметр флагов.
2. Получив приглашение, вставьте начальную дискету, которую создали раньше, и нажмите <Enter>, чтобы продолжить.
3. Продолжайте установку в соответствии с указаниями программ дистрибутива.

Следующие указания демонстрируют, как начать установку с CD-ROM.

1. Определите имя устройства CD-ROM с помощью команды *show dev*, показанной выше. Если у вас CD-ROM с интерфейсом SCSI, его имя будет начинаться с *dk*. Если с интерфейсом IDE, то с *dq*. Предположим, что в нашем случае устройство называется *dka400*.
2. Начните установку с CD-ROM с помощью команды такого типа:

   ```
   >> boot dka400 -file vmlinux.gz -fl "root=/dev/sr0"
   ```

   В случае IDE CD-ROM нужно будет изменить строку флагов в соответствии с соглашениями, принятыми в Linux для имен устройств IDE. Кроме того, учтите, что многие новые дистрибутивы заранее сконфигурированы на загрузку с помощью *aboot*. Обратитесь к документации дистрибутива за указаниями по предварительному конфигурированию способов загрузки.
3. Продолжайте установку в соответствии с указаниями программ дистрибутива.

Если вы предпочли запустить установку через Milo и правильно настроили BIOS для загрузки Milo, то должны увидеть следующее приглашение:

```
MILO>
```

Поскольку Milo представляет собой микроядро, у него много параметров, с которыми может потребоваться ознакомиться, прежде чем загружать ядро Linux. Чтобы увидеть параметры Milo, вызовите команду справки Milo:

```
MILO> help
```

Для того чтобы увидеть, как настроен Milo, какие устройства он видит и какие файловые системы распознает, введите команду *show*:

```
MILO> show
```

Если все выглядит нормально, можно продолжить установку и загрузить Linux из подготовленного образа ядра Linux.

Загрузка ядра выполняется очень просто. Это можно сделать одной командой.

1. Чтобы загрузить ядро Linux с первого диска, введите следующее:

   ```
   MILO> boot fd0:vmlinux.gz root=/dev/fd0 load_ramdisk=1
   ```

   Обратите внимание, что это подразумевает использование образа с гибкого диска, который вы создали из образа с жесткого диска. Milo по умолчанию предполагает раздел с файловой системой *ext2* и устройство дискет *fd0*. Если вы хотите выполнить загрузку с дискеты, отформатированной под MS-DOS, например, с ранее подготовленной резервной дискеты, можно ввести следующее:

   ```
   MILO> boot fd0 -t msdos -fi vmlinux.gz load_ramdisk=1
   ```

2. После приглашения вставьте гибкий диск с ramdisk.
3. Запустите программы установки и конфигурации с CD-ROM.
4. По окончании установки установите Milo на небольшой раздел диска на вашей системе для использования при повторном конфигурировании. Если вы хотите загружать Milo при загрузке, этим разделом должен быть первичный раздел MS-DOS. Вы можете создать его при помощи *fdisk* для MS-DOS или Windows NT.

После установки: настройка и замечания

Теперь ваша система Linux должна загружаться и нормально работать. Мы все же рекомендуем внести некоторые усовершенствования.

Настройка ядра

Завершив установку, скомпилируйте свое собственное ядро, поскольку в ядре из вашего дистрибутива, вероятно, больше драйверов устройств, чем вам нужно. Информацию по компиляции ядра можно найти на веб-сайте AlphaLinux (*http://www.linuxalpha.org*), а также в «Kernel Howto», доступном на любом архивном сайте Linux.

Производительность и настройка библиотек

Для увеличения производительности системы AlphaLinux можно заменить стандартные математические библиотеки на переносимую математическую библиотеку Compaq Portable Math Library (CPML). CPML по содержанию идентична Compaq Tru64 Unix *libm* и непосредственно заменяет AlphaLinux *libm*. Для получения более подробных сведений о CPML посетите:

http://www.unix.digital.com/linux/cpml.htm.

Полезные ресурсы для настройки работы AlphaLinux можно найти по адресу:

http://cyclone.ncsa.*uiuc.edu/PCA/PerformanceTuning.html*

Двоичная эмуляция

AlphaLinux в основном совместима с Tru64 Unix (Digital Unix или DU) на уровне исполняемых файлов. Однако не все системные вызовы поддерживаются. Для запуска двоичных файлов Tru64 Unix в AlphaLinux вам необходимы некоторые разделяемые библиотеки из Tru64 Unix. Они сейчас поставляются в нескольких дистрибутивах Linux и могут быть загружены вместе с Netscape Communicator с сайта Compaq. Более подробную информацию по подобной настройке можно найти в AlphaLinux FAQ на *http://www. linuxalpha.org/faq/FAQ.html*.

AlphaLinux может также выполнять программы i386 Linux посредством программы, называемой em86. Информация об этом также доступна в AlphaLinux FAQ. Обратите внимание, что исправление ядра теперь не является необходимым, хотя вам придется включить поддержку программ i386 при компиляции ядра.

Замечания по поводу графического броузера

Netscape не перенесла Netscape Communicator на AlphaLinux. Но это не должно огорчать, поскольку «родные» программы x86 и Tru64 будут работать в AlphaLinux в режиме двоичной эмуляции x86 и Tru64 Unix. Загрузить Netscape RPM с необходимыми библиотеками Tru64 можно с:

http://www.compaq.com/partners/netscape/downloads/register_nav4_Linux.html

Броузер Mozilla также может быть альтернативой. Сведения о Mozilla и других броузерах можно найти на *http://www.linuxalpha.org/software*. Скомпилированные RPM можно также взять на *ftp://ftp.mozilla.org/pub/mozilla/releases/*. Выберите каталог, соответствующий версии Mozilla, затем каталог RedHat 6.x или 7.x и потом каталог alpha.

Также можно воспользоваться файловым менеджером KDE из системы KDE (*http://www.kde.org*).

Библиография

В настоящей библиографии указан ряд книг по Linux и родственным темам. Она ни в коей мере не является исчерпывающей – опубликовано более 500 книг по Linux.

Здесь также приведены некоторые наиболее полезные ссылки из Linux Documentation Project, часть которых доступна в печатном виде, а все они есть на веб-сайте LDP по адресу *http://www.tldp.org/guides.html* и на его зеркальных сайтах.

Руководства Проекта документирования Linux

Cooper, Mendel. *Advanced Bash-Scripting Guide.*

Erlich, Shie. *Custom Linux: A Porting Guide–Porting LinuxPPC to a Custom SBC.*

Jobst, Jennifer. *Emacspeak User's Guide.*

Welsh et al, Matt. *Linux Installation and Getting Started.*

Komarinski, Mark F., Jorge Godoy, and David C. Merrill. *LDP Author Guide.*

Seifried, Kurt. *Linux Administrator's Security Guide.*

Drake, Joshua. *Linux Consultants Guide.*

Beekmans, Gerard. *Linux From Scratch.*

Aivazian, Tigran. *Linux Kernel 2.4 Internals.*

Burkett, B. Scott, Sven Goldt, John D. Harper, Sven van der Meer, and Matt Welsh. *Linux Programmer's Guide.*

Frampton, Steve. *Linux System Administration Made Easy.*

Greenfield, Larry. *Linux User's Guide.*

Mourani, Gerhard. Securing and Optimizing Linux Red Hat Edition—A Hands-on Guide. Barnson, Matthew P. *The Bugzilla Guide.*

Stutz, Michael. *The Linux Cookbook: Tips and Techniques for Everyday Use.*

Rusling, David A. *The Linux Kernel.*

Johnson, Michael K. *The Linux Kernel Hackers' Guide.*

Pomerantz, Ori. *The Linux Kernel Module Programming Guide.*

Kirch, Olaf. *The Linux Network Administrators' Guide.*

Wirzenius, Lars, Joanna Oja, and Stephen Stafford. *The Linux System Administrators' Guide.*

Документы FAQ Проекта документирования Linux

Merrill, David. *Linux Frequently Asked Questions with Answers.*
Zanella, Neil. *Brief Linux FAQ (BLFAQ).*
Sullivan, Andrew. *AfterStep FAQ.*
De Wit, Johan. *FTape FAQ.*
Kirsch, Mathew E. *Linux ATAPI FAQ.*
Walton, Sean. *Linux Threads FAQ.*
Leblanc, Gregory. *Linux-RAID FAQ.*
Wolff, R. E. *The Sig11 FAQ.*
Ridgway, Douglas. *Wine FAQ.*
Moen, Rick. *WordPerfect on Linux FAQ.*

Документы HOWTO Проекта документирования Linux (частичный перечень)

Gonzato, Guido. *From DOS/Windows to Linux HOWTO.*
Tranter, Jeff. *Linux CD-ROM HOWTO.*
Langfeldt, Nicolai, and Jamie Norrish. *Linux DNS HOWTO.*
Grennan, Mark. *Linux Firewall and Proxy Server HOWTO.*
Pritchard, Steven. *Linux Hardware Compatibility HOWTO.*
Ward, Brian. *Linux Kernel HOWTO.*
Heuser, Werner. *Linux Laptop HOWTO.*
Lawyer, David S. *Linux Modem HOWTO.*
Barr, Tavis, Nicolai Langfeldt, Seth Vidal, and Tom McNeal. *Linux NFS HOWTO.*
Drake, Joshua. *Linux Networking HOWTO.*
Hinds, David. *Linux PCMCIA HOWTO.*
Taylor, Grant. *Linux Printing HOWTO.*
Wood, David. *Linux SMB HOWTO.*
Tumenbayar, Enkh. *Linux SMP HOWTO.*
Lawyer, Dave S. *Linux Serial HOWTO.*
Tranter, Jeff. *Linux Sound HOWTO.*
Arcomano, Roberto. *Linux Wireless HOWTO.*
Raymond, Eric S. *Linux XFree86 HOWTO.*
Raymond, Eric S. *Unix Hardware Buyer HOWTO.*

Книги по Linux общего характера

Siever et al, Ellen. *Linux in A Nutshell: A Desktop Quick Reference*. O'Reilly & Associates.[1]

LeBlanc, Dee-Ann, Melanie Hoag, and Evan Blomquist. *Linux for Dummies*. John Wiley & Sons.

Ricart, Manuel Alberto. *The Complete Idiot's Guide to Linux*. Alpha Books.

Ball, Bill, Stephen Smoogen, and Ryan K. Stephens. *Sams' Teach Yourself Linux in 24 Hours*. Sams.

Tranter, Jeff. *Linux Multimedia Guide*. O'Reilly & Associates.

Phillips, Dave. *Linux Music and Sound*. No Starch Press.

Negus, Christopher. *Red Hat Linux 7.3 Bible*. John Wiley & Sons.

Unix и оболочки Unix

Todino-Gonguet, Grace, John Strang, and Jerry Peek. *Learning the Unix Operating System, 5th Edition*. O'Reilly & Associates.

Bach, Maurice. *Design of the Unix Operating System*. Prentice Hall.

Kernighan, Brian W, and Rob Pike. *The UNIX Programming Environment*. Prentice Hall.[2]

Vahalia, Uresh. *UNIX Internals: The New Frontiers*. Prentice Hall.

Kochan, Stephen G., and Patrick H. Wood. *UNIX Shell Programming*. Sams.

Программные приложения

Stallman, Richard M. *GNU Emacs Manual*. GNU Press.

Robbins, Arnold, and Linda Lamb. *Learning the vi Editor*. O'Reilly & Associates.

Rosenblatt, Bill, Eric S. Raymond, and Debra Cameron. *Learning Gnu Emacs*. O'Reilly & Associates.

Lampaport, Leslie. *LaTeX: A Document Preparation System*. Addison Wesley.

Reese, George, Randy Jay Yarger, Tim King, and Hugh E. Williams. *Managing and Using MySQL*. O'Reilly & Associates.

Greenspan, Jay, and Brad Bulger. *MySQL/PHP Database Applications*. John Wiley & Sons.

DuBois, Paul, and Michael Widenius. *MySQL*. New Riders.

Meadhra, Michael. *StarOffice for Linux For Dummies*. Hungry Minds.

Knuth, Donald E. *The TEX book*. AddisonWesley.

---

[1] Э. Сивер, С. Спейнауэр, С. Фиггинс и Д. Хекман «Linux. Справочник», 3-е издание. – Пер. с англ. – СПб: Символ-Плюс, 2001.

[2] Брайан Керниган и Роб Пайк «UNIX. Программное окружение». – Пер. с англ. – СПб: Символ-Плюс, 2003.

Oualline, Steve. *Vi iMproved (VIM)*. New Riders.

Levine Young, Margaret, David C. Kay, David Guertin, and Kathy Warfel. *WordPerfect for Linux for Dummies*. Hungry Minds.

Интернет

Laurie, Peter, and Ben Laurie. *Apache: The Definitive Guide*. O'Reilly & Associates.

Aulds, Charles. *Linux Apache Web Server Administration*. Sybex.

Musciano, Chuck, and Bill Kennedy. *HTML & XHTML: The Definitive Guide*. O'Reilly & Associates.[1]

Niederst, Jennifer. *HTML Pocket Reference*. O'Reilly & Associates.

Williams, Hugh E., and David Lane. *Web Database Applications with PHP & MySQL*. O'Reilly & Associates.

Сети и коммуникации

Albitz, Paul, and Cricket Liu. *DNS and BIND*. O'Reilly & Associates.[2]

Comer, Douglas E, and David L. Stevens. *Internetworking with TCP/IP*. Prentice Hall.

Kirch, Olaf, and Terry Dawson. *Linux Network Administrator's Guide*. O'Reilly & Associates.

Stern, Hal, Mike Eisler, and Ricardo Labiaga. *Managing NFS and NIS*. O'Reilly & Associates.

Costales, Bryan, and Eric Allman. *Sendmail*. O'Reilly & Associates.

Stevens, W. Richard, and Gary R. Wright. *TCP/IP Illustrated*. Addison Wesley.

Hunt, Craig. *TCP/IP Network Administration*. O'Reilly & Associates.[3]

Eckstein, Robert, and David Collier-Brown. *Using Samba*. O'Reilly & Associates.

Программирование и внутреннее устройство Linux

Stevens, W. Richard. *Advanced Programming in the UNIX Environment*. Addison Wesley.

Matthew, Neil, and Richard Stones. *Beginning Linux Programming*. Wrox Press.

---

[1] Чак Муссиано и Билл Кеннеди «HTML и XHTML. Подробное руководство», 4-е издание. – Пер. с англ. – СПб: Символ-Плюс, 2002.

[2] Пол Альбитц и Крикет Ли «DNS и BIND», 4-е издание. – Пер. с англ. – СПб: Символ-Плюс, 2002.

[3] Крэйг Хант «TCP/IP. Сетевое администрирование», 3-е издание. – Пер. с англ. – СПб: Символ-Плюс, 2004.

Stallman, Richard M., Roland Pesch, and Stan Shebs. *Debugging with GDB*. GNU Press.

Keogh, Jim. *Linux Programming for Dummies*. John Wiley & Sons.

Niemeyer, Pat, and Jonathan Knudsen. *Learning Java*. O'Reilly & Associates.

Schwartz, Randal L., and Tom Phoenix. *Learning Perl*. O'Reilly & Associates.

Rubini, Alessandro, and Jonathan Corbet. *Linux Device Drivers*. O'Reilly & Associates.

Wall, Larry, Tom Christiansen, and Jon Orwant. *Programming Perl*. O'Reilly & Associates.[1]

Loukides, Mike, and Andy Oram. *Programming with GNU Software*. O'Reilly & Associates.

Kernighan, Brian W., and Dennis M. Ritchie. *The C Programming Language*. Prentice Hall.

Eckel, Bruce. *Thinking in Java*. Prentice Hall.

Bovet, Daniel P., and Marco Cesati. *Understanding the Linux Kernel*. O'Reilly & Associates.

Системное администрирование

Nemeth, Evi, Garth Snyder, and Trent R. Hein. *Linux Administration Handbook*. Prentice Hall.

Shah, Steve. *Linux Administration: A Beginner's Guide*. Osborne McGraw-Hill.

Collings, Terry, and Kurt Wall. *Red Hat Linux Networking and System Administration*. John Wiley & Sons.

Maginnis, Tobin. *Sair Linux and GNU Certification Level I, Installation and Configuration*. John Wiley & Sons.

Вопросы безопасности

Hatch, Brian, James Lee, and George Kurtz. *Hacking Linux Exposed*. Osborne McGraw-Hill.

Silverman, Richard, and Daniel J. Barrett. *SSH, the Secure Shell: The Definitive Guide*. O'Reilly & Associates.

Bauer, Michael D. *Building Secure Servers with Linux*. O'Reilly & Associates.

Garfinkel, Simson, and Gene Spafford. *Practical UNIX & Internet Security, 2nd Edition*. O'Reilly & Associates.

---

[1] Ларри Уолл, Том Кристиансен и Джон Орвант «Программирование на Perl», 3-е издание. – Пер. с англ. – СПб: Символ-Плюс, 2002.

Алфавитный указатель

Специальные символы

& (амперсанд)
    работа в фоновом режиме, 409
| (вертикальная черта), 120, 281
? (вопросительный знак), 117
- (дефис), 491
^ (диакритический знак), 115
* (звездочка), 118
    в файле passwd, 169
@ (знак at), 491
> (больше), перенаправление вывода, 118
< (меньше), перенаправление ввода, 120
% , для номеров заданий, 123
# (решетка)
    как приглашение root, 146
    символ комментария, 90
/ (слэш), каталоги, 109
~ (тильда)
    в редакторе *vi*, 314
    каталоги, 109
. (точка), текущий каталог, 122
.. (точка-точка), родительский каталог, 110

A

AbiWord, 340
Abiword, текстовый процессор, 439
Acorn Disk Filing System (ADFS), 180
action, поле (inittab, файл), 159
Adaptec, интерфейсы, 73
AddIcon, директива, 613
AddIconByEncoding, директива, 613
AddIconByType, директива, 613
adduser, команда, 174
ADSL (Asynchronous Digital Subscriber Line), 598
Advanced Power Management (APM), 43
afio, утилита, 262
agetty, программа, 162
aha152x=, параметр загрузки, 73
aha1542=, параметр загрузки, 73
aic7xxx=, параметр загрузки, 73
alias, команда, 145
Alpha, системы, 685–701
    идентификация, 690
    история и современное состояние, 686
    требования к памяти, 689
    установка Linux, 696
    установка утилиты Milo, 693, 697
AlphaBIOS, 693–694, 698
AlphaLinux
    источники информации, 688
        сайты, списки рассылки, 689
    совместимость с Tru64 Unix, 701
    эмуляция i386 Linux, 701
Amateur Radio, поддержка, 247
AmigaOS, файловые системы, 180
ANSI C, стандарт, 475
Anyware Office, офисный пакет, 339
a.out, файл, 477
Apache, веб-сервер, 42
    установка, 611
apachectl, утилита, 616
APM (Advanced Power Management), поддержка, 43
append, строки, 154
Apple Mac HFS, 180
apropos, команда, 124
APSfilter, пакет, 292
ARPAnet и TCP/IP, 562
ASCII и утилита lptest, 275
ask, режим (vga), 151
Asynchronous Digital Subscriber Line (ADSL), 598

Athena, набор графических примитивов, 544
attach, команда (gdb), 526
Autofill, режим (Emacs), 336
autofs, пакет, 188
aztcd=, параметр загрузки, 74

B

Back, кнопка (Konqueror), 609
bash, оболочка, 34, 113
.bash_profile, стартовый файл, 132
.bashrc, стартовый файл, 132
Berkeley Software Distribution (*см.* BSD), 25
BFS, файловая система, 180
-bg, параметр (xterm), 407
/bin, каталог, 134
binary, режим, 100
BIOS, 72
  и ограничения начальной загрузки, 66
bmouse=, параметр загрузки, 75
Bochs (программы эмуляции Windows), 470
BogoMips, показатель, 157
/boot, каталог, 134
  /boot/lilo.conf, файл, 150
  /boot/map, файл, 151
Bourne Again Shell (*см.* bash), 34
br, переменная (printcap), 284
break, команда (gdb), 531
BSD (Berkeley Unix)
  роль в разработке Linux, 25
  утилиты печати, 294, 297
    устранение неполадок, 305
Buslogic SCSI-интерфейсы, 73
buslogic=, параметр загрузки, 73
bzip2, утилита, 207

C

C, C++, языки, 472, 483
  компилятор, обновление, 222
  компиляция совместно используемых библиотек, 481
calls, программа, 538
cancel, команда, 381
'cannot access memory at address...', ошибка, 525
cat, команда, 111, 275
cd, команда, 109, 110
CD-проигрыватель, 45
CD-ROM приводы
  Alpha-системы с, 692
  драйверы, 248
  параметры загрузки, 74
/cdrom, каталог, 186
cdu31a=, параметр загрузки, 74
chat, утилита, 582, 584
chgrp, команда, 129
chmod, команда, 104, 131
chown, команда, 129, 177
chsh, команда, 114
ci, команда, 547
CLASSPATH, переменная окружения, 511
cm206=, параметр загрузки, 74
Coherent, файловая система, 180
combine, утилита (ImageMagick), 362
Common Unix Printer System (CUPS), 306
CONNECT, сигнал, 585
continue, команда (gdb), 531
convert, утилита (ImageMagick), 361
copyright для Linux, 46
core, файлы
  отладка и, 32
CorelDRAW, 38
cp, команда, 149
CPML (Compaq Portable Math Library), 700
cron, утилита, 264
csh (C оболочка), 113
.cshrc, стартовый файл, 133
Ctrl-Alt-Del, комбинация клавиш, 91, 162
CUPS (Common Unix Printer System), 306
CVF-FAT, расширение DOS-FAT, 181
CVS (Concurrent Version System)
  через Интернет, 554
CVSROOT, переменная окружения, 551

D

[data], раздел (smb.conf, файл), 464
Data Display Debugger (DDD), 535
dd, команда, 145, 198
  копирование файлов образа на дискеты, 61
  создание загрузочной дискеты, 149

DDD (Data Display Debugger), 535
DDoS, сетевая атака, 638
debug, параметр загрузки, 72
DefaultIcon, директива, 613
define-key, функция, 335, 336
/dev, каталог, 76, 135, 143, 200, 583
  /dev/hda#, файлы, 200
  /dev/MAKEDEV, сценарий, 202
  /dev/lp#, файлы, 275
  /dev/mouse, файл, 203
  /dev/null, файл, 120, 200
  /dev/ttys# и ttyS#, файлы, 275
  /dev/zero, файл, 200
devfs, файловая система, 182
'device full', ошибка, 100
diff, программа, 555
disass, команда (gdb), 532
display, команда (gdb), 533
display, утилита (ImageMagick), 360
Distribution HOWTO, 59
DMA (прямой доступ к памяти), 95
  список используемых каналов, 167
dma, файл (/proc), 167
DNS (Domain Name Service), 630
  настройка, 587
doc, формат файла, 469
Docbook, 36, 346
domainname, команда, 603
DOS-совместимость, утилиты для, 43
DOSEMU, 44
DOS-FAT, файловая система, 180
DoubleSpace, 181
down, команда (lpc), 299
drag-and-drop (KDE), 411
'drive not bootable', ошибка, 102
DriveSpace, 181
du, команда, 120
.dvi, файлы, 344

E

EFS, файловая система, 180
Emacs
  запуск, 324
  макросы, 331
  программирование, 331
Emacs, редактор, 33, 323, 443
  как программный интерфейс, 546
  настройка, 325
.emacs, файл, 334
.emacs, стартовый файл, 133
Enhanced IDE/MFM/RLL, поддержка жестких дисков, 246
enscript, утилита, 384
/etc, каталог, 134
  /etc/fstab, файл, 186
    редактирование, 89
  /etc/group, файл, 169
  /etc/hosts, файл, 577, 578
  /etc/hosts.deny, файл, 644
  /etc/init.d, каталог, 161
  /etc/ld.so.cache, 219
  /etc/ld.so.conf, 219
  /etc/lilo.conf, файл, 153
  /etc/lptest, файл, 275
  /etc/networks, файл, 578
  /etc/passwd, файл, 87, 168, 170, 311
  /etc/printcap, файл, 273, 280, 287, 380
    устранение неполадок в системе печати, 303
  /etc/rc.d/boot, каталог, 164
  /etc/rc.d/init.d, каталог, 162
  /etc/rc.d/rc.local, файл, 162
  /etc/rc.d/rc.sysinit, сценарий, 160
  /etc/shadow, файл, 87, 171
  /etc/skel, каталог, 88, 135, 176
  /etc/syslog.conf, файл, 269
ether=, параметр загрузки, 75
Ethernet, карты, 569
  устранение неполадок, 580, 581
Ethernet, настройка, 570
Evolution
  адресная книга, 437
  календарь, 436
  электронная почта, 435
.exrc, стартовый файл, 133
ext2fs, в сравнении с reiserfs, 181
ext2fs, файловая система, 83
Eyes (GNOME), 431

F

FAQ, Linux, 54
FAT, файловая система, 453
fbdev, сервер, 392
fd#, файлы, 77
fdisk
  параметры, 78
  создание разделов в Linux, 78
fdisk, утилита, 67, 102, 153, 692
fetchmail, пакет, 628

ff, переменная (printcap), 286
fg, команда, 123
-fg, параметр (xterm), 407
FHS (Filesystem Hierarchy Standard), стандарт иерархии файловой системы, 69
'file not found', ошибка, 93, 100
find, команда, 263
finish, команда (gdb), 520
floppy=, параметр загрузки, 75
FM-синтез (звук), 366
fo, переменная (printcap), 286
Forward, кнопка (Konqueror), 609
-fpic, ключ (gcc), 481
Free Software Foundation (Фонд свободного программного обеспечения), 24, 46
Free Software, проекты (веб-сайты), 679
free, команда, 166
пространство для свопинга, 197
fsck, программа, 309
проверка файловых систем, 193
fstab, файл, 89, 186
настройка NFS, 600
ftape, драйвер, 251
FTP (File Transfer Protocol), 101
  XFree86 через, 393
  получение Linux с анонимного FTP-сервера, 60
  сайты с программным обеспечением, 678
  через World Wide Web, 607
Fundamental, режим (Emacs), 324
Future Domain SCSI-интерфейсы, 74
fvwm2, 41
  менеджер окон, 409

G

g++, сценарий оболочки, 483
gcc, компилятор GNU C, 472
  обновление, 222
  совместно используемые библиотеки, 481
gdb, отладчик GNU, 39
gecos, поле (passwd файл), 169
General Public License (*см.* GPL), 24
-geometry, параметр (xterm), 407
getty, программа, 137
Ghostscript, 287
gid, поле (passwd файл), 169
GID, UID и NFS, 601
GIMP, приложение, 362
GIMP Toolkit (GTK), 543, 544
globbing, функция оболочки, 117
GNOME, настольная среда, 41, 406, 427
  взаимодействие с, 428
    панель, 430
  установка, 427
.gnome, стартовый каталог, 134
GNU Emacs, текстовый редактор, 323
GNU gcc, компилятор C, 39
GNU Project, GPL (General Public License), 24
Gnumeric, электронная таблица, 437
GNUS, программа чтения, 332
GoldStar CD-ROM-интерфейсы, 74
gPhoto, 438
GPL (General Public License), 24
  обзор, 46
gprof, утилита, 39, 538
groff, процессор текста, 35
group, файл, 169, 173
groupmod, команда, 177
groups, команда, 173
gscd=, параметр загрузки, 74
GTK (GIMP Toolkit), 543, 544
GUI (Graphical User Interface)
  X Window System, 40
  графические среды, 41
gunzip, утилита, 206
gzip, утилита, 205, 208, 262, 312
  tar утилита и, 214

H

hd#, файлы, 77
hd=, параметр загрузки, 73, 100
High Sierra, файловая система, 182
HiSax, драйвер (ISDN), 591
/home, каталог и резервные копии, 259
homedir, поле (passwd файл), 169
hostname, команда, 579
HOWTO, документы Linux, 54, 59
HPFS, файловая система, 180
hsfs, файловая система, 182
HTML, 36
htools, пакет, 44
httpd, демон, 610
запуск, 616

I

ID группы, 169, 172
ID пользователя, 177
IDE (интегрированные среды разработки), 558
IDE, диски, 73
  поддержка системами Alpha, 690
identify, утилита (ImageMagick), 362
if, переменная (printcap), 284
ifconfig, команда, 570, 574
  устранение неполадок, 581
ImageMagick, пакет, 279, 360
import, утилита (ImageMagick), 360
include, каталог, 135
indent, программа, 557
info, программа, 125
Info, система (Emacs), 327
init, программа, 158
inittab, файл, 159, 579
insmod, команда, 252, 312
Intel, платформы, совместимость с, 31
Internet Relay Chat (IRC), 677
interrupts, файл (/proc), 167
ioports, файл, 166
IP (Internet Protocol), 564, 571
IP-адрес, 563
  hosts, файл, 577
IP-пакеты, фильтрация, 646
  примеры, 653
iptables, команда, 648, 649
IRC (Internet Relay Chat), 677
IRQ (линия запроса прерывания), 95
ISDN, подсистема, 247
ISDN, PPP поверх, 589
  настройка оборудования ISDN, 590
  синхронный PPP, 593
  устранение неполадок, 596
ISO 9660, файловая система, 180, 182

J

Java, интерпретатор, 506
Java, язык
  JVM (Java Virtual Machine), 506
Java, поддержка для, 39
javac (Java-компилятор), 510
JDK (Java Developers Kit), 508
JIT (Just-in-Time), компиляторы, 507
Joliet, файловая система, 180
JVM (Java Virtual Machine), 506

K

kdbg, отладчик, 535
KDE (K Desktop Environment), 41, 406, 410
  KWord, процессор слов, 340
  системы Alpha, 701
  установка, 412
kdebase, пакет, 412
kdeedu, пакет, 413
kdegames, пакет, 412
kdegraphics, пакет, 413
kdelibs, пакет, 412
kdemultimedia, пакет, 413
.kde/share/config, стартовый каталог, 133
kdeutils пакет, 413
KDevelop IDE, 558
'kernel panic', ошибка, 147
'kernel too big', ошибка, 250
kerneld, демон, 254
KGhostview, утилита, 424
kill, команда, 137
Konqueror, 426, 608
konsole, 420
ksh (Korn-оболочка), 113
KWord, 340

L

LAMP, набор программных пакетов, 659
LAN (локальная сеть), NFS и NIS, 600
lastlog, файл, 271
LaTeX, текстовый процессор, 341
  форматирование и печать, 344
ldconfig, команда, 219, 221
ldd, утилита, 218, 482
LDP (Linux Documentation Project), 13, 605, 678
less, команда, 111
lf, переменная (printcap), 283
libc, библиотека, обновление, 219
libm библиотека, обновление, 219
libncurses, библиотека, 220
LILO (загрузчик Linux), 70, 85, 102, 156
  в качестве вторичного начального загрузчика, 153
  задание геометрии диска, 97
  настройка, 150
  параметры начальной загрузки, 153

удаление, 155
установка, 85
lilo.conf, файл, 150
Linux
группы пользователей и пакеты дистрибутивов, 60
достоинства, 11
знание Unix, необходимый уровень, 52
история, 22
копирование дистрибутивов, ограничения, 60
кто использует, 25
начальная загрузка, 75, 102, 156
номера версий, 27
обзор характеристик, 29
полезность, 19
получение
заказ по почте, 60
из Интернета, 60
разработка приложений, пригодность для, 40
установка на системах Alpha, 693
Linux Distribution HOWTO, 59
Linux Documentation Project (LDP), 678
Linux Journal, 55
Linux Magazine, 55
linux, команда, 75
LISP, язык, 514
list, команда (gdb), 518
ln, команда, 112, 203, 221
local, каталог, 135
lock, параметр загрузки, 72
.login, стартовый файл, 133
Logitech, мыши, 398
Lout, пакет (печати), 279
lp, команда, 381
lp, переменная (printcap), 283
lp#, файлы, 275
lpadmin, команда, 381
lpc, утилита, 290, 380
реализация, 299
lpd, демон, 273, 299
устранение неполадок, 305
lpq, команда, 379
lprm, команда, 380
lpstat, команда, 381
lptest, команда, 275, 298
ls, команда, 110
lsmod, команда, 251
LUN, логический номер устройства, 73
Lynx, броузер, 610

M

Magic-Filter, 279
make bzImage, команда, 249
make menuconfig, программа, 243
make xconfig, программа, 243
MAKEDEV, сценарий, 202
Makefile, файл, 236
makeinfo, команда (Texinfo), 357
man, каталоги, 352
man, команда, 88, 124
MANPATH, переменная, 125
map, файл, 151
max_scsi_luns, параметр загрузки, 73
mbadblocks, утилита, 451
MBR (master boot record), главная загрузочная запись, 64, 155
mc, переменная (printcap), 286
mcd, утилита, 451
mcd=, параметр загрузки, 74
mcopy, команда, 450
mcopy, утилита, 451
mdel, утилита, 451
mdeltree, утилита, 451
mdir, команда, 450
mdir, утилита, 451
mdu, утилита, 452
mem=, параметр загрузки, 72, 155
meminfo, файл, 166
METAFONT, система, 37
mformat, команда, 450
mformat, утилита, 452
MIDI (Musical Instrument Digital Interface), 367
Milo, утилита, 693
загрузка, 698
ограничения мини-загрузчика, 695
minfo, утилита, 452
minicom, эмулятор терминала, 585
Minix, операционная система, 23
Minix, файловая система, 179
Mitsumi CD-ROM-интерфейсы, 74
mke2fs, команда, 83
mkfs, команда, 191, 193
mkmanifest, утилита, 452
mknod, команда, 201
mkswap, команда, 82, 198
mlabel, утилита, 452

mmd, утилита, 452
mmount, утилита, 452
mogrify, утилита (ImageMagick), 362
montage, утилита (ImageMagick), 361
more, команда, 111
  каналы и, 121
Motif, библиотека, 543
mount, команда, 183, 188
'mount-point busy', ошибка, 186
Mozilla, броузер, 610, 701
MP3-проигрыватели, 376
mrd, утилита, 452
mren, утилита, 452
MS-DOS, 44
  начальная загрузка, 152
  разделы/файловые системы, 180
  монтирование, 453
  эмуляция, 44
msmouse=, параметр загрузки, 75
mt, команда, 260
mtools, 44
MTools, пакет, совместимость с Windows, 449
mtools.conf, файл, 449
mtype, утилита, 452
MUD, игры, 45
Musical Instrument Digital Interface (MIDI), 367
mx, переменная (printcap), 284
MySQL, 38

N

n, команда, fdisk, 79
named, демон, 573, 577
Nautilus (менеджер рабочего стола и файлов GNOME), 431
nenscript, утилита, 291, 386
NET-4, 561
  файлы конфигурации, 570
NETBEUI, протокол, 455
netfilter, брандмауэр, 648
netmask, параметр (ifconfig, команда), 575
netstat, команда,
  проверка настройки TCP/IP, 580
newgrp, команда, 172
.newsrc, стартовый файл, 133
next, команда (gdb), 519
nexti, команда (gdb), 532
NFS (Network File System), сетевая файловая система, 26, 43
NIS (Network Information Service), 599, 602
nmbd, демон (Samba), 465
'no such file or catalog', ошибка, 527
nosmp, параметр загрузки, 72
NTFS, файловая система/разделы, 180
  монтирование, 453
null, файл, 120
  перенаправление стандартного вывода в, 268

O

of, переменная (printcap), 284
Open Source, веб-сайты, 679
optcd=, параметр загрузки, 74
Optics Storage Interface CD-ROM-интерфейсы, 74
OSS/Free, 368
'out-of-memory', ошибка, 93
Overwrite, режим (Emacs), 336

P

p, команда, fdisk, 79
PAM (Pluggable Authentication Methods), подключаемые методы аутентификации, 171
pas16=, параметр загрузки, 74
passwd, команда, 88, 107, 177
passwd файл, 87, 168, 311
password, поле (passwd, файл), 169
patch, программа, 241, 555
PC card, *см.* PCMCIA, 43
PCI-шина
  сообщения ядра при начальной загрузке, 157
  устройства, поддержка, 246
PCMCIA, поддержка, 43
PDL (язык описания страниц), 278
Perl, язык программирования, 496
'permission denied', ошибка, 93, 100, 104, 184
Philips CD-ROM-интерфейсы, 74
PHP, язык программирования, 669
PHP4 как модуль Apache, 672
pl, переменная (printcap), 285
Plex86, проект (программы эмуляции Windows), 44, 470
Pluggable Authentication Methods (PAM), 171
POP3, протокол, 629

Postfix, программа, 618
PostScript, 278
KGhostview и, 424
nenscript, фильтр, 291
POVRAY, программа, 363
poweroff, команда, 91
PPP (Point-to-Point Protocol), 42, 569
модемы и, 581
настройка DNS, 587
поверх ISDN
настройка оборудования ISDN, 590
синхронный PPP, 593
устранение неполадок, 596
устранение неполадок в настройке, 588
pppd, демон, 582
print, команда (gdb), 520, 527
printcap, файл, 273, 280, 378, 380
устранение неполадок в системе печати, 303
PRINTER, переменная окружения, 382
Pro Audio Spectrum SCSI, интерфейс, 74
/proc, каталог, 135
/proc, файловая система, 90, 165, 180
/proc/dma, файл, 167
/proc/filesystems, файл, 179
/proc/interrupts, файл, 167
/proc/ioports, файл, 166
/proc/meminfo, файл, 166
ps, команда, 138, 298
ptype, команда (gdb), 527
pw, переменная (printcap), 285
pwconv, команда, 171
pwd, команда, 109
pwuconv, команда, 171
px, переменная (printcap), 285
py, переменная (printcap), 285

Q

q, команда, fdisk, 79
QNX4, файловая система, 180
Qt, модуль (Perl), 504
Qt, пакет, 543
quit, команда (gdb), 521

R

ramdisk=, параметр загрузки, 72, 154
ranlib, команда, 480
RAWRITE.EXE, программа, копирование файлов образа на дискеты, 61
.rc, файлы, 134, 160
сетевые, 573
rc.d/init.d, каталог, 162
rc.d/rc, сценарий, 161
RCS (Revision Control System), 546
rdev, команда, 151
rdev, утилита, создание загрузочной дискеты, 148
'read error, file not found', ошибка, 99
'read_intr: 0x10', ошибка, 100
README, файлы и установка Linux, 61
recode, утилита, 468
Red Hat Package Manager (RPM), менеджер пакетов Red Hat, 224
reiserfs, в сравнении с ext2fs, 181
resolv.conf, файл, 579
rg, переменная (printcap), 286
rm, команда, 110, 145, 203
rm, переменная (printcap), 285
rmmod, команда, 251
ro, параметр загрузки, 72, 154
Rock RIdge Extensions (для ISO 9660), 182
RockLinux, Polish Linux Distribution и Kondara, 687
ROM, файловая система, 179, 181
root (/), каталог, 64, 147, 187
только для чтения, 154
root, корневая файловая система, 64, 187
root, пользователь
опасность, 145
пароль, 103
учетная запись, 87
root, учетная запись, 143, 146, 168
каталог для, 170
root=, параметр загрузки, 154
route, команда, 575
routed, демон, 576, 577
rp, переменная (printcap), 285
RPM (Red Hat Package Manager), 224
rs, переменная (printcap), 285
rsh, команда, 554
run, команда (gdb), 518
rw, параметр загрузки, 72
rw, переменная (printcap), 283

S

Samba, 42
разделяемые ресурсы SMB, 458
установка, 459
mount/submount, утилиты, 456
Sanyo CD-ROM-интерфейсы, 75
/sbin, каталог, 134
/sbin/lilo, команда, 152
sbpcd, параметр загрузки, 75
sc, переменная (printcap), 286
Scheme, интерпретаторы, 515
ScriptAlias, директива, 614
SCSI-контроллеры, 71, 158
компиляция ядра, 247
устранение неполадок, 98
SCSI-устройства, 76, 91, 247
именование, соглашения, 77
устранение неполадок, 98
SCSI HOWTO, документ, 98
sd, переменная (printcap), 283
Seagate ST-Ox SCSI-интерфейсы, 74
Second Extended Filesystem (*см.* ext2fs), 83
Server Resource Map, файл, 616
setterm, программа, 306
seyon, эмулятор терминала, 585
sf, переменная (printcap), 286
SGML, 346
sh, (оболочка Борна), 113
sh, переменная (printcap), 285
shadow, файл, 87, 171
shareware, 46
shell, поле (passwd, файл), 169
single, параметр времени загрузки, 154
sjcd=, параметр загрузки, 75
skel, каталог, 88, 135, 176
SlashApp (GNOME), 431
SLIP (Serial Line Internet Protocol), 42, 561, 569, 581
SMB (Server Message Block), протокол, 455
SMB, разделяемые ресурсы, 458
smb.conf, файл, 464
smbclient, утилита, устранение неисправностей с разделяемыми ресурсами Windows, 458
smbd, демон (Samba), 465
soncd535=, параметр загрузки, 74
Sony CD-ROM-интерфейсы, 74
Sony CDU-535, интерфейсы, 74
sort, команда, 120
SoundBlaster Pro CD-ROM-интерфейсы, 75
src, каталог, 135, 236
SRM, консоль, 694
srm.conf, файл, 616
st0x=, параметр загрузки, 74
ST-506, приводы жестких дисков, 73
Stacker, 181
StarOffice, 37
startkde, сценарий оболочки, 414
step, команда (gdb), 519, 531
stop, команда (lpc), 299
'stopped (tty input)', ошибка, 123
strace, программа, 538
su, команда, 144
SuperProbe, программа, 390
SuSE Linux
системы Alpha, 687
SVGA, чипсеты, 390
swapoff, команда, 199
swapon, команда, 84, 199
Swing, пакет (Java), 509
sync, команда, 198, 199
syslog.conf, файл, 269
syslogd, демон, 269, 577
System V IPC, включение поддержки, 246
System V, файловая система, 180

T

t, команда, fdisk, 81
t128=, параметр загрузки, 74
Tab, символ, 487
tar, утилита, 208, 213
gzip утилита и, 213
ошибки, 100
приемы работы, 215
сжатие, 262
создание резервных копий, 258
Tcl, язык, 545
TCP (Transmission Control Protocol), 564
TCP/IP, 43, 560–581
настройка Ethernet, 570
обзор, 560
поддержка, 43
сетевое взаимодействие, 30
TCP wrappers, 643
tcsh, оболочка, 113

testparm, утилита, 464
TeX, текстовый процессор, 35, 341, 346
Texinfo, текстовый процессор, 35, 353, 359
Thinkpad, системы, 75
Tk, модуль (Perl), 504
Tk, программное расширение, 545
tmc8xx=, параметр загрузки, 74
tmp, каталог и cron, 265
Transmission Control Protocol (TCP), 564
Trantor T128 SCSI-интерфейсы, 74
troff, программа, 558
ttys# и ttyS#, файлы, 275
tunelp, команда, устранение неполадок в системе печати, 304
tunelp, утилита, 302

U

UDP (User Datagram Protocol), 564
UFS, файловая система, 180
uid, поле (passwd, файл), 169
umask, команда, 131
umount, команда, 184
UMSDOS, файловая система, 180, 181
Uniform Resource Locator (URL), 606
UNIX
  Perl, язык, 496
  история, 22
  команды, 33, 105
update, программа, 164
URL (Uniform Resource Locator), 606
USB, поддержка устройств, 448
User Datagram Protocol (UDP), 564
useradd, программа, 87, 176
userdel, команда, 176
usermod, команда, 177
username, поле (passwd, файл), 168
/usr, каталог
  /usr/bin, каталог, 134
  /usr/include, каталог, 135
  /usr/lib, каталог, 134, 481
  /usr/lib/autofs, каталог, 188
  /usr/local, каталог, 135
  /usr/sbin, каталог, 134
  /usr/src, каталог, 135
  /usr/src/linux, каталог, 240, 258
  /usr/X11R6/lib/X11, файл, 135
utmp, файл, 271

V

/var, каталог, 134
  /var/log/lastlog, файл, 271
  /var/log/utmp, файл, 271
  /var/log/wtmp, файл, 271
  /var/spool, каталог, 134, 283, 294
  /var/spool/lpd, каталог, 378
VFAT, файловая система/разделы, монтирование, 453
'VFS: Unable to mount root', ошибка, 93
vga, режимы, 151, 154
vga=, параметр времени загрузки, 154
vi, редактор, 33
  запуск, 314
  клоны, 314
  обзор, 313
  расширение, 322
VMWare, 44, 470

W

w, команда, fdisk, 79, 81
watch, команда (gdb), 519, 531
what, команда, 549
Win4Lin, 470
Windows
  загрузка вместо Linux, 103
  поддержка драйверов устройств, 447
  совместимость
    MTools и, 449
    дисковые разделы, монтирование, 453
    поддерживаемые форматы документов, 468
    утилиты для, 43
  совместное использование файлов, 455
  совместный доступ к файлам и принтерам, 42
  стиль для KDE, 418
  установка вместе с Linux, 62
Wine, проект, эмулятор Windows, 43
  программы, 469
WordPerfect Office 2000, 38
World Wide Web, 605
  документация по Linux, 54
  настройка сервера, 610
'wrong fs type', ошибка, 185
wtmp, файл, 271

WYSIWYG, обработка слов, 34, 36
WYSIWYG, процессоры слов, 37, 339

X

x, команда (gdb), 529
X Window System, 41, 387
  Emacs и, 443
  make xconfig, программа, 243, 252
  Wine, эмулятор, 43
  настройка среды, 405
  обзор, 40
  приложения для KDE, 419
    совместимость с KDE, 415
  просмотр документов TeX, 344
  процессы, 137
  ресурсы (X-приложения), 440
  серверы, 388, 394
  средства создания интерфейса, 542
  устранение неисправностей, 404
  часы, 423
X11R6, 387
Xaw3D, инструментарий, 543
.Xdefaults, стартовый файл, 133
xdm, 40
xdvi, утилита, 37, 279
XEmacs, текстовый редактор, 323
xf86config, программа, 396
XFree86, 40, 387, 403
  запуск, 403
  настройка, 395
  установка, 393
Ximian Evolution (пакет для GNOME), 433
xinit, программа, 403
.xinitrc, стартовый файл, 133
.xinitrc, файл, 403
  X-ресурсы, 440
xman, 40
XML, обработка текста, 346
Xmms (MP3-проигрыватели), 377
xmodmap, команда, 335
XT, дисковые контроллеры, 246
xterm, 40, 420
xtp, утилита (ImageMagick), 362
xvidtune, программа, 402
XView, интерфейс, 543
xxgdb, отладчик, 534

Y

yast2, программа, 573
ypwhich, команда, 604

Z

zcat, команда, 207
zsh (Z оболочка), 113

А

автоматическое обнаружение аппаратных устройств, 70
автомонтирование устройств, 188
автомонтировщик, 191
администрирование, 141
  останов системы, установка прав для, 91
  сопровождение служб печати, 301
  электронная почта, 630
адрес
  принтера параллельного порта, устранение неполадок, 306
  хранения переменной, 530
адреса
  IP, 562–563, 571
    динамические, 562
  networks, файл, 578
  кольцевой интерфейс, 568
  портов, 563
  регистрация, 630
  сетей, 563
  совместно доступной памяти, 95
  хостов, 563
  широковещательные, 572, 575
адресная книга (Evolution), 437
альфа-версии, 50
аппаратное обеспечение
  группы пользователей, 174
  жесткие диски, подготовка к установке Linux, 62
  именование, 76
  необнаружение при загрузке, 153
  несовместимость, 93
  обработчики прерываний и ядро, 30
  параметры для обнаружения во время установки Linux, 70
  принтеры, оптимизация, 302
  устранение неполадок, 94, 99
  факсы, 279
апплеты
  Java, 507
  панель GNOME, 430
арифметический сопроцессор, эмуляция, сборка ядра и, 245

архивирование файлов, 205, 215
tar, утилита, 208, 213
gzip утилита и, 213
особые приемы tar, 215
утилиты архивирования, 205
ассоциативные массивы, 497
атаки из Интернета, виды, 637
аудио, поддержка, 45
аутентификация, 171
PAM (Pluggable Authentication Methods), подключаемые методы аутентификации, 171
изменение метода, 171
протоколы для, 588

Б

база данных X-ресурсов, 440
базы данных (коммерческие), 38
байт-коды, 506
безопасность
Java и, 507
аутентификация, изменение метода, 171
компьютерных систем, 636
контроль ревизий
RCS, 516
RCS, утилита, 546–550
обновление программного обеспечения и, 235
пароль, установка, 107
права доступа, 126
системное администрирование, 143
библиотеки
обновление, 217
поддержка, 32
программирование, 473, 479
распространение программного обеспечения, 235
совместно используемые, 481
нарушение ссылок, 311
блоки, 82
проверка и исправление, 196
блокировка учетных записей пользователей, 177
брандмауэры, 646
броузеры в AlphaLinux, 701
буферы, Emacs, 324

В

ввод, перенаправление, 120
ввод/вывод
отображаемый в память, 98
проверка адресов в /proc/ioports, 166
веб-сайты
Abiword, текстовый процессор, 439
ftformat, утилита, 260
GNOME-приложения, 439
GPL, 46
KDE, документация, 416
Linux Documentation Project, 348
Linux, получение, 60
PAM, информация о, 171
Plex86, 44
RAWRITE.EXE, программа, 61
USB-устройства, 448
VMWare, 44
Wine, проект, 44
каталоги приложений, 45
службы печати, информация о, 273
стандарты, 683
ядро, история версий, 240
версии Linux, 216
вывод данных о, 245
описание ранних версий, 23
разрабатываемые, 28
система нумерации, 27
стабильные, 28
видеокарты, 390
виртуальная память, поддержка, 32
виртуальные
консоли, поддержка, 29, 162
машины и совместимость с Windows, 44
устройства, 297
файловые системы, 182
вирусы, сетевая атака, 637
владельцы (права доступа), 127
изменение, 129
возврата каретки символы, вставка, 290
волшебные фильтры (печати), 292
восстановление файлов из резервных копий, 312
восстановление файловых систем, 309
вставка текста, Emacs, 445
вставка файлов в *vi*, 319
встроенные процессоры, поддержка, 29
Вторая расширенная файловая система, 179

вход в систему, устранение неполадок, 103
вывод данных, перенаправление, 118
выполнение, право на, 126

Г

генерация отчетов, механизм Perl, 501
геометрия диска, 73, 97, 100, 153
гибкие диски
  файловые системы, создание на, 191
главная загрузочная запись (MBR), 64, 155
графика
  GIMP, приложение, 362
  ImageMagick, пакет, 279, 360
  POVRAY, программа, 363
  обзор, 359
графические среды (GUI), 41
групповые учетные записи, 172
группы пользователей
  и пакеты дистрибутивов, 60
  права доступа, 127
    изменение, 129

Д

дамп памяти, 517, 526
данные, скорость обмена, 284
двоичные эмуляторы AlphaLinux, 701
декомпрессия файлов, 206
демоны
  automount, 188
  служб печати, 273
  учетные записи, 168
деревья проектов, CVS, 550
динамические IP-адреса, 562
дискеты, 183
  создание резервных копий на, 261
дисковые разделы
  log-файлы, 271
  обзор, 63
  размер создаваемого раздела, 80
  требования, 64
дистрибутивы Linux, 27, 51, 59
  различия инсталляции, 83
  rc, файлы, 573
  выбор пакетов, 84
  история, 10
  копирование, ограничения, 60
  нумерация версий, 28
  распространение, 60
  типы, 59
  требования к дисковым разделам, 64
  установка, 69
  устранение неполадок, 91–104
документация
  Emacs, 327
  FHS (Filesystem Hierarchy Standard), 69
  LILO, конфигурирование, 156
  Linux Documentation Project (LDP), 13, 54, 678
  Perl, 497
  TeX, 346
  Texinfo, 353, 359
  vi, редактор, 323
  страницы руководства, 123
документы, совместимость с Windows, поддерживаемые форматы, 468
домашний каталог, 169
доменное имя, регистрация, 630
дописывание к файлам, 119
дополнительный номер устройства, 201
доступ только для чтения, монтирование файловой системы, 195
драйверы устройств, поддержка в Windows, 447

Ж

жесткие диски
  загрузка Linux с, 102, 156
  геометрия диска, 97, 100, 153
  дисковые разделы
    обзор, 63
    переразбиение (изменение размеров), 67
    требования, 64
  обнаружение, устранение неполадок, 96
  подготовка к установке Linux, 62
журналирование (файловые системы), 181
журналы
  ошибок принтера, 283
  ротация, 272
  системное администрирование, 269

З

завершение слов, 116

зависание системы, 92
зависимости, отслеживание, 84
заглушки-функции, 218
заголовок, Texinfo, 353
загрузка
альтернативных ОС
LILO, 44
установка, 85
поддержка, 29
модулей
автоматическая, 254
поддержка, 245
ядра, 246
системы Alpha, 698
страниц исполняемых модулей
по запросу, 31
с установочного носителя, 70
загрузочная дискета, 70, 85, 92, 101, 149
замеры (звук), 365
запись, право на, 126
заплатки (патчи), 725
запуск команд (файлы rc), 160
защита системы, организация, 639
защищенный режим процессора (Intel), совместимость, 31
звук, 368
обзор, 364
обработка, 45
цифровой, 365
звуковые драйверы, 369
звуковые карты
поддержка, 248, 369
установка/конфигурирование, 370
plug and play, 371
обзор, 370
тестирование, 373
устранение неполадок, 375
ядро, 372
значки для документов Web, 613

И

игры, поддержка, 45
имена
последовательных устройств, 583
принтеров, 281
серверов, 573
хостов
распознавание, host.conf, файл, 578
установка, 579
именование разделов, соглашения, 76
индексирование, Texinfo, 356
инкрементальные резервные копии, 263
инкрементный поиск, 330
инсталляция
Linux, 51, 91
драйверы устройств, 254
обновление против реинсталляции, 217
интегрированные среды разработки (IDE), 558
интернационализация рабочего стола KDE, 418
Интернет
CVS, утилита, 554
WWW (*см.* World Wide Web), 605
получение Linux из, 61
регистрация доменного имени, 630
интерфейсы пользователя
X Window System, 40
графические среды, 41
интерфейсы, GNOME, 428
панель, 430
история Linux, 10, 22, 25
история команд, 116
исходный код, 473

К

кабели, 98
канал (pipe), 120
каталоги, 108, 134
. (текущий), 122
Samba, 459
домашний, 169
права доступа, 126
принтера, 294
сохранение структуры в tar, 215
файлов устройств, 200
клавиши сокращенного набора, привязка клавиш (Emacs), 334
классы приложений, 440
кластеры, 45
клиент X, 388
книги по Linux, 54, 704
код, 475
оптимизация, 478, 518
кольцевой интерфейс, адрес, 568
кольцо удаления, 328
командная строка
расширение имен файлов, 117

редактирование, 114
клавиши стрелок, 116
командные сценарии (shell scripts), 34
команды
(*см.* утилиты), 109
Unix, 121
канал (pipe), 120
монтирование файловых систем, 183
обзор, 33
оболочка и vi, 319
стартовые, в файлах rc, 160
комментарии в
Emacs, 334
Texinfo, 353
коммерческое ПО, 49
веб-сайты, 683
копирование, ограничения, 60
коммутируемые линии и PPP, 581
компиляторы, 39
GNU C, gcc, 472
Java, 510
компиляция в Emacs, 333
компоновщик, 473
компрессия, 215
консоли
виртуальные, 162
драйверы, 248
программное обеспечение систем Alpha, 693
сообщения ядра при начальной загрузке, 157
контроль версий
CVS (Concurrent Version System), 550
через Интернет, 554
восстановление из резервных копий CVS, утилита, 550
символические ссылки, 112
контроль ревизий
RCS (Revision Control System), 546
восстановление из резервных копий RCS, утилита, 546
контрольные точки, 519, 531
конфигурирование
DNS, 587
Emacs, 325, 534
Ghostscript, 287
ISDN, оборудование, 590
KDE, рабочий стол, 416
Konqueror, 609
LILO, 150
NET-4, 570
NFS, 600
NIS, 602
PPP, 582
имена последовательных устройств, 583
синхронный PPP, 593
создание сценария для chat, 584
требования, 582
TCP/IP, 570
звуковые карты
ISA plug and play, 371
обзор, 370
тестирование, 373
устранение неполадок, 375
ядро, 372
учет изменений, 307
конфликт по адресам или IRQ, 95
копирование
дистрибутивов Linux, ограничения, 60
структур каталогов с помощью tar, 215
корневая (root) файловая система, 64
монтирование только для чтения, 195
обслуживание, 194
корневой (/) каталог, 147, 187

Л

ленточные приводы
без обратной перемотки, 260
Линус Торвальдс, 46
лицензии, 60
X11R6., 387
логические разделы, 64
лэптоп-компьютеры, поддержка, 43

М

магнитные ленты, создание резервных копий на, 259
макросы
Emacs, 331
make, 487
маршрутизаторы, 566
маска подсети, 571
менеджер файлов, Nautilus (GNOME), 431
менеджеры окон, 41, 409

микроядро, 31
микширование звука, 366
многозадачность, поддержка, 29
модемы, настройка PPP для, 582
модули
автоматическая загрузка, 254
драйверы устройств, 251
модули ядра, 71
монитор заряда аккумулятора, (GNOME), 431
мониторы, 400
частота обновления и синхронизации, 400
монолитное ядро, 31
монтирование
дисковых разделов Windows, 453
файловых систем, 183, 188
автоматическое, 191
мультимедиа, 25
MP3-проигрыватели, 376
звук
обзор, 364
поддержка, 45
цифровой, 365
полезность Linux, 27
мышь, система Alpha, 690

Н

настольные среды, 405
настройка
DNS, 587
Emacs, 325, 534
Ghostscript, 287
ISDN, оборудование, 590
KDE, рабочий стол, 416
Konqueror, 609
LILO, 150
NET-4, 570
NFS, 600
NIS, 602
PPP, 582
имена последовательных устройств, 583
синхронный PPP, 593
создание сценария для chat, 584
требования, 582
TCP/IP, 570
X Window System, среда, 405
звуковые карты
ISA plug and play, 371
обзор, 370
тестирование, 373
устранение неполадок, 375
ядро, 372
собственного веб-сервера, 610
учет изменений, 307
начальная загрузка
init, программа, 158
LILO
в качестве вторичного начального загрузчика, 153
настройка, 150
Linux, 75, 101, 156
проверка файловых систем, 195
загрузочная дискета, 147
устранение неполадок, 101
параметры начальной загрузки, 153
последовательность операций, 91
с установочного носителя
устранение неполадок, 92
сообщения ядра, 156
установочный диск, 62
устранение неполадок, 91–104
файловая система, 147
начальные загрузчики, 29, 86
LILO, 149
в качестве вторичного начального загрузчика, 153
Windows и LILO, 150
неверный формат, 99
независимость от платформы, 506
номера портов, httpd, 612
ноутбук-компьютеры, поддержка, 43

О

обновление, 237
библиотек, 217
компилятора, 222
патчи для ядра, 241
программного обеспечения
обзор, 215
оболочки (shell), 34, 112–114
vi и команды оболочки, 319
команды в проектах make, 490
популярные оболочки, 34
редактирование командной строки
клавиши стрелок, 116
оборудование
AlphaLinux, подготовка к установке , 698
ISDN, настройка, 590
X Window, требования, 389

обработка
документа, 338
слов, обзор, 34
текста, 338
LaTeX, 341
форматирование и печать, 344
SGML/XML/Docbook, 346
TeX, 341
обзор, 34
обработчики прерываний и ядро, 30
образ ядра и загрузочная дискета, 148
объектные файлы, 473
однопользовательский режим, Linux, 154, 159, 163
окна, управление, 389
операционные системы
загрузка неверной, 103
оптимизация кода, 475, 478, 479, 518
основной номер устройства, 201
останов системы, 90, 162, 164, 165
отладка, 72
gdb, отладчик, 39, 479
log-файлы, 269
выполняющихся программ, 527
интерактивные отладчики, 534
на уровне инструкций процессора, 532
обзор, 32
переназначение вывода ошибок, 119
таблиц маршрутизации, 580
отладчики, gdb, 39, 479
отмена изменений (vi редактор), 316
отображаемый в память ввод/вывод, 98
отслеживание зависимостей, 84
очередь печати, 379
управление, 299
ошибки
'cannot access memory at address...', 525
'device full', 100
'drive not bootable', 102
'file not found', 93, 100
'kernel panic', 147
'kernel too big', 250
'mount-point busy', 186
'no such file or catalog', 527
'out-of-memory', 93
'permission denied', 93, 100, 104, 184
'read error, file not found', 99
'read_intr: 0x10', 100
'stopped (tty input)', 123
'VFS: Unable to mount root', 93
'wrong fs type', 185

П

пакеты, 564
Perl, 502, 504
для научных приложений, 45
установка с помощью RPM, 224
память, 13, 32, 197
'cannot access memory at address...', ошибка, 525
du, команда, 120
адрес хранения переменной, 530
виртуальная, поддержка, 32
для заданий печати, 284
зависание системы, 92
нарушение границ, 521
нарушение доступа, 540
неправильное завершение работы, 90
проверка в файловой системе /proc, 166
размер RAM, сообщение ядру, 155
требования для резервных копий, 257
панель GNOME, 430
панель задач, KDE, 414
параллельные вычисления, 45
параллельный порт, поддержка, 246
параметры
fdisk, 78
для обнаружения аппаратной части, 70
начальной загрузки, 72
файла lilo.conf, 150
пароли, 103
root, 146
группы, 173
для доступа к ресурсам Windows, 457
создание учетных записей пользователей, 88
теневые пароли, 170
установка, 107
патчи (заплатки), ядро, 239
применение, 241, 555
песочница, 550
первичные разделы, 64

переключатель рабочих пространств (GNOME), 431
перемещение по тексту
в Emacs, 325
в vi, 317
перенаправление ввода, 120
перенаправление вывода
сохранение данных, 118
стандартного в файл null, 268
переносимость, 29
переразбиение жесткого диска, 63
требования Linux, 67
перечень файлов, 110
печатные ресурсы Linux, 54
печать, 386
BSD, 294, 297
CUPS, 306
enscript, утилита, 384
Ghostscript, конфигурирование, 287
KGhostview, печать из, 425
LaTeX, 344
lpc, утилита, реализация, 299
nenscript, утилита, 386
nenscript, фильтр, 291
TeX-документов , 345
имена принтеров, 281
предварительный просмотр, 37
принципы функционирования, 381
программное обеспечение для, 277
сопровождение, 301
спулинг, 378
удаленная печать, 285
управление, 272
устранение неполадок в системе печати, 303
утилиты, 279
факсимильные устройства, 279
фильтры, 288
волшебные, 292
планирование
Evolution, календарь, 436
заданий с помощью cron, 264
повреждение сжатых архивных копий, 262
поврежденные файлы, 310
повторная установка системы в сравнении с обновлением, 216
подсказка, Emacs, 327
поиск и замена
Emacs, 330
vi, 318, 320
поисковые машины и помощь по Linux, 56
полоса прокрутки, Emacs, 444
получение
исходного кода ядра, 240
носителя с Linux через заказ по почте, 60
помощи, 58, 88
пользователи
root, 146, 168
для задач cron, 269
журнал подключения, 271
учетные записи, 88, 177
создание/удаление, 177
помощь, 13
источники, 55
оперативная, 88
порты принтеров, 273
последовательные порты
скорость обмена, определение, 284
последовательные устройства
имена, 583
потоки вывода, 119
потоки, поддержка, 31
почтовые списки рассылки Интернета, 677
документация по Linux, 55
права доступа, 126
владельцы и группы, 127
изменение, 129
для останова системы, установка, 91
отказ при начальной загрузке, 93
системное администрирование, 143
для служб печати, 295–297
устранение неполадок, 305
устройства, 202
файловые системы только для чтения, 184
правила суффиксов (make), 489
преобразование адресов, 573
префиксные клавиши (Emacs), привязка, 336
приводы
CD-ROM с системами Alpha, 692
дискет, поддержка, 246
привязка ресурсов X Window, 442
приглашение
login, 92
пользователя root, 146
регистрации, 106

приложения
KDE, 419
X Window
работа в среде KDE, 416
коммерческие, 37
поиск веб-каталогов для, 45
промышленные, полезность Linux, 26
разработка, пригодность Linux для, 40
принтеры, совместный доступ, 42
программирование
в Emacs, 331
полезность Linux, 26
программное обеспечение
FTP -сайты, 678
GNOME, 439
Open Source, 48
shareware, 46
альфа- и бета-версии, 50
для печати, 277
для факсимильных устройств, 279
коммерческое
веб-сайты, 683
контроль качества, 49
копирование, ограничения, 60
обновление ПО, 215, 223, 234, 237
библиотек, 217
компилятора, 222
сборка нового ядра, 237
средства измерения производительности, 535
установка, 83
RPM, 224
Проект документирования Linux, 54
производительность, 11
AlphaLinux, 700
Java-программы, 506
оптимизация принтера, 301
средства измерения производительности, 535
пространство для свопинга, 158, 200
включение/отключение, 199
разбиение жестких дисков
на разделы и, 65
своп-раздел, 82
создание, 82, 198
установка Linux, 62
управление, 196
протоколы, поддержка
коммутируемых соединений, 42
профилирование, средства, 535
процессоры
поддержка Linux многопроцессорных систем, 29
сборка ядра и, 245
сообщения ядра при начальной загрузке, 156
процессоры слов, 339
процессы, 136
прямой доступ к памяти (DMA)
список используемых каналов, 167
псевдоустройства, 200
путь поиска, 122
включаемых файлов, 480
страницы руководства, 125

Р

работа в сети, 43
компиляция ядра, 246
рабочий стол, Nautilus (GNOME), 431
рабочий экран, KDE, 414
разделение ресурсов между Windows и Linux
утилиты преобразования файлов, 467
разделяемый ресурс, 464
разделы на жестких дисках, 67
для свопинга, 196
обнаружение, устранение неполадок, 96
сжатые, доступ к, 181
совместимость с Windows, 453
создание разделов Linux, 62, 78
разделы/файловые системы
/etc/fstab, файл, 89
виртуальные, 182
имена, 81
исправление, 196
компиляция ядра, 248
монтирование, 83, 188
автоматическое, 191
разделы Windows (shares)
монтирование, 453
разделы на жестких дисках, 192
переразбиение, 68
размонтирование, 184
своп, 198
создание, 83
типы, 183
требования Linux, 67

управление, 196
файловые системы в разделах, 192
размонтирование файловых систем, 184
распознаватель имен, 579
расширение имен файлов, 117
расширенные разделы, 64
регистрация, 103, 106
доменного имени, 630
пароли, установка, 107
приглашение, 106
регистры, 320
регрессивные тесты, 49
регулярные выражения, 497
в сравнении с расширением имен файлов, 118
поиск в (Emacs), 330
редактирование командной строки, 114
клавиши стрелок, 116
редакторы, текстовые, 339
резервные копии
cron и, 268
важность, 307
восстановление из, 312
инкрементальные, 263
на дискетах, 261
сжатие и, 262
создание простых, 258
реинсталляция против обновления, 217
репозиторий, 550
ресурсы
DNS, 565
X-ресурсы, 440
книги, 54
Samba, 458
SLIP, 569
TCP/IP, 561
TeX, 341
сетевые файловые системы, 183
системное администрирование, 142
по звуку, 377
получение помощи, 55
приложения, веб-каталоги для, 45
сетевые, 677, 684
телеконференции, 55
электронные документы, 54
ротация log-файлов, 272

С

сборка ядра, 250
аппаратное обеспечение и, 244
свободно распространяемое программное обеспечение, 48
свопинг, 32
своп-разделы, 196
сервер
World Wide Web, 610
X (*см.* X Window System), 388
буфера кадров, 392
имен, 573
сетевое взаимодействие
поддерживаемые устройства, 247
поддержка, 30
сетевые возможности, 26
полезность Linux, 26
сетевые приложения
поддержка, 42
поддержка, X Window System, 40
сети
Ethernet, настройка, 570
PPP, 569
модемы и, 581
rc, файлы, 573
Samba, разделяемые ресурсы SMB, 458
SLIP, 569, 581
TCP/IP, обзор, 560
X System и (*см.* X Window System), 388
адрес, 563
установка Linux на системы Alpha, 692
шлюзы, 566
сжатие, 215, 262
bzip2, утилита, 207
compress, программа, 207
gzip, утилита, 205, 208
tar-утилита и, 214
log-файлы, 271
резервных копий, 261
файлов, утилиты, 205
сжатые разделы, доступ к, 181
символические ссылки, 112
в файлах rc, 162
символы-заместители, 117
символьные устройства, 248
симметричные многопроцессорные платформы, поддержка, 245

система безопасности, 144
аутентификация, изменение метода, 171
пароли, установка, 107
права доступа, 126
системное администрирование
автомонтирование устройств, 188
безопасность, 143
компиляция нового ядра, 250
пространство для свопинга, 196
включение/отключение, 199
создание, 198
системные журналы, 269
утилита cron и мнимые пользователи, 269
учетные записи пользователей, 167, 172
passwd, файл, 168
модификация, 177
создание, 174
теневые пароли, 170
удаление, 176
файловые системы, 178
монтирование, 183
проверка и исполнение, 193
создание, 191
системные вызовы, 30
безопасность и, 143
отображение с помощью strace, 538
системный диск, создание, 68
скорость обмена, 284
скрытые файлы, 110
совместимость
CPU, 29
загрузка альтернативных ОС, 29
совместно используемые библиотеки, 481
совместное использование файлов с машинами Windows, 455
создание страниц руководства, 349
сокрытие IP-адресов, 562, 649
пример, 657
сообщения
загрузка ядра, 158
об ошибках
устранение неполадок при установке системы, 99
сохранение выводимых данных, 121
сохранение файлов
в Emacs, 326
в vi, 318
список окон (GNOME), 431
спулинг печати, 378
каталоги, 294
средства создания интерфейса, 542
ссылки символические, 112
стандарт иерархии файловой системы (FHS), 69
стандартное устройство сообщений об ошибках, 119
стандартный вывод, 119
ошибок, 119
перенаправление в null, 268
стандарты
веб-сайты, 683
совместимость, 29
стартовые команды, 160
стартовые файлы, 134
статические библиотеки, 474, 481
стек вызовов, 524
стиль для рабочего стола KDE, 418
стирание файлов
восстановление, 256
страницы, пустые, 306
страницы печати
высота, 285
ширина, 285
страницы руководства, 88, 123
mount, команда, 187
создание в groff, 349
строка зависимости, 484
суперблоки файловой системы, 186

Т

таблица разделов, 64
таблицы маршрутизации, отладка, 580
тайм-аут, ошибка, 98
текстовые редакторы, 33, 339
привязка к командной строке, 117
телеконференции, 677
X Window System, 404
документация по Linux, 55
программы чтения, 332
теневые пароли, 170
терминалы
setterm, утилита, 306
регистрация, 162
устройства, 162
эмуляция, 585
терминаторы SCSI, 98

тестирование
настройки TCP/IP, 579
подключения принтеров, 275
точки прерывания, 531
трассировка программ, 517
трассировщик лучей,программа, 364
требования к системе, 66
(*см. также* установка)
разделы на жестких дисках, 66
Третья расширенная файловая система, 179

У

удаление
LILO, 155
файлов, опасность непреднамеренного, 145
удаление и вставка
в Emacs, 328
в vi, 316, 320
удаленная печать, 285
узлы, Textinfo, 354
управление
заданиями, cron утилита, 264
сеансом, 411
уровень патчей, номера версий, 29
уровни исполнения, 158, 161
установка
Alpha, системы, 693, 696
GNOME, 427
KDE, 412
LILO, 85
Linux, 68
RPM, 224
Samba, 459
mount/submount, утилиты, 456
XFree86, 393
ведение учета, 63
загрузочная дискета, создание, 85
звуковые карты
plug and play, 371
обзор, 370
тестирование, 373
устранение неполадок, 375
ядро, 372
методы, 83
нового ядра, 250
обнаружение аппаратных устройств, 70
обновление в сравнении с повторной установкой, 216
программного обеспечения, 83
пространство для свопинга, создание, 82
разделы Linux, создание, 78
разделы на жестких дисках, изменение размеров, 67
реальные файловые системы, 83
страницы руководства, 351
с установочного носителя, 70
требования к системе, 66
устранение неполадок, 91–104
начальная загрузка с установочного носителя, 92
устранение неисправностей
log-файлы и, 270
mkfs, команда, 193
PPP, при настройке, 588
синхронный PPP, 596
аварийные ситуации, 307
аппаратное обеспечение, 94–99
дамп памяти, 521
загрузочная дискета, 101
настройка звука, 375
начальная загрузка, 102
после установки системы, 101
при входе в систему, 103
при работе с системой, 104
при установке системы, 92
программное обеспечение, 99
разделяемые ресурсы Windows, 458
регистрация, 103
сети, 580
системы печати, 303
файловые системы, 196, 309
устройства, 297
виртуальные, 297
драйверы, загрузка, 254
именование, 76
обработчики прерываний и ядро, 30
создание, 202
файлы устройств, 143, 203, 583
утилиты
(*см. также* команды), 33, 109
fdisk, 67
GNU, 25, 33
MTools, 451
tunelp, 302
Windows/MS-DOS совместимость, 43
архивирование и сжатие файлов, 205

обзор, 33
обработка текста, 35
печать, 279
поддержка лэптоп-компьютеров, 43
программирование, 38
сетевые, 43
создание учетных записей пользователей, 87
учет, ведение при установке Linux, 63
учетные записи пользователей, 88, 177
администрирование, 167
passwd, файл, 168
теневые пароли, 170
группы, 172
модификация, 177
создание, 86, 174
удаление, 176

Ф

факсимильные устройства, 279
файловые системы
MS-DOS, 181
proc, 165
автомонтирование, 188
администрирование, 178
восстановление, 309
в режиме только для чтения, 72, 154, 184, 194
журналирование, 181
монтирование, 183
начальная загрузка и, 147
поддержка, 30
проверка и исполнение, 193
разбиение жестких дисков на разделы и, 64
реальные, 83
совместимость с другими платформами, 182
создание, 191
установка Linux, 62
типы, 179
установка нескольких, 64
файлы
inittab, 159
passwd, 168
proc/filesystems, 179
rc, 160
архивирование, утилиты для, 205
блокировка (lpc), 299
восстановление из резервных копий, 256, 312
дописывание к, 119
исходного кода, 477
конфигурации
резервные копии и, 258
конфигурации ресурсов, 132
перечень, 110
поврежденные, 310
права доступа, для печати, 295
применение заплаток, 555
проектов, 483
make, синтаксис, 487
команды оболочки, 490
сжатие, утилиты для, 205
скрытые, 110
совместный доступ, 42
создание резервных копий, 264
сообщений ядра при начальной загрузке, 156
удаление, опасность непреднамеренного, 145
устройств, 200, 203
безопасность и, 143
местонахождение, 200
правила именования, 201
фиктивные объекты, 486
философия Linux, 48
фильтрация IP-пакетов, 646
фильтры печати, 284, 288, 382
nenscript, 291
волшебные, 292
фон, цвет
xterm, окно, 408
фоновый режим, процессы, 123
формат
printcap, файл, 280
Texinfo, 357
страницы руководства, 351
форматирование, LaTeX, 344
фреймы, Emacs, 445
функции, tar команда, 208

Х

хакерство, поддержка, 248
характеристики
Linux, обзор, 29
программного обеспечения, 33
хост, адрес, 563
хранилище, CVS, 550

Ц

цвет
Emacs и, 443
текста, 446
KDE, рабочий стол, фон, 416
целевые объекты, 483, 486
цилиндры, 73, 80
цифровые камеры, программное обеспечение GNOME, 438
ЦП, загрузка (GNOME), 430

Ч

часы, 423
чипсеты
AlphaLinux, 687
X Window и, 390
чтение, право на, 126

Ш

шаблоны
make, правила составления для, 489
поиск и замена
Emacs, 330
vi, 320
шлюзы, 566, 572

Э

электронная почта, 617
cron и, 268
Evolution, 435
fetchmail, пакет, 628
администрирование, 630
очереди, 630
электронная таблица (Gnumeric), 437
электронные диски, 93, 154
электронные документы (ресурсы Linux), 54

Я

ядро, 32
загружаемые драйверы устройств, 254
загрузка альтернативных, 151
загрузка в системах Alpha, 698
загрузочная дискета и, 147
звуковые драйверы, 369
исходный код, получение, 240
компиляция, 250
аппаратная часть и, 244
местонахождение, 148
монолитное, 31
нумерация версий, 27
обзор, 30
патчи для, 240
применение, 241
перекомпиляция, 250
сборка, 242
создание резервной копии, 258
сообщение panic, 147
сообщения ядра при загрузке, 156
язык в KDE, 418
языки форматирования, обработка текста, 36
языки программирования, 38
C, C++, 472, 483
Java, 39
LISP, 514
Perl, 496
PHP, 669
Scheme, 515
Tcl, 545
Tk, 545
сетевые ресурсы, 680

По договору между издательством «Символ-Плюс» и Интернет-магазином «Books.Ru – Книги России» единственный легальный способ получения данного файла с книгой ISBN 5-93286-069-3, название «Запускаем Linux, 4-е издание» – покупка в Интернет-магазине «Books.Ru – Книги России». Если Вы получили данный файл каким-либо другим образом, Вы нарушили международное законодательство и законодательство Российской Федерации об охране авторского права. Вам необходимо удалить данный файл, а также сообщить издательству «Символ-Плюс» (piracy@symbol.ru), где именно Вы получили данный файл.