По договору между издательством «Символ-Плюс» и Интернет-магазином «Books.Ru-Книги России» единственный легальный способ получения данного файла с книгой ISBN 5-93286-084-7 «UNIX. Практическое пособие администратора. 2-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» (www.symbol.ru), где именно Вы получили данный файл.

HIGH TECH

UNIX

ПРАКТИЧЕСКОЕ ПОСОБИЕ АДМИНИСТРАТОРА

Второе издание,
дополненное

Филипп Торчинский

СИМВОЛ®

Санкт-Петербург — Москва
2005

Серия «High tech»

Филипп Торчинский

UNIX. Практическое пособие администратора, 2-е издание

Главный редактор	*А. Галунов*
Зав. редакцией	*Н. Макарова*
Редактор	*В. Овчинников*
Художник	*В. Гренда*
Корректор	*О. Макарова*
Верстка	*Н. Гриценко*

Торчинский Ф.
UNIX. Практическое пособие администратора, 2-е издание. – СПб: Символ-Плюс, 2005. – 400 с., ил.
ISBN 5-93286-084-7

Книга адресована системным администраторам, часто вынужденным в поисках информации обращаться к разным источникам. Для ее прочтения опыт работы в UNIX не обязателен – достаточно представлять себе, как работает любая многопользовательская система – Novell Netware, Windows NT или VAX VMS. Те, кто не знаком с UNIX, смогут приступить к изучению этой системы, установив ее в соответствии с описанием, приведенным в книге. Много внимания уделено самым популярным в России системам UNIX – FreeBSD и Linux, однако почти весь материал применим и к другим диалектам UNIX. Рассмотрены установка «стандартного UNIX-сервера», командные процессоры, настройка POP3- и IMAP-серверов, установка системы, работа в текстовом редакторе, настройка PAM-модулей и некоторые другие задачи.

ISBN 5-93286-084-7

© Торчинский Ф. И., 2005
© Издательство Символ-Плюс, 2005

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 08.08.2005. Формат 70x100 1/16 . Печать офсетная.
Объем 25 печ. л. Тираж 3000 экз. Заказ N
Отпечатано с готовых диапозитивов в ГУП «Типография «Наука»
199034, Санкт-Петербург, 9 линия, 12.

Оглавление

Предисловие к первому изданию 11

Предисловие ко второму изданию 16

1. История UNIX 19

Дела минувших дней 19
UNIX сегодня 21
Проект GNU в жизни UNIX 23

2. Установка UNIX 26

На какие машины можно ставить UNIX 26
Выбор конфигурации компьютера 26
Конфигурация операционной системы 29
Разделы дисков в UNIX 30
Русификация 32
Процедура установки 32

3. Начало работы 39

Включение и выключение компьютера 39
Вход в систему и выход из системы 40
Режимы работы системы 40
Понятие терминала 41
Управляющие комбинации клавиш 43
Чтение документации. Организация man-страниц 43
Где хранятся настройки ОС? 45

4. Пользователи в системе UNIX 47

Пользователи и группы 47
Учетная запись пользователя 49
Предопределенные пользователи и группы 50
Привилегированный пользователь 51
Как стать привилегированным пользователем 51
Что делать, если вы забыли пароль суперпользователя 52
Управление учетными записями 53

5. Файловая система UNIX 54

Имена файлов и каталогов 56
Основные каталоги 57
Файлы и каталоги в UNIX 58
Права доступа к файлам и каталогам 65
Типы файловых систем 71
Таблица индексных дескрипторов 75
Монтирование и демонтирование файловой системы 78
Применение символических ссылок 79
Поиск файлов 81

6. Процессы 83

Распределение памяти. Свопинг 84
Идентификаторы процесса 85
Приоритеты процесса 85
Доступ процессов к файлам 86
Запуск процесса от чужого имени 86
Интерактивные и фоновые процессы 88
Взаимодействие процессов 89
Управление процессами 97
Работа с заданиями 101
Планирование запуска процессов 102
Регулярно выполняемые системные скрипты 105

7. Работа с текстами 106

Термины и кодировки текста 106
Случай с порнографическими изображениями 107
Зачем системному администратору работа с текстом 108
Редактор vi 109
Программы поиска и замены в текстах 113
Регулярные выражения 116
Дополнительная литература 119

8. Командные процессоры 120

Обзор командных процессоров 120
Действия командного процессора при запуске 122
История команд 123
Псевдонимы команд 124
Подстановка имен файлов. Шаблоны 125
Подстановка результата выполнения 125
Подсказка имен файлов 126
Средства редактирования командной строки 126
Командный процессор с ограничениями возможностей 127

Повторение ранее введенных команд (с модификацией и без) 128
Переменные среды окружения 128
Предопределенные переменные 130
Язык написания скриптов 131
Установка приглашения командного процессора 136
Перечень спецсимволов 136
Разбор командной строки 137
Дополнительная литература 137

9. Резервное копирование 138

Расчет на авось против резервного копирования 138
Как использовать dump и restore 141
Копирование файлов на удаленный компьютер: программа rsync 144

10. Работа UNIX в сети 146

Сетевые стандарты и организации 147
Модель межсетевого взаимодействия 148
Протокол TCP/IP 151
Настройка TCP/IP 158
Маршрутизация 164
Где находятся базовые настройки сети в UNIX 169
Планирование адресации в сети 172
Сетевые службы в UNIX 177
Процесс inetd 178
Страж tcp-wrapper 180
Proxy-серверы 181
NAT 183
Защита сетей с помощью фильтров пакетов 191
Виртуальные частные сети 194
Мониторинг сети и борьба со сбоями сети 198
Дополнительная литература 208

11. Основы системы имен доменов (DNS) 209

Как работает DNS 212
Полностью определенное доменное имя 214
Настройка сервера имен 214
Получение информации от DNS 223
Правовые аспекты регистрации доменов 224

12. Процедуры запуска и останова системы 226

Режимы работы UNIX: runlevels 228
Загрузка систем BSD 229
Загрузка систем System V 233

13. Установка новых программ в UNIX 235

Компиляторы и компоновщики 235
Программа make 236
Несколько способов установить программу 240
FreeBSD: ports collection, команда pkg_add 240
Linux: команда rpm 242
Debian Linux: команда dselect 242
Дополнительная литература 242

14. Конфигурация ядра системы 243

Модули vs статические драйверы 243
Конфигурация основного ядра 244
Загружаемые модули ядра 247

15. Настройка удаленного доступа pppd 250

16. Сетевая файловая система NFS 256

Общие соображения 256
Принцип работы NFS 257
Настройка NFS-сервера 258
Блокировка файлов на NFS-сервере 265
Настройка NFS-клиента 271

17. Печать в UNIX 276

18. Время в UNIX 279

Который час 279
Синхронизация времени: timed 280
Синхронизация времени: ntpd 283
Дополнительная литература 290

19. Установка и настройка почтового сервера 291

Что такое почтовый сервер 291
Задачи сервера передачи почты (MTA) 292
Задачи почтового клиента (MUA) 293
Задачи сервера выдачи почты (MAA) 294
Установка и начальная настройка sendmail 297
Sendmail: куда обратиться за черным списком? 303
Sendmail: как установить внешний фильтр спама и вирусов? 304
Установка и начальная настройка postfix 304
Postfix: куда обратиться за черным списком? 305
Установка POP3-сервера 306
Настройка IMAP-сервера 307

20. Основы работы с веб-сервером Apache 308

Как устроен сервер Apache 308
Установка Apache 309
Основные настройки Apache 311
Работа с SSI 312
Как заставить выполняться CGI-скрипты 313
Обработка скриптов на PHP 314
Обработка скриптов на Perl 315
Протоколирование запросов и событий 316
Безопасность httpd 316
Аутентификация в Apache 318
Ограничение количества одновременных соединений 319

21. Настройка http-cache Squid 321

Установка и основные настройки 321
Борьба с баннерами 324
Ограничение доступа в Сеть 326
Ведение и анализ протоколов 328

22. UNIX в роли Windows NT (пакет Samba) 330

23. Установка и настройка СУБД MySQL 336

Установка MySQL 336
Настройка mysql: права доступа, создание БД 337
Распространенные ошибки 339
Дополнительная литература 340

24. Аутентификация. PAM-модули 341

Аутентификация в любой программе 341
Аутентификация с помощью PAM 342
Аутентификация на UNIX через домен Windows-сети 347

25. Основы настройки X Window 350

Настройка с помощью xf86setup 352
Русификация X Window 353
Где узнать больше об X? 353

26. Русификация UNIX: обзор 354

Проблемы русификации UNIX 354
Локализация и русификация 355
Символы и кодировки 356
Русификация bash и vi 357
Перекодировка русскоязычных текстов 358

Русскоязычные имена файлов . . . 359
Русификация Solaris . . . 359
Дополнительная литература . . . 360

27. Наблюдение за системными ресурсами . . . 361

28. Предотвращение сбоев . . . 368

29. Полезные демоны и утилиты . . . 373

Архиваторы и упаковщики . . . 373
Антивирусы . . . 375
Борьба с вирусами: Anomy Sanitizer . . . 377
Борьба с вирусами: ClamAV . . . 378
Атаки . . . 379
Протоколирование событий: syslogd . . . 379
Дополнительная литература . . . 380

A. Краткий справочник по командам и файлам конфигурации UNIX . . . 381

Инициализация среды работы пользователя . . . 381
Настройка сетевых параметров . . . 382
Конфигурация системы . . . 387

B. Указатель веб-адресов . . . 390

Алфавитный указатель . . . 393

Посвящается моему дедушке
Павлу Дмитриевичу Астапенко

Предисловие к первому изданию

Тот, кто умеет, тот делает, кто не умеет – учит.
Б. Шоу

Зачем читать эту книгу?

Есть старая шутка о том, что было бы, если операционные системы были бы авиакомпаниями.

Unix Airways описана в ней так: «Каждый пассажир приносит с собой в аэропорт деталь самолета. Затем они встречаются на взлетной полосе и начинают собирать его, бесконечно споря, какой именно самолет они делают. Когда пассажиры наконец поднимаются на борт, один из них назначается пилотом и следует в кабину. Там он находит инструкцию, описывающую в мельчайших деталях устройство самолета, но не дающую никаких указаний по пилотажу».

Задача этой книги – быть хорошим руководством по пилотажу, которое будет вам полезно независимо от того, кто, когда и как собрал ваш самолет.

Я много лет работаю системным администратором, и часто мне приходится искать ответы на свои вопросы в разных книгах по UNIX. Бывает, целые главы книг оказываются устаревшими, и довольно редко нужные сведения находятся под одной обложкой. Приходится просмотреть две-три книги и почитать руководство, чтобы найти решение задачи.

Идея этой книги родилась в минуту, когда я очередной раз раскладывал на столе пасьянс из толстых томов. Не исключено, что вы уже прочли несколько книг по UNIX, принадлежащих перу иностранных авторов, и почерпнули из них много полезного. Однако пока мне не встречалась ни одна иностранная книга, которая бы учитывала опыт работы с UNIX в России. Согласитесь, задачи системных администраторов американского университета с тысячами UNIX-станций отличаются от забот системного администратора интернет-провайдера в Горно-Алтайске,

где этот администратор в одиночку управляется с немаленькой сетью большого района.

Эта книга рассчитана, прежде всего, на российского системного администратора, т. к. в ней рассмотрены задачи, характерные для компьютерных сетей и систем в России. Одна из таких задач – установка «стандартного UNIX-сервера». Обычно под таковым понимают компьютер, который работает веб-сервером, почтовым сервером и маршрутизатором одновременно. Благодаря надежности и дешевизне такого решения оно приобрело популярность везде, где нагрузка на такой компьютер сравнительно невелика, – в офисах, небольших институтах и бизнес-центрах.

В России есть, на мой вкус, всего одна практически всеобъемлющая книга по UNIX [1]. Она поистине прекрасна! Однако она написана американскими специалистами для американских специалистов. В книге, которая лежит перед вами сейчас, я стараюсь описывать UNIX так, как он в действительности используется в России. Для тех, кто уже знаком с книгой Немет и других [1], я расскажу, чем моя книга отличается от классического «Руководства системного администратора».

Во-первых, она рассчитана на достаточно опытных людей, но опыт работы в UNIX для ее прочтения не обязателен. Эта книга проще, с ее помощью можно осваивать UNIX с нуля. Достаточно представлять себе, как работает любая многопользовательская система. Novell Netware, Windows NT или VAX VMS – не важно. Надеюсь, что эта книга покажется читателю менее сложной.

Во-вторых, здесь рассказано об аспектах UNIX, опущенных в [1] из-за кажущейся простоты или вследствие их новизны: от написания книги до ее появления на полках магазинов проходит несколько месяцев, а если книга переводная, то и больше. Например, рассмотрены разные shell'ы, подробнее описана настройка POP3 и IMAP-серверов, установка системы, работа в текстовом редакторе, настройка PAM-модулей и некоторые другие задачи.

В-третьих, не описано то, чем не пользуются системные администраторы на практике в отечественных сетях.

Поэтому хочу надеяться, что мне удалось написать достаточно короткую информативную книгу, которая будет хорошим справочником по системе. Это никак не отменяет необходимость читать руководство по UNIX, поставляемое вместе с системой.

Тот, кому UNIX внове, сможет изучить систему в деталях с этой книгой в руках. Установка системы описана уже в главе 2, что дает читателю возможность поставить систему и учиться с ней работать по мере чтения.

Тот, чья цель – быстро настроить «стандартный UNIX-сервер», может сделать это, следуя советам в главах 2, 10–12, 19–21.

Самые популярные в России системы UNIX – FreeBSD и Linux, поэтому в книге много внимания уделяется именно им. Однако почти все, что в ней рассказано, применимо и к другим системам UNIX. Если какое-то описание относится только к конкретной системе, это будет отдельно оговорено в тексте.

Предполагается, что читатель имеет опыт работы с компьютером, знает, что такое файл и каталог, не боится работать без графического интерфейса, знает английский язык на уровне способности читать сообщения системы и страницы руководства. Последнее весьма важно. Эта книга написана по-русски, потому что она адресована, прежде всего, русскоязычным читателям. Однако чтобы разобраться, как работает программа, отчего новая версия почтового сервера не функционирует, почему не загружается старая программа на новой версии UNIX и т. д., придется читать руководство, устанавливаемое как часть системы (команда `man`), и статьи в конференциях в Интернете. Без знания английского вы пропадете. Если язык вам пока не был нужен, то это – хороший повод его выучить.

Всякий раз, когда я рассказываю о UNIX и не могу ответить на чей-то вопрос, меня раздирают два противоречивых чувства: во-первых, жгучее желание найти ответ, а во-вторых, воспоминание о фразе Бернарда Шоу, которая вынесена в эпиграф предисловия. И так много лет подряд, потому что интересные вопросы не иссякают: ведь UNIX совершенствуется и растет вместе с новыми сетями и службами. Часть таких вопросов легла в основу некоторых разделов этой книги. Поэтому я прошу читателей присылать мне вопросы. Часть ответов на них будет опубликована на веб-сайте этой книги *http://unixbook.spb.ru*.

Пожалуйста, не надо этих приторных «www» перед *unixbook.spb.ru*, они так и вязнут в зубах! Вам нравится произносить «вэ-вэ-вэ»? Если да, купите книжку по Windows, в этом слове есть целых две буквы «вэ».

Если ответов наберется достаточно для второго издания книги, то их даже будет можно купить.

Мне кажется, что писать о себе в третьем лице – это нездорово и неприлично. Поэтому я не буду в книге писать «автор думает» о том, что думаю я. Однако из скромности не хочу все время писать «я». Буду писать «мы», полагая, что если уж читатель дочитал до этого места, то он со мной в чем-то соглашается, и это «мы» будет означать меня и читателя.

Ответственность

Материал, использованный при подготовке этой книги, взят из надежных источников. В большинстве примеров файлы конфигураций

и скрипты не придуманы специально для книги, а взяты из реальных систем, работающих в разных организациях.

Однако я не исключаю, что в процессе подготовки книги могли быть допущены ошибки или опечатки, которые могли повлиять на функциональность примеров. Пожалуйста, напишите мне по адресу *filip@unix-book.spb.ru*, если заметите ошибку.

Ни издательство, ни автор книги не берут на себя ответственность за то, что может случиться, если вы будете использовать эту книгу в своей работе. Если, несмотря на это, вы будете следовать инструкциям, данным в книге, делайте это на свой страх и риск.

Администрируя системы, от которых зависит жизнь и безопасность людей, проверяйте каждую мелочь перед тем, как применить ее на практике. Обязательно сверяйтесь с руководством по системе (в том числе с `man`).

Если вы, прочтя этот раздел, по-прежнему готовы работать по моим рецептам, то я желаю вам удачи!

Благодарности

Эта книга была бы невозможна без добрых советов моих друзей. Прежде всего, хочу поблагодарить Максима Рязанова, советы которого серьезно обогатили материал нескольких разделов книги. Спасибо Игорю Николаеву за то, что он всегда отвечал на все мои самые трудные вопросы. Кроме того, он является автором лучшей инструкции по настройке X-Window, которая только есть в Интернете.

Я благодарен моим друзьям и коллегам из Института высокомолекулярных соединений РАН, где я приобрел интереснейший опыт работы, а также Компьютерной академии Софтджойс, где многие слушатели прошли мои курсы по UNIX.

Огромная благодарность также Европейскому университету в Санкт-Петербурге, где я работаю в настоящее время. Приобретенный в его стенах опыт, несомненно, помог мне сделать книгу лучше.

Особенно я благодарю Ларису, которая лучше всех моих друзей знает, как важна поддержка любимого человека в любой серьезной работе.

Как пользоваться книгой

Эта книга задумана как справочник для администратора ОС и как руководство для того, кто только собирается администрировать UNIX. Поэтому можно сразу переходить к той главе, которая описывает интересующую вас тему или программу, а можно читать все подряд.

В книге вы найдете ответы на вопросы, возникающие в работе администратора, но не подробное описание каждой из системных утилит UNIX. Пожалуйста, читайте руководство по системе (команда **man**) для того, чтобы получить детальную информацию о программах. Более того, вы будете совершенно правы, читая руководство даже по знакомым программам в новой для себя системе. Синтаксис или ключи команд могут меняться от версии к версии.

Шрифты и условные обозначения в этой книге практически совпадают с теми, что приняты в книге [1]:

- названия и аргументы команд, которые надо набирать на клавиатуре без изменений, даны шрифтом **courier bold**;
- аргументы команд и другие фрагменты, вместо которых надо подставлять конкретные значения, даны шрифтом *courier italic* (курсив);
- имена файлов, а также все, что выводят на экран программы UNIX, даны шрифтом courier;
- адреса электронной почты и Интернета выделяются *курсивом*;
- при описании синтаксиса команд UNIX используются те же символы, что и в руководствах по системе:
  - текст, заключенный в квадратные скобки [], может отсутствовать;
  - в шаблонах имен файлов используются символы * (означает любое число любых символов) и ? (один любой символ).

Дополнительная литература

В России издано множество книг по UNIX. Наиболее достойные, с моей точки зрения, перечислены ниже, и я искренне благодарю их авторов за идеи, которые помогли мне в работе, в том числе и над этой книгой.

[1] Немет Э., Снайдер Г., Сибасс С, Хейн Т. «UNIX: руководство системного администратора». – СПб: Питер, 2002 (это третье издание данной книги, самое лучшее, хотя и несколько устаревшее; первое издано в 1997 г.: Немет Э., Снайдер Г. «UNIX: руководство системного администратора». – BHV-Киев).

[2] Робачевский А. «Операционная система UNIX». – СПб: BHV, 2000.

[3] Дунаев С. «UNIX-сервер. Настройка, конфигурирование, работа в операционной среде, Internet-возможности». – СПб: BHV, 1998 (или предыдущее издание: Дунаев С. «Операционная система UNIX System V Release 4.2». – М.: Диалог-МИФИ, 1995).

Предисловие ко второму изданию

Системные администраторы, директора IT-отделов, программисты, студенты, школьники и другие читатели книг о UNIX ищут ответы на свои вопросы в Интернете. Зря ли вы потратили деньги, заплаченные за эти книги? Этот вопрос вряд ли волнует тех, чьи покупки оплатила компания, но что думать тем, кто выложил свои кровные средства за четыреста-пятьсот страниц мелким шрифтом?

Пока любители книг могут быть спокойны: книги выигрывают у Сети по нескольким пунктам:

- Читателю книг безразлично, доступен ли веб-сервер, на котором хранятся нужные вам документы или форум.
- Книга не скажет вам «404 Document not found», если только младший брат не вырвал из нее несколько страниц.
- Текст в книге не меняется со временем, и если на странице 420 раньше описывалась ваша версия Interbase, то через год там не появится описание более новой, и менять старую добрую версию СУБД не придется только потому, что вы забыли, как с ней управляться, а документацию больше не можете найти.
- Книгу удобно читать, сидя в метро, а читать в метро, глядя на экран компьютера, пока еще не слишком приятно – глаза напрягаются сильнее, чем хотелось бы.
- В книге нет рекламы порносайтов, равно как и прочей рекламы, ни вверху, ни справа, ни слева на странице.
- В книге нет ссылки на спонсора внизу каждой страницы.
- Ссылку на книгу не надо искать на Google, если вы забыли, как она называется или где лежит (хотя жаль, что Google не знает, где лежат разные нужные вещи у нас дома – например, куда делся этот чертов эпилятор или где же старый USB-дисковод от ноутбука, который как раз сейчас понадобился).

При подготовке второго издания этой книги мы старались к этим преимуществам добавить еще одно, характерное для Интернета: актуальность. С момента выхода предыдущего издания разработчики UNIX ушли довольно далеко вперед. Появились новые версии систем, возникли и были решены новые задачи.

Основные отличия второго издания книги от первого:

- Более подробно описаны новые возможности файловых систем UNIX, в частности назначение прав доступа к файлам и каталогам для отдельных групп и пользователей (ACL, extended access rights).
- Полностью изменено описание почтовых систем, сделан обзор методов борьбы со спамом и фильтрации вирусов в письмах, добавлен раздел про настройку postfix, намного подробнее описана настройка sendmail.
- Глава «Резервное копирование» пополнена информацией о новых устройствах для резервного копирования, расширен обзор практического опыта, добавлен раздел, рассказывающий о программе rsync, гениально синхронизирующей содержимое каталогов по сети.
- Существенно дополнен предметный указатель в конце книги, который, к сожалению, был малоинформативен в первом издании.
- Исправлены некоторые неточности (и, скорее всего, добавлены новые, поскольку ничто не совершенно в этом мире).

Я благодарен тем читателям, которые откликнулись на призыв писать мне о найденных в книге опечатках и неточностях, и, прежде всего, Николаю Друганову из Хабаровска, выигравшему конкурс на поиск опечаток и ошибок «Ловим блох», объявленный на моем сайте *unix-book.spb.ru*. Спасибо и Евгению Жукову из Воронежа, и Владимиру Демидову, замечания которых помогли сделать второе издание лучше.

1

История UNIX

Дела минувших дней

В 1969 году сотрудники подразделения Bell Labs фирмы AT&T Деннис Ритчи (Dennis Ritchie) и Кен Томпсон (Ken Thompson) создали новую ОС, которая была задумана как многопользовательская и многозадачная. Первая версия UNIX была написана на ассемблере для компьютера PDP-7. Томпсон загорелся идеей написать код новой операционной системы на языке высокого уровня. Вскоре после этого он разработал язык B, а Ритчи усовершенствовал его и назвал новый язык C, и друзья решили написать на нем весь код новой системы. Летом 1972 года они завершили эту работу и создали первую в мире систему, полностью написанную на языке высокого уровня, что позволяло перенести ее на любую аппаратную платформу. Первым компьютером, которым она управляла, стал новый компьютер PDP-11, представлявший собой настолько совершенную разработку, что его советские аналоги успешно проработали в России до начала 90-х годов, после чего были сданы в утиль. Они были сделаны из элементов, содержавших много золота, и их сгубила корыстная любовь человека к цветным металлам.

Во второй половине 70-х годов XX века выпускники университета в Беркли (University of California-Berkeley) начали работу над собственной версией UNIX. Дальнейшее развитие UNIX иллюстрирует рис. 1.1, основанный на материале, размещенном по адресу *http://www.ehlis.com/adam/solaris/history.html*.[1] В 1977 году вышла первая

[1] В 2005 году эта ссылка стала недоступной; более свежую информацию об истории UNIX можно почерпнуть, например, по адресу *http://www.levenez.com/unix/*.

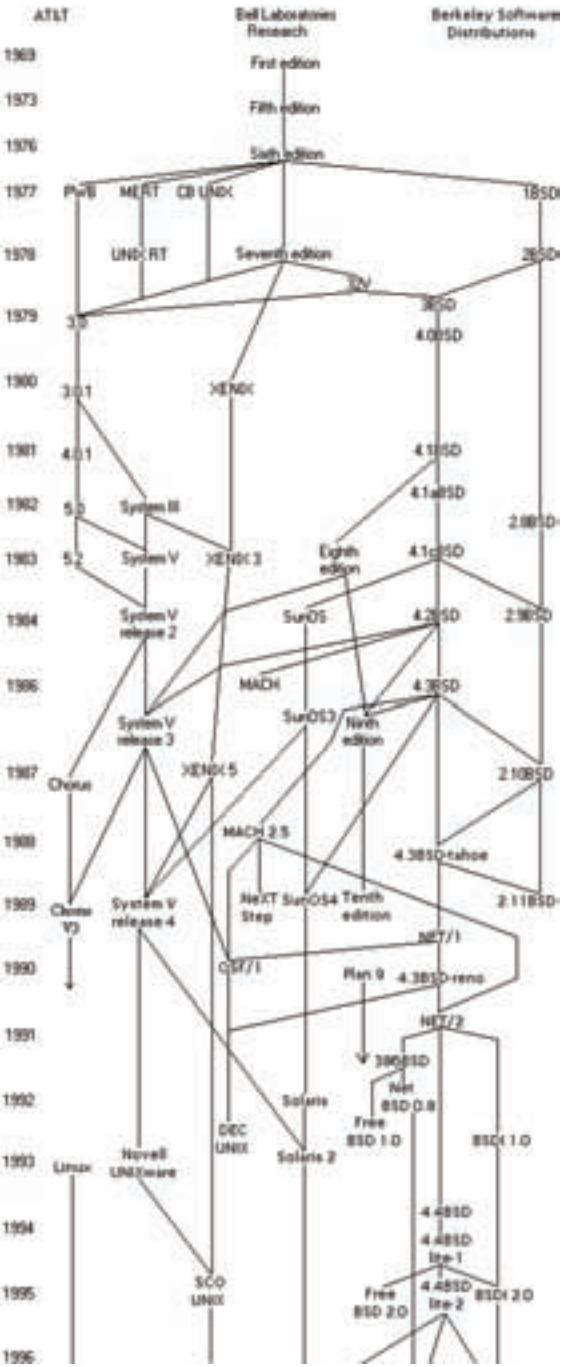


Рис. 1.1. *Рост дерева UNIX*

версия Berkeley Software Distribution – популярной университетской версии UNIX. Поскольку эта система была основана на коде AT&T, для ее использования требовалась лицензия AT&T. Лицензии на код ОС быстро росли в цене вместе с популярностью UNIX. Поэтому разработчики из Беркли решили убрать код AT&T из своей версии UNIX. Работа была долгой и трудной, и они завершили ее к 1991 году, выпустив систему Net/2. Вскоре после этого Исследовательская группа вычислительных систем (Computer Systems Research Group) была расформирована из-за недостатка финансирования. Код Net/2 был взят за основу несколькими компаниями, в частности BSDI (Berkeley Software Design, Inc.). Разработки UNIX двинулись в двух направлениях: одни в качестве исходной версии использовали версию Беркли, другие – версию System V Release 4.2, которая была разработана AT&T.

Разнообразие версий UNIX вызвало у разработчиков желание стандартизировать систему; они объединились в консорциум X/Open, который создал спецификацию Spec1170. Консорциум был создан в 1984 году, первоначально в него входили компании Bull, ICL, Siemens, Olivetti и Nixdorf. Спецификация Spec1170 содержит API (Application Program Interface) UNIX, и ее поддерживают все основные производители UNIX.

В 1988 году IBM, DEC, HP, Sun и AT&T основали Open Software Foundation (OSF). В июне 1993 года компания Novell приобрела торговую марку UNIX у AT&T и передала ее консорциуму X/Open в ноябре 1993 года.

В 1996 году X/Open и OSF объединились в The Open Group. В настоящее время этот независимый международный консорциум (*http://www.opengroup.org*) является владельцем торговой марки UNIX. Консорциум спонсируется компаниями Compaq, Fujitsu, HP, Hitachi, IBM, Motorola, Sun.

Сейчас принято говорить о двух ветвях UNIX: System V (эта ветвь ведет свою родословную от кода System V release 4) и BSD (ведет начало от Net/2). Относящиеся к ним ОС отличаются друг от друга некоторыми архитектурными и административными особенностями. Многое в этих системах одинаково или схоже, поэтому в данной книге мы будем специально отмечать отличия.

UNIX сегодня

Самые известные диалекты UNIX – это Linux и FreeBSD. Они бесплатны, свободно распространяются в исходных текстах и достаточно надежны для того, чтобы их эксплуатировали в самых серьезных приложениях. Более того, в Германии недавно был принят закон, запрещающий использовать в государственных учреждениях операционные системы, код которых не является открытым. Фактически сейчас это означает, что можно работать только с Linux и FreeBSD.

Мы будем называть различные UNIX диалектами системы, если они разработаны разными производителями, и говорить о версиях каждого

из диалектов. Будем также помнить, что Linux называют любую из ОС UNIX, работающую с ядром Linux. Сейчас это ядро разрабатывает команда программистов во главе с Линусом Торвальдсом (Linus Torvalds). В остальном разные Linux обычно ведут себя как потомки UNIX System V, отличаясь, однако, друг от друга в деталях и комплектом поставки. Разные диалекты Linux появились потому, что поставкой систем на базе одного и того же ядра занялись разные компании, предлагающие разные дополнительные программы в дистрибутиве ОС, разные графические среды, разные условия технической поддержки.

Широко известны продукты фирмы Sun Microsystems – SunOS и Solaris. Последняя версия Solaris 10 продолжила линейку хорошо себя зарекомендовавших серверов приложений. Чем отличаются SunOS и Solaris? Алан Бикрафт (Alan Beecraft), инженер из Sun Developer Technical Support, объясняет это в форуме разработчиков Sun так: SunOS – это «операционная система», одна из компонент продукта «Операционная среда Solaris», который обычно называют просто Solaris. SunOS была самостоятельной системой до того, как стала частью Solaris; ее последняя самостоятельная версия – SunOS 4.1. Начиная с версии 5.x SunOS – это часть Solaris версии x. Например,

Solaris 2.6 включает SunOS 5.6
Solaris 7 включает SunOS 5.7
Solaris 8 включает SunOS 5.8

Переход от Solaris 2.6 к Solaris 7 означал переход к первой 64-разрядной версии Solaris – Solaris 7. В Solaris команда `uname -a`, которая сообщает версию системы, выводит «SunOS 5.x».

Наиболее известны следующие коммерческие UNIX: IRIX от Silicon Graphics, AIX от IBM, HP-UX от Hewlett Packard, Unixware от Santa Cruz Operation (это результат слияния двух ОС – SCO от Santa Cruz Operation и Unixware от Novell). Особняком стоит система реального времени QNX, отличающаяся гарантированным временем реакции на внешние сигналы и применяемая в системах управления технологическими процессами, а также в военных целях. QNX вызвала бурю восторга российских пользователей, когда в конце 90-х годов стала распространяться ее минимальная версия, занимавшая всего одну дискету и позволявшая работать в графической среде, дозваниваться до провайдера, работать в Интернете и отправлять/получать почту.

UNIX – многоплатформенная ОС, и в табл. 1.1 показано, на каких платформах работают разные версии UNIX. Эта таблица неполна. Например, надо иметь в виду, что существуют многочисленные разработки Linux для многих платформ. Более детальную информацию о том, какие Linux на каких платформах работают, можно получить, например, в документе

http://www.redhat.com/mirrors/LDP/FAQ/Linux-FAQ/intro.html#PLATFORM-SUPPORT

Таблица 1.1. Платформы, поддерживаемые некоторыми из UNIX

Система	Свежайшая рабочая версия (beta-версии не учитываются) по сост. на май 2005 г.	Производитель	Платформа
Solaris	10	Sun Microsystems	Sun, i386
FreeBSD	5.3	BSDI	i386, Alpha, PC98
Red Hat Linux	9.0	Red Hat	i386, Alpha
Mandrake Linux	10.0	MandrakeSoft	i386, PowerPC
AIX	5L	IBM	PowerPC, RS/6000
Digital UNIX, OSF/1	Tru64 V5.1B-2	DEC (компания куплена Compaq и Samsung), в настоящее время Tru64 выпускается компанией HP, поглотившей Compaq	Alpha, IA-64
HP-UX	11i version 2	Hewlett Packard	HP
IRIX	6.5	SGI	SGI
Unixware	7.1.4	SCO	i386

Проект GNU в жизни UNIX

Многие справедливо полагают, что программное обеспечение должно быть бесплатным. Так считают и те люди, которые зарабатывают себе на хлеб написанием программ и администрированием систем. Спрашивается, как же можно зарабатывать деньги на программном обеспечении, если оно ничего не стоит?

Ответ весьма прост. Сами программы бесплатны, но ведь ими надо уметь пользоваться. Стало быть, авторы бесплатных программ могут зарабатывать на обучении работе с ними. Кстати, так они и делают. Более того, кто-то ведь будет применять эти программы. И за это он получит деньги. Согласитесь, когда ремонтируют вашу квартиру, вам ведь все равно, заплатил маляр за кисти и краску или нет. Вы платите за работу, а стоил ли инструмент мастеру хоть грош, вас не очень волнует.

Многие очень полезные программы в мире UNIX были написаны администраторами или программистами для облегчения собственного труда, поскольку не было удобных программ, которые выполняли бы нужную работу. Поднаторевшие в практическом применении своих удобных программ и в их совершенствовании специалисты начинали выполнять заказы коммерческих компаний. Это не мешало им заниматься бесплатным ПО. Наоборот, бесценный опыт программирова-

ния помогал им выполнить менее общие, специфические задачи коммерческих проектов.

Практика показывает, что безоглядное взимание денег за каждый шаг тормозит прогресс, потому что настоящий специалист думает о том, как сделать работу лучше и быстрее, а не как заработать на ней побольше. Репутация людей, которые писали sendmail, MySQL, Linux, Perl и другие некоммерческие продукты, такова, что компании готовы платить им за возможность указать их в числе работников. Так что специалисты в любом случае не бедствуют.

Проект GNU стартовал в 1984 году с легкой руки Ричарда Столмэна (Richard M. Stallman). Цель проекта – создание и распространение свободного программного обеспечения. Программы, написанные в рамках проекта GNU, можно бесплатно копировать, распространять и модифицировать, а также распространять модифицированные копии. Бесчисленное множество программ, созданных в рамках проекта GNU, распространяется с лицензией GPL (General Public License). Эта лицензия предполагает «copyleft» на распространяемые объекты. Смысл «copyleft» (термин введен участниками проекта GNU), в отличие от «copyright», в том, чтобы гарантировать пользователю объекта copyleft (программы, текста книги и т. п.) полную свободу распоряжения этим объектом.

Важный момент, связанный с copyleft, состоит в том, что человек, получивший продукт на правах copyleft и распространяющий его, должен обеспечить трансляцию прав copyleft на всех тех, кому он передал этот продукт.

В частности это означает, что, даже продавая продукт copyleft, вы обязаны предоставить покупателю все те же права, что и ваши собственные. Публикуя на своем сайте текст, распространяющийся на правах copyleft, вы автоматически обязаны разрешить всем и каждому копировать этот текст бесплатно (или за деньги, по желанию потребителя) и распространять его бесплатно или за деньги. Поэтому часто финансово невыгодно включать продукты, защищенные правом copyleft, в продукты, продаваемые за деньги: последние автор обязан сделать публично доступными.

В жизни UNIX проект GNU (GNU – рекурсивная аббревиатура GNU's Not UNIX) сыграл очень важную роль. Много утилит UNIX распространяется в рамках этого проекта и было написано его участниками.

Некоторые программы имеют как коммерческую, так и бесплатную версию. Так, пакет библиотек языка C под UNIX может называться libc (не-GNU-версия) и glibc (GNU-версия). Иногда в одной системе уживаются GNU-версии программ и их коммерческие версии. Для того чтобы подчеркнуть, что утилита создана в рамках проекта GNU, ей может быть дано имя, начинающееся с «g»: gcc, gzip, gdb.

К девяностым годам XX века участники проекта GNU нашли или написали все основные компоненты операционной системы, кроме одного – ядра. Тогда Линус Торвальдс (Linus Torvalds) разработал Linux – ядро, исходный текст которого распространялся на условиях GPL. Комбинация Linux и почти завершенной системы GNU создала полную операционную систему: систему GNU, основанную на Linux. Современные UNIX-продукты на ее основе было бы правильнее называть GNU/Linux.

2

Установка UNIX

На какие машины можно ставить UNIX

Большинство современных систем UNIX могут быть установлены на компьютеры с разными архитектурами. Например, FreeBSD устанавливается на компьютеры с архитектурой Intel (i386), а также PC98, Alpha. Linux в зависимости от диалекта может устанавливаться на i386, Alpha и многие другие.

В книге мы будем рассматривать установку только FreeBSD и Linux на компьютеры с архитектурой i386. Часто архитектуру Intel называют i386: так назывался первый процессор фирмы Intel, который поддерживал защищенный режим доступа к оперативной памяти. На компакт-диск с дистрибутивом UNIX бывают записаны файлы, необходимые для установки системы на компьютеры с разной архитектурой.

Поэтому на таком диске могут быть каталоги i386, alpha и т. п. Название архитектуры i386 не означает, что системе нужен именно процессор i386, а Pentium IV не годится. Все процессоры Intel совместимы между собой снизу вверх (более мощные умеют выполнять программы, написанные для менее мощных). FreeBSD и Linux будут одинаково хорошо работать на процессорах i386, i486 и на любом процессоре Pentium. Как правило, эти системы отлично работают и с клонами процессоров Intel (например, Cyrix и AMD), но иногда ядро системы требует дополнительной настройки для оптимизации работы с этими процессорами.

Выбор конфигурации компьютера

Перед установкой операционной системы UNIX необходимо определить, компьютер какой конфигурации вам нужен. Это зависит от функ-

ций, которые он будет выполнять. Если нам нужен маршрутизатор, главная функция которого – пересылать пакеты между локальной сетью и медленным внешним соединением с Интернетом, то надо позаботиться о надежных сетевых картах и модеме, а объем жесткого диска и оперативной памяти не имеет значения. Если задача будущего компьютера – работать в качестве веб-сервера с высокой загрузкой (например, вы договорились о синхронном переводе и размещении у себя ленты новостей CNN), то надо позаботиться о высокоскоростном канале (нужна быстрая и надежная сетевая карта), быстром процессоре и большой оперативной памяти (для одновременной обработки нескольких тысяч сетевых запросов).

Однозначных рекомендаций в отношении идеальной конфигурации компьютера под UNIX дать нельзя. Надо лишь иметь в виду, что:

- UNIX отлично работает даже на медленных процессорах, и если ваши задачи не требуют высокой производительности, старого компьютера с любым процессором начиная с i386 будет достаточно.
- Если вы не планируете запускать на UNIX графические программы (для обычного сервера это не нужно), то 32 Мбайт оперативной памяти будет достаточно для большинства систем с умеренной нагрузкой (например, единицы одновременных обращений к веб-серверу при запущенных почтовом сервере и сервере DNS).
- Для размещения необходимых системных файлов требуется не менее 64 Мбайт дискового пространства, а для нормальной работы – не менее 300 Мбайт. Стало быть, годится любой современный жесткий диск, так как обычно его объем в несколько сотен раз больше. Однако если вы планируете использовать UNIX-сервер как хранилище файлов или почтовых ящиков или для размещения на нем большой базы данных, то размер жесткого диска надо определять, исходя из запросов самой требовательной к дисковому пространству функции компьютера.
- Если нужен компьютер с высокой скоростью доступа к данным на диске (сервер баз данных, сильно нагруженный веб-сервер), подумайте об установке нескольких SCSI-дисков, пусть и сравнительно небольшого объема. Распараллеливание нагрузки увеличивает скорость работы.
- Модель и объем памяти видеокарты не играют никакой роли для сервера, так как с ним обычно работают в текстовом режиме или через сеть, и в последнем случае видеосистема сервера просто не используется – он прекрасно обойдется вообще без видеокарты.
- Если вы устанавливаете графическую станцию под UNIX и думаете о качественном графическом изображении, обязательно приобретайте видеокарту производителя, известного своей лояльностью к стандартам, ибо UNIX не любит нестандартных карт. Для работы с графикой UNIX требует большого объема оперативной памяти (не менее 64 Мбайт, а лучше 128 Мбайт). Не жалейте денег на каче-

ственную видеокарту с большим объемом памяти и поддержкой высокой кадровой частоты, а также на хороший монитор.

Последнее требование касается только ваших глаз, поскольку качество монитора UNIX-машине безразлично.

Помните о сетевых картах

В отличие от других операционных систем, дистрибутив UNIX имеет весьма ограниченный набор драйверов сетевых карт. *Перед* тем как ставить систему, пожалуйста, уточните, какая сетевая карта установлена в вашем компьютере. Если не можете это установить точно, поставьте другую сетевую карту, название и параметры которой будут точно известны. Обязательно выясните, заглянув на сайт производителя вашей версии UNIX, поддерживается ли такая сетевая карта в этой версии. Если нет, лучше смените карту или найдите ту версию UNIX, которая ее поддерживает.

Администраторы иногда устанавливают UNIX без поддержки сетевой карты в надежде, что они потом добавят такую поддержку каким-нибудь образом. Этот «какой-нибудь образ» обычно отнимает столько времени, что к середине процесса настройки администратор продолжает его лишь из упрямства. Я не раз убеждался, что существенно проще и дешевле поставить в компьютер абсолютно стандартную карту типа NE2000 или SMC9432, чем заставить UNIX дружить с какой-нибудь потрясающе хорошей, но нестандартной сетевой картой.

Поэтому *до* установки UNIX необходимо установить в компьютер как можно более стандартную и надежную сетевую карту и точно знать ее параметры (например, IRQ и порт, если карта установлена в слот ISA).

Некоторые сетевые карты поставляются с дискетами, содержащими драйверы для UNIX. Прежде чем ставить такую карту, убедитесь, что на дискете есть драйвер именно для вашего диалекта системы и вашей версии. Предположение, что годится драйвер для любой системы UNIX, неверно.

Собираясь установить драйвер сетевой карты с дискеты, имейте в виду, что во время установки не будет возможности[1] сказать: «А вот тут на дискете еще драйвер лежит, его тоже поставить надо». Поэтому вы сначала поставите систему, а потом будете модифицировать ее ядро, добавляя туда драйвер. Это нетрудно, просто утройте время, которое вы отвели в своем ежедневнике на установку системы.

---

[1] На самом деле все зависит от диалекта UNIX, версии системы и программы установки. Поэтому готовьтесь к худшему, чтобы лучшее стало приятным сюрпризом.

Конфигурация операционной системы

UNIX состоит из ядра, системных утилит и стандартных сетевых приложений. Последние обеспечивают реализацию базовых сетевых служб, таких как веб-сервер, почтовый сервер, telnet- и ftp-серверы. Сетевые приложения и другие системные программы, обслуживающие запросы других программ, в UNIX называются *демонами* (*daemons*).

В компьютерном сообществе ведется дискуссия о происхождении этого термина. На одном из православных форумов я даже встретил объяснение защитника UNIX'овских демонов. Он писал, что «нехороший, демонический» демон – это demon, а «демон» в UNIX'е зовется словом daemon, что читается как «даймон» или «дееймон», и по-английски означает «дух» в широком смысле этого слова. Так что неправы православные, отказывающиеся использовать UNIX из-за того, что в нем системные службы называются «демонами».

В разных источниках указывается, что слово «daemon» прикрепилось к системным программам потому, что они работают независимо от пользователя, олицетворяя собой некий «дух системы».

С другой стороны, словарь компьютерного жаргона (*http://developer.syndetic.org/query_jargon.pl?term=daemon*) рассказывает другую легенду, никак не связанную с важной ролью демонов в системе. Утверждается, что слово daemon было впервые употреблено разработчиками UNIX в отношении сервисных программ, используемых системой. Более того, одна из первых программ в UNIX, выполнявших резервное копирование на ленту, называлась DAEMON.

Чтобы не нарушать сложившиеся традиции перевода, мы будем употреблять термин «демон». Если же среди читателей есть те, кому неприятно это слово, пусть они убедят издательство продавать pdf-версию книги – в ней можно будет сделать контекстную замену на «даймона», более соответствующего духу UNIX.

В разных диалектах UNIX используются разные ядра. Для каждой аппаратной платформы требуется свое собственное ядро. При установке системы на жесткий диск обязательно копируется ядро системы и основные системные утилиты и приложения. Какие дополнительные приложения и утилиты будут скопированы, зависит от того, какой вариант установки системы выбран.

Основные утилиты для работы с файлами, дисками, процессами и сетью устанавливаются всегда. Имейте в виду, что для нормальной работы под UNIX помимо основных утилит понадобятся компилятор языка C, программа **`make`** и демон **`inetd`**. Должны быть установлены системные библиотеки (каталоги `/lib`, `/usr/lib` не должны быть пусты). К сожалению, ваше мнение о том, какие вам нужны программы, может не совпадать с мнением поставщика дистрибутива.

Устанавливая UNIX в первый раз, не задумывайтесь, какие именно дополнительные пакеты выбрать. Для начала будет достаточно минимального варианта любой системы. К счастью, большинство программ довольно легко установить в UNIX в любой момент. Если же вы точно знаете, какие программы вам нужны, не забудьте отметить соответствующие флажки в программе-установщике. Сделать это вручную легче, чем выбрать какой-нибудь «стандартный» вариант от поставщика системы и обнаружить после установки, что лично вам «стандарт» представлялся совершенно иным.

Впрочем, надо признать практически идеальными с этой точки зрения установщики FreeBSD и Mandrake Linux, а при определенном везении вы сможете сказать такое и про установщик Solaris.

Чтобы обдуманно выбрать все программы, которые надлежит установить одновременно с системой, заранее составьте список на листе бумаги. Налейте чашечку кофе, устройтесь в кресле поудобнее и спланируйте, как вы будете использовать компьютер. Будет ли он выступать в качестве почтового сервера, потребуется ли хранить на нем файлы многих пользователей, будет ли он сервером печати, поставите ли вы на него СУБД и веб-сервер. Запишите, какие программы необходимы для того, чтобы новый компьютер под UNIX сыграл свою роль так, как вы этого хотите. Этот листок вам пригодится, когда вы будете отвечать на вопросы программы установки.

Коммерческие системы (Solaris и другие) могут поставляться без компилятора языка C. Для того чтобы работать в дальнейшем было удобнее, стоит скачать из Интернета бесплатный компилятор для вашего диалекта UNIX. Следите за тем, чтобы компилятор был предназначен именно для вашей аппаратной платформы.

Большинство современных систем можно устанавливать с компакт-диска, с дискет, через сеть (по ftp или через NFS) или с жесткого диска. Коммерческие системы, особенно для старого оборудования, часто устанавливаются с магнитной ленты. На одиночный компьютер удобнее всего устанавливать UNIX с загрузочного компакт-диска. Устанавливая систему на несколько компьютеров в сети, удобнее переписать на один из них дистрибутив, а на остальные компьютеры устанавливать систему через сеть.

Разделы дисков в UNIX

В компьютере с архитектурой i386 на любом жестком диске может быть не более четырех разделов. Один из них может быть расширенным (extended), другие – первичными (primary). Расширенный раздел может быть разбит на несколько дополнительных логических дисков, первичный содержит всего один логический диск.

В UNIX нет понятия «логический диск», из-за чего возникает терминологическая путаница. То, что в других системах называют логическим диском, в UNIX называют разделом. Поэтому мы будем говорить «раздел UNIX» и «раздел диска» там, где из контекста не ясно, какой именно раздел имеется в виду.

В BSD-системах принята иная схема: при установке среди разделов жесткого диска надо выбрать тот, что будет считаться разделом FreeBSD. Этот раздел надо будет поделить на подразделы (slices), которым будут сопоставлены разделы UNIX.

Каждый раздел UNIX должен иметь *точку монтирования* – каталог из общего дерева каталогов файловой системы. Подробнее это будет рассмотрено в главе 5. Основными каталогами (и точками монтирования) в UNIX обычно являются каталоги `/` (*корневой каталог*), `/var` и `/usr`. В Linux также важен каталог `/boot`, в котором по умолчанию размещается ядро системы. По соображениям удобства администрирования эти каталоги размещаются на отдельных разделах. Это, однако, не влияет на их полные имена (то есть каталог `/var` всегда называется `/var` независимо от того, размещен ли он на отдельном разделе или на одном разделе с родительским каталогом).

Имеет смысл при установке создать четыре раздела UNIX: корневой, swap-раздел, разделы `/var` и `/usr`. Размеры этих разделов для «стандартного UNIX-сервера» составят 64 Мбайт, 96 Мбайт, 256 Мбайт и все остальное пространство на жестком диске соответственно.

Таким образом, после разметки жесткого диска надо создать разделы UNIX на выбранных для установки UNIX разделах жесткого диска. Кроме этого, надо выбрать один раздел UNIX, предназначенный для свопинга. Для всех разделов UNIX, кроме swap-раздела, необходимо назначить точки монтирования.

Linux, в отличие от некоторых других диалектов UNIX, поддерживает свопинг (swapping) в файл, а не на отдельный раздел. Все же рекомендуется для свопинга всегда назначать именно раздел. Некоторые системы не будут корректно установлены, если забыть это сделать. Подробнее о назначении свопинга рассказано в главе 6.

При установке UNIX существуют ограничения на разбиение диска. Следует убедиться, что корневой раздел располагается до 1024-й дорожки – для того чтобы BIOS могла загрузить с него ядро. Это ограничение BIOS компьютера с архитектурой i386, а не UNIX. Для диска SCSI это обычно означает, что корневой раздел располагается в первых 1024 Мбайт диска. Для дисков IDE это означает, что корневой раздел размещается в первых 504 Мбайт.

Ниже будет рассмотрена установка UNIX на компьютер с аппаратной платформой i386 на примере FreeBSD 4.9 и RedHat Linux 7.2.

Русификация

Русификация UNIX – очень непростая тема. Во-первых, потому что в большинстве версий UNIX нет «сквозной» русификации всех программ в системе одновременно. Во-вторых, русификация графической среды выполняется отдельно от русификации других программ. Будет конкретная программа понимать символы кириллицы или нет, зависит от того, как написана эта программа.

Те, кому нужна полностью русифицированная система, могут обратиться к Black Cat Linux. Насколько нам известно, это наиболее полно русифицированная версия UNIX, в которой собраны одинаковым образом русифицированные программы: *http://www.inp.nsk.su/~baldin/Cyrillic-HOWTO-russian/Cyrillic-HOWTO-russian.html*. В последние два года также широко распространяется ALT-Linux (*http://www.altlinux.ru*) – дистрибутив, собранный российскими разработчиками.

Некоторые вопросы русификации освещены в главе 26. Русификация X Window кратко описана в главе 25.

При установке ОС UNIX имеет смысл стремиться к русификации системы, только если работа с символами кириллицы в системе действительно необходима. Большинство стандартных приложений при использовании UNIX-машины в качестве сервера (веб-сервера, почтового сервера и т. п.) не требует никаких настроек, касающихся локализации системы.

Если же локализация (русификация) при установке все-таки важна, посетите соответствующий пункт меню программы-установщика (что-нибудь вроде Language или Language support). RedHat Linux прямо спрашивает, нужна ли вам поддержка национального языка.

Процедура установки

Способы установки UNIX могут быть разными для разных систем. Многие коммерческие UNIX-системы поставляются предустановленными, вместе с оборудованием, и тем, кто покупает такую машину, думать об установке ОС не придется. В целом есть две идеологии установки UNIX. Условно назовем одну Windows-установкой, другую – Linux-установкой.

Первая предполагает определенный уровень привычного комфорта администратора. Он может при установке указывать ее параметры, используя псевдографический или графический интерфейс. Все параметры устанавливаются в одной и той же программе, после установки на диске находится работоспособная система.

Вторая идеология предполагает участие администратора в операциях копирования файлов из дистрибутива или из Интернета и их раскладывания по каталогам. Никакого графического интерфейса при уста-

новке системы здесь нет и в помине. После процедуры установки на диске находится начальный загрузчик и рабочее ядро, все необходимые программы приходится устанавливать отдельно.

Обе идеологии имеют свои плюсы. Первая удобнее, вторая дает больший контроль над процессом установки. Первая принята во всех UNIX, кроме некоторых диалектов Linux, например Slackware. Вторая распространена меньше, но имеет своих поклонников. Это не значит, что Slackware можно установить только вторым способом. Просто второй способ в Slackware тоже возможен.

Здесь мы разберем два варианта установки, следующие первой идеологии. Вторая не будет рассмотрена потому, что если вы устанавливаете UNIX впервые, начать лучше с простого. Если вы UNIX-гуру и тысячи раз устанавливали разные системы, то и без книги отлично знаете, как устанавливать систему.

Следите за регистром букв

В UNIX повсюду – в именах пользователей и файлов, названиях команд, ответах на вопросы программ – важен регистр букв. Если требуется ввести `y` или `n`, проявите внимательность: введите именно то, что просят, а не `Y` или `N`. Если во время установки системы проигнорировать регистр букв, то результат может оказаться неожиданным. Поставив систему, помните об этом правиле, в противном случае вряд ли можно предсказать, как система поведет себя в ответ на ваши команды.

Установка FreeBSD

Рассмотрим процесс установки FreeBSD с загрузочного компакт-диска.

После загрузки с него в память копируется стандартное ядро системы, управление передается ядру, затем происходит монтирование временной файловой системы в памяти компьютера и управление передается программе-установщику `sysinstall`. Впоследствии эту же программу можно использовать для установки дополнительных программ и настройки системы. Вначале следует выбрать тип интерфейса установки. Наиболее удобен полноэкранный режим (full-screen visual mode).[1] Затем `sysinstall` предложит изменить список устройств, поддерживаемых ядром системы, или поменять их параметры. Если в компьютере есть устройства с нестандартными прерываниями или присоединенные к нестандартным портам (например, модем на COM2 с IRQ5), обязательно измените параметры соответствующих устройств.

Удалять поддержку отсутствующих устройств при установке не обязательно. Она увеличивает размер ядра, больше ни на что не влияя. Оптимизировать ядро по размеру можно позже – когда система будет уста-

[1] Во FreeBSD 5.x остался только он – вопросов установщик не задаст.

новлена, все устройства будут работать, а также будут установлены все необходимые программы.

Изменив при необходимости параметры конфигурации устройств, вы переходите к разметке жесткого диска, на который будет установлена система.

В этот момент `sysinstall` предложит текстовое полноэкранное меню, позволяющее изменить параметры установки системы.

Для разметки жесткого диска надо выбрать пункт Partitions. Вначале на диске выбираются или создаются разделы жесткого диска, предназначенные для установки FreeBSD. Затем они делятся на подразделы (slices). Не забудьте указать, какой раздел будет активным (с него будет загружаться компьютер!). Если указать раздел неправильно или не указать вовсе, BIOS компьютера не будет знать, откуда загружать систему, и начнет пугать вас жуткими сообщениями после перезагрузки.

После этого надо перейти к пункту меню Label и назначить точки монтирования для этих подразделов.

После назначения точек монтирования следует выбрать, куда установить начальный загрузчик (boot manager). Начальный загрузчик – это программа, которая найдет на диске ядро системы, загрузит его в память и передаст ему управление. Начальный загрузчик предоставляет возможность выбрать загружаемую систему. На одном компьютере может быть установлено несколько операционных систем и boot manager может загрузить любую по вашему выбору. Каждый диалект UNIX имеет свой собственный начальный загрузчик, который устанавливается в MBR (и затирает старый загрузчик, например от Windows NT, если таковой был записан в MBR ранее). Начальный загрузчик некоторых диалектов UNIX (например, FreeBSD) может быть записан в загрузочный сектор раздела жесткого диска, а не в MBR. При выборе места установки начального загрузчика надо выбрать пункт boot-sector, если MBR нельзя изменять. Наконец, можно выбрать вариант leave MBR untouched, что позволит вообще не устанавливать начальный загрузчик. Этот вариант хорош только в том случае, если вы собираетесь загружать установленную на жесткий диск систему с другого носителя (например, с дискеты). Обычно это не практикуется.

Затем надо в пункте Options включить протоколирование установки системы, изменив значение параметра Debug на YES. Если это сделать, то на втором виртуальном терминале в процессе установки можно будет наблюдать за ее ходом. Переход к виртуальному терминалу происходит по нажатию клавиш <Alt>+<Fn>, например ко второму терминалу – <Alt>+<F2>. Подробнее о виртуальных терминалах рассказано в главе 3.

Теперь следует определить набор компонентов системы для установки (Distribution). Можно выбрать каждый из пакетов программ для установки индивидуально, но это требует немало времени. В случае с FreeBSD

разработчики программы установки системы постарались и подготовили продуманные готовые наборы компонентов для установки.

Наиболее универсальным вариантом является набор Kern-Developer. При его выборе программа установки скопирует все основные части системы. Кроме них копируются исходные тексты ядра, все библиотеки и заголовочные файлы (`*.h`), необходимые для компиляции программ. В результате вы получите систему, которая после установки будет готова к любой работе, а при необходимости сможете модифицировать ее ядро, не устанавливая ничего более с дистрибутива.

После определения набора компонентов остается только указать, откуда будет производиться установка (с компакт-диска, с другого компьютера в сети, с жесткого диска, с раздела FAT). Выбрав установку с удаленного компьютера, будьте готовы тут же ввести параметры для настройки вашей сетевой карты (IP-адрес и др.).

Программа установки спросит, надо ли устанавливать ports collection. Подробнее об этой коллекции рассказывается в главе 13. Если вы не уверены в том, что коллекция вам нужна, *не ставьте* ее! По умолчанию предлагается вариант Yes, выберите No перед нажатием клавиши <Enter>.

У людей с древних пор развивался хватательный рефлекс. Надо было быстро хватать добычу, пока не убежала. Или пока кто другой не схватил. Сейчас первобытный инстинкт сублимируется в мгновенное нажатие клавиши <Enter> при виде любого вопроса или кнопки на экране. Вдруг убежит, «Enter'ом» ее скорее! Самые стойкие системные администраторы удерживаются от того, чтобы «давить» – они сначала читают, что написано. Самые опытные из них даже стараются понять смысл прочитанного.

Если случайно выбрать установку ports collection, процедура удлинится вдвое за счет копирования 70 Мбайт информации, которой вы вряд ли воспользуетесь.

Золотое правило номер один

НИКОГДА не устанавливайте программу, назначение которой вам непонятно или полезность которой неочевидна.

Для начала копирования файлов дистрибутива на диск следует выбрать пункт Commit. В этот момент все определенные ранее разделы будут физически созданы на диске. Если на нем была какая-то информация, которую предполагалось уничтожить, в этот момент она уничтожится. Если вы хотели ее сохранить, надеюсь, вы сделали ее резервную копию.

Теперь начинается установка системы. Компьютер Pentium-100 с медленным диском и 16 Мбайт памяти справится с установкой минут за двадцать, Pentium-III с быстрыми дисками – минут за пять.

После установки надо согласиться выполнить постустановочную настройку (post-install configuration). В новом меню следует выбрать как минимум установку root password и указание параметров сетевых интерфейсов – IP-адреса, маски сети и т. п. (Networking>Interfaces). Обратите внимание, что выбор пункта делается клавишей <Space>, а не <Enter>. Если вы не знаете, что такое IP-адрес, обратитесь к главе 10. Настройку других параметров выполните по необходимости. Например, в разделе post-install configuration можно указать, какие демоны должны запускаться при старте. Часто такие настройки делаются позже, когда система уже установлена.

По окончании постустановочной настройки вам останется только перезагрузиться, удалить компакт-диск (или дискету, если компьютер загружался с дискеты) из компьютера и ждать загрузки свежеустановленной системы.

Установка RedHat Linux

Рассмотрим процесс установки RedHat Linux с загрузочного компакт-диска. Программа установки RedHat Linux начиная с версии 6.1 может работать в графическом режиме. Установка из дистрибутива RedHat 7.2 в этом режиме достаточно удобна, в отличие от первой версии графической программы установки из версии 6.1. С ростом номера версии процесс установки становился все более похожим на установку Windows, так что к версии 9, последней бесплатной версии Red Hat, стал совсем привычным тем, кто по нескольку раз в месяц переустанавливал Windows-системы. Существенной разницы в том, какие именно действия потребуется выполнить, устанавливая версию 7.2 и 9.0, вы, скорее всего, не почувствуете.

Мы будем говорить об установке RedHat Linux 7.2 в графическом режиме. После загрузки ядра и передачи управления программе установки **anaconda** последняя предлагает выбрать режим установки системы: графический или текстовый. Нажмите клавишу <F2>, если не знаете, что лучше выбрать и что нажимать для выбора режимов.

Затем надо сообщить программе установки, какие у вас мышь и клавиатура. По умолчанию программа сама определяет их тип (обычно правильно). Затем следует выбрать тип установки. Чтобы иметь больший контроль над происходящим, выбираем Custom (Ручная установка).

Выберите язык установки (например, русский). Имейте в виду, что даже если в качестве языка установки выбран русский, подсказки частично даются на английском языке, поскольку переведены они не полностью. По умолчанию в дальнейшем основным языком системы будет считаться тот, который выбран языком установки. Не забудьте потом добавить поддержку английского языка!

Разбиение дисков надо выполнять вручную, потому что только вы знаете, как именно будет использоваться компьютер. Обычно для это-

го применяют утилиту fdisk, которая вызывается из программы-установщика. В Linux есть альтернатива – программа Disk Druid, которой мы рекомендуем воспользоваться.

Сделайте побольше (от 256 Мбайт до нескольких гигабайт) раздел `/var`, если планируете отвести UNIX-машине роль почтового сервера или http cache. Не забудьте о swap-разделе, он должен быть не меньше 64 Мбайт. Часто размер swap-раздела делают равным удвоенному объему оперативной памяти. Однако не стоит злоупотреблять этой формулой. Если память компьютера превышает сотни мегабайт, стоит ограничиться для swap-раздела объемом 96 – 512 Мбайт. Если в старых версиях некоторых систем Linux сделать swap-раздел слишком большим (256 Мбайт и более), производительность системы снижается. Всегда предпочтительнее иметь 256 Мбайт оперативной памяти и swap-раздел размером 96 Мбайт, чем наоборот. Однако в современных версиях снижения производительности из-за слишком большого размера swap-раздела не наблюдалось.

После разметки диска требуется указать, какой начальный загрузчик вы хотите установить. Можно установить GRUB (только в некоторых Linux) или Lilo (стандартный загрузчик, используемый в любой версии Linux).

Часто возникает вопрос о совместной установке нескольких операционных систем на один жесткий диск. В мире есть миллионы людей, успешно сделавших это. Следуйте за ними только на свой страх и риск. Для построения устойчивой системы или сети не обязательно, чтобы один компьютер в разное время выполнял совершенно разные функции, требующие установки разных систем. Исключением являются компьютеры разработчиков, тестирующих свои программы для нескольких систем. Но и в этом случае предпочтительнее иметь несколько компьютеров, каждый со своей системой (так работа не только упростится, но и пойдет быстрее).

Выбрав начальный загрузчик, определите конфигурацию сети. Потребуется задать IP-адрес компьютера, на который ставится UNIX, маску сети, шлюз (gateway) и IP-адрес DNS-сервера. Можно установить различные уровни секретности при работе с сетью (такая установка специфична только для RedHat Linux). В общем случае имеет смысл выбрать средний уровень, установить фильтр пакетов (firewall) и указать eth0 (интерфейс сетевой карты) в качестве доверенного устройства (proxy).

Перевод текста в окне помощи в программе установки RedHat 7.2 неплох, но не идеален, например «check the checkbox» означает не «проверить», а «установить флажок».

После этих установок выбирается язык системы (поддержка английского и русского достаточна для большинства случаев) и часовой пояс.

Затем выполняется настройка системы аутентификации. Вы придумаете пароль администратора и сможете добавить в систему новых пользователей. В RedHat 7.2 пароли шифруются алгоритмом MD5, что позволяет иметь пароль длиной до 256 символов.

Затем выбираются пакеты программ для установки. Базовая установка для сервера (без поддержки графики) занимает 330 Мбайт, с графикой и графической средой GNOME – 805 Мбайт. Чтобы гарантировать установку всех необходимых пакетов программ, отметьте их вручную. Стандартные наборы («Сервер» и т. п.), предлагаемые разработчиками системы, укомплектованы довольно странно. Компилятор языка C (**gcc** или **cc**) надо ставить самому.

Программа **inetd** (демон, среди прочего управляющий запуском telnet- и ftp-серверов) в RedHat 7.2 (и в некоторых других Linux) называется **xinetd**.

Теперь все готово для копирования файлов на диск. Не забудьте прочесть сообщение о том, куда будут записаны протоколы (logs) установки системы, и нажмите <Enter>. На пятом виртуальном терминале (<Alt>+<F5>) будет виден ход установки. Для нее потребуется немного больше времени, чем для установки FreeBSD.

После установки в минимальной комплектации система займет 77 Мбайт в корневом каталоге, 203 Мбайт на разделе `/usr` и 14 Мбайт на `/var`.

Выбор поддержки русского языка в системе не означает, что после установки система будет полностью русифицирована. Подробно настройки русификации UNIX рассмотрены в главе 25.

3

Начало работы

Включение и выключение компьютера

Что происходит после нажатия на кнопку компьютера «Вкл»?[1] Начинается загрузка – сначала BIOS, а затем операционной системы. При загрузке UNIX сначала загружается ядро, затем все остальное. Подробно этот процесс описан в главе 12.

На всех терминалах после загрузки будут светиться приглашения `login:`.

Во время загрузки будут запущены сетевые и системные приложения, такие как почтовый сервер, сервер баз данных, веб-сервер.

Перед выключением компьютера под управлением UNIX надо дать команду **`shutdown`**. В разных системах эта команда имеет разные ключи и совершает разные действия по умолчанию. Часто **`shutdown`** без параметров выдает на все активные терминалы, подключенные к системе, сообщение о завершении работы, ждет одну минуту, затем останавливает систему. Детальную информацию о работе **`shutdown`** в данной системе можно получить, дав команду **`man shutdown`**. В Linux и FreeBSD есть команда **`halt`**, которая вызывает немедленный останов системы без выдачи предупреждений на терминалы. Перезагрузку системы можно выполнить командой **`reboot`**.

Выключать UNIX простым отключением питания без предварительного предупреждения (в виде **`shutdown`** или **`halt`**) нежелательно, так как данные, еще не записанные на диск и хранящиеся в оперативной

---

[1] Вообще-то там написано «ON» или «power», но вы же догадались, что такое «Вкл»?

памяти, будут потеряны. Иногда это приводит к ошибкам в файловой системе на диске и может потребовать их исправления программой **fsck** (аналог scandisk в Windows).

Вход в систему и выход из системы

Для входа в систему необходимо набрать имя пользователя (login) и пароль (password). Если пароля у пользователя нет, система его и не спросит. Для нормальной работы системы не требуется, чтобы кто-либо входил в нее, все системные и сетевые приложения запускаются независимо от того, работают ли в системе пользователи.

После входа в систему для пользователя запускается программа – интерпретатор команд. Эта программа называется командным процессором (shell). В среде этой программы проходит весь сеанс работы пользователя с системой. В случае если пользователь входит через окно графического терминала, он будет работать в среде программы-менеджера окон. Попробуйте это сделать, вы же не так часто входите куда-нибудь через окно.

Мы не будем говорить здесь о том, как надо работать в графической среде, поскольку нажать большую кнопку с надписью Logoff для выхода из системы вы догадаетесь и без чужих советов. Практически все, что вы прочтете ниже, относится к работе с UNIX в текстовом режиме.

Для выхода из системы надо дать команду **logout**. Кроме этого, можно просто нажать <Ctrl>+<D> (большинство командных процессоров воспримут это как сигнал к окончанию работы) или дать команду **exit** (это эквивалентно <Ctrl>+<D>). Если в процессе работы было запущено несколько командных процессоров, то давать команду **exit** или нажимать клавиши <Ctrl>+<D> придется до тех пор, пока вы не доберетесь до самого первого, который запустился при входе в систему.

Режимы работы системы

UNIX может работать в одном из двух режимов: однопользовательском (single-user mode) или многопользовательском (multi-user mode).

Обычно система загружается для работы в многопользовательском режиме. В нем пользователи могут входить в систему локально или через сеть. В этом режиме несколько пользователей могут одновременно работать в системе. Их количество ограничивается размером таблиц ядра и, как правило, не должно превышать 256.

Однопользовательский режим применяется системным администратором для настройки, резервного копирования или ремонта системы (например, администратор запускает программы **dump** – резервное копирование или **fsck** – проверку дисков). Для загрузки системы в однопользовательском режиме следует дать соответствующую команду про-

грамме-загрузчику. В разных диалектах UNIX загрузчики и команды разные. Во FreeBSD, например, в ответ на приглашение `boot:` надо сказать **`boot -s`** (просто **`-s`** во FreeBSD версии младше 3.2).

Для перехода из многопользовательского режима в однопользовательский в системах System V следует дать команду **`init s`** или **`init 1`** (зависит от системы, детали описаны в man init). Во FreeBSD для этого используется **`shutdown now`** без ключей или **`kill -TERM 1`**.

Для того чтобы перейти из однопользовательского режима в многопользовательский, надо перезагрузить компьютер (годится для любого диалекта системы) или выйти из однопользовательского командного процессора shell (**`exit`** или <Ctrl>+<D>, работает в большинстве случаев).

Вообще говоря, существует несколько «уровней выполнения» (runlevels) операционной системы. Уровень с номером 1 обычно соответствует однопользовательскому режиму, а 2 или 3 – многопользовательскому. Уровень номер 0 – это останов (на этот уровень систему переводит команда **`shutdown`**). Остальные номера обычно зарезервированы для особых конфигураций. Об уровнях выполнения в вашем диалекте UNIX легко узнать подробнее при помощи команды **`man init`**. Дополнительную информацию об уровнях выполнения можно найти в главе 12.

Понятие терминала

Терминал – это экран и клавиатура, с помощью которых пользователь дистанционно связывается с компьютером. Терминал нужен только для связи с компьютером – для ввода информации и ее отображения на экране. Терминалы бывают физическими (еще их называют реальными), виртуальными и псевдотерминалами (то есть программами, которые притворяются терминалами).

Физический терминал – это устройство, состоящее из экрана и клавиатуры, обычно подключенное к компьютеру через последовательный интерфейс. Характер соединения может быть разным – через кабель или по модему. Пользователь, который соединяется с UNIX-машиной при помощи эмулятора терминала через модем своего компьютера, позвонив на другой модем, подключенный непосредственно к UNIX-машине, с точки зрения UNIX работает на физическом терминале.

Основа виртуального терминала – это монитор и клавиатура, непосредственно подключенные к компьютеру. Однако, хотя монитор всего один, виртуальных терминалов несколько. Вы можете переключаться между ними, нажимая клавиши <Alt>+<F*n*>, где *n* – номер терминала. С монитором и клавиатурой связаны несколько независимых виртуальных терминалов. При работе в графическом режиме переключение между виртуальными терминалами выполняется посредством нажатия клавиш <Ctrl>+<Alt>+<F*n*>.

Виртуальные терминалы используются в UNIX на платформе i386, потому что терминалы, подключаемые к компьютерам на других платформах, обычно работают в графическом режиме. В этом случае можно просто открыть несколько окон, в каждом из которых запустить терминальную программу.

Монитор и клавиатура, подключенные непосредственно к UNIX, называются консолью. Применительно к виртуальным терминалам консолью является первый виртуальный терминал (<Alt>+<F1>).

Псевдотерминал – это программа-эмулятор терминала, которая соединяется с UNIX-компьютером через сеть.

Терминалы бывают графические и текстовые. Графические терминалы могут работать в графическом режиме. Графическим может быть физический, виртуальный или псевдотерминал. Эмулятор графического терминала базируется на программе вывода графики, которая называется X-Server. Подробнее об X-server'ах и о системе X Window рассказывается в главе 24.

Текстовые терминалы могут отображать только текст и псевдографику. Будем рассматривать приемы работы с UNIX в предположении, что работаем с текстовым терминалом.

Существует много типов текстовых терминалов. Наиболее известны ANSI и VT-100. Терминалы отличаются друг от друга управляющими последовательностями. Например, чтобы дать терминалу VT-100 команду «поднять курсор на одну строку вверх», надо передать последовательность `<Esc>[A`. Описания терминалов (и соответствующих управляющих последовательностей) содержатся в файле `/etc/termcap` и базе данных в подкаталогах `/usr/share/terminfo`.

Переменная среды окружения `TERM` должна иметь значение типа терминала, на котором вы работаете. Возможно, ее придется переопределить, если в начале работы пользователя система неверно определила тип его терминала. Фактический тип терминала может отличаться от принятого в системе по умолчанию. Переопределение выполняется путем присваивания переменной соответствующего значения:

```
TERM='VT100'; export TERM
```

Значение переменной **TERM** можно вывести на экран командой

```
echo $TERM
```

Если переменная `TERM` определена неверно, терминал будет вести себя непредсказуемо. Например, при нажатии клавиши <Enter> не произойдет перевода строки или при нажатии стрелки вверх на экране появится `[[A`, а курсор вверх не сдвинется. Некоторые терминалы практически совместимы между собой, например ANSI и VT-100. Однако некоторые управляющие последовательности могут не совпадать. Поэтому лучше, чтобы в `TERM` был записан тип именно того терминала, на котором вы работаете.

Управляющие комбинации клавиш

Терминал обычно воспринимает и передает активной программе коды следующих управляющих комбинаций клавиш, которые любая программа должна интерпретировать стандартным образом (табл. 3.1).

Таблица 3.1. Управляющие комбинации клавиш

Клавиши	Действие
<Ctrl>+<C>	прерывание выполнения программы (termination)
<Ctrl>+<D>	конец ввода текста
<Ctrl>+<S>	остановка вывода на экран
<Ctrl>+<Q>	продолжение вывода на экран
<Ctrl>+<U>	стирание введенной строки
<Ctrl>+<Z>	остановка выполнения программы (suspend)
<Backspace>	удаление последнего введенного символа

Некоторые терминалы не воспринимают клавишу <Backspace> стандартным образом, вместо нее на таких терминалах можно использовать клавишу <Delete>.

В старых системах клавиша <Delete> могла применяться вместо <Ctrl>+<C> для прерывания работы программы.

Настройка терминала: команда stty

Проверить, как настроен терминал, можно с помощью программы **stty**:

```
stty -a
```

Для настройки правильной реакции на стандартные комбинации клавиш (см. табл. 3.1) надо выполнить команды

```
stty susp '^z'
stty intr '^c'
stty erase '^h'
```

и т. д. Список параметров установки (`susp`, `intr`, `erase` и др.) можно увидеть по команде **stty -a** или в руководстве (**man stty**).

Чтение документации. Организация man-страниц

Невозможно ни изучать, ни пользоваться UNIX без руководства по системе. Печатное руководство поставляется только с коммерческими системами, да и то не всегда. Кроме того, печатные руководства имеют скверную привычку теряться. Потом они отыскиваются, но только не в тот момент, когда нужны вам позарез.

К счастью, любой диалект UNIX устанавливается вместе с электронной копией документации. Эта документация состоит из нескольких разделов (обычно их семь или восемь, иногда больше). Каждый раздел содержит статьи. Статья посвящена одной команде, файлу, понятию или системному вызову UNIX. Любая статья имеет свой заголовок, в котором есть имя команды (файла, вызова и т. п.) и ее очень краткое описание, например:

```
cp - copy files and directories
```

Руководство вызывается командой **man**. Программа **man** форматирует страницу руководства и показывает ее на экране с помощью программы форматирования **nroff** и программы поэкранной выдачи текста **more** (или ее аналога **page**, **pg**, **less**). На диске страницы руководства хранятся в неформатированном виде, так как это экономит место на диске. Страницы руководства хранятся в каталогах /usr/share/man/manN (N – номер раздела) или /usr/man/manN, или в каком-нибудь похожем подкаталоге /usr. Однажды отформатированные страницы **man** складывает в подкаталоги catN. Иногда они расположены там же, где и manN, иногда – нет. Программу **man** можно запустить с ключом **-M**, чтобы указать, в каком каталоге искать страницы руководства. Иногда при установке некоторых программ руководства по ним размещаются вне стандартного дерева каталогов **man**, тогда этот ключ может пригодиться.

Ссылки на man-страницы в литературе обычно включают номер раздела, в котором расположена статья, например, **inetd.conf(5)** означает статью о конфигурационном файле демона **inetd** в пятом разделе руководства. В этой книге мы тоже будем так ссылаться на статьи руководства по системе.

В табл. 3.2 перечислены разделы руководства, к которым обращается **man**. В BSD и System V руководства организованы немного по-разному.

Таблица 3.2. Содержание разделов man в System V и в BSD

Раздел	System V	BSD
1	команды	
2	системные вызовы	
3	библиотечные несистемные вызовы (например, из библиотеки Perl)	
4	драйверы устройств, файлы и протоколы	
5	разные документы	файлы конфигурации
6	игры и демонстрационные программы	
7	драйверы устройств и сетевые протоколы	тексты, кодировки, выражения
8		утилиты системного администрирования

В Linux руководство организовано по типу BSD.

Чтобы получить справку по любой команде UNIX или даже понятию, наберите

man *то,_что_вас_интересует*

Набирайте **man -k** *тема* для поиска темы в заголовках статей руководства.

Команда **man -k** предполагает, что существует индексированный список заголовков статей руководства. Этот список создается командой **catman** или каким-нибудь ее аналогом. Синоним **man -k** – команда **apropos**. Поиск информации о программах выполняется еще и командой **whatis**.

Некоторые команды в UNIX – это встроенные команды командного процессора, а не отдельные программы. Например, **alias** и **jobs**. Поэтому, если вы не можете найти справку по команде, попробуйте почитать man по командному процессору, с которым вы работаете (скорее всего, **sh** или **bash**).

Кроме **man** документация содержится в гипертекстовых файлах формата info и файлах howto. Файлы info следует читать программой **info**. Можно ее запустить без параметров и затем дать команду **?** для знакомства с системой встроенных команд **info**. Она отличается от **man** своей «гипертекстовостью»: можно переходить по ссылкам как внутри документа, так и между документами.

Howto – это набор документации, распространенный в Linux. В документах howto разъясняются многие действия, которые могут понадобиться при администрировании. Howto распространяется в формате html или в виде текстовых файлов. В отличие от man, не существует жесткой структуры howto. Это просто набор документов, и на одном сайте он может оказаться более полным, чем на другом.

В Интернете есть несколько сайтов, на которых размещен перевод на русский язык man и howto в формате html. Перевод иногда хорош, иногда дурен. Если вы не в состоянии понять, что написано в переводе документации, остается в сердцах ругать переводчика и обращаться к оригиналу.

Где хранятся настройки ОС?

Большинство файлов конфигурации UNIX расположены в каталоге `/etc`. Практически все файлы конфигурации представляют собой обычные текстовые файлы. Довольно часто их содержание вполне сносно откомментировано. Все строки, которые в файле конфигурации начинаются со знака # (решетка), представляют собой комментарии. Исключение составляют файлы описания доменов для сервера

имен **named**, в которых признаком комментария служит символ ; (точка с запятой). Формат файлов конфигурации бывает разным и зависит от того, чей это файл конфигурации.

Файлы конфигурации пользовательских приложений, особенно тех, что инсталлированы после установки системы без компиляции из исходных текстов, могут находиться не в `/etc`, а в каталогах типа `/usr/local/etc` или внутри дерева каталогов, куда установлен пакет программ (`/usr/local/apache/conf`).

Файлы конфигурации можно редактировать в обычном текстовом редакторе. Некоторые файлы автоматически сгенерированы системой или какой-нибудь утилитой на основании других; в начале таких файлов обычно содержится предупреждение, что для реального изменения конфигурации надо редактировать не этот файл, а источник, на основании которого файл конфигурации создан.

В приложении «Краткий справочник по командам UNIX» вы тоже найдете несколько указаний на каталоги, в которых различные системы UNIX хранят настройки.

4

Пользователи в системе UNIX

Пользователи и группы

Каждый пользователь UNIX имеет уникальное имя, которое он должен сообщить системе при входе. Имя должно состоять из латинских символов нижнего регистра и цифр. Не допускается, чтобы имя начиналось со знака «-» (дефис). В имени не должно быть точек и букв верхнего регистра; это может свести с ума почтовые программы. Имя может иметь длину до шестнадцати символов. В некоторых ОС UNIX, например в Solaris, буквы верхнего регистра в имени допустимы, но имя всегда должно начинаться с буквы нижнего регистра.

Причины такого ограничения кроются в глубине истории UNIX. Известно, что были люди, желавшие изменить эту традицию в угоду эстетике. Однако этот краеугольный камень устоял. Теоретически можно изменить значение константы `UT_NAMESIZE` в файле `/usr/include/utmp.h` и перекомпилировать всю систему и все приложения в ней при условии, что вы располагаете полным комплектом ее исходных текстов. Так, впрочем, можно навредить, поскольку потеряется совместимость с другими UNIX-системами. В NIS (Network Information Service) – службе, предоставляющей возможность использовать одни и те же учетные записи на разных компьютерах, – длина имени ограничена еще сильнее – 8 символами. Однако в Linux имя может быть длиной до 32 символов. Если пользователь просит сделать имя подлиннее, чтобы у него был красивый почтовый адрес вида *Ivan.Ivanovich.Ivanov@mail.com*, просто создайте для его имени vanya почтовый псевдоним в файле `/etc/mail/aliases`.

Имя пользователя назначает системный администратор, когда вносит новую информацию в список пользователей.

В руководствах имя пользователя часто называют «логин» (login) или «username».

При входе в систему пользователь должен набрать свое имя в ответ на приглашение `login:`. Затем в ответ на `password:` следует набрать пароль.

Пароль назначает системный администратор при добавлении пользователя в систему. В некоторых UNIX программа добавления пользователя может сама придумать пароль, легкий для запоминания и сложный для отгадывания.

Пользователь или системный администратор могут в любой момент изменить пароль с помощью команды **passwd** *имя_пользователя*. Системный администратор может изменить пароль любого пользователя, все остальные – только собственный.

Любой пользователь может изменить свой собственный пароль. Для этого достаточно дать команду

passwd

Команда **passwd** без параметров меняет пароль того пользователя, который ее запустил. При изменении своего пароля надо набрать старый пароль, а затем дважды новый пароль, чтобы система была уверена в вашем выборе.

Пароль может содержать любые символы. В зависимости от уровня секретности, принятого в системе, допустимая минимальная длина пароля может быть от 1 до 6 символов. Редко допускается иметь пустой пароль. Системный администратор, которому дорого его дело, никогда не будет устанавливать пустой пароль себе и запретит это делать пользователям. Максимальная длина пароля обычно равна 8 символам. В тех новых системах, где пароли шифруются алгоритмом MD5, допускаются пароли длиной до 256 символов. Символы разных регистров в паролях, как и повсюду в UNIX, считаются различными.

Пароль не должен легко отгадываться. Многие системы при назначении пароля сверяются со словарем широко известных паролей. Похожим словарем пользуются программы взломщиков систем при подборе пароля. Если придуманный пароль содержится в этом словаре или слишком прост (короток, содержит только однородные символы), система предложит выбрать другой пароль. Лучше последовать совету. Помните: взлом вашего пароля означает не только доступ взломщика ко всей вашей информации в системе, но и облегчение взлома всей системы от вашего имени. Вы хотите, чтобы именно ваше имя сообщили шефу, когда тот поинтересуется, из-за какого разини взломщик умыкнул секретную информацию из базы данных?

Идеальный пароль в UNIX состоит из восьми символов, среди которых есть латинские буквы разных регистров, цифры и знаки препинания.

Не гордитесь своей изобретательностью, набирая на клавиатуре пароль латинскими буквами, соответствующими русскому слову, – эти комбинации и комбинации последовательных клавиш типа qwerty давно попали в словари взломщиков.

Так же как в секретном институте нельзя одолжить чужой пропуск на денек, пользователям нельзя делиться с коллегами своим паролем. Строго запрещено писать пароль на бумажке и приклеивать ее к столу или на монитор.

Если под одним именем и паролем работает несколько человек, нельзя определить, кто конкретно совершил в системе действие от этого имени.

Например, из финансового отчета стерли пару строк, от которых зависит решение совета директоров. Доступ к отчету имел всего один пользователь, но под его именем работала толпа людей. Кто виноват?

Чтобы было удобнее администрировать систему, пользователей объединяют в группы. Дело в том, что часто права доступа намного удобнее назначать группе, чем конкретным пользователям. Например, над проектом работают Иван, Миша и Галя. Им необходим доступ к данным, с которыми работает каждый из них. Можно назначить права доступа каждому из них персонально. Представьте, что Миша уволился, и работу над проектом вместо него продолжает Лена. Придется аннулировать доступ Миши ко всем файлам проекта, а Лене такой доступ предоставить. Намного быстрее заранее назначить права доступа к нужным файлам группе, а когда Мишу сменит Лена, исключить Мишу из группы, а Лену включить в нее.

Поэтому концепция прав доступа в UNIX требует объединять пользователей в группы всегда, когда следует предоставить одинаковые права доступа к файлам или каталогам группе людей.

Список пользователей системы хранится в файле `/etc/passwd`, список групп – в `/etc/group`.

Учетная запись пользователя

Имя и пароль пользователя – это часть учетной информации, которая содержится в файле `/etc/passwd`. В этом файле содержатся учетные записи (accounts) пользователей. Каждая учетная запись – это одна строка, состоящая из полей, разделенных двоеточиями:

```
ivan:*:1001:100:Ivan Ivanov-room127:/home/ivan:/bin/bash
```

1001 в нашем примере – это идентификатор пользователя – UID (User IDentifier), а 100 – идентификатор его главной группы (login group) – GID (Group IDentifier).

В `/etc/passwd` за именем пользователя следует звездочка. В старых системах в этом поле находился зашифрованный пароль. Однако `/etc/passwd` доступен для чтения любому пользователю. Это сделано для

того, чтобы все могли идентифицировать хозяина любого файла или каталога в системе. Файловая система хранит только идентификатор хозяина файла, но не его имя. Если о файле известно, что он принадлежит пользователю 1001, много ли нам это скажет? Увидев, что им владеет ivan, вы поймете, кто из ваших коллег хозяин этого файла. В некоторых новых системах зашифрованные пароли хранятся вдали от /etc/passwd. Обычно пароли содержатся в файлах /etc/shadow или /etc/master.passwd (последнее характерно для BSD-систем).

За идентификатором группы следует полное имя пользователя – произвольное текстовое поле. Оно может включать реальное имя пользователя (Иван Иванов), его рабочий телефон и т. п. После полного имени идет домашний каталог пользователя (home directory), а затем программа-оболочка, запускаемая для пользователя при входе в систему (login shell).

В каждый момент времени какой-то каталог файловой системы UNIX является текущим для пользователя. При входе пользователь попадает в свой домашний каталог. Из него он может перейти в любой другой при условии, что обладает требуемыми правами доступа к новому каталогу.

Программа-оболочка может быть настоящим командным процессором, способным выполнять команды пользователя в интерактивном сеансе работы. Пользователям, которым не следует работать в системе интерактивно, например, тем, кто лишь получает почту с UNIX-машины, назначают псевдооболочки (/bin/date, /nonexistent и т. п.). Если login shell пользователя не настоящий командный процессор, а такая псевдооболочка, то при попытке входа этого пользователя она запустится и тут же завершится. Интерактивный сеанс, в котором пользователь мог бы ввести команды UNIX, даже не начнется. Если в качестве login shell указана несуществующая программа, пользователь также не сможет войти в систему.

Для всех пользователей, которым не надо работать интерактивно, а требуется лишь получать почту, скачивать файлы по ftp, размещать на вашем UNIX-компьютере свои веб-сайты и т. п., надлежит устанавливать в качестве login shell что-нибудь типа /bin/date.

Предопределенные пользователи и группы

При установке системы в файлы /etc/passwd и /etc/group записывается информация о предопределенных пользователях и группах. Это необходимо для того, чтобы было проще управлять правами доступа к системным файлам.

Предопределенные пользователи и группы имеют «говорящие» имена. Интуитивно ясно, что от имени пользователя mail будет работать почтовый сервер, а пользователю bin принадлежат многие системные утилиты.

В разных системах могут быть разные предопределенные пользователи и группы. Некоторые программы (например, **mysqld**, **proftpd**) требуют, чтобы их запускали от имени определенного пользователя.

Во многих системах есть пользователь nobody, от имени которого запускают веб-сервер. Дело в том, что веб-сервер взломать легче, чем многие другие программы. Пользователь nobody обычно имеет минимальные права в системе. Взломщик же при успешном взломе программы обычно получает доступ в систему с правами того пользователя, от имени которого запущена взломанная программа.

Предопределенные пользователи и группы имеют не только заранее заданные имена, но и совершенно определенные идентификаторы (UID). Избегайте удаления и переопределения UID этих пользователей и групп! Удалив их или назначив им другие UID, вы внесете путаницу в права доступа в системе. Путаница может откликнуться неудобством администрирования или уязвимостью системы.

Привилегированный пользователь

Один из предопределенных пользователей – это администратор системы с именем root и UID, равным нулю.

Пользователь с таким UID называется суперпользователем (superuser) или привилегированным пользователем и всегда имеет имя root. Переопределять это имя не надо: некоторые скрипты, предназначенные для установки программ, написаны в расчете на то, что пользователь с идентификатором 0 – root. Этот пользователь имеет неограниченные права на доступ к любому объекту в системе.

Тот, кто владеет паролем суперпользователя, должен хорошо знать базовые процедуры администрирования UNIX и работать в системе осторожно, чтобы не навредить ей. Системный администратор отвечает за безопасность системы и ее правильную настройку, добавление и исключение пользователей, регулярное резервное копирование файлов и т. п. Он должен хранить пароль суперпользователя как зеницу ока и доверять его только надежному персоналу.

Вход в систему под именем root разрешен только с тех терминалов, которые присоединены непосредственно к UNIX-машине. Это умолчание, которое изменить можно, но по соображениям безопасности делать этого не следует. Root никогда не должен иметь пустой пароль.

Как стать привилегированным пользователем

В UNIX можно перевоплотиться в любого пользователя. Для этого служит команда **su** (switch user):

```
su ivan
```

Команда **su** без параметров эквивалентна **su root**.

Для перевоплощения в другого пользователя надо знать его пароль. А для того чтобы стать пользователем root, помимо этого необходимо еще и быть достойным такого высокого звания. В некоторых диалектах UNIX этот статус означает принадлежность к определенной группе (например, wheel для FreeBSD). В других системах достаточно лишь знать пароль пользователя root.

После «перевоплощения» вы будете работать от имени другого пользователя. Пользователю root для того, чтобы выполнить команду **su user** и начать работать от имени любого пользователя, пароль не нужен.

Команда **su** имеет ключи, определяющие, насколько полным будет «перевоплощение».

В некоторых системах есть программа **sudo**, с помощью которой любой пользователь может выполнить команду от имени другого пользователя. Системный администратор должен заранее отредактировать файл /etc/sudoers, в котором определяется, кто и что может запускать от чужого имени. При выполнении программа **sudo** спросит у пользователя его пароль, чтобы убедиться, что ее запускает не тот, кто случайно подошел к терминалу с незакрытой сессией работы. **Sudo** будет регулярно переспрашивать пароль (по умолчанию – раз в пять минут), чтобы не быть обманутой. Эта программа позволяет системному администратору делегировать часть своих полномочий в системе доверенным пользователям. Например, при администрировании веб-сайта редактирование файла конфигурации веб-сервера можно поручить вебмастеру.

Что делать, если вы забыли пароль суперпользователя

Если вы забыли пароль обыкновенного пользователя, достаточно сменить этот пароль командой **passwd**, зайдя в систему от имени суперпользователя (root). Но как быть, если забыт пароль его самого?

Без паники! В разных версиях UNIX эта проблема решается по-разному. Но в любом случае надо загрузить систему в однопользовательском режиме и изменить пароль root. Во FreeBSD следует, как только появится приглашение boot: при загрузке компьютера, набрать **boot -s** (для версий младше 3.2 просто **-s**). Это можно сделать, только работая у консоли! Не забывайте этот пароль вдали от UNIX-компьютера.

Вы попадете в среду командного интерпретатора, работающего от имени суперпользователя (да-да, без пароля!). Затем сделайте следующее:

```
mount -u /    # перемонтировать раздел / в режиме чтения-записи
mount -a      # смонтировать остальные разделы
passwd root   # изменить пароль суперпользователя
exit          # выйти из однопользовательского режима в обычный
```

В Linux такой простой способ не поможет, здесь потребуется загрузиться с дискеты, смонтировать файловую систему жесткого диска в какой-нибудь каталог (например, /mnt) и текстовым редактором удалить из файла с учетными записями (в нашем примере – /mnt/etc/passwd) все символы из поля пароля в строке пользователя root. В других диалектах UNIX эта задача решается схожим образом.

Управление учетными записями

Перед первым входом пользователя в систему системный администратор должен внести имя нового пользователя в файл /etc/passwd. Добавить пользователя можно командой **adduser** (во FreeBSD и многих других системах) или **useradd** (в Linux). В разных диалектах UNIX синтаксис и ключи **adduser** и **useradd** отличаются. Перед добавлением пользователей в новой для себя системе стоит почитать man по **adduser** или **useradd** соответственно.

Для удаления пользователя предназначена команда **userdel** или **rmuser**, для изменения информации о нем – **usermod**, **chpass**.

Редактирование файла /etc/passwd

Администратор может прибегнуть к специальным системным утилитам для модификации учетных записей пользователей (типа **usermod**), а может просто отредактировать **/etc/passwd** в текстовом редакторе. Для этого отлично подходит редактор **vi**. Затем надо синхронизировать файл /etc/passwd с /etc/shadow (/etc/master.passwd во FreeBSD). Для этого следует дать команду **pwconv**. Можно сразу воспользоваться командой **vipw** для редактирования – при выходе из редактора файлы будут синхронизированы автоматически.

По договору между издательством «Символ-Плюс» и Интернет-магазином «Books.Ru-Книги России» единственный легальный способ получения данного файла с книгой ISBN 5-93286-084-7 «UNIX. Практическое пособие администратора. 2-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» (www.symbol.ru), где именно Вы получили данный файл.

5

Файловая система UNIX

Термином «файловая система» в литературе обозначаются три разных понятия:

1. Файловая система – это набор правил и конструкций, описывающих способ сохранения файлов на диске. В этом смысле мы употребляем, например выражение «файловая система FAT 32», и «файловая система» здесь тождественна понятию «тип файловой системы».
2. Файловая система – это совокупность всех файлов, хранимых в компьютере.
3. Файловая система – это совокупность всех файлов на разделе диска или устройстве (данное значение термина характерно именно для UNIX-систем).

В этой книге мы будем говорить о файловых системах во всех трех смыслах, и там, где из контекста неясно, что имеется в виду, будем отдельно оговаривать, в каком смысле употребляется это выражение.

В разных диалектах UNIX поддерживается множество разных типов файловых систем. Однако все они имеют схожую структуру. Файловая система UNIX независимо от типа содержит четыре основных компонента с управляющей информацией: загрузочный блок, суперблок, таблицу индексных дескрипторов (i-node table) и каталоги. Более подробно о типах файловых систем можно узнать из раздела «Типы файловых систем».

В System V каждый раздел жесткого диска форматируется (размечается) для размещения на нем файловой системы UNIX. В BSD один раздел жесткого диска разбивается на подразделы, каждый из которых форматируется. Мы называем «разделом UNIX» такой размеченный

в формате файловой системы UNIX раздел (System V) или подраздел (BSD).

Загрузочный блок (*boot block*) – это один или несколько секторов в начале раздела диска. В загрузочном блоке записана *метка диска* (*disk label*) и маленькая программа, которая при старте системы загружает ядро ОС с диска в оперативную память. Загрузочный блок имеет смысл только для первого раздела жесткого диска, однако место для него резервируется в каждом разделе.

Суперблок (*superblock*) содержит общую информацию о файловой системе как совокупности файлов на данном разделе жесткого диска, в частности размер раздела UNIX, количество свободных и занятых блоков и индексных дескрипторов и флаг целостности файловой системы. Этот флаг устанавливается при успешном завершении работы с файловой системой, например при корректной остановке операционной системы. В случае если компьютер выключили рубильником, не дождавшись корректной остановки системы, при следующем старте системы программа **fsck** автоматически начнет проверку дисков и выдаст сообщение «clean flag is not set in superblock». Копии суперблока для большей надежности многократно записываются в нескольких несмежных блоках внутри каждого раздела. В файловой системе UFS записывается по одной копии суперблока на каждую группу цилиндров. Если выяснится, что оригинал суперблока в начале раздела поврежден, будет использована копия суперблока.

Таблица индексных дескрипторов (*i-node table*) хранит дескрипторы файлов. Дескриптор файла содержит сведения о типе файла, размещении файла на диске, правах доступа к нему, UID владельца файла, GID группы файла, время последнего доступа к файлу, время последней модификации файла, время последней модификации самого индексного дескриптора. Подробнее структура таблицы индексных дескрипторов рассмотрена в разделе «Таблица индексных дескрипторов: детали».

Размер таблицы индексных дескрипторов фиксирован и задается при создании файловой системы на разделе. Программе **mkfs** (используется для создания новой файловой системы, в некоторых диалектах – **newfs**) можно явно указать требуемое количество индексных дескрипторов на разделе.

Файлы в UNIX разложены по каталогам. Каталоги образуют древовидную структуру: есть корневой каталог, которому предшествует знак «/» (слэш), и его подкаталоги. В каждом из последних есть свои подкаталоги и т. д. Ограничений на количество файлов в каталоге нет. Разные каталоги («ветви» дерева каталогов) могут размещаться на разных дисках, это незаметно для пользователя. Чтобы обратиться к файлу, не надо знать, на каком физическом устройстве или разделе диска записан файл. Это означает, что лихорадочный поиск по дискам `C:`, `D:`, `E:`, `K:`, `R:`, `Y:` незнаком человеку, работающему с UNIX. Он ли-

хорадочно ищет потерянные файлы в густой кроне UNIX-каталогов, начиная от корня.

Между прочим, забавно, что уже много лет все упорно называют деревом структуру каталогов, у которой корень находится наверху, а его ближайшие отростки-подкаталоги называются каталогами верхнего уровня... Кто из нас видел деревья, растущие кронами вниз?

Имена файлов и каталогов

Имена файлов и каталогов в UNIX могут состоять из латинских букв верхнего и нижнего регистра, цифр и знаков препинания. Лучше всего из последних использовать только точку «.», дефис «-», подчеркивание «_».

Теоретически возможно присутствие в именах файлов и других знаков (скобок, запятых, слэшей, вопросительного и восклицательного знаков, звездочек и т. п.), но это неудобно. Дело в том, что обработчики команд в UNIX интерпретируют такие знаки специальным образом (например, как шаблоны имен файлов или модификаторы команд shell, подробнее об этом говорится в главе 8). Обращаться к файлу с именем `,/?*&^q-` придется тоже специальным образом, а это быстро надоест.

Буквы верхнего и нижнего регистра в UNIX считаются разными символами, имена «Advert» и «advert» – это разные имена, хотя и отличаются всего одной буквой.

В UNIX нет понятия «расширение имени файла», точка считается равноправным символом в имени, поэтому имя «advert.of.company.on.Monday.morning» смотрится в UNIX не более экзотично, чем «data.txt». UNIX не делает предположений о назначении или содержимом файла в зависимости от того, какое у него имя и какие символы стоят справа от самой правой точки в имени.

Сделать заключение о содержимом файла, не открывая его, можно с помощью программы **file**:

```
file data.txt
English text
```

Программа **file** использует специальный файл образцов `/etc/magic`, содержащий информацию о том, как должен выглядеть файл, чтобы его можно было отнести к какому-либо известному типу: англоязычный текст (English text), текст программы на языке C (C program), исполняемый код (executable file), данные (data) и т. д.

В разных диалектах UNIX действуют разные ограничения длины имени файла. В старых системах (например, HP-UX 8.0) длина ограничивалась 14 символами. В большинстве новых UNIX длина имени ограничена 255 символами. Впрочем, такое имя нелегко читать в списке файлов каталога. Право, удобнее именовать файлы так, чтобы имя уме-

щалось в одной строке экрана, укладываясь в прокрустово ложе восьмидесяти символов.

Имя может быть полным (абсолютным) или относительным. Полное имя файла – это имя с указанием пути к файлу от корневого каталога, например `/usr/local/apache/conf/httpd.conf`. Полное имя легко узнать: оно всегда начинается с символа «/». Относительное имя файла может быть очень коротким, например просто `f`. Если в имени файла вообще нет знака «/» (слэш), то имя относится к файлу текущего каталога. Если слэш есть (но не в начале имени, например `src/support/mod_info.c`), то все, что расположено слева от первого в имени слэша, расценивается как имя подкаталога текущего каталога.

Полное имя любого файла не должно быть длиннее 1023 символов. Однако это ограничение можно обойти. Для того чтобы обратиться к файлу, спрятанному глубоко в структуре каталогов, следует перейти в промежуточный каталог командой **cd** и оттуда обратиться к файлу по относительному имени, например, вместо

```
/usr/local/content/french/Libertй/Йgalitй/fraternitй/show.pl
```

можно обратиться к тому же файлу так:

```
cd /usr/local/content/french/Libertй/Йgalitй/fraternitй/
show.pl
```

Имя файла иногда называют «путем файла», или «путевым именем файла»,[1] или «путем к файлу».

Символ «~» (тильда) обозначает домашний каталог пользователя: `~anna/` – домашний каталог пользователя anna, ~ без имени означает домашний каталог текущего пользователя.

Основные каталоги

В большинстве диалектов UNIX названия и назначения каталогов верхнего уровня стандартизованы. Вы можете записывать в эти предопределенные каталоги свои собственные рабочие файлы, но это считается дурным тоном. Как плевок в пепельницу на деловой встрече.

Назначение основных каталогов верхнего уровня поясняется в табл. 5.1.

UNIX развивается, и вместе с ним развивается структура каталогов. Так, в последнее время в Linux стало традиционным размещать деревья каталогов больших приложений (СУБД Oracle и т. п.) в каталоге `/opt`. Например, в `/opt/oracle` могут располагаться все подкаталоги, необходимые СУБД для работы.

---

[1] Называть-то его так называют, но лучше бы этого не делали: от «путевого» и до «непутевого» недалеко. – *Примеч. ред.*

Таблица 5.1. Основные каталоги файловой системы UNIX

Каталог	Содержимое	Примечание
/bin	утилиты типа **cp**	
/sbin	системные утилиты типа **fsck**	
/etc	файлы конфигурации и стартовые скрипты	в System V – не просто стартовые скипты, а целые каталоги со стартовыми скриптами
/dev	файлы устройств	
/usr	каталоги пользователей, приложения, установленные отдельно	
/home	каталоги пользователей	часто упрятан в /usr и называется /usr/home
/var	временная информация, которую жаль стирать сразу	почта пользователей, которую они заберут с клиентских машин, или файлы протоколов, временные файлы подсистемы печати
/tmp	временные файлы, которые не жаль	обычно очищается при перезагрузке
/usr/local	место для приложений, установленных отдельно от системы: apache, informix и т. д.	
/usr/bin	место для несистемных утилит	
/usr/sbin	место для системных утилит, установленных отдельно от системы	все, что «установлено отдельно», представляет собой нужные и полезные приложения, которые вы, скорее всего, скачали из Интернета и установили на ваш компьютер несколькими днями (месяцами) позже самой ОС

Файлы и каталоги в UNIX

Типы файлов

Тип файла указывается первым символом первой колонки в выводе команды

```
ls -l
```

Файлы бывают такими (табл. 5.2):

Таблица 5.2. Типы файлов UNIX

Обозначение	Тип	Пояснение
-	обычный файл (regular file)	самый обычный файл
D	каталог (directory)	содержит список файлов, как и полагается добропорядочному каталогу
L	символическая ссылка (symbolic link, soft link)	ярлык (shortcut), то есть ссылка по имени; подробнее о ссылках рассказано в этой главе в разделах «Жесткие ссылки», «Символические ссылки»
C	файл символьного устройства	символьное устройство, например терминал
B	файл блочного устройства	блочное устройство, например жесткий диск
P	канал (pipe)	специальный файл, создается для организации канала обмена данными между процессами, используется в системах System V
S	сокет (socket)	выполняет ту же функцию, что и канал (pipe), но в BSD

Файлы

Обычный файл может содержать текст или данные. Пустой файл можно создать, например, командой **touch**:

```
touch filename
```

Его можно копировать командой **cp**:

```
cp file1 file2
```

Переименование файла выполняется при помощи команды **mv**:

```
mv file1 file2
```

Командой **mv** файл можно переместить в другой каталог:

```
mv file /usr/home/useful/
```

При копировании или переносе файла или нескольких файлов в каталог имя последнего лучше завершать слэшем (например, */usr/home/useful/*). Тогда программа **mv** или **cp** будет знать, что подразумевается именно каталог. В этом случае, если вы ошиблись и последний аргумент команды оказался не каталогом, а файлом, программа сообщит об этом. Если слэш не указать, то в случае ошибки старое содержимое файла, в который что-то копируется, будет утеряно.

Файл можно удалить командой **rm**. В UNIX не предусмотрена возможность восстановления файлов (undelete), поэтому все, что стерто, пропадает навсегда. В UNIX нет «мусорной корзины» (trash), в которую файлы попадают перед окончательным удалением.

В некоторых графических оболочках, созданных для UNIX, корзину реализовали. Однако возможность восстановления стертых файлов предоставляется не операционной системой, а графической оболочкой, и поэтому лучше не рассчитывайте, что всегда будете работать с такой оболочкой.

Вывести файл на экран можно командами **cat**, **more**, **less**, **pg**, **page**.

Только **cat** и **more** есть в любом диалекте UNIX, остальные команды представляют собой варианты команды **more** и присутствуют не всегда. Назначение **more** и ее коллег – поэкранный вывод длинных файлов. При выводе информации программа **more** останавливается, заполнив экран содержимым очередной страницы файла и ждет команды. По нажатию <Enter> обычно выводится следующая строка, по нажатию <Space> – следующая страница. Можно искать подстроку в тексте, дав команду

```
/подстрока
```

Клавишами <Ctrl>+<B>, <Ctrl>+<F> производится перемещение назад и вперед на страницу, клавишей <q> – выход из программы.

Каталоги

Термин «каталог» – это синоним термина «директория» (directory). В руководстве по системам Windows в том же значении употребляется слово «папка» (folder). В русскоязычной литературе по UNIX принят термин «каталог».

Каталог физически представляет собой таблицу из двух столбцов, в первом из которых записывается имя файла, а во втором – номер индексного дескриптора, описывающего файл. В одном каталоге не может быть двух файлов с одинаковыми именами.

Каталог содержит файлы и подкаталоги. Каталог можно создать командой **mkdir**. Пустой каталог можно удалить командой **rmdir**, непустой каталог удаляется командой

```
rm -rf каталог
```

например, по команде

```
rm -rf /usr/local/apache/
```

будет удален каталог /usr/local/apache, все файлы в нем и все его подкаталоги.

Каталог может быть переименован или перенесен командой **mv**, для копирования каталогов вместе с подкаталогами применяется команда **cp -Rp**.

Список файлов в каталоге можно вывести командой **ls**. Без параметров она выдает только список имен файлов. Команда имеет множество ключей, из которых более всего полезны приведенные в табл. 5.3.

Таблица 5.3. Некоторые ключи программы ls

Ключ	Назначение
l (long)	вывести полную информацию о файле
a (all)	вывести все файлы, в том числе те, чьи имена начинаются с точки
i (i-nodes)	вывести номера индексных дескрипторов
t (time)	отсортировать файлы по времени последней модификации

При создании любого каталога в нем всегда появляются две записи – «.» (точка) и «..» (две точки). «Точка» ссылается на сам текущий каталог, а «две точки» – на родительский каталог. В корневом каталоге обе эти записи указывают на сам корневой каталог.

По умолчанию программа **ls** выводит информацию обо всех файлах, за исключением тех, имена которых начинаются с символа «.» (точка). При указании ключа **-a** эти файлы тоже присоединяются к общему списку.

Перейти в каталог можно по команде

```
cd каталог
```

Команда **cd** без аргументов вызывает переход в домашний каталог текущего пользователя и эквивалентна **cd ~**.

Вывести на экран полное имя текущего каталога (если вы заблудились) можно командой **pwd**.

Ссылки

Ссылки в UNIX бывают жесткими и символическими.

Жесткие ссылки

Жесткой ссылкой на существующий файл называется запись в каталоге, которая ссылается на индексный дескриптор этого файла:

```
ls -li
-rwxrwrxrwx  2 root root qq 1567
-rwxrwrxrwx  2 root root qw 1567
```

Обратите внимание, что в каталоге есть два файла с разными именами, но с одинаковыми номерами индексных дескрипторов. Число во вто-

ром столбце показывает количество жестких ссылок на этот файл. Физически файл один, так как местоположение содержимого файла на диске определяется индексным дескриптором. Просто можно обратиться к этому файлу, называя его разными именами. Фактически после создания жесткой ссылки на файл определить, какое из имен было придумано раньше, невозможно. Права доступа ко всем жестким ссылкам на файл одинаковы, поскольку определяются одним и тем же индексным дескриптором.

Смысл жесткой ссылки в том, что можно поместить в разные каталоги записи об одном и том же файле, не копируя его многократно во все каталоги, где нужна запись о нем. При модификации файла независимо от того, через какую именно жесткую ссылку к нему обратились, изменяется информация в самом файле, и, соответственно, при обращении к нему через другую жесткую ссылку отображается последняя, обновленная версия.

Жесткие ссылки создаются командой **ln**:

```
ln старое_имя новое_имя
```

При создании жесткой ссылки сам файл не модифицируется, старое имя никак не изменяется, просто появляется еще одно имя этого файла.

Нельзя создать жесткую ссылку на каталог или на файл, который располагается на другом разделе UNIX. Дело в том, что таблица индексных дескрипторов в каждом разделе UNIX своя, а указать в каталоге индексный дескриптор чужого раздела невозможно.

Файл в UNIX считается удаленным только тогда, когда удалены все жесткие ссылки на него. Удаление жесткой ссылки выполняется командой **rm** и внешне ничем не отличается от удаления файла.

Найти все жесткие ссылки на файл можно с помощью команды **find**:

```
find откуда_начать_поиск -inode номер_индексного_
дескриптора_файла
```

Символические ссылки

Символической ссылкой на файл называется запись в каталоге, указывающая на файл с определенным именем.[1] Индексный дескриптор этой записи содержит только путь к файлу, на который она ссылается:

```
ls -l
lrwxrwxrwx 1 root root …. qq ->/usr/local/apache/qq
```

Можно создать символическую ссылку на любой каталог, а также на файл, находящийся в другом разделе UNIX. Символическая ссылка представляет собой аналог ярлыка (shortcut) в системах Windows. При

[1] Как правило, этот файл находится в каталоге, отличном от каталога символической ссылки.

удалении символической ссылки с файлом, на который она ссылается, ничего не происходит. А если удалить сам файл, она «повисает в воздухе», ссылаясь на пустоту. В последнем случае при обращении к такой «висячей» ссылке вы получите сообщение `file not found`, несмотря на то, что сама ссылка будет доступна и видна в списке файлов.

Обычно по команде **`ls -l`** отображается не только информация о типе файла `l`, если это символическая ссылка, но и путь к файлу, на который она указывает. В некоторых системах для вывода этой информации следует выполнить команду **`ls`** с ключом **`-F`**.

Файлы устройств

Каждое физическое устройство в UNIX имеет не только свой собственный драйвер, но и специальный файл устройства. Все файлы устройств лежат в каталоге `/dev`. Многие поставщики UNIX выделяют в отдельные подкаталоги файлы однотипных устройств: дисков, терминалов, ленточных накопителей и т. п. Рассмотрим сведения о файле устройства:

```
ls -l /dev
...
crw-------   1 root     root       4, 0  May  6  1998 tty0
...
```

tty0 – это файл первого физического терминала, то есть первого последовательного порта (COM1). Как видно из сообщения программы **`ls`**, это файл символьного устройства, право на чтение и запись в которое имеет только владелец – `root`. Никто больше не имеет прямого доступа к устройству. Вместо длины файла показаны два числа, которые называют *major*- и *minor*-номерами устройства. Эти два числа используются ядром для того, чтобы общаться с физическим устройством. В ядре хранится список всех доступных внутри ядра драйверов устройств. Major-номер указывает на нужный драйвер в ядре, minor-номер указывает на номер устройства данного типа.

Фактически с помощью major-номера вычисляется смещение в ядре, по которому расположен драйвер устройства, а minor-номер нужен для ссылки на конкретный экземпляр устройства такого типа. В нашем примере запись в каталоге `/dev` говорит о том, что к устройству tty0 надо обращаться через драйвер физического терминала, номер драйвера – 4, а номер терминала – 0.

Одни и те же устройства в разных диалектах UNIX носят разные имена. Размещение драйверов в ядрах разных диалектов и даже версий UNIX может не совпадать, поэтому никакой совместимости по major- и minor-номерам устройств нет.

Схема именований разделов дисков обычно подразумевает указание номера контроллера диска, номера жесткого диска, номера раздела.

В Solaris используется также SCSI-идентификатор устройства. В Linux применяется более простая схема именований разделов: задается только порядковый номер диска и буквой указывается раздел на нем.

Например, во FreeBSD зарезервированы следующие имена:

- /dev/ad0 – нулевой atapi (IDE) контроллер, первый жесткий диск;
- /dev/ad1 – нулевой atapi (IDE) контроллер, второй жесткий диск;
- /dev/ad2 – первый atapi (IDE) контроллер, первый жесткий диск;
- /dev/ad3 – первый atapi (IDE) контроллер, второй жесткий диск.

Полное имя файла устройства, указывающего на раздел UNIX, будет, например, таким: /dev/ad0s1a – нулевой atapi (IDE) контроллер, первый жесткий диск (ad0), первый раздел жесткого диска (s1), первый подраздел (a).

Подраздел, обозначаемый буквой «c», например /dev/ad3c, ссылается на весь жесткий диск в целом.

В Linux драйвер ATAPI устройств называется hd, а соответствующие устройства называются hda, hdb, hdc, hdd.

В Solaris первый UNIX-раздел первого SCSI-диска будет описываться файлом устройства /dev/dsk/c0t3d0s0, где c0 – номер контроллера, t3 – SCSI-идентификатор, d0s0 – disk 0, slice 0.

В некоторых системах последовательный порт для ввода и вывода информации описывается разными файлами устройств. Для ввода – /dev/tty*, для вывода – /dev/cua*.

Файлы устройств создаются при установке системы. Дополнительные файлы устройств при необходимости можно создать командой **mknod**. Для этого, правда, надо знать major- и minor-номера устройств.

Во FreeBSD для полуавтоматического создания файлов устройств применяется скрипт **MAKEDEV** из каталога /dev.

По командам

```
cd /dev
./MAKEDEV
```

будут сгенерированы недостающие файлы устройств.

В HP-UX полуавтоматическое создание файлов устройств выполняется так: при старте системы отрабатывает скрипт **/sbin/ioinitrc**,[1] который, если обнаруживает новые платы и устройства, запускает команду их инициализации, настройки и создания необходимых для них файлов устройств.

В HP-UX к созданию файлов устройств относятся команды **mksf**, **insf**, **ioscan**.

---

[1] Вызов этого скрипта содержится в /etc/inittab.

В Solaris для создания файлов новых устройств надо при загрузке компьютера выйти в boot manager и загрузиться с ключом **-r**:

```
boot -r
```

Аналогичный эффект даст создание файла `/reconfigure` и последующая перезагрузка, вот так:

```
touch /reconfigure
reboot
```

При этом все необходимые файлы устройств для вновь подключенного оборудования должны будут создаться автоматически. Этого не произойдет, если подключено устройство, незнакомое системе. Сверяйтесь с HCL перед установкой нового оборудования!

Каналы (pipes) и сокеты (sockets)

Вместо термина «сокет» в русскоязычной переводной литературе иногда употребляют малопонятный буквальный перевод слова «socket» – «гнездо». Ради благозвучности и для того чтобы исключить неуместные зоологические ассоциации, мы будем «переводить» термин «socket» калькой «сокет». Ведь привыкли же мы к слову «компьютер».

Каналы и сокеты обеспечивают взаимодействие процессов, поэтому подробно они будут рассмотрены в главе 6, в соответствующем разделе. Файлы типов `p` и `s`, представляющие каналы и сокеты, могут быть созданы в файловой системе для доступа разных процессов к именованным каналам и сокетам.

Права доступа к файлам и каталогам

Вам наверняка не раз приходилось натыкаться на таблички с надписями: «Посторонним вход запрещен!», «С собаками не входить!»

В любой многопользовательской системе системный администратор хочет развесить такие таблички по каталогам, чтобы доступ к информации имели только те, кому он разрешен.

Вся идеология безопасности в UNIX держится на правах доступа к объектам файловой системы – файлам и каталогам. Каждый файл и каталог имеет одного владельца и одну группу, которая является групповым владельцем файла. Группового владельца файла будем для краткости называть группой файла.

Дав команду **ls -l** в любом каталоге, мы увидим нечто вроде

```
-rw-rw-r--   1 filip   wheel  290 Jul 16  2001 wget.log
```

Первый столбец – это тип файла (первый символ) и права доступа (остальные десять символов). Затем следуют: количество жестких ссылок

на файл, владелец и группа файла, размер файла в байтах, дата последней модификации в формате «Месяц Число Год» и имя файла.

Существует три группы прав, применяемых соответственно к владельцу файла, группе файла и ко всем остальным (последние обозначаются термином other или world). Девять бит слова прав доступа делятся на три части по три бита:

```
rwx | rw- |r--|
 u    g     o
```

Часть, обозначенная буквой u (user), определяет права доступа владельца файла, буквой g – права группы файла, o – всех остальных. Существуют права трех типов: на чтение (r, read), запись (w, write) и запуск на выполнение (x, execute). Пользователю, который хочет обратиться к файлу, предоставляются права доступа либо владельца, либо группы файла, либо права всех остальных. Проверка прав доступа на примере слова прав доступа `rwxrw-r--` показана на рис. 5.1.

Когда пользователь обращается к файлу, система выясняет, является ли он владельцем файла. Если да, то он получает права доступа, определенные для владельца файла. Если он не владелец, но входит в группу, которая является группой файла, то он получает права доступа, определенные для группы файла. Если же он не владелец и не входит в группу, он получает права, определенные для всех остальных. Права не складываются, то есть если доступ к файлу хочет получить его владелец и он входит в группу файла, но права группы шире прав владельца (например, слово прав доступа `r-xrwxr--`), то он все равно получит права владельца (в последнем примере – `r-x`). На практике права владельца обычно шире, чем права группы.

Слово прав доступа представляет собой последовательность из двенадцати бит (активно используются младшие девять), поэтому часто за-

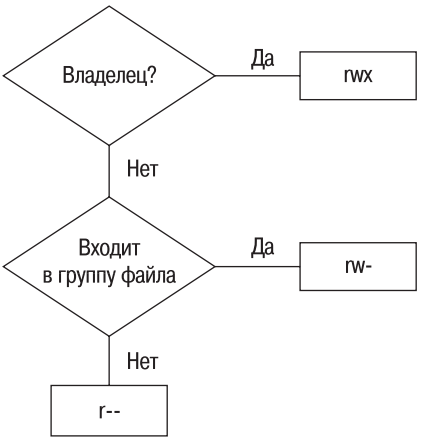


Рис. 5.1. *Алгоритм получения доступа к файлу*

писывается в виде трех десятичных чисел, представляющих права владельца, группы и всех остальных соответственно. Установленный в единицу бит говорит о наличии соответствующего права:

- мнемоническое обозначение слова прав доступа:`rwxrw-r—`
- двоичное представление слова прав доступа: `111110100`
- десятичное представление слова прав доступа:`764`

В документации по UNIX часто можно встретить указание при установке, допустим, ftp-сервера установить права доступа для такого-то каталога **755**. Имеются в виду, как можно видеть из приведенного примера, права **rwxr-xr-x**, что соответствует полному доступу владельцу и доступу на чтение и запуск группе и всем остальным.

Право запуска в отношении каталога определяется как право поиска в каталоге, что на практике означает право на чтение индексных дескрипторов файлов и подкаталогов этого каталога. Действительно, возможность узнать, какие именно файлы есть в каталоге, определяется правом на чтение каталога. А вот более подробная информация о файлах (кто их хозяин, где лежат и т. д.) доступна только тем, кто имеет право на поиск в каталоге.

Старшие три бита слова прав доступа относятся к запуску файла, поэтому детально они рассмотрены в главе 6. Вот полный вид слова прав доступа:

```
su   Sg   t   r   W   x   r   w   x   r   w   x
```

Старшие три бита: бит установки владельца при запуске файла (suid), бит установки группы (sgid) при запуске файла и бит запрета выгрузки файла на диск при выполнении (t).

Для каталога биты suid и sgid имеют другое значение. По умолчанию при создании файла он получает владельца и группу по наследству от процесса, который его создал. Аналогичное правило действует и на каталоги. Однако если каталог имеет установленный бит suid, то созданный в нем файл или подкаталог будет принадлежать не владельцу по умолчанию, а владельцу каталога с битом suid.

В некоторых диалектах UNIX это правило распространяется только на группу файла, но не на владельца. То есть владельцем нового файла будет владелец создавшего его процесса независимо от наличия бита suid у каталога, где создан файл. Но если каталог имеет установленный бит sgid, то группой нового файла станет группа процесса-создателя.

Бит запрета выгрузки файла на диск при выполнении (sticky bit) в старых версиях UNIX, будучи установленным, указывал, что при запуске программы, записанной в этом файле, ее страницы нельзя выгружать на диск. Например, это может быть нужно для программ управления технологическим процессом, чтобы не терять время на подкачку страниц с диска. В настоящее время бит запрета задает особые

условия удаления файлов из каталога. Если этот бит установлен для каталога, то все пользователи, кроме root, могут удалять из каталога только те файлы, которые им принадлежат, независимо от прав доступа к каталогу. Во FreeBSD бит запрета, установленный для файлов, игнорируется. В разных диалектах он может интерпретироваться по-разному, уточнить это можно командами `man sticky` и `man chmod`.

Расширенные права доступа к файлам и каталогам

Для более гибкого управления доступом к файлам и каталогам удобно назначать определенные права доступа не одной группе пользователей, а нескольким. Например, доступ к каталогу с финансовыми данными компании должны иметь сотрудники бухгалтерии и топ-менеджмент, к обновляемым файлам данных в этом каталоге – сотрудники отдела IT или внешней компании, выполняющей обновление программ финансового учета.

Как обеспечить доступ разного типа к одному каталогу разным группам пользователей? В рамках прежней концепции прав доступа, когда права назначались либо владельцу файла, либо группе, либо всем остальным вообще, это сделать тяжело. К тому же старая концепция мешает сделать файловый сервер на базе UNIX для рабочих станций под управлением Windows – ведь последние «привыкли» работать с правами доступа к объектам, назначаемыми различным группам, а не одной.

В современных файловых системах UNIX введены новые права доступа, дополняющие набор старых, классических. Это так называемые «списки управления доступом» (Access Control Lists, ACL), или «расширенные права доступа». Поддержка расширенных прав доступа присутствует в системах FreeBSD 5.x и новее, Linux[1] с ядром 2.4 и новее, Solaris 2.5.1 и выше и других новых системах.

Механизм расширенных прав доступа позволяет назначить права доступа отдельным пользователям, а не только владельцу файла, а также целым группам. Каждой группе можно назначить ее собственные права доступа. Это делается с помощью программы `setfacl`. Получать информацию об установленных правах доступа следует с помощью `getfacl`.

Для каталогов можно установить расширенные права доступа по умолчанию (именно они назначаются новым подкаталогам и файлам, создаваемым в таком каталоге). Установить права доступа по умолчанию можно посредством команды `setfacl -d`, а узнать, каковы они, с помощью `getfacl -d`.

---

[1] В Linux может потребоваться монтирование файловой системы с параметром `acl` для того, чтобы была включена фактическая поддержка расширенных прав доступа, например, вот так:

```
mount -o acl /dev/hda1 /home
```

На фактические права доступа к каталогу в файловой системе влияет также маска расширенных прав доступа, которая может задавать более строгие общие ограничения в доступе, чем определено индивидуальными правами доступа к файлу или каталогу.

Например, чтобы для каталога `/opt` установить права **rwx** группам itstaff и managers, а также права **rx** группе developers, и, кроме того, указать права доступа по умолчанию для новых файлов и подкаталогов в `/opt` как **rwx** для владельца – пользователя database, **rx** для группы файла (каталога) database и ничего для всех остальных, следует дать такие команды:

```
setfacl -m group:itstaff:rwx /opt
setfacl -m group:managers:rwx /opt
setfacl -m group:developers:rx /opt
setfacl -m default:user:database:rwx /opt
setfacl -m default:group:database:rx /opt
setfacl -m default:other: /opt
```

Следует обратить внимание на то, что при использовании расширенных прав доступа задействуется еще один объект – маска расширенных прав доступа. По умолчанию она равна **rwx** (это самые широкие права). Изменяя маску, легко ограничить в правах всех, кто имеет доступ к файлу, так как фактические права доступа вычисляются как наиболее строгие из заданных прав и маски. Например, установлены такие права:

```
setfacl -m group:itstaff:rwx /opt
setfacl -m mask:rwx /opt
```

Фактические права группы itstaff на каталог /opt будут установлены в значение **rx**.

Теневые индексные дескрипторы

Механизм расширенных прав доступа основан на использовании теневых индексных дескрипторов (shadow i-nodes). Они содержат информацию о специальном файле, в котором записаны расширенные права доступа (ACL) файла или каталога. Такие расширенные права доступа могут быть назначены в отношении каталогов, обычных файлов, символических ссылок и именованных каналов (FIFO).

В момент назначения файлу расширенных прав доступа (например, с помощью команды `setfacl`) создается теневой индексный дескриптор для этого файла и в обычный индексный дескриптор файла записывается ссылка на теневой индексный дескриптор. Он не имеет специального формата и представляет собой точно такой же индексный дескриптор, как и обычный, но с указанием в поле типа дескриптора на то, что он теневой.

Расширенные права доступа хранятся в специальном файле, который создается в момент назначения таких прав. Теневой дескриптор указывает на этот специальный файл. В Solaris 9 разрешено создавать не более 1024 записей в таком файле, следовательно, расширенные права доступа могут содержать не более 1024 специфических определений прав. Этого с избытком хватает для установки каких угодно специфических прав доступа.

В других UNIX могут быть другие ограничения, например, в Linux и FreeBSD максимальное количество записей ACL на файл определяется константой ACL_MAX_ENTRIES в файле `/usr/include/sys/acl.h`. По умолчанию во FreeBSD 5.x оно равно 32.

Каждая запись ACL занимает 12 байт, поэтому обычно эти записи не требуют много места на диске. Однако при назначении специфических прав доступа очень большому количеству объектов в файловой системе следует помнить о том, что для этого требуется дополнительный объем дисковой памяти.

Команды **chmod** и **setfacl** имеют влияние друг на друга, равно и изменение владельца, или группы файла, или каталога влияет на фактические права доступа к нему.

Команды chmod, chown, chgrp

Права доступа к файлу изменяются командой **chmod**, причем права доступа задаются в символьном виде:

```
chmod u+rw файл   #добавить владельцу право на чтение и запись
chmod g-x файл    #лишить группу права на запуск
chmod o=rx файл   #присвоить остальным право чтения и запуска
```

или в числовом виде:

```
chmod 755 файл    #присвоить файлу права доступа rwxr-xr-x
```

Программа **chmod** не изменяет права на символические ссылки. Впрочем, это не является недостатком, так как права символических ссылок (права доступа, определенные в дескрипторе файла, описывающем символическую ссылку) никогда не используются. Однако **chmod** изменяет права доступа к файлу, связанному с символической ссылкой, заданной в командной строке. При этом **chmod** по умолчанию игнорирует символические ссылки, если они встречаются во время рекурсивной обработки каталогов по команде **chmod -R**.

Владельца файла можно изменить командой **chown**, группу файла – командой **chgrp**. Эти команды имеют ключ **-R**, который обязывает рекурсивно изменить права доступа, владельца или группу каталога и всех его файлов и подкаталогов, если команда задана в виде

```
команда -R каталог
```

Во FreeBSD группу файла можно менять одновременно со сменой владельца командой **chown**:

```
chown владелец.группа файл
```

Команда umask

Права доступа по умолчанию назначаются системой в зависимости от маски прав доступа. Можно посмотреть или изменить текущую маску командой **umask**. Обычно маска по умолчанию имеет значение 022.

При создании нового каталога права доступа для него вычисляются как разность между правами доступа 777 и маской (если маска 022, то права получаются 755). При создании нового файла маска вычитается из 666. Эта подробность опускается в большинстве руководств, наверное, из неприязни к «числу дьявола».

Если системный администратор хочет установить маску, отличную от стандартной, то в один из стартовых скриптов системы или в файл `/etc/profile` надо вписать команду

```
umask маска
```

Типы файловых систем

Изначально в UNIX использовалась файловая система s5, в которой допускались имена файлов длиной до 14 символов, блок данных на диске имел размер от 512 байт до 2 Кбайт. В начале 1980-х годов в SunOS была введена UFS, которую отличала длина имени файла до 255 символов и размер блока от 4 до 8 Кбайт. В 1983 году аналог этой системы появился в 4.2BSD под названием FFS (Fast File System). Новшеством этой файловой системы стало логическое деление диска на *группы цилиндров* (*cylinder groups*). В каждой такой группе хранится копия суперблока и часть таблицы индексных дескрипторов, отвечающая за файлы, записанные в этой группе.UFS отличается от FFS тем, что в UFS реализована концепция виртуальной файловой системы.

Виртуальная файловая система – это некая абстракция и соответствующий ей API. В этой абстракции существуют vnodes (виртуальные индексные дескрипторы), которые соответствуют обычным индексным дескрипторам, но не зависят от типа файловой системы, позволяя монтировать файловые системы любых типов. Виртуальные индексные дескрипторы не записаны на диск, они существуют только в памяти драйвера файловой системы. Впервые концепция виртуальной файловой системы реализована компанией Sun Microsystems.

UFS используется во многих диалектах UNIX. Сетевая файловая система NFS обеспечивает доступ к локальной файловой системе удаленным компьютерам. Ее поддерживают любые диалекты UNIX.

Тем не менее в разных диалектах UNIX поддерживаются разные файловые системы. Любая система UNIX умеет работать с родными и с неродными файловыми системами. В Linux родными считаются ext2 и ext3. Диалект OpenLinux имеет встроенную поддержку системы ReiserFS. Это сравнительно новая файловая система с журналированием, разработанная командой энтузиастов под руководством Ханса Рейзера (Hans Reiser). Более подробно об этой новой файловой системе рассказано на сайте ее разработчиков *www.namesys.com*. Solaris, FreeBSD и многие другие диалекты считают родной систему UFS. Некоторые диалекты используют модифицированные варианты UFS.

Поставщики систем UNIX иногда придумывают свои собственные файловые системы, добавляя в ядро их поддержку.

Родная файловая система HP-UX называется HFS. Есть сомнение в том, что это та же самая HFS, которая поддерживается MacOS и OpenServer Linux.

Unixware работает с vxfs, bfs и s5. Документацию по Unixware и SCO UNIX сейчас надо искать по адресу *http://docsrv.caldera.com*.

Часто определенный диалект UNIX может работать как с родной файловой системой, так и с файловыми системами других диалектов UNIX, а также и с файловыми системами, вообще не имеющими отношения к UNIX (например, с файловой системой компакт-дисков ISO 9660, которую иногда называют cdfs или cd9660).

Все файловые системы UNIX ведут свой род от классической файловой системы, разработанной в конце 60-х – начале 70-х годов XX века. Наиболее близка к прародительнице система UFS. Все они имеют одинаковую структуру управляющей информации (суперблок, таблица индексных дескрипторов, каталоги).

Некоторые файловые системы (ext3, ReiserFS) включают поддержку журналирования. Это значит, что все выполняемые с файлами и каталогами операции записываются в специальную область раздела UNIX – журнал. Запись из журнала удаляется только в случае успешного завершения операции. Если во время операции произошел сбой (например, выключилось питание), то после восстановления работоспособности компьютера файловая система будет автоматически приведена в то состояние, в котором она находилась до начала операции.

В табл. 5.4 перечислены распространенные (и не очень) файловые системы и диалекты UNIX[1], в которых они поддерживаются, а табл. 5.5 помогает сравнить самые распространенные в мире UNIX файловые системы.

[1] Полнота таблиц не гарантируется: постоянно появляются новые файловые системы и новые реализации прежних систем для разных диалектов UNIX.

Таблица 5.4. Файловые системы

Название файловой системы	Происхождение/ назначение	Где поддерживается
NFS	Network File System	в любых UNIX, есть клиенты NFS для всех известных ОС
SMBFS	Server Message Block File System	в UNIX обычно посредством пакета Samba, в системах Microsoft – встроенная поддержка
Coda	экспериментальная сетевая файловая система, разработанная в университете Carnegie Mellon	Linux, подробности на *http://www.coda.cs.cmu.edu*
DevFS	Device File System	Linux
CRAMFS	Compressed ROM File System	Linux
NTFS	NT File System	Windows NT, Linux
ISO9660 (CDROM)	компакт-диски	везде
UDF	Universal Disk Format, файловая система для DVD-дисков, поддерживает режим пакетной записи для дисков CD-R и CD-RW	Linux
BFS	Boot File System	Unixware
EFS	Extent File System (не-ISO 9660 CDROM, IRIX < 5.3)[a]	IRIX младше 5.3
XFS	Extended File System	IRIX 6.2 и более поздние
FAT/MSDOS/ VFAT	File Allocation Table, Microsoft Disc Operating System, Virtual File Allocation Table	MS DOS, Linux, все Windows-системы, FreeBSD
HFS	Hierarchical File System, Macintosh	MacOS, OpenServer Linux
HPFS	High Performance File System	OS/2, Linux
JFFS	Journaling Flash File System	Linux (требуется флэш-память)
QNX4	QNX	QNX
TMPFS	(RAM disk file system)	Linux
Minix	старый диалект UNIX – Minix	Minix
SYSV	SCO/Xenix/Coherent	SCO/Xenix/Coherent
UFS	BSD-системы	BSD, Solaris

[a] Есть еще Encrypting File System (Windows NT/2000/XP) и Enhanced Filing System (Linux). Последняя позволяет администратору определить монтируемую файловую систему на наборе блочных устройств (либо жестких дисках, либо разделах).

tmpfs – виртуальная файловая система, данные которой хранятся в оперативной памяти компьютера. От RamDisk отличается тем, что данные не обязательно все время находятся в оперативной памяти, так как могут быть перемещены в swap. Подробности можно найти по адресу *http://www-106.ibm.com/developerworks/linux/library/l-fs3.html*.

Таблица 5.5. Сравнение основных файловых систем

Параметр	**UFS**[a]	**ext2**	**ext3**	**ReiserFS**
Максимальный размер раздела	8 Тбайт	4 Тбайт	4 Тбайт	1 Тбайт
Максимальное количество файлов на разделе	сколько угодно	сколько угодно	сколько угодно	32 К каталогов, 4,2 миллиарда файлов
Максимальный размер файла	4 Тбайт	1 Тбайт	1 Тбайт	1 Тбайт
Длина имени	255	255	255	255
Длина полного имени	1023	1023	1023	n/a
Минимальный/максимальный размер блока по умолчанию	1024/ 4096 байт	1024/ 4096 байт	1024/ 4096 байт	n/a
Распределение индексных дескрипторов по умолчанию	1 на каждые 4096 байт	1 на каждые 4096 байт	1 на каждые 4096 байт	n/a
Максимальное количество монтирований до принудительной проверки файловой системы	n/a	20	20	n/a
Журналирование	нет	нет	есть	есть

[a] Значения указаны для блока размером 4 Кбайт.

Максимальные размеры файла в UFS для FreeBSD (при разных размерах блока) приведены в табл. 5.6.

Таблица 5.6. Максимальные размеры файла в USF для FreeBSD

Размер блока	**2.2.7-stable**	**3.0-current**	**Работает**	**Должно быть**
4 Кбайт	4T-1	4T-1	4T-1	>4T
8 Кбайт	>32G	8T-1	>32G	32T-1
16 Кбайт	>128G	16T-1	>128G	32T-1
32 Кбайт	>512G	32T-1	>512G	64T-1
64 Кбайт	>2048G	64T-1	>2048G	128T-1

Реальным ограничением количества блоков в одном файле в UFS в версиях FreeBSD 2.x и 3.x был блок 1G-1, хотя теоретическое ограничение – блок 2G-1. Приближение количества блоков в файле к теоретическому ограничению может приводить к возникновению ошибок. При размере блока 4 Кбайт такое количество блоков недостижимо, а при размере блока 8 Кбайт оно достижимо, но реально вводится ограничение блок 1G-1.

Разработчикам программного обеспечения неведомо, что:

(1) от добра добра не ищут

(2) лучшее – враг хорошего

(3) благими намерениями выстлана дорога в ад

Поэтому, несмотря на постоянное исправление ошибок в том, что они уже разработали, они находят время искать новые (лучшие?) решения. Неизвестно, станет ли новая файловая система таким стандартом, как классическая UFS, но мы приведем для сведения пытливого читателя короткое описание ReiserFS от самих ее разработчиков: «ReiserFS (последняя версия 3.2.25) – это экспериментальная файловая система с журналированием, которую можно установить вместе с системами OpenLinux Server и OpenLinux Workstation. ReiserFS сложнее, чем стандартная файловая система Linux, и пока находится в стадии тестирования. К преимуществам ReiserFS можно отнести лучшее использование дискового пространства, более быстрые доступ к диску и восстановление после сбоя».

Поддержка файловой системы означает, что в файловую систему компьютера можно вмонтировать другую файловую систему стандартной командой `mount`. Такая поддержка гарантирует, что после монтирования вы сможете работать с файлами и каталогами смонтированной системы так же, как и с любыми другими файлами и каталогами своего компьютера.

Если вам надо прочесть файловую систему, которую не поддерживает ваш диалект UNIX, попробуйте поискать программы, которые работают с чужими файловыми системами напрямую, без монтирования. Например, в старых версиях некоторых диалектов UNIX были программы типа `doscp`, которые могли копировать файлы с дискеты или раздела DOS без монтирования файловой системы FAT.

Таблица индексных дескрипторов

В ранних версиях разных диалектов UNIX таблица индексных дескрипторов занимала фиксированное пространство в начале устройства; в файловых системах современных диалектов эта таблица разбита на участки, распределенные по диску.

Например, в файловой системе FFS диск разбит на группы цилиндров. Каждая группа цилиндров содержит копию суперблока, битовую карту свободных блоков для данного участка и таблицу индексных дескрипторов для тех файлов, которые расположены в пределах этой группы цилиндров. Такая распределенная структура имеет два преимущества:

1. Ускорение доступа к системным структурам данных (когда системные данные расположены вблизи от блоков пользовательских данных, уменьшается расстояние, на которое перемещаются головки дисковода при переходе от служебной информации к запрошенным данным).
2. Повышенная устойчивость к сбоям носителя. При повреждении участка поверхности диска теряется лишь небольшая часть системных данных.

Каждый индексный дескриптор хранит информацию о свойствах файла и его размещении на диске:

- тип файла;
- права доступа к файлу;
- идентификатор владельца файла;
- идентификатор группы файла;
- время последней модификации файла;
- время последнего доступа к файлу;
- время последней модификации индексного дескриптора;
- длину файла в байтах (для файлов устройств это поле имеет другой смысл – см. раздел «Типы файлов» ранее в этой главе);
- идентификатор файловой системы, в которой расположен файл;
- количество связей файла;
- структуру, описывающую размещение файла на диске.

Структура, описывающая физическое размещение файла на диске в системах s5, Xenix, FFS, UFS, представляет собой последовательность из номеров блоков.

Если файл имеет размер более десяти блоков (то есть его длина больше $10 \times 512 = 5120$ байт), то предпоследние три номера обозначают не номера блоков данных, а номера косвенных блоков (indirect blocks), в которых хранятся указатели на следующие блоки данных и, возможно, на следующие косвенные блоки.

Размер индексного дескриптора в s5 составляет 64 байта. Он содержит тринадцать полей номеров блоков по 3 байта каждое. Размер поля определяет максимальный размер адресуемого раздела на диске – до 2^{24} блоков. Это 16 777 216 блока (если блоки по 512 байт, то это соответствует 8 Гбайт, а блоками по 2 Кбайт набирается целых 32 Гбайт).

Первые десять полей содержат номера блоков данных. Одиннадцатое поле – это номер косвенного блока первого уровня. В блоке первого уровня содержится до 170 адресов блоков данных (речь идет о 512-байтных блоках).

Двенадцатое поле содержит номер косвенного блока второго уровня. Косвенный блок второго уровня содержит 170 номеров косвенных блоков первого уровня, таким образом, через косвенный блок второго уровня адресуется до 170^2 блоков данных.

В тринадцатое поле записан номер косвенного блока третьего уровня. Косвенный блок третьего уровня содержит 170 номеров косвенных блоков второго уровня, так что через косвенный блок третьего уровня адресуется до 170^3 блоков данных.

Файл не может располагаться на разных разделах UNIX. Файл может быть фрагментирован, хотя файловая система построена так, чтобы свести фрагментацию к минимуму. Теоретически не рекомендуется заполнять файловую систему более чем на 70%, чтобы не увеличивать фрагментацию при работе.

Максимальный размер файла может быть легко посчитан, так как мы знаем правила адресации блоков данных. Если в блоке 512 байт, то:

блоки, напрямую адресуемые из индексного дескриптора, дают 10×512 байт = 5 120 байт

адресуемые через номер косвенного блока первого уровня – 170×512 байт = 87 040 байт

адресуемые через номер косвенного блока второго уровня – $170^2 \times 512$ байт = 14 796 800 байт

адресуемые через номер косвенного блока третьего уровня – $170^3 \times 512$ байт = 2 515 456 000 байт

Итого 2 530 344 960 байт, или примерно 2,5 Гбайт.

Свежие диалекты Unix, в частности последние версии FreeBSD, используют 128-байтный индексный дескриптор (например, в файловой системе UFS), который вмещает больше информации, и номер блока в нем занимает не 3 байта, а 4. Поэтому адресовать можно куда большие разделы.

В этом «увеличенном» дескрипторе 28 байт отводится под разные «расширения», включая место для 32-разрядных идентификаторов владельца и группы файла, а также под 64-разрядные поля времен модификации. Введение 64-разрядных полей времени в UNIX вместо прежних 32-разрядных нужно для того, чтобы избежать «проблемы 2031 года», так как именно в этом году 32 байт станет недостаточно для представления времени в системах UNIX. Разработчики обоснованно полагают, что к этому времени все существующие системы UNIX будут заблаговременно переведены на 64-разрядную архитектуру.

Монтирование и демонтирование файловой системы

Для того чтобы файловая система, размещенная на каком-то устройстве (например, на разделе жесткого диска), стала доступной для использования, ее надо смонтировать. Монтирование означает сопоставление файла устройства и точки монтирования. Точка монтирования – это один из каталогов всей файловой системы компьютера.[1] Монтирование выполняется командой **mount**. Для успешного монтирования необходимо, чтобы каталог, указанный программе mount в качестве точки монтирования, существовал и был пустым.

При старте системы программа **mount** запускается из стартового скрипта и автоматически монтирует файловые системы, указанные в файле /etc/fstab. В Solaris и HP-UX этот файл называется /etc/vfstab. Параметры монтирования указываются в этом же файле. Файловая система может быть смонтирована только для чтения (read-only) или для чтения и записи.

После монтирования файловой системы в каталог в нем появляются все файлы и каталоги, расположенные в смонтированной файловой системе.

При монтировании файловой системы в непустой каталог прежнее содержимое этого каталога должно исчезнуть, если оно хочет себя вести соответственно классической документации по UNIX. На самом деле во всех известных нам диалектах UNIX оно просто становится временно недоступным. После размонтирования этой файловой системы (команда **umount**) старое содержимое каталога вновь становится доступным.

Синтаксис команды **mount** требует указания файла того устройства, файловая система которого монтируется, и точки монтирования (каталога, куда монтируется файловая система):

mount *файл_устройства точка_монтирования*

При монтировании неродной файловой системы следует указать ее тип. Какую именно систему программа **mount** в вашем диалекте UNIX считает родной, можно узнать, дав команду **man mount**. Например, при монтировании дискеты, отформатированной под FAT, в каталог /mnt/ floppy в Linux надо дать команду

```
mount -t msdos /dev/fd0 /mnt/floppy
```

Здесь /dev/fd0 – файл устройства «дисковод», а /mnt/floppy – заранее созданный пустой каталог. После выполнения команды **mount** в /mnt/floppy «появится» (станет доступным) содержимое дискеты.

[1] Здесь имеется в виду файловая система во втором смысле этого термина, то есть совокупность всех файлов компьютера.

Важно, что каталог, указываемый в качестве точки монтирования, должен существовать до начала монтирования.

Список всех смонтированных в системе файловых систем можно получить по команде **mount** без аргументов.

Для демонтирования файловой системы применяется команда

umount *точка_монтирования*

Демонтировать занятую файловую систему нельзя. Занятой считается файловая система, с которой работает хотя бы один процесс. Например, если один из пользователей открыл файл внутри дерева каталогов файловой системы. Если чей-нибудь текущий каталог находится в файловой системе, которую необходимо демонтировать, пользователь должен покинуть этот каталог, т. к. с текущим каталогом работает shell. Иногда системный администратор пытается демонтировать файловую систему при том, что его текущий каталог находится внутри нее. Это тоже ошибка: сначала надо сказать что-нибудь вроде **cd /** (если только демонтируется не корневой каталог).

Применение символических ссылок

Символические ссылки часто помогают системным администраторам исправлять собственные старые ошибки и упущения. При установке UNIX надо заранее планировать, как будет организовано дисковое пространство. Например, если UNIX-машина будет использоваться как http-cache большой организации, надо предусмотреть большой UNIX-раздел для хранения кэшированных html-, gif-, jpg- и mp3-файлов. Типы файлов, конечно, могут быть и другими, эти просто наиболее популярны. Раздел не просто должен быть большим, он должен содержать значительное количество индексных дескрипторов, ведь при прогулках по Интернету, как вы знаете, выкачивается несметное множество мелких gif- и jpg-файлов – кнопочек, линеечек и прочих дизайнерских изысков. Места они занимают немного, но каждому из них нужен индексный дескриптор. Значит, надо при создании файловой системы подумать, не лучше ли увеличить количество индексных дескрипторов, создаваемых по умолчанию.[1]

Заранее трудно предсказать, как много почты будут получать пользователи UNIX-машины. Если они работают локально (то есть входят в систему и работают с командным процессором в интерактивном сеансе), то они, очевидно, будут хранить свои почтовые ящики на сервере. Более того, они будут упорно хранить там все письма, полученные ими со времен розового детства их младшего сына, да-да, того, который скоро поступает в университет.

[1] Как это ни странно, в большинстве случаев делать это не стоит. В UNIX приняты весьма разумные значения по умолчанию.

Почтовые клиенты[1] обычно работают именно с почтовым ящиком в /var/mail, а для подключения к ящикам из других мест они требуют давать им ключи. Вы когда-нибудь видели пользователя, который бы помнил ключи каких-то странных программ? А такого, что не только помнил, но и не ленился их вводить?

Пользователи, работающие с сервером удаленно, как правило, не задумываются о том, где именно они хранят почту, но предпочитают это делать там, где она хранится по умолчанию. Чтобы не совершать лишних действий. А по умолчанию она обычно хранится именно в /var/mail.

Почтовые программы, работающие по протоколу IMAP (например, TheBat!), более благосклонны к системным администраторам и часто хранят почту пользователя в файле mbox в его домашнем каталоге.

Так или иначе, вы можете столкнуться с нехваткой места в каком-то важном каталоге. Предположим, это каталог /var/mail, в котором хранятся почтовые ящики пользователей. «Тупо» перенести такой каталог в другой раздел UNIX довольно сложно. Ведь при этом изменится полное имя каталога, а к нему «привыкли» почтовый сервер sendmail, почтовые серверы, отдающие клиентам почту: imapd, pop3d. Может быть, он прописан в каких-нибудь скриптах, о которых вы уже давно забыли, но они еще запускаются автоматически раз в неделю.

Выход состоит в том, чтобы физически перенести каталог в другой раздел, где больше места, но логически оставить его на месте. Проще говоря, вы находите другой раздел, где места много больше, чем там, где сейчас лежит /var/mail. Может быть, даже подключаете новый жесткий диск и создаете на нем этот большой раздел. Потом перемещаете туда каталог /var/mail. Затем (подкаталога mail в /var в этот момент уже нет – он перенесен на новый раздел и называется, допустим, /new/ mail) создаете символическую ссылку mail в каталоге /var, она будет указывать на /new/mail.

Полностью процедура выглядит так:

```
# сначала остановить sendmail
ps -ax | grep sendmail
28059   ?        S      2:52 sendmail: accepting connections
kill 28059
# sendmail остановлен
# копируем /var/mail, mv использовать нельзя
cp -Rp /var/mail/  /new/
# удаляем старый каталог и создаем символическую ссылку
на новый
rm -rf /var/mail/
ln -s /new/mail/ /var/mail
# запускаем sendmail вновь
sendmail -bd -q30m
```

[1] Это такие программы, которые называются неблагозвучной аббревиатурой MUA (Mail User Agent) и служат для чтения и отправки почты.

Останавливать **sendmail** надо для того, чтобы предотвратить запись в почтовые ящики во время копирования каталога с ними. Вообще говоря, было бы неплохо остановить и клиентские операции с почтовыми ящиками, запретив на время демону **inetd** запускать imapd и pop3d. Можно вместо этого (если позволяет ситуация) выключить на время UNIX-машину из компьютерной сети, по которой к ней обращаются.

После остановки процессов, работающих с почтовыми ящиками, надо поместить копию старого каталога на новое место. Программа **mv** не умеет переносить целые каталоги с одной файловой системы на другую, поэтому используем копирование. Ключ **-R** команды **cp** – рекурсивное копирование каталогов, ключ **-p** – сохранение прав доступа к каталогам при копировании.

Затем создаем символическую ссылку и не забываем запустить **sendmail** снова с теми же ключами, с которыми он был запущен ранее. Какие были ключи, можно посмотреть в стартовых файлах системы. Где именно, зависит от диалекта UNIX. Например, в `/etc/rc.conf` во FreeBSD.

Теперь любой процесс, желающий записать что-либо в `/var/mail`, будет с успехом делать это, но фактически запись будет происходить в `/new/mail`, так как файловая система при обращении к `/var/mail` будет автоматически подставлять `/new/mail`.

Поиск файлов

Тяжело найти черную кошку в темной комнате.
Особенно если ее там нет.
Конфуций

Весьма мощные возможности поиска в файловой системе всегда помогут отыскать файл, если о нем хоть что-то известно.

Во-первых, можно воспользоваться программой **locate**. Она быстро просматривает индексированную базу данных, которая содержит имена всех файлов в системе. Эту базу надо сначала создать, а затем регулярно обновлять. Во FreeBSD программа обновления называется **/usr/libexec/locate.updatedb**, в некоторых диалектах – просто **updatedb**. Как правило, название программы обновления есть в **man locate**. Во FreeBSD вызов **locate.updatedb** включен в ежедневно выполняемый скрипт **/etc/daily** (в версиях 4.x и выше – **/etc/periodic/daily**).

Во-вторых, существует программа **whereis**.

Программы **locate** и **whereis** есть не во всех диалектах UNIX. Зато программа **find** гарантированно есть в любом UNIX.

Синтаксис ее вызова таков:

```
find каталог_откуда_начать_поиск -параметр_поиска аргумент
```

Например, для поиска файла `perl` можно использовать команду

```
find /usr/bin -name perl
/usr/bin/perl
```

С помощью **find** можно искать файл по имени. Допускаются шаблоны типа `*.pl`, но помните, что их надо заключать в кавычки, чтобы shell не пытался их интерпретировать по-своему. Кроме этого, файл можно искать по дате модификации, длине, правам доступа и т. д. Полный список возможностей содержится в **man find**.

Отличие **locate** и **whereis** от **find** состоит в том, что **find** ищет файлы в каталогах на диске, а **locate** и **whereis** – в заранее составленных списках файлов. Поэтому **find** всегда выдает актуальную информацию, а **locate** и **whereis** могут ошибиться и сообщат об удаленном или переименованном файле. Ненадежность **locate** и **whereis** отчасти компенсируется скоростью их работы. Программа **find** работает намного медленнее, и, чем больше файлов на диске, тем медленнее будет работать **find**.

При запуске **find** обычным непривилегированным пользователем программа время от времени будет натыкаться на закрытые для нее каталоги, куда этому пользователю нет доступа. Поскольку **find** выполняет рекурсивный поиск по дереву каталогов, это неизбежно при поиске, начатом от корневого каталога. Вообще говоря, риск натолкнуться на запертую дверь в такой ситуации всегда велик, если только пользователь начал поиск не в своем домашнем каталоге.

Между тем, пользователь обычно не знает, где именно лежит нужный файл. И начать поиск он хочет с корневого каталога. Поэтому, чтобы не получать вереницу сообщений «permission denied», имеет смысл поток ошибок (`stderr`) программы **find** перенаправить в `/dev/null`. Это можно сделать командой

```
find / -name "что_искать" 2>/dev/null
```

Команда не работает в **csh** и **tcsh**, потому что там нет перенаправления стандартных потоков по номеру внутреннего дескриптора. Возможно, скоро появятся версии, в которых оно будет.

6

Процессы

Процесс – это единица распределения ресурсов системы в UNIX. Процессу выделяется память и процессорное время, ему отводится строка в таблице процессов. Одновременно в UNIX может быть запущено не более 65 535 процессов (для 64-разрядного UNIX – не более $2^{64}-1$ процесс). Это ограничение теоретическое, потому что на практике количество одновременно работающих процессов ограничивается размером виртуальной памяти системы и другими ее ресурсами.

Каждый процесс имеет свой приоритет.

Планировщик задач раз в 100 микросекунд анализирует очередь процессов, ждущих передачи им управления. В соответствии с приоритетами он передает управление тому процессу, который в данный момент имеет наивысший приоритет.

Одна программа может породить несколько параллельных процессов. Например, веб-сервер, отвечая на внешние запросы, порождает несколько параллельных процессов, каждый из которых обрабатывает один запрос.

Процесс может находиться в нескольких состояниях: запущенном (running, потребляет процессорное время в настоящий момент), спящем (suspended, ничего не делает, но контекст процесса сохранен, можно продолжить выполнение в любой момент), выгруженном на диск (swapped, страницы памяти процесса записаны в swap-раздел диска), в ожидании завершения операции ввода/вывода (I/O wait, запустили чтение/запись и ждем, пока завершится).

Распределение памяти. Свопинг

Поскольку процессы в многопроцессной системе могут занимать больше места, чем установлено оперативной памяти, в таких системах используют виртуальную память. Общий объем виртуальной памяти складывается из объема оперативной памяти и объема swap-разделов, размещенных на жестких дисках компьютера. Вся память в UNIX разделена на страницы по 4 Кбайт каждая. Ядро и процесс swapper всегда находятся в памяти, остальные процессы могут быть выгружены на диск полностью или частично. В разных диалектах UNIX процесс, отвечающий за обмен страницами памяти между диском и оперативной памятью, называется по-разному, но обычно в его имени присутствует слово «swap».[1] Процесс обмена страницами имеет очень высокий приоритет, потому что ни один процесс не сможет выполняться, если его исполняемый код еще не загружен в оперативную память.

Традиционно этот процесс обозначается неблагозвучным термином «свопинг» (swapping). При этом происходит почти полная выгрузка страниц процесса на диск. На самом деле менеджер виртуальной памяти в UNIX выполняет «подкачку» (paging). Это означает частичную выгрузку страниц на диск. Алгоритм, применяемый для выгрузки страниц в UNIX, называется «алгоритмом часов». Предполагается, что стрелка этих «часов» время от времени указывает на случайно выбранную страницу памяти. Как только это произошло, страница становится кандидатом на выгрузку. Если в течение определенного промежутка времени к ней не было обращений, ее и в самом деле выгружают на диск. Детали этого алгоритма могут отличаться в разных диалектах UNIX.

При создании swap-раздела обычно рекомендуют делать его равным удвоенному объему оперативной памяти. Однако практика показывает, что производительность многих Linux-систем падает, если объем swap-раздела превышает 96 Мбайт. Разумеется, большой swap-раздел не нужен и в системе со значительным объемом оперативной памяти (256 Мбайт и больше), если только вы не собираетесь запускать сервер баз данных или http cache, которым нужно столько памяти.

Конечно, мы должны отдавать себе отчет в том, что через год-другой компьютеры с 1–2 Гбайт оперативной памяти распространятся повсеместно, а программное обеспечение станет еще более требовательно к объему оперативной памяти. Поэтому, прежде всего, надо хорошо знать, сколько памяти необходимо тем программам, которые будут одновременно работать на вашем компьютере. Создание огромного swap-раздела, который никогда не будет использован даже на 40%, нерацио-

[1] Например, во FreeBSD этот процесс называется swapper, а в Linux – kswapd. Однако всюду он является частью ядра и поэтому даже не всегда присутствует в таблице процессов.

нально, равно как и создание слишком маленького swap-раздела, потому что последнее приведет к нехватке виртуальной памяти в системе.

При нехватке памяти ядро UNIX просто уничтожает процессы с низким приоритетом, чтобы освободить место для других, более важных для системы процессов. Иногда ядро убивает процессы молча, даже не сообщая об этом на консоль, поэтому в условиях нехватки памяти компьютер может стать «вещью в себе»: он будет работать, но перестанет отвечать на внешние запросы. В таком случае помогает только аппаратный перезапуск.

Идентификаторы процесса

Процесс всегда имеет свой уникальный номер – PID (process ID). PID – это целое число от 0 до 65 535. Номер 0 обычно имеет процесс свопинга, фактически являющийся частью кода ядра. Процесс с номером 1 – это **init**, процесс, который порождает все остальные процессы в системе при ее старте. У всех процессов, кроме **init**, есть родительский процесс, и каждому процессу сопоставляется PPID (parent process ID, номер родительского процесса). Номера присваиваются ядром последовательно, каждый новый процесс получает следующий свободный номер из таблицы процессов. К началу таблицы ядро возвращается только после того, как дойдет до конца.

В таблице процессов, кроме PID и PPID, хранятся идентификаторы eUID (effective UID) владельца процесса и eGID (effective GID) группы процесса. Владелец и группа процесса – это пользователь и группа, от имени которых был запущен процесс. Идентификаторы eUID и eGID по умолчанию передаются процессу по наследству от родительского процесса. При входе пользователя в систему для него и от его имени запускается командный процессор (программа-оболочка). Группой процесса этой оболочки является главная группа пользователя, вошедшего в систему.

Любой процесс входит в какую-либо терминальную группу, то есть группу процессов, запущенных с одного терминала. Каждый процесс имеет TTYID – идентификатор терминала, с которого он был запущен. Процесс характеризуется приоритетом, в зависимости от которого запросы процесса к системе выполняются медленнее или быстрее.

Полный список процессов в системе можно получить, выполнив команду **ps -auxw** (BSD) или **ps -ef** (System V).

Приоритеты процесса

Процесс имеет базовый приоритет (параметр NICE) и динамический приоритет (PRI). Базовый приоритет может принимать значение от –20 до +20, в некоторых диалектах – от 0 до 39. Чем меньше значение

NICE, тем выше приоритет. Процесс с приоритетом –20 – самый высокоприоритетный. Динамический приоритет вычисляется планировщиком задач на основе базового приоритета в зависимости от того, как много процессорного времени уже потребил процесс и как давно он стоит в очереди на получение управления. Чем дольше выполняется процесс, тем ниже становится его динамический приоритет. Это сделано для того, чтобы дать возможность выполняться более новым и более коротким процессам.

Базовый приоритет в русскоязычной литературе иногда называют «фактором уступчивости» или «показателем любезности».

Доступ процессов к файлам

Процедура получения доступа к каталогам и файлам одинакова, так что в выражениях типа «доступ к файлу (каталогу)» слово «каталог» мы будем опускать.

Эффективные идентификаторы владельца (eUID) и группы (eGID) процесса определяют, какие права доступа к файлу есть у процесса. Когда процесс обращается к файлу, ядро проверяет, кем по отношению к этому файлу является владелец процесса. Если эффективный идентификатор владельца процесса совпадает с идентификатором владельца файла, то процесс получает права доступа к файлу, определенные для владельца файла. Если владелец процесса входит в группу файла, но не является владельцем файла, он получает права, определенные для группы файла. Если ни то и ни другое, процесс получает права доступа к файлу, определенные для «всех остальных».

Новый файл по умолчанию получает идентификаторы владельца и группы по наследству от процесса, который его создал.

Обратите внимание: эффективный идентификатор группы процесса вообще не играет роли в вычислении прав доступа к файлу. Он нужен только для того, чтобы передать его по наследству новому файлу, который будет создан процессом.

Запуск процесса от чужого имени

Идентификаторы владельца и группы процесса называются эффективными из-за того, что они не всегда передаются по наследству. На деле они вычисляются, исходя из специального права доступа, о котором сейчас пойдет речь. Вот почему результат этих вычислений решили назвать «эффективным», то есть тем, что действительно оказывает воздействие на права процесса. Дело в том, что иногда надо запустить процесс от имени другого пользователя. Конечно, не для того, чтобы начинающий программист мог послать веселую картинку начальнику от имени вашего секретаря.

Например, вы хотите изменить свой пароль. Для этого вам надо дать право записи в файл `/etc/shadow`. С другой стороны, нельзя давать вам это право: вдруг вы поменяете не только свой пароль? Пусть даже и по ошибке, а не по злому умыслу. Выход нашли: право записи в `/etc/ shadow` дали не пользователю, а программе **passwd**. Однако в UNIX нет механизма, который позволил бы наделять правами отдельный процесс, потому что вся система безопасности построена на распределении прав на объекты между пользователями и группами. Поэтому сделали так: право записи в `/etc/shadow` дали пользователю root, а программу **passwd** разрешили запускать от имени ее владельца – root. Теперь для модификации пароля программу **passwd** вы запускаете не от своего имени, а от имени root.

Не подозревали об этом? Не удивительно: вам же не пришлось ничего говорить системе, чтобы **passwd** запустилась от имени root. За пользователя это сделал системный администратор, который и должен следить за программами, которые надо запускать от чужого имени.[1] От имени своего владельца запускаются только те файлы, в правах доступа к которым выставлен бит suid (set UID). Вспомним полный вид слова прав доступа:

```
su   sg   t   r   w   x   R   w   x   r   w   x
```

Старшие три бита – suid, sgid и sticky bit (t).

Suid и Sgid

Установить бит suid или sgid можно, только указав **chmod** права доступа в числовом виде:

chmod 4755 *файл*

В некоторых системах верно обратное: бит suid можно установить, только дав команду:

chmod u+s *файл*

Если в правах доступа к файлу установлен бит suid, то идентификатор владельца не передается по наследству новому процессу при запуске этого файла. Вместо этого эффективному идентификатору владельца процесса присваивается значение идентификатора владельца запускаемого файла. Аналогично действует бит установки группы (sgid) при запуске файла.

---

[1] Честно говоря, программе **passwd** о том, что она должна запускаться от имени root, сказал не системный администратор, а программа-установщик операционной системы, поскольку в любом UNIX программа **passwd** должна запускаться от имени root.

При выводе списка файлов по команде **ls -l** права доступа к файлам с установленными битами suid и sgid выглядят так (вместо x появляется s):

- rwsr-xr-x – установлен suid;
- rwxr-sr-x – установлен sgid;
- rwsr-sr-x – установлены и suid, и sgid.

Если у владельца файла нет прав запуска файла, но в правах доступа установлен suid, буква s в описании прав доступа владельца становится прописной:

```
chmod 4755 file
chmod u-x file
ls -l
…
-rwSr-xr-x    1    root    root    3587    Jul 20 2002    file
```

Такая комбинация прав доступа, при которой владелец файла не имеет права запуска файла, а остальным разрешено запускать файл от имени владельца, довольно редко встречается на практике.

Системный администратор должен внимательно относиться к файлам, которые можно запускать от чужого имени. Иногда появление таких файлов в системе облегчает жизнь взломщику. Программа **passwd**, например, написана таким образом, что злоумышленник точно не сможет с ее помощью сделать ничего, кроме изменения собственного пароля. Поэтому ей можно доверить запускаться с правами root. Устанавливайте бит suid только тем программам, в которых вы уверены на сто процентов!

Во FreeBSD в сценарии /etc/daily, ежедневно проверяющем состояние системы, по умолчанию выполняется поиск добавленных за сутки файлов с установленным битом suid. Обязательно удостоверьтесь, что все эти файлы появились легально. Если в вашей системе нет такой проверки, напишите сами простой скрипт, который будет вызывать программу **find** для поиска файлов с битом suid и отправлять вам по почте результаты поиска.

Интерактивные и фоновые процессы

Процесс, имеющий доступ к клавиатурному вводу, называется интерактивным (foreground). Такими процессами, например, могут быть текстовый редактор или почтовая программа **mail**. Однако интерактивный процесс не обязательно должен принимать клавиатурный ввод, он должен лишь иметь такую возможность. Например, по команде **ls -l** запускается интерактивный процесс – программа **ls**. Она ничего не ждет от пользователя и только выводит на экран информацию. Любая программа, запущенная обычной командой, например **ls**, **ps** или **mail**, представляет собой интерактивный процесс.

Фоновый (background) процесс выполняется «на заднем плане», не имея возможности получить данные с клавиатуры и (как правило) вывести их на экран.

Некоторые особенно упорные фоновые процессы все же выводят на экран данные, причем тогда, когда пользователь этого не ждет. Пугаться этого не следует: фоновый процесс не в состоянии вмешаться в работу интерактивного процесса таким образом. Например, вы работаете в текстовом редакторе **vi**, и вас раздражают сообщения, которые вывела на экран программа **find**, выполняющаяся в фоновом режиме. Достаточно нажать клавиши <Ctrl>+<L> для обновления экрана, и сообщения исчезнут.

Фоновый процесс запускается из оболочки командой, за которой следует знак «&» (амперсанд):

```
команда &
```

Можно запустить компиляцию длинной программы в фоновом режиме и, пока она компилируется, работать над статьей:

```
cc -o the_best long_code.c &
vi article1.txt
```

Взаимодействие процессов

Процессы взаимодействуют, передавая друг другу данные, управление или конкурируя за ресурсы системы.

Каждый процесс имеет три связанных с ним потока данных: стандартный ввод (stdin, внутренний дескриптор файла – 0), стандартный вывод (stdout, внутренний дескриптор файла – 1) и стандартный поток ошибок (stderr, внутренний дескриптор файла – 2). По умолчанию все три потока связаны с тем терминалом, с которого запущен процесс (стандартный ввод – с клавиатуры, вывод данных и ошибок – на экран). Стандартный ввод еще называют входным потоком, а стандартный вывод – выходным.

Внутренний дескриптор файла – это номер, по которому к файлу (потоку данных) можно обратиться «изнутри» программы. Номер представляет собой беззнаковое целое число. То есть где-то в программе можно написать:

```
print (1,*)"hello,world"
```

Пример написан на Фортране, но обратиться к файлу, указав его внутренний дескриптор, позволяют и другие языки программирования. Для того чтобы обратиться к файлу, надо сопоставить номер внутреннего дескриптора с именем файла. В UNIX для стандартных потоков это делается автоматически, потому что никаких имен в файловой системе они не имеют. Для файлов, которые программа открывает для

чтения или записи, внутренний дескриптор создается системным вызовом `open`. Внутренний дескриптор файла существует только в пределах того процесса, который его создал. У разных процессов могут быть дескрипторы с одинаковыми номерами, но это будут физически разные дескрипторы.

При записи в файл или чтении из него с дескриптором файла сопоставляется смещение в байтах от начала файла, которое является текущей позицией в файле при чтении или записи в него.

Перенаправление потоков

Потоки данных, связанные с процессом, могут быть перенаправлены. Например, если требуется, чтобы программа **ls** выдала список имен файлов не на экран, а в файл `list`, мы скажем:

```
ls > list
```

Символ **>** означает перенаправление выходного потока в файл. Пробелы вокруг знака **>** игнорируются, так что их можно вовсе не ставить.

При таком перенаправлении потока файл `list` будет создан, если он еще не существует, а если он есть, то его старое содержимое безвозвратно потеряется. Если мы хотим добавить к содержимому файла то, что выводится в выходной поток, мы должны использовать конструкцию **>>**:

```
ls >> list
```

В этом случае файл `list` будет создан, если он еще не существует, а если он есть, то все, что будет выведено в выходной поток программой **ls**, добавится в конец файла.

Можно перенаправить ввод из файла. Например, можно заранее подготовить письмо в текстовом редакторе и отправить его командой

```
mail адресат@сервер < файл_с_письмом
```

Конструкция «встроенный документ»[1] помогает перенаправить ввод в процесс не из файла, а прямо из командной строки. Правда, эту конструкцию разумно использовать не при наборе команд вручную, а в скрипте:

```
mail адресат@сервер <<END
Дорогой $USER,
помни об ужине!
Автоинформатор
END
```

[1] По-английски эта конструкция называется «here document», что в русскоязычной литературе часто переводят как «документ здесь». Нам представляется более понятным термин «встроенный документ».

Данный скрипт будет отсылать по указанному адресу письмо, в котором вместо `$USER` будет подставлено имя того пользователя, который запустил этот скрипт. Переменная среды окружения USER содержит это имя, а конструкция $USER требует от командного процессора подставить значение переменной USER.

При использовании конструкции `<<` то, что следует непосредственно за ней до ближайшего символа-разделителя, считается **ограничителем текста**. Символ-разделитель – это пробел, табуляция или конец строки. Все, что идет за первым ограничителем текста, передается во входной поток запущенного командой процесса до тех пор, пока в начале новой строки не встретится точно такой же ограничитель текста. Как именно выглядит ограничитель – будет это слово «END», как в нашем примере, или слово «finish», или просто символ . (точка) – совершенно не играет роли.

Кроме перенаправления потока в файл или из файла, возможно перенаправление потока во входной поток другого процесса. Для этого ядро организует программный канал. Например, можно выходной поток программы **ls** направить во входной поток программы **more**, чтобы длинный список файлов вывести поэкранно:

```
ls -l | more
```

Символ «|» (вертикальная черта или «сплюснутое двоеточие»; как именно вы видите этот символ, зависит от того, на какую клавиатуру вы смотрите) означает перенаправление выходного потока программы, указанной слева, во входной поток программы, указанной справа.

Иногда необходимо перенаправить выходной поток первого процесса на вход второго, выходной поток второго – на вход третьего и т. д. Такая конструкция в UNIX называется конвейером. Действительно, данные идут от одного процесса к другому, и каждый процесс может их как-то обработать. Очень похоже на обычный конвейер на заводе.

Например, можно отправить по почте большой файл в виде приложения. Это не так просто, потому что передача слишком больших файлов по почте – дурной тон. Большие файлы надо сжимать перед отправкой, такое письмо дойдет быстрее и места в почтовом ящике получателя займет меньше. С другой стороны, по почте нельзя передавать ничего, кроме текста. Если ваша любимая игрушка называется bigtetris.exe и занимает 10 Мбайт на диске, ее нельзя отправить командой

```
mail friend@mail.ru < bigtetris.exe # так нельзя
```

Потому что `bigtetris.exe` кроме символов, допустимых в тексте, содержит и непечатаемые символы, которые по почте не передаются. Поэтому для отправки двоичного или сжатого файла нужно сначала его закодировать, а затем отправить. Многие почтовые программы кодируют прикладываемые к письму файлы незаметно для пользователя. Однако **mail** – очень простая программа, которую написали тогда,

когда электронная почта была нужна только для обмена текстовыми сообщениями. Поэтому вызвать программу кодировки нам придется самостоятельно.

Воспользуемся программой **gzip** для сжатия (ключ **-c** нужен для вывода сжатого файла в стандартный поток вывода) и **uuencode** – для кодировки:

```
gzip -c < bigtetris.exe | uuencode bigtetris.exe.gz | mail \
friend@mail.ru
```

Аргумент программе **uuencode** дается для того, чтобы она знала, как назвать кодируемый файл, потому что код, который выдает **uuencode**, должен включать имя файла. Конвейеры могут использоваться и при обработке текста, об этом будет рассказано в главе 7.

Сигналы, команда kill

Процессы могут посылать друг другу сигналы. Их список в UNIX ограничен тридцатью двумя, из которых наиболее употребительны сигналы TERM, KILL и HUP. Каждый сигнал имеет свое мнемоническое обозначение и номер.

Процесс, запущенный от имени обычного пользователя, может послать сигнал только процессу, запущенному тем же пользователем. Процесс, работающий от имени root, может послать сигнал любому процессу в системе.

В табл. 6.1 приведен список сигналов, определенный стандартом POSIX 1.1.

Таблица 6.1. Сигналы POSIX 1.1

Сигнал	Номер	Значение
SIGHUP	1	Разрыв связи с управляющим терминалом или управляющим процессом
SIGINT	2	Прерывание с клавиатуры
SIGQUIT	3	Сигнал выхода дан с клавиатуры
SIGILL	4	Недопустимая инструкция
SIGABRT	6	Сигнал abort получен от вызванной кем-то функции abort
SIGFPE	8	exception: ошибка вычислений с плавающей запятой
SIGKILL	9	Безусловное завершение процесса
SIGSEGV	11	Неверный адрес памяти
SIGPIPE	13	Запись в несуществующий канал
SIGALRM	14	Сигнал timer от функции alarm
SIGTERM	15	Завершение

SIGUSR1	30,10,16	Определяется пользователем
SIGUSR2	31,12,17	Определяется пользователем
SIGCHLD	20,17,18	Дочерний процесс остановлен или принудительно завершен
SIGCONT	19,18,25	Продолжить выполнение, если оно было остановлено
SIGSTOP	17,19,23	Остановить процесс
SIGTSTP	18,20,24	Остановить ввод с терминала
SIGTTIN	21,21,26	Ввод с терминала для фонового процесса
SIGTTOU	22,22,27	Вывод с терминала для фонового процесса

Каналы (pipes) и сокеты (sockets)

Каналы и сокеты служат для межпроцессной коммуникации, то есть для передачи данных между одновременно запущенными процессами. Для работы с файлом, каналом или сокетом процесс должен обратиться к нему, используя дескриптор (описатель) файла или канала.

Дескриптор файла создается при выполнении системных вызовов `open` и `socket`. Системный вызов `open` получает в качестве аргументов имя файла и тип доступа, который определяет, должен ли файл открываться для чтения, для записи или для обеих операций. При открытии несуществующего файла для записи файл будет создан. Системные вызовы `read` и `write` применяются для передачи данных из файла и в файл. Системный вызов `close` уничтожает дескриптор. Обычно дескрипторы представляют три типа объектов: файлы, каналы и сокеты.

Файл – это последовательность байтов, имеющая по крайней мере одну жесткую ссылку на него в файловой системе. Процессы работают с устройствами ввода/вывода как с файлами.

Канал – это последовательность байтов, такая же, как файл, но используемая исключительно как однонаправленный поток ввода/вывода. У обычного канала нет имени, он не может быть открыт при помощи вызова `open`. Он создается вызовом `pipe`. Бывают также именованные каналы. Они имеют те же самые свойства, что и обычные каналы, но располагаются в файловой системе и поэтому могут быть открыты системным вызовом `open`. Процессы, которые хотят обмениваться данными, могут открыть такой именованный канал: один процесс открывает его для чтения, а другой – для записи.

Сокет (socket) является промежуточным объектом, который используется для межпроцессных коммуникаций; он существует, пока какой-либо процесс хранит дескриптор, ссылающийся на него. Сокет создается системным вызовом `socket`, который возвращает его дескриптор. Имеется несколько типов сокетов, которые поддерживают различные возможности передачи данных.

Для каждого процесса ядро хранит таблицу внутренних дескрипторов.[1] Она наследуется от родительского процесса, поэтому вместе с ней наследуется и доступ к объектам, на которые ссылаются дескрипторы (файлам, каналам). Основными способами, при помощи которых процесс может получить дескриптор, являются открытие или создание объекта, а также наследование от родительского процесса. Когда процесс завершается, ядро освобождает все дескрипторы, которые использовались этим процессом. Если процесс хранил последнюю ссылку на файл или канал (то есть и в файловой системе больше нет записей об этом объекте), то менеджер объектов уведомляется о том, что надо выполнить окончательное удаление файла или уничтожение сокета.

Семафоры

Семафоры – это механизм, предназначенный для контроля доступа нескольких процессов к одному ресурсу. Есть несколько реализаций API, связанного с семафорами: вариант System V IPC (Inter-Process Communication), BSD и POSIX 1003.1b. Последний реализован во многих современных UNIX.

Идея семафоров сводится к следующему. Семафор – это глобальная целая переменная. Он может иметь положительное значение, говорящее о том, что ресурс, контролируемый этим семафором, свободен. Нулевое значение говорит о том, что ресурс занят. Отрицательное значение свидетельствует о том, что ресурс занят и к нему выстроилась очередь из процессов.

Семафор не имеет процесса-владельца. Однако созданный процессом семафор передается по наследству его дочерним процессам. При этом дочерний процесс не обязан инициализировать или открывать семафор заново для начала работы с ним. Бывают именованные и неименованные семафоры. Неименованные семафоры служат для синхронизации родительского и дочерних процессов, именованные семафоры – для синхронизации любых процессов, в том числе не состоящих в отношениях родства.

Семафор создают для контроля доступа к какому-нибудь общему для нескольких процессов ресурсу: физическому устройству, области разделяемой памяти и т. п.

Синхронизация процессов с помощью семафора происходит так. Перед началом работы с ресурсом создается семафор, который инициализируется положительным числом. Число (обычно 1) означает количе-

[1] В оригинале разработчики назвали эту таблицу «descriptor table», но чтобы не путать ее с таблицей дескрипторов в файловой системе, в нашей книге она называется таблицей внутренних дескрипторов. Слово «внутренних» подчеркивает, что эти дескрипторы существуют только в пределах процесса, который пользуется ими для обращения к файлам.

ство процессов, которым может быть дозволено одновременное обращение к ресурсу. Например, если наш принтер умеет обрабатывать по два задания одновременно, мы можем инициализировать семафор, контролирующий доступ к нему, числом 2.

Процесс, желающий занять ресурс, проверяет значение семафора. Если значение положительно, то процесс занимает ресурс и вычитает единицу из значения семафора. По окончании работы с ресурсом процесс добавляет единицу к значению семафора (освобождает ресурс).

Если значение равно нулю, то процесс становится в очередь и вычитает единицу из значения семафора. Есть два варианта системных вызовов, контролирующих доступ к занятому ресурсу по семафору: один мгновенно завершается с определенным кодом ошибки, если ресурс занят, другой блокирует вызвавший его процесс до освобождения ресурса.

Если значение семафора отрицательно, процесс становится в очередь (или, если вызван соответствующий системный вызов, продолжает работу, не получив доступа к ресурсу).

Согласно стандарту POSIX 1003.1b семафор представляет собой глобальную переменную, хранящую значение независимо от того, используется ли она. Так, если последний процесс, работавший с семафором, закрыл его вызовом `sem_close()` при значении семафора 4, то следующий процесс, который откроет этот семафор вызовом `sem_open()`, получит семафор со значением 4. Поэтому рекомендуется инициализировать семафор перед завершением работы с ним.

Контролировать доступ к ресурсу с помощью семафоров можно только в том случае, если все процессы, работающие с этим ресурсом, подчиняются одинаковым правилам использования семафора и работают с ним корректно. Очевидно, что если одни программы проверяют значение семафора перед доступом к ресурсу, а другие – нет, то в работе будут серьезные сбои.

Это напоминает оформление багажа в аэропорту: если вы полагаете, что летите рейсом 5479 в Боготу, а сотрудник авиакомпании отправляет ваш багаж рейсом 576 в Амстердам, не заглянув в билет, то налицо будет некоторая несогласованность ожиданий и реальности. Билет здесь играет роль семафора, значение которого должны одинаково понимать и проверять кассир, вы, сотрудник авиакомпании, служба безопасности и стюардесса.

Кстати, интерфейс системных вызовов, связанных с семафорами, в POSIX 1003.1b намного лаконичнее и проще, чем в классическом API System V или BSD. Поэтому, если вам придется разрабатывать программное обеспечение в системах, не отвечающих стандарту POSIX 1003.1b, лучше не связывайтесь с семафорами. Иначе мы вам не завидуем.

Некоторые серверы приложений, например СУБД Oracle, требуют наличия определенного количества семафоров в ядре UNIX. Это количество контролируется файлом конфигурации ядра. Например, в Solaris

число семафоров определяется в /etc/system (если там нет определения, стало быть, количество семафоров в системе по умолчанию оставлено без изменений).

В Linux, по крайней мере для последних версий ядра, начиная с 2.2, установки семафоров находятся в /usr/src/Linux/include/Linux/sem.h. Играют роль параметры SEMMNI, SEMMSL и SEMOPN. В этот файл надо вносить изменения при желании изменить число семафоров. Делать это следует до перекомпиляции ядра.

Во FreeBSD для использования POSIX-семафоров в файле конфигурации ядра должно быть указано:

```
options "P1003_1B"
```

FreeBSD также поддерживает и IPC System V, а именно совместно используемую (разделяемую) память, сообщения и семафоры. Чтобы они работали, надо добавить следующие строки в файл конфигурации ядра:

```
options    SYSVSHM
options    SYSVSHM          # enable shared memory
options    SYSVSEM          # enable for semaphores
options    SYSVMSG          # enable for messaging
```

Во FreeBSD 3.2 и выше эти параметры уже включены в ядро GENERIC по умолчанию. Количество семафоров должно устанавливаться в секции файла конфигурации, которая называется «System V semaphores and tunable parameters»:

```
options  SEMMAP=31   # размер таблицы семафоров
options  SEMMNI=11   # количество идентификаторов в системе
options  SEMMNS=61   # количество семафоров в системе
options  SEMMNU=31   # количество структур undo в системе
options  SEMMSL=61   # максимум семафоров на идентификатор
options  SEMOPM=101  # максимум операций на один вызов semop
```

Как и в Linux, после изменений в файле конфигурации ядра надо перекомпилировать ядро и установить новое ядро.

В Solaris количество семафоров определяется в /etc/system:

```
set shmsys:shminfo_shmmin=1
set shmsys:shminfo_shmmax=4294967295
set shmsys:shminfo_shmmni=200
set shmsys:shminfo_shmseg=25
set semsys:seminfo_semmns=500
set semsys:seminfo_semmni=70
```

Посмотреть текущие значения этих параметров в Solaris можно командой:

```
sysdef
......
* IPC Semaphores
```

```
*
    10  entries in semaphore map (SEMMAP)
    70  semaphore identifiers (SEMMNI)
   500  semaphores in system (SEMMNS)
    30  undo structures in system (SEMMNU)
   150  max semaphores per id (SEMMSL)
    10  max operations per semop call (SEMOPM)
    10  max undo entries per process (SEMUME)
 32767  semaphore maximum value (SEMVMX)
 16384  adjust on exit max value (SEMAEM)
*
* IPC Shared Memory
*
4294967295      max shared memory segment size (SHMMAX)
     1  min shared memory segment size (SHMMIN)
   200  shared memory identifiers (SHMMNI)
    25  max attached shm segments per process (SHMSEG)
```

Текущие наборы семафоров в Solaris можно просмотреть при помощи команды **ipcs -sb**.

Управление процессами

Команды наблюдения за процессами

Главная команда системного администратора – **ps**. С ее помощью вы получите мгновенный «снимок» таблицы процессов, запущенных в системе. Если дать ее без аргументов, то она выведет список процессов, запущенных текущим пользователем в текущей сессии. Это не очень интересно. Наиболее часто **ps** применяется так:

```
ps -auxw          # BSD version
```

или

```
ps -ef          # System V version
```

Программа **ps** не может иметь аргументов, только ключи, поэтому иногда она «обижается» на пунктуальный «минус» перед ключами. Такая «обидчивость» характерна для BSD-версий программы **ps**. В таких диалектах ее можно запускать так:

```
ps auxw
```

Программа **ps** – редкое исключение в UNIX, так как обычно разница между ключами программ в системах BSD и System V невелика.

С ключами **auxw** или **ef** соответственно **ps** выводит информацию обо всех процессах, запущенных в системе:

```
ps auxw
USER PID %CPU %MEM  VSZ RSS TTY  STAT START TIME COMMAND
```

```
root  1   0.0  0.0  1064 76 ?     S    Jun11 0:21 init [3]
root  2   0.0  0.0   0   0  ?     SW   Jun11 0:00 [keventd]
root  303 0.0  0.0  1136 164 ?    S    Jun11 4:25 syslogd -m 0
root  324 0.0  0.0  1280 124 ?    S    Jun11 0:02 crond
root  337 0.0  0.0  1080  64 ?    S    Jun11 0:00 inetd
root  389 0.0  0.0  1032  4  tty1 S    Jun11 0:00 /sbin
                                        /mingetty tty1
root  390 0.0  0.0  1032  4  tty2 S    Jun11 0:00 /sbin
                                        /mingetty tty2
root  564 0.0  0.3  2128  740 ?   S    Jun11   1:41 dhcpd
root 24561 0.0 8.2 18132 15716 ?  S    Jul30   0:00 /usr/sbin
                                        / squid -sY
filip 24765 0.8 0.6 2152 1284 pts/0 S  00:01   0:00 -bash
```

USER показывает владельца процесса, PID – идентификатор процесса, затем идет загрузка процессора и памяти этим процессом, VSZ (Virtual Size) – общий размер процесса в памяти, RSS (Resident Set Size) – размер той части процесса, которая находится в оперативной памяти, TTY – терминал, с которого запущен процесс, STAT – статус процесса, START – время запуска процесса, TIME – сколько времени процесс отнял у процессора, COMMAND – какой командой был запущен процесс.

В оперативной памяти обычно находится не весь процесс; частично он может быть выгружен на диск. Чем ближе RSS к VSZ, тем быстрее (потенциально) работает процесс. Знак вопроса в поле TTY означает, что процесс запущен не с терминала (при старте системы или через **inetd**).

Процесс может иметь статус:

- R – выполняется (running)
- S – спит (sleeping)
- D – спит беспробудно (uninterruptible sleep)
- T – остановлен (terminated) или трассируется в процессе отладки (traced)
- Z – зомби (zombie)

Вторая буква в поле статуса (W) означает, что процесс не имеет резидентных страниц в памяти.

Третья буква N указывает на то, что процесс имеет положительное значение базового приоритета NICE (NI).

Кроме программы **ps** для наблюдения за процессами удобно использовать программу **top**, она подробно описана в главе 27.

Процессы-зомби

Все статусы достаточно понятны, за исключением беспробудного сна и зомби. Что имеется в виду? Процесс-зомби (zombie) – это процесс, который освобождает ресурсы системы, но не освобождает строчку в таблице процессов. Обычно родительский процесс ждет завершения

дочернего, для чего вызывает функцию **wait()**. Если функция не вызвана вследствие безвременной смерти родительского процесса, а дочерний процесс завершился, то он становится зомби. Процессу-зомби некому передать свой код завершения. Правильнее было бы назвать процесс-зомби «сиротой». Такие процессы берет под свою опеку процесс **init**. Он принимает их код завершения и выполняет все действия для корректного завершения зомби.

Появление процесса-зомби говорит об ошибках в работе родительского процесса (он не дождался завершения своего потомка). Если процессы-зомби множатся в системе и не исчезают со временем, это говорит о сбое в работе **init** или ядра ОС.

При перезагрузке системы процессы-зомби удаляются. Зомби не влияют на работу системы, и можно не обращать на них внимания. Однако они занимают место в таблице процессов и раздражают системного администратора. Статус процесса uninterruptible sleep может говорить об ошибке драйвера устройства или чего-либо в этом роде, такой процесс безнадежно уснул в ожидании конца операции, которой не суждено завершиться. Можно попробовать завершить процесс командой **kill -9** *PID*.

Команды управления процессами

Можно передать процессу сигнал командой **kill**. Имя этой команды связано с тем, что раньше ее применяли для остановки процессов, так как при запуске без ключей она передаст процессу сигнал TERM, требующий остановки процесса. Синтаксис **kill**:

```
kill [-ключи] номер_процесса
```

Пример:

```
ps auxw | grep sendmail      # хотим убить sendmail
root   28059  0.0  0.2  2664  444 ? S  Jul08 3:24 sendmail:
accepting connections       # смотрим номер процесса sendmail
kill 28059                  # посылаем сигнал TERM
kill 28059                  # еще раз, для проверки
kill: (28059) - No such pid # раз такого номера нет,
                            # процесс завершился
```

Однако программу **kill** можно применять и для передачи других сигналов. Для передачи определенного сигнала надо указать в качестве ключа числовое или мнемоническое обозначение этого сигнала. Например, известно, что во многих UNIX демон **inetd** перечитывает свой файл конфигурации, если получает сигнал с номером 1.

```
ps auxw | grep inetd # хотим попросить inetd перечитать
                     inetd.conf, смотрим номер процесса inetd
filip 21336 0.0 0.7 836 348  p1 S    16:57   0:00 grep inetd
root    262 0.0 0.1 776  52  ?  S   Jun 26   0:00 inetd
```

Здесь целых два процесса, в описании которых есть слово `inetd`. Так бывает при выполнении этой команды, поскольку **ps** и **grep** запускаются одновременно, и **ps** показывает текущее состояние таблицы процессов. Однако физически запуск **grep** происходит чуть-чуть позже, чем **ps**, поэтому последняя может завершиться раньше, чем **grep** начнется (ведь работы у **ps** мало – только показать таблицу процессов). В этом случае **grep** в списке найденных процессов не будет. Продолжаем:

```
kill -1 262 # командуем "перечитать файл конфигурации"
```

Как мы отмечали раньше, не все демоны по сигналу HUP (номер 1) перечитывают файл конфигурации. Прежде чем давать демону сигналы, узнайте в **man**, как он на них реагирует.

Посылать сигналы можно только тому процессу, которым вы владеете. Суперпользователь может послать любой сигнал любому процессу.

Посылка сигнала KILL (номер 9) вызывает немедленное завершение процесса:

```
kill -9 PID
```

или

```
kill -KILL PID
```

При выполнении команды **kill** старайтесь задавать мнемонические, а не числовые обозначения сигналов. В разных диалектах UNIX сигналы с одинаковыми номерами могут иметь разные значения, а мнемонические обозначения всегда соответствуют смыслу, который в них вложил разработчик. Так, если вы командуете **kill -HUP** *<PID>*, то даже если в вашем диалекте UNIX сигнал HUP имеет номер 11 вместо 1, он все равно будет верно интерпретирован как сигнал HUP.

В некоторых системах (например, FreeBSD и Linux) есть еще одна полезная программа – **killall**. Это посылка процессу сигнала прерывания по имени процесса. Например, команда **killall portmap** вызывает завершение программы **portmap**.

При запуске процесса можно менять его базовый приоритет. Простой пользователь может только понижать приоритет процесса (то есть делать значение NICE больше, чем установлено по умолчанию). Суперпользователь может менять NICE в обе стороны. Для этого применяется команда **nice**, которая в разных системах имеет разные ключи и немного разный синтаксис. Обычно она выглядит так:

```
nice приращение команда
```

например

```
nice -5 ls
```

По умолчанию это «-5» воспринимается как увеличение значения NICE на 5. Пользователь root может дать команду вида **nice -n -4 ls**.

Это будет воспринято как увеличение базового приоритета, то есть уменьшение значения NICE на 4.

Во многих диалектах можно менять базовый приоритет уже запущенного процесса командой **renice**:

```
renice изменение PID
```

Кроме программы **ps** для наблюдения за процессами можно использовать программу **top**. Подробнее о программе **top** рассказывается в главе 27.

Работа с заданиями

Несколько программ можно запустить параллельно. Каждая из этих программ может породить несколько процессов. Вся совокупность таких одновременно запущенных из shell процессов называется **заданием**. Работа с заданием сводится к приостановке выполнения задания, продолжению выполнения, прерыванию выполнения или его завершению. Задание может выполняться интерактивно или в фоне. Управление заданиями осуществляется средствами командного процессора. Все командные процессоры, кроме **sh**, имеют встроенные команды управления заданиями.

Список заданий можно получить, выполнив команду **jobs**. А список PID процессов, входящих в задание, – **jobs -l**.

Например, запускаем программу **find** для поиска файлов `*.gif`, список найденных файлов перенаправляем в `~/gifs` (то есть в файл `gifs` нашего домашнего каталога):

```
find / -name "*.gif" > ~/gifs &
jobs -l
[1] + 3612 Running          find/-name      "*.gif">~/gifs&
```

То, что в квадратных скобках [ ], и есть номер задания. Задания имеют уникальные номера в пределах текущего сеанса работы с shell. Задание можно перевести в интерактивный режим командой **fg** (foreground):

```
fg  %1 # %1 - номер задания
```

Теперь задание выполняется в интерактивном режиме. Можно остановить его, переведя в режим suspend (sleeping, спящий):

```
<Ctrl>+<z>
….
```

Теперь можно продолжить выполнение задания в фоне командой **bg** (background):

```
bg  %1
```

Команды управления заданиями бывают полезны, когда вы забываете поставить знак & в конце команды, но при этом хотите, чтобы она вы-

полнялась в фоне. Вы можете приостановить выполнение команды по <Ctrl>+<Z>, а затем перевести ее в фон командой **bg**. В разных командных процессорах **bg** и **fg** могут принимать разные варианты аргументов. Знак процента, за которым следует номер задания, все shell трактуют одинаково.

Планирование запуска процессов

at

Иногда системному администратору очень хочется, чтобы компьютер сам выполнил работу за него, причем часа в два ночи. В таком случае нет иного выхода, чем запланировать действие заранее, потребовав у системы его выполнения в точно назначенное время.

Для планирования однократного запуска задач в определенное время предназначена программа **at**. Она работает совместно с демоном **atd**, создавая для него список задач для выполнения. Синтаксис **at**:

at *time date*

Время (time) может быть представлено в форме HH:MM (часы:минуты) или просто словами midnight, noon или teatime (4pm), или временем с суффиксом AM или PM в американском формате. Дата должна быть представлена либо в формате *имя_месяца день [год]*, либо *MMDDYY*, либо *MM/DD/YY*, либо *DD.MM.YY*. Также можно указать время в виде **now+** *n единиц*, где единицы – это **minutes**, **hours**, **days**, **weeks**. Можно указать для времени суффикс **today** или **tomorrow**.

Указание *time* без *date* означает выполнение задачи в текущие сутки, а если *time* сегодня уже прошло, то назавтра в указанное время.

Программа **at** принимает список задач для выполнения во входной поток:

- **at -l** (или **atq**) распечатывает список поставленных в очередь задач;
- **at -r** позволяет удалить задачу по идентификатору, который показывает **at -l**.

В некоторых системах **at -r** называется **atrm**.

Очередь задач для **atd** программа **at** обычно держит в **/var/spool/at**.

Суперпользователь всегда может запустить **at**. Для ограничения этой возможности для обычных пользователей предназначены файлы /etc/at.allow и /etc/at.deny. Если существует файл /etc/at.allow, то с **at** разрешено работать только тем пользователям, которые в нем перечислены. Если /etc/at.allow нет, то выполняется чтение /etc/at.deny и всем, кто в нем НЕ перечислен, разрешается использование **at**. Пустой /etc/at.deny означает, что выполнять **at** можно всем. Если файлов /etc/at.* нет вообще, то выполнять **at** может только **root**. По умолчанию работать с **at** можно всем.

Задание выполняется от имени пользователя, который поставил задание в очередь. Стандартный вывод и стандартный поток ошибок задания **at** высылает почтой этому пользователю. Программа **at** выполняет запланированные задачи в среде /bin/sh, то есть расценивает их как команды для shell.

batch

Программа **batch** работает в паре с **at**, запланированные с ее помощью задачи тоже запускает **atd**. Однако ее почти никто не использует, так как она предназначена для запуска задач с самым низким приоритетом. Применяйте ее только для самых несрочных задач, поскольку на постоянно загруженном компьютере такая задача почти не будет выполняться, а на даже очень незначительно загруженном сервере будет выполняться во много раз медленнее обычного. Ключи **batch** такие же, как и у **at**, синтаксис аналогичный.

cron

Демон **cron** ежеминутно сверяется со своей таблицей запланированных задач: не пора ли выполнить какую-нибудь из них? В некоторых системах демон называют **crond** или как-то похоже. Программа **crontab** облегчает планирование периодических задач. Она подготавливает таблицу для **cron**.

У каждого пользователя своя таблица **crontab**. Ее можно отредактировать командой **crontab -e**. По этой команде вы окажетесь в редакторе **vi**, а в качестве файла для редактирования получите вашу таблицу **crontab**. Если ее раньше ничем не заполняли, то файл будет пустым.

Формат таблицы **crontab**:

```
mm        hh        dd        mo        dw        command
```

- mm – минуты от 0 до 59
- hh – часы от 0 до 23
- dd – числа от 1 до 31
- mo – месяцы от 1 до 12
- dw – дни недели от 0 до 7, 0 и 7 – это воскресенье
- command – команда, которую надо запустить

Каждое числовое поле в таблице может содержать число, последовательность чисел, разделенных запятыми, диапазон чисел (два числа через дефис) или звездочку, обозначающую «каждую» единицу времени. Например, звездочка в поле часов будет означать «каждый час».

Если в строке указаны конкретные и число месяца, и день недели, то для выполнения команды достаточно, чтобы наступило либо это число, либо этот день недели.

Например, строка

```
45 12 * * 0 rm -rf /var/cache/logs/*.[0-9]
```

означает, что в 12:45 каждое воскресенье надо удалять старые протокольные файлы `*.[0-9]` из `/var/cache/logs`. Однако если было бы написано:

```
45 12 20 * 0 rm -rf /var/cache/logs/*.[0-9]
```

это значило бы «в 12:45 по воскресеньям и двадцатым числам», а не «в 12:45 двадцатого числа, если оно приходится на воскресенье».

Команда запускается от имени владельца таблицы, в качестве shell по умолчанию выбирается `/bin/sh`.

В первых строках таблицы могут быть установлены переменные среды окружения. Наиболее полезна может оказаться переменная среды окружения `PATH`:

```
PATH=/usr/bin:/usr/sbin:/usr/local/bin
```

В ней можно задать каталоги, которые по умолчанию не входят в `PATH`.

Кроме `PATH` можно установить переменные `SHELL` и `HOME`, а также `MAILTO`. По умолчанию `HOME` (домашний каталог) берется из `/etc/passwd`. Переменная `LOGNAME` устанавливается автоматически при запуске команды, но ее нельзя переопределить. Ей присваивается имя пользователя, от имени которого запускается команда. В BSD-системах она может называться `USER`. `LOGNAME` и `USER` нельзя переопределить, `HOME` и `SHELL` – можно. **Cron** пошлет сообщение владельцу таблицы **crontab**, если найдет повод это сделать (например, произошла ошибка выполнения команды). Если переменная `MAILTO` существует и имеет некое значение, то **cron** воспримет его как имя пользователя, которому надо послать письмо вместо владельца **crontab**. Если переменная определена, но ее значение – пустая строка, то почта послана не будет. Установка любой переменной среды окружения в таблице **crontab** осуществляется так:

```
имя_переменной = значение
```

Пробелы вокруг знака равенства могут отсутствовать, и любые последующие пробелы после значения будут трактоваться как часть значения. Значение может быть заключено в апострофы или кавычки для явного указания пробелов в начале и конце значения.

Для просмотра таблицы **crontab** применяется **crontab -l**, для удаления таблицы – **crontab -r**. Можно создать таблицу **crontab** заранее и установить ее в систему командой **crontab** *имя_файла_с_таблицей*. Но этого лучше не делать, потому что старое содержимое таблицы потеряется навсегда. Имеет смысл по крайней мере записать его резервную копию командой:

```
crontab -l > имя_резервной_копии
```

Регулярно выполняемые системные скрипты

Разработчики системы разумно предположили, что надо помочь системному администратору. Есть задачи, которые необходимо выполнять в UNIX независимо от диалекта, версии и назначения системы. Поэтому такие решения подобных общих задач были оформлены в виде скриптов. Эти скрипты системе предписано выполнять ежедневно и еженедельно соответственно. Запускает их на выполнение **cron**. Во FreeBSD они называются **/etc/daily**, **/etc/weekly**. В них заранее записаны некоторые необходимые команды. Можно отредактировать эти файлы, добавить в них что-то необходимое именно вашей системе. Результатом работы этих скриптов по умолчанию являются, в частности, письма-отчеты системному администратору, которые высылаются пользователю root.

В RedHat ежечасное, ежедневное и еженедельное выполнение задач организуется по-другому: необходимые скрипты просто помещаются в каталоги:

```
/etc/cron.hourly/
/etc/cron.daily/
/etc/cron.weekly/
```

В некоторых диалектах UNIX эти скрипты размещаются в каталогах `/var/admin/cron` или `/etc/periodic`. В этих каталогах может быть организована структура подкаталогов, например:

```
hourly/
dayly/
weekly/
monthly/
```

Такая же схема размещения скриптов принята в Solaris, в свежих версиях FreeBSD и даже в Mac OS X (последняя выполнена на базе FreeBSD).

Имеет смысл ежечасно проверять размер больших файлов протоколов, которые следует вовремя архивировать (например, протокол демона **squid** в сильно загруженных прокси-серверах), ежедневно – размер очереди почтового сервера (не слишком ли там много писем?), права доступа к файлам (не появились ли новые файлы с битом suid?), заполненность диска (есть ли свободное место?), еженедельно – состояние очереди почтового сервера, статистику использования сети и почтовой системы, ежемесячно надо архивировать файлы протоколов.

Кроме этого в любой системе можно запланировать ежедневные, еженедельные и ежемесячные задачи с помощью обычного **crontab** для пользователя root.

7

Работа с текстами

Термины и кодировки текста

При работе с текстом в UNIX принято называть строкой любую последовательность символов, заканчивающуюся символом перевода строки (<Ctrl>+<J>, десятичный код ASCII 12, иногда отображается как ^J). Подстрокой называют любую часть строки.

В UNIX конец строки обозначается одним символом, в отличие от Microsoft Windows и DOS, где конец строки обозначается двумя символами («возврат каретки» и «перевод строки» – ^M^J).

Такое различие связано, скорее всего, с тем, что при разработке UNIX программисты думали о том, как сделать систему эффективнее, а затраты на ее использование – меньше. Ресурсы аппаратуры тогда были весьма ограничены, и экономное расходование дискового пространства представлялось практичным и разумным. Современный подход, при котором расточительность новой операционной системы заставляет потребителя покупать все новые и новые компьютеры и диски, выгоден не потребителю, а только поставщику оборудования.

Представляете, сколько места можно сэкономить, если хранить тексты в формате, принятом в UNIX! Если ваша диссертация содержит 160 страниц по 25 строк в каждой, то экономия на «возвратах каретки» составит 4000 символов, почти 4 Кбайт. А если на диске хранится полная документация по UNIX (несколько десятков тысяч страниц), то можно сэкономить один-два мегабайта лишь на менее расточительной кодировке текста. Храните тексты в форматах UNIX, экономьте мегабайты!

При копировании текста между разными ОС (например, DOS и UNIX) с помощью обычных программ копирования (например, `cp`) текст не

переформатируется автоматически. Это же относится и к передаче текстовых файлов по `ftp`. Если текстовый файл скопировать из UNIX в DOS (Windows), то получится одна длинная строка, а если из DOS (Windows) в UNIX, то каждая строка будет заканчиваться символом ^M. Последнее, однако, не будет видно при выдаче текста на экран командой **cat** или **more**, поскольку символ ^M непечатный. Однако если вызвать для редактирования такого файла редактор **vi**, то он покажет эти символы в конце строк.

Для вставки символа ^M перед копированием файла из UNIX в DOS (Windows) можно использовать команду **unix2dos**, а после копирования из DOS (Windows) в UNIX – команду **dos2unix**. Эти команды обычно представляют собой символьные ссылки на одну и ту же программу, которая при старте проверяет, с каким именем ее вызвали, и выполняет соответствующее имени действие. В разных диалектах UNIX эта программа может носить разные имена. Несколько лет назад существовала программа **todos**, написанная в КИАЭ РАН (Курчатовском институте атомной энергетики). Она не только вставляла или убирала ^M, но и переводила русский текст из одной кодировки в другую. Если нужную программу не найти, можно либо воспользоваться ее аналогом (например, **grep**), либо поискать справку – **man -k dos**. Обычно «dos» – хорошее ключевое слово для поиска такой программы.

В UNIX принято кодировать текст, пользуясь кодовой таблицей ASCII. Кодировка символов национальных алфавитов в разных диалектах UNIX может быть разной. Коммерческие поставщики UNIX обычно предлагают собственные решения локализации систем. Никаких проблем с хранением или передачей текстов с символами из второй половины таблицы ASCII в UNIX не возникает. Будут ли они корректно отображаться на вашем терминале – вопрос правильности самой терминальной программы или знакогенератора драйвера монитора (если вы работаете за виртуальным или физическим терминалом). Более подробно о русификации систем UNIX рассказано в главе 25.

Случай с порнографическими изображениями

Однажды ко мне пришел студент, пользователь нашей сети, с жалобой, что его раздражает коллега, сидящий рядом с ним в компьютерном классе, тем, что постоянно просматривает порнографические веб-сайты в учебное время. Разумеется, такое времяпрепровождение в учебном классе не поощряется. Поэтому я отправился в класс попросить нарушителя прекратить смущать соседей и получать удовольствие за казенные деньги (входящий трафик оплачивает учебное заведение). Однако когда я пришел, студент уже углубился в изучение какого-то научного документа. Естественно, он заявил, что и не думал просматривать порнографию в классе.

«Обманывать нехорошо», – подумал я и решил выяснить, кто из двух студентов меня обманул – тот, что жаловался на коллегу, или тот, кто якобы смотрел порнографические картинки.

Во многих организациях все исходящие http-запросы проходят через кэширующий proxy-сервер, который ведет протокол обращений к нему (подробнее о таких серверах сказано в главе 20). У нас было сделано именно так, и я углубился в изучение протокола нашего proxy-сервера.

Системный администратор подобен птице, отыскивающей съедобное зерно в куче мусора. Мне надо было проанализировать огромный протокол работы, чтобы найти требуемую запись. Однако с помощью программы **grep** я довольно быстро отыскал список последних запросов с компьютера, на котором работал любитель «клубнички». Студент, который пожаловался на соседа, оказался прав: почти все запросы направлялись к сайтам с «говорящими» названиями типа *http://www.sex-files.nl*. О характере запрашиваемой информации говорило и то, что запрашивались только файлы `jpg` объемом 50–100 Кбайт, что характерно для фотографий. Обманщик был изобличен и предупрежден о том, что его поведение неэтично.

В дальнейшем нам пришлось принять технические меры к тому, чтобы ограничить доступ к порносайтам из учебного класса. К сожалению, полностью запретить его технически невозможно, и приходится полагаться в равной степени на программное обеспечение и порядочность пользователей.

Зачем системному администратору работа с текстом

Как видно из предыдущего примера, задачи, связанные с анализом больших файлов протоколов, возникают в работе системного администратора постоянно. Чтобы облегчить этот труд, применяют программы-фильтры для работы с текстом. Наиболее известны программы **grep**, **sed**, **tr**. Нередко для той же цели применяют скрипты, написанные на **awk** или **perl**.

Для создания и модификации текстовых файлов в UNIX (в частности, файлов конфигурации) принято использовать текстовый редактор **vi** (призносится «ви-ай», хотя русскоязычные специалисты иногда предпочитают жаргонное «ви»). Есть и другие более совершенные редакторы под UNIX, однако **vi** хорош тем, что поставляется с любой системой UNIX независимо от диалекта и версии.

Работая с графическими оболочками и программами администрирования системы с оконным интерфейсом, вы можете неожиданно оказаться в среде редактора **vi**, если пожелаете изменить настройки какой-нибудь подсистемы UNIX. Таковы, например, условия работы в SAM (System Administration Manager) в HP-UX. Имеет смысл заранее по-

знакомиться с правилами работы в **vi**, так как они несколько непривычны для тех, кто привык к Microsoft Word, Notepad или даже MultiEdit.

До того как Билл Джой (Bill Joy) в 1976 году написал **vi**, тексты в UNIX редактировали в **ed**. Это был не оконный, а строчный редактор, то есть редактируемый текст не отображался на экране. Можно было просмотреть лишь его часть, а для редактирования была доступна только одна текущая строка. Сейчас этот редактор не применяется. Однако его команды унаследовал потоковый редактор **sed**, который иногда бывает весьма удобен. Например, для замены одной подстроки другой подстрокой сразу в нескольких файлах. Поэтому в описании редактора **vi** будет уделено внимание режиму совместимости с редактором **ed**. В этом режиме можно вводить команды редактора **ed**, и **vi** будет их выполнять так же, как выполнил бы **ed**.

Отличные от **vi** редакторы: **ee**, **emacs**, **jstar**, встроенный редактор Midnight Commander (mc).

Последний работает под X Window (то есть имеет графический оконный интерфейс), остальные – в текстовом режиме. **Emacs** считается одним из самых мощных текстовых редакторов в мире, представляет собой элегантную интегрированную среду работы и напоминает MultiEdit под DOS или Windows. **Emacs** получил довольно широкое распространение за рубежом, но в России я не встречал людей, которые бы с ним работали.

Редактор vi

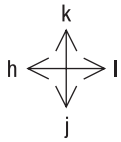
Vi может находиться в одном из двух режимов – командном режиме или режиме ввода. При запуске редактор начинает работу в первом из них. В этом режиме все, что пользователь набирает на клавиатуре, интерпретируется как команда. Команды в **vi** короткие: почти все состоят из одной или двух букв. Некоторые команды редактора **vi** отображаются в служебной строке. Служебной строкой в **vi** считается последняя строка экрана. Большинство команд редактора никак не отображаются, хотя выполняются. Это не очень удобно, если у пользователя дрожат руки или он не помнит, какую клавишу только что нажал. Так можно легко стереть текст или модифицировать его неожиданным образом. Избегайте случайных нажатий на клавиши в командном режиме!

Vi отображает вводимую команду в служебной строке, если это команда поиска или если команда вводится в режиме совместимости с редактором **ed**. Обычно служебная строка используется для текста файла наравне с другими строками. При отображении в ней служебной информации текст, который до этого отображался в ней, никак не меняется, несмотря на то что временно исчезает с экрана.

Для перехода в режим ввода дайте команду **a**, **i** или **o**. После этого текст можно будет вводить соответственно в позиции курсора, в позиции, следующей за позицией курсора, или в начале новой строки, которая появится под текущей строкой. В режиме ввода нажатия всех клавиш, за исключением клавиши <Esc>, интерпретируются как символы набираемого текста. По нажатию клавиши <Esc> выполняется переход из режима ввода в режим команд.

В режиме ввода не всегда можно пользоваться клавишами передвижения по тексту (стрелками, <PgUp>, <PgDn>, <Home>, <End> и т. д.). Если терминал настроен не совсем корректно, то нажатие, например, клавиши «стрелка вверх» **vi** может воспринять как нажатие <Esc>, за которым следует [B. Поэтому может случиться так, что, нажав клавишу «стрелка вверх» в режиме ввода, вы незаметно для себя перейдете в режим команд. Если продолжить ввод текста, то он будет воспринят как команда. Таким образом, иногда можно передвигаться по тексту в режиме ввода, но лучше сначала выйти в командный режим, затем перейти по тексту в нужное место и вернуться в режим ввода.

Переход по тексту в командном режиме выполняется клавишами передвижения по тексту (стрелками, <PgUp>, <PgDn>, <Home>, <End>), а также <Ctrl>+<F> (forward, на страницу вперед), <Ctrl>+<B> (backward, на страницу назад). Бывает, что клавиши-«стрелки» не работают. И тогда вместо них можно задействовать клавиши <h>, <j>, <k>, <l> следующим образом:



Эти клавиши были выбраны только потому, что они расположены рядом, а буквы на них не соотносятся с английскими словами, которые можно было бы ассоциировать с передвижениями вверх-вниз, вправо-влево.

В табл. 7.1 перечислены основные команды редактора **vi**. Полный список можно найти в руководстве по **vi** (**man vi**), в Интернете (например, по адресу *http://asu.pstu.ac.ru/book/unix/vi/*) или в книге Питера Дайсона «Операционная система UNIX. Настольный справочник». – М.: Лори, 1997.

Все команды, начинающиеся с двоеточия, – это команды режима совместимости с редактором **ed**. Ввод двоеточия интерпретируется **vi** как переход к этому режиму. Введенное двоеточие сразу отобразится в начале служебной строки, и оставшуюся часть команды вы будете набирать, видя ее в этой строке.

Таблица 7.1. Основные команды редактора vi

Команда	Действие
a	перейти в режим ввода, начать ввод в позиции, следующей за позицией курсора
i	перейти в режим ввода, начать ввод в позиции курсора
o	перейти в режим ввода, добавить пустую строку под текущей строкой и начать ввод в новой строке
	операции с буфером
dd	вырезать текущую строку в буфер
x	вырезать текущий символ в буфер
u	отменить последнее действие
yy	копировать текущую строку в буфер
p	вставить строку из буфера под текущей
	навигация по тексту
0	перейти в начало строки
$	перейти в конец строки
j	перейти на строку вниз
k	перейти на строку вверх
h	перейти на символ влево
l	перейти на символ вправо
	выйти из редактора, записать файл
:q	выйти из редактора
:q!	выйти без записи
:w	записать изменения
:wq	записать и выйти
:w!	записать, даже если нет права записи в файл (срабатывает, если вы – root)
:w *имя файла*	записать в файл с другим именем, в дальнейшем будет считаться, что редактируется именно этот файл с другим именем
	поиск и замена
/	поиск вперед
?	поиск назад
:s/что/на что/[g]	поиск и замена
	прочее
<Esc>	перейти в командный режим
rn	заменить символ в позиции курсора на n

Таблица 7.1 (продолжение)

Команда	Действие
%	перейти к парной скобке
^G	сообщить имя файла и местоположение в нем

Моя самая любимая команда редактора vi – это %. Нет слов, насколько удобно искать таким образом парные скобки в программах на C, и даже в файлах конфигурации UNIX это помогает быстро отыскать ошибки.

Перед любой командой **vi**, где это осмысленно, за исключением команд, начинающихся с двоеточия, можно набрать число, которое будет интерпретироваться как требование повторить следующую за ним команду это число раз. Например,

```
15dd
```

означает вырезать в буфер пятнадцать строк, начиная с текущей.

Важная часть команд – команды поиска и замены. Обычный поиск выполняется командой **/***образец*, то есть знак «слэш», за которым следует образец для поиска. При поиске назад по файлу вместо слэша указывается вопросительный знак. Продолжение поиска – слэш или вопросительный знак без образца. Если в образце встретится слэш или вопросительный знак, **vi** сочтет их ограничителем образца и проигнорирует остаток образца. Поиск и замена выполняется командой **:s/***образец***/***на_что_менять***/**. В ней указывается образец для поиска и строка, которой следует его заменить. После завершающего слэша может стоять модификатор **g**. Если его не поставить, заменен будет только первый образец в строке. Если в строке есть еще подстроки, отвечающие образцу, они останутся нетронутыми. Если поставить модификатор **g**, будут заменены все подстроки, отвечающие образцу.

Все команды режима совместимости с редактором **ed** могут быть предварены выражением *n,m*, где *n,m* – номера строк, ограничивающих диапазон выполнения команды. Если такого выражения нет, команда выполняется для текущей строки. Например,

```
:23,33s/black/white/g
```

означает, что требуется заменить все вхождения «black» на «white» во всех строках с 23 по 33 включительно. В выражении, описывающем диапазон, допустимы числа и символы «.» и «$», а также арифметические выражения. Символ «.» обозначает текущую строку, символ «$» – последнюю строку файла. Например:

```
:.,.+7s/black/white/g
```

означает замену «black» на «white» в восьми строках, начиная с текущей. Допустим, мы находимся в первой строке. Тогда выражение

```
.,.+7
```

фактически означает

```
1,1+7
```

то есть

```
1,8
```

Команда

```
:.,$s/yellow/blue/g
```

означает замену «yellow» на «blue» во всех строках, начиная с текущей и до конца файла.

В режиме совместимости можно не только выполнять команды поиска и замены. Здесь есть еще команда удаления – **d**. Команда

```
:1,.d
```

удалит все строки с первой по текущую.

Диапазон может состоять из одной строки, например, команда

```
:4d
```

означает требование удалить четвертую строку.

Ввод числа за двоеточием без всяких команд означает переход к строке с указанным номером, например:

```
:56
```

вызовет переход к 56-й строке.

Программы поиска и замены в текстах

grep

Если вы – системный администратор UNIX-системы, то собьетесь, пытаясь сосчитать, сколько раз в день используете **grep**. Эта программа чаще всего применяется при анализе файлов протоколов, выделении нужных процессов из большого списка и т. п. Работает **grep** так: если ей передан один аргумент, она берет исходный текст из входного потока. Если аргументов два и более, она считает все аргументы, начиная со второго, именами файлов, из которых она будет брать исходный текст. Первый аргумент **grep** – это подстрока, которую **grep** ищет во всех строках исходного текста. Обычно каждая успешно сопоставленная строка копируется на стандартный вывод; если исходных файлов

несколько, перед найденной строкой выдается имя файла. Подстрока может содержать регулярные выражения (подробно регулярные выражения рассмотрены в разделе «Регулярные выражения» в этой главе).

В дистрибутиве FreeBSD и Linux поставляется GNU **grep**, который по умолчанию поддерживает расширенные регулярные выражения. В тех диалектах, где установлена другая версия **grep**, надо запускать **egrep**, чтобы искать подстроки с помощью расширенных регулярных выражений.

Есть еще один вариант **grep** – **fgrep**, программа, выполняющая буквальный поиск подстроки. Программа **fgrep** просматривает входные файлы в поиске строк, содержащих заданную подстроку. Отличие команды **fgrep** от **egrep** и **grep** в том, что она выполняет сопоставление с подстрокой в точности так, как она написана, а не с шаблоном, заданным регулярным выражением; **fgrep** более быстра и компактна. Программа **fgrep** интерпретирует специальные символы регулярных выражений $, *, [ ], ^, |, ( ) и \ буквально, а не как элементы регулярных выражений. Однако для экранирования этих символов от интерпретации командным процессором проще всего заключать подстроку в вызове **fgrep** в апострофы.

Все программы семейства **grep** имеют одинаковые ключи. Ключ **-v** указывает, что выдавать надо только строки, **не** содержащие искомую подстроку, а ключ **-f** *файл* требует прочесть искомые подстроки из отдельного файла. Обратите внимание: так можно искать несколько разных подстрок в тексте!

При работе с версией **grep** от GNU (например, в Linux и FreeBSD) вызов **egrep** почти эквивалентен вызову **grep -E**, а **fgrep** строго эквивалентен **grep -F**.

При задании нескольких файлов в командной строке **grep** по умолчанию выдает в выходной поток найденные строки, предваряя их именем файла, в котором они нашлись. Ключ **-h** подавляет вывод имен файлов в начале строки.

Ключ **-n** требует от **grep** выводить номер строки в файле перед найденной строкой.

Ключ **-r** задает рекурсивное чтение всех файлов каждого подкаталога, начиная с текущего.

Ключ **-i** требует от **grep** игнорировать регистр букв при поиске.

Ключ **-s** (поддерживается не всегда) нужен для подавления вывода сообщений о нечитаемых файлах или каталогах. Это возможно при рекурсивном поиске по дереву каталогов, в котором есть недоступные каталоги и файлы.

Пакетный редактор sed

Для однотипного редактирования нескольких файлов можно использовать пакетный редактор **sed**. Он также годится для быстрого внесения однотипных исправлений в десятки или тысячи строк в одном файле. Программа **sed** читает свой стандартный ввод, применяет переданное ей в качестве аргумента правило редактирования к каждой строке и выдает в стандартный вывод результат. Как и **grep**, **sed** принимает в качестве второго и последующих аргументов имена файлов, которые нужно читать, вместо входного потока. Программа **sed** ничего не меняет в самих файлах, которые она читает. Если вы хотите модифицировать файл с помощью **sed**, например изменить все вхождения подстроки 'abc' на подстроку 'xyz', то надо воспользоваться промежуточным файлом как хранилищем модифицированного текста:

```
sed 's/abc/xyz/g' < исходный_файл  > временный_файл
mv -f временный_файл исходный_файл
```

Команды редактора **sed** ведут свой род от **ed**, поэтому здесь работают любые команды режима совместимости редактора **vi**. Команда, переданная **sed**, автоматически применяется к каждой строке входного потока или файла.

Однако если в команде указан конкретный номер строки, то она будет применяться лишь к строке с этим номером. Помните, что номером строки может быть не только число, но и выражение /*подстрока*/, вместо которого подставляется номер строки, в которой впервые встречается указанная *подстрока*. Кроме этого номер строки может быть знаком «$» и указывать на последнюю строку. Указатель текущей строки (знак «точка») в **sed** не имеет смысла, так как все строки файла по очереди становятся текущими.

В **sed** можно в подстроке, на которую требуется заменить искомую подстроку, использовать символ «&». Он будет заменен на исходную подстроку. Например, для замены «htm» на «html» можно дать команду

```
sed 's/htm/&l/g' исходный_файл > результат
```

Транслитератор tr

Транслитерация – это замена одних символов другими. Смена регистра букв в слове – типичный пример транслитерации. С помощью **tr** можно менять одни символы в тексте на другие, например маленькие буквы на большие:

```
tr "[a-z] [A-Z]" < исходный_файл > результат
```

Регулярные выражения

Регулярные выражения представляют собой шаблоны для поиска и замены, в которых используются символы со специальными значениями. Регулярные выражения также могут использоваться в программах на Perl, Java, VisualBasic.Net, C#, потому что их поддержка реализована в этих языках.

По-моему, раздел «Регулярные выражения» – один из самых полезных в этой книге, потому что многие задачи системного администрирования без регулярных выражений не решить.

Существуют основные (basic) и расширенные (extended) регулярные выражения. Обычно по умолчанию программа полагает, что ей в качестве аргумента передано базовое регулярное выражение. Однако в некоторых случаях программа считает по умолчанию, что получила расширенное выражение. Например, GNU-реализация программы **grep** (GNU grep) по умолчанию работает с расширенными регулярными выражениями.

Разные типы регулярных выражений отличаются синтаксисом (применяемыми спецсимволами) и функциональностью (возможностями задать более сложное условие для поиска). В GNU-версии **grep** базовые и расширенные регулярные выражения имеют одинаковую функциональность и синтаксис, в отличие от подавляющего большинства программ.

GNU **grep** поставляется с дистрибутивами Linux и FreeBSD. С коммерческими диалектами UNIX (Solaris, HP-UX) поставляется другая версия **grep**. Она не удовлетворяет стандарту XPG4 (The X/Open Portability Guide, Issue 4 – XPG4), и поэтому другая некоммерческая и отвечающая этому стандарту версия также входит в комплект поставки Solaris и хранится в каталоге `/usr/xpg4/bin`.

Вначале мы рассмотрим расширенные регулярные выражения, после чего будут описаны их отличия от базовых. В расширенных регулярных выражениях используют символы со следующими значениями:

- **^** – начало строки
- **$** – конец строки
- **[a-z]** – любой символ из диапазона
- **[^a-z]** – любой символ, за исключением символов из диапазона
- **[abil]** – любой символ из набора
- **[^azx]** – любой символ, за исключением символов из набора
- **.** – любой символ
- ***** – любое число повторений символа, стоящего перед *
- **+** – любое ненулевое число повторений символа, стоящего перед +

- **?** – один символ, стоящий перед ?, или пустая подстрока (ноль таких символов)
- **&** – в выражении, указывающем подстроку, на которую надо заменить найденную, означает повторение найденной подстроки

Набор символов, заключенный в квадратные скобки, соответствует одному любому символу из набора, а если первый символ набора – «^» (крышка), то такая конструкция соответствует любому символу, не входящему в набор. Например, [0123456789] означает «любая цифра». Можно указать диапазон символов, поставив знак «-» (минус) между первым и последним символом диапазона. В диапазоне недопустим последний символ, стоящий в таблице символов ASCII до первого.

Для облегчения указания диапазонов есть предопределенные классы символов с «говорящими» названиями: **[:alnum:]**, **[:alpha:]**, **[:cntrl:]**, **[:digit:]**, **[:graph:]**, **[:lower:]**, **[:print:]**, **[:punct:]**, **[:space:]**, **[:upper:]** и **[:xdigit:]**. Например, **[[:alnum:]]** означает [0–9 A–Z a–z]. Впрочем, диапазон [0–9 A–Z a–z] корректен при условии, что английский язык по умолчанию задан в настройке языка системы, и годится только для таблицы ASCII. Форма **[[:alnum:]]** независима от языка системы и кодовой таблицы, поэтому она более универсальна. В скриптах такая форма предпочтительнее, если их планируется переносить на другую платформу. Внутренняя пара скобок в **[[:alnum:]]** представляет собой часть названия класса символов и должна быть обрамлена внешней парой, указывающей, что это диапазон. Большинство специальных символов регулярных выражений теряют свое специальное значение в диапазонах. Для того чтобы включить в диапазон символ «]», поместите его куда угодно, кроме конца диапазона, для включения «^» поставьте его куда угодно, кроме начала диапазона, а для включения символа «-» поместите его в конец диапазона.

Символ «.» (точка) означает любой символ. Конструкция **\w** – синоним **[[:alnum:]]**, **\W** – **[^[:alnum:]]**. «Крышка» ^ и «доллар» $ вне диапазона обозначают начало и конец строки.

\b означает границу слова, **\B** – отсутствие границы слова. Например, если есть файл qq, содержащий строки

```
west
east
best
qwerty,
```

в результате выполнения команды **grep '\bw' qq** будет выведено

```
west,
```

а команда **grep '\Bw' qq** даст

```
qwerty,
```

потому что в слове «west» «w» стоит на границе слова, а в слове «qwerty» – нет.

За регулярным выражением может следовать один из операторов повторения:

- **?** – означает, что предыдущий элемент выражения повторяется один раз
- ***** – предыдущий элемент повторяется неотрицательное число раз (ноль и более)
- **+.** – предыдущий элемент выражения повторяется положительное число раз (один и более)
- **{***n***}** – предыдущий элемент выражения повторяется ровно *n* раз
- **{***n***,}** – предыдущий элемент выражения повторяется *n* или более раз
- **{***n***,***m***}** – предыдущий элемент выражения повторяется как минимум *n* раз, но не более *m* раз

Регулярные выражения могут быть объединены оператором «|» (вертикальная черта); такое выражение соответствует любой строке, отвечающей любой его части.

Оператор повторения имеет большую силу, чем объединение выражений. Часть выражения может быть объединена в скобки, чтобы пересилить это правило.

Выражение \q, где q – любая цифра, означает подстроку, ранее нашедшую соответствие с выражением в q-тых скобках регулярного выражения.

В основных (basic) регулярных выражениях символы ?, +, {, |, (и) теряют свое специальное значение. Классический **egrep** не воспринимает «{» как спецсимвол.

В GNU **grep -E** трактует «{» как спецсимвол, если он встречается в выражении {2}, но не считает его таковым, если выражение может вызвать синтаксическую ошибку, как в выражении {1. Например, **grep -E '{1'** ищет строку {1 вместо того, чтобы указать на ошибку выражения. В GNU **egrep** ведет себя как классический **egrep**, игнорируя специальное значение {}.

При поиске регулярное выражение считается соответствующим первой отвечающей ему подстроке. Если таких подстрок в строке несколько или последующая начинается до того, как закончилась предыдущая, первой все равно будет найдена та подстрока, которая начинается раньше.

В табл. 7.2 приведены примеры регулярных выражений и соответствующих им подстрок.

Таблица 7.2. Примеры регулярных выражений

Выражение	Соответствует	Не соответствует
a.*b	ab, auub	ba
a*b	aaaab, ab, b	aub
a[0-9]b	a0b, a1b	a00b, ab
a.+b	a0b	ab
a.?b	ab, aub	auub
[^a-z]b	0b	az, ab, zb

При указании диапазона считается, что диапазон символов соответствует таблице ASCII. Первый символ диапазона должен предшествовать в этой таблице последнему символу диапазона, поэтому диапазон [a–Z] будет считаться неверным, так как заглавные буквы в ASCII расположены раньше строчных. Помните, что диапазон репрезентирует один символ. Если надо указать, что в подстроку входит последовательность символов из диапазона, следует использовать диапазон, за которым стоит звездочка, например [a-z]*.

С помощью регулярных выражений можно искать подстроки, содержащие специальные символы ([,*,.,],?,^); для этого применяется символ экранирования «\» (обратный слэш). Этот символ экранирует следующий за ним спецсимвол от интерпретации. Поэтому для поиска подстроки 2*2 регулярное выражение должно иметь вид `2\*2`, а для подстроки C:\ – `C:\\` (первый обратный слэш экранирует следующий за ним от интерпретации в качестве спецсимвола).

Если в вашей системе работают не все из приведенных примеров, попробуйте выяснить, какой набор выражений (базовый или расширенный) используется по умолчанию. Иногда достаточно дать программе **grep** ключ, указывающий, что надо работать с расширенным набором. Подробнее этот аспект освещен в описании **regexp(3)**, **grep(1)**. О регулярных выражениях можно рассказать намного подробнее, здесь мы привели лишь самые основные примеры и самые популярные шаблоны. За подробностями обращайтесь к дополнительной литературе.

Дополнительная литература

[4] Jeffrey E. F. Friedl. Mastering Regular Expressions, 2nd Edition. – O'Reilly & Assosiates, 2002.[1]

[1] Фридл. Д. «Регулярные выражения. Библиотека программиста». – СПб: Питер, 2001.

8

Командные процессоры

Обзор командных процессоров

Интерпретатор команд пользователя в UNIX называется командным процессором (shell). Есть несколько вариантов интерпретаторов команд. Самый первый был написан Стивом Баурном (Steve Bourne) в 1976 году (по другим данным, в 1974), когда Стив работал в Bell Laboratories в AT&T. Затем в Berkeley был написан C-shell (csh, произносится «си шелл»), а потом количество командных процессоров быстро выросло. Наиболее известны Korn shell (ksh) Дэвида Корна (David Korn), Bourne again shell (bash), разработанный в рамках проекта GNU (GNU's Not UNIX), и tcsh – расширенная версия csh.

C-shell был разработан студентом университета в Беркли (University of California Berkeley) Биллом Джоем (Bill Joy), который до поступления в Berkeley закончил Университет Мичигана в Энн-Арборе (University of Michigan, Ann Arbor), а по окончании Berkeley основал компанию Sun Microsystems. Билл Джой также написал редактор **vi** и был одним из авторов первой версии BSD UNIX.

Shell (командный процессор) не только интерпретирует команды, но и выполняет ряд важных функций, которые суммированы в табл. 8.1. Вот почему называть его «интерпретатором команд» не совсем корректно. Термин же «оболочка» не прижился в русскоязычной литературе по UNIX, и это очень хорошо, поскольку оболочкой сейчас называют почти все, что угодно. С другой стороны, вся работа shell заключается в исполнении команд. Поэтому мы будем употреблять хотя и длинный, но правильный термин «командный процессор».

Таблица 8.1. Популярные командные процессоры UNIX

Название	Имя в системе	Комментарий
Bourne Shell	sh	Самый первый командный процессор, немного примитивный, зато идеальный для скриптов.
C Shell	csh	По сравнению с sh, в csh добавлены некоторые возможности, такие как управление заданиями и редактирование командной строки. Однако он плох для скриптов и содержит немало ошибок.
Korn Shell	ksh	Совместим с sh по командам и содержит ряд возможностей csh. Добавлена возможность редактирования команд, введенных ранее (history).
Bourne Again Shell	bash	Похож на ksh по возможностям, но имеет несравнимо более удобный интерфейс. В России наиболее популярен, стандарт де-факто в Linux.
tcsh	tsch	Расширенная версия csh с добавлениями полезных функций из ksh и bash, написана в Cornell University.

Командные процессоры выполняют следующие задачи в системе:

- интерпретацию команд пользователя, в том числе разбор командной строки;
- запуск программ;
- организацию перенаправлений потоков между процессами;
- интерпретацию языка скриптов и их выполнение;
- управление заданиями;
- интерпретацию шаблонов имен файлов;
- подстановку имен файлов в командную строку.

Не все командные процессоры умеют выполнять все перечисленные задачи, а некоторые умеют выполнять одну из задач лучше, чем это получается у других.

По табл. 8.1 можно сравнить возможности командных процессоров. Более подробно о выполняемых функциях рассказано в этой главе ниже.

В некоторых диалектах UNIX команда `csh` представляет собой символическую ссылку на `tcsh` (то есть, вызвав `csh`, мы на самом деле вызовем `tcsh`). В других системах вызов `sh` приводит к автоматическому вызову `ksh`. Обнаружить это можно при помощи `man`. Если по команде `man csh` выводится справка по `tcsh`, самое время заподозрить неладное. Кроме этого можно дать команду `ls -l /bin/*sh`, чтобы посмотреть, какие файлы являются символическими ссылками, а какие нет.

У каждого командного процессора есть приверженцы. Равно как и противники. Например, по адресу

http://www.faqs.org/faqs/unix-faq/shell/csh-whynot/

можно прочесть пламенную и весьма взвешенную статью о том, почему неудобно работать с **csh**. Лично я предпочитаю **bash** для интерактивной работы и добрый старый **sh** для написания скриптов. **Bash**, кстати, выполняет скрипты **sh** без сучка и задоринки.

Таблица 8.2. Основные задачи командных процессоров

Функция	sh	csh	tcsh	ksh	bash
history	Нет	Да	Да	Да	Да
псевдонимы	Нет	Да	Да	Да	Да
скрипты	Да	Да	Да	Да	Да
подстановка и завершение имен файлов	Нет	Да[a]	Да	Да[a]	Да
редактирование командной строки	Нет	Нет	Да	Да[a]	Да
управление заданиями	Нет	Да	Да	Да	Да

[a] По умолчанию выключено

Действия командного процессора при запуске

При запуске командный процессор выполняет стартовые файлы. Вначале выполняются его системные (общие для всех) стартовые файлы, а затем пользовательские (индивидуальные). Общие лежат в `/etc`, индивидуальные – в домашнем каталоге пользователя. Если командный процессор запускается при входе пользователя в систему (то есть это login shell), то после стартовых файлов командного процессора выполняются его login-файлы в том же порядке: сначала его общий login-файл, а затем индивидуальный, из домашнего каталога. Login-файлы командных процессоров расположены в `/etc` (общие) и в домашнем каталоге (индивидуальные). Если вы, находясь в среде командного процессора, запустите его еще раз, вот так:

```
sh
```

то при запуске этой новой копии командного процессора его стартовые файлы выполнятся, а login-файлы – нет.

Некоторые командные процессоры при входе в систему выполняют сначала login-файлы, а затем стартовые файлы, поскольку это зависит от того, как они были скомпилированы.

Некоторые командные процессоры выполняют стартовые файлы лишь при входе пользователя в систему. Как правило, у командного процессора есть ключ, при помощи которого можно сымитировать вход в систему и заставить его выполнить login-файлы.

В **bash** реализован более сложный, по сравнению остальными командными процессорами, алгоритм выполнения стартовых и login-файлов. Если **bash** запущен как login shell, то он выполняет `/etc/profile`,

если тот существует. Затем ~/.bash_profile. Если такого нет, делается попытка выполнить ~/.bash_login, а если и такого нет, то ~/.profile. Если **bash** запущен как обыкновенный интерактивный командный процессор (не login shell), то при старте он выполняет только ~/.bashrc.

Таблица 8.3. Стартовые и login-файлы командных процессоров

shell	Стартовые файлы	login-файлы
sh		/etc/profile, ~/.profile
Csh	/etc/csh. cshrc, ~/.cshrc	/etc/csh. login, ~/.login
ksh		/etc/profile, ~/.profile
bash	~/.bashrc	/etc/profile, ~/.bash_profile ~/.profile

Если же **bash** был запущен как **sh** (то есть **sh** представляет собой символическую ссылку на **bash**), то при старте он ведет себя так же, как и **sh**.

История команд

History – это функция запоминания введенных команд. Командный процессор помнит все команды, введенные пользователем в текущем сеансе. По окончании сеанса он вносит их в файл истории. Файл истории хранится в домашнем каталоге пользователя. В **sh**, **csh** и **tcsh** он называется ~/.history, в **bash** – ~/.bash_history, в **ksh** – ~/.shhistory, а в бесплатной версии Korn Shell (public domain Korn shell – **pdksh**) его по умолчанию нет вообще.[1]

Многие командные процессоры позволяют вызвать ранее введенную команду по ее номеру или начальным буквам. О том, как это делать, подробнее сказано в разделе «Повторение ранее введенных команд (с модификацией и без)».

Файл истории – это потенциально уязвимое место системы UNIX. Имейте в виду, что по умолчанию он может быть доступен для чтения не только тому, кто вводил команды, но и другим пользователям. В этом случае избегайте вводить любые пароли в командной строке. Допустим, броузер **lynx** или программа доступа к серверу баз данных **mysql** были вызваны из командной строки с указанием пароля:

```
lynx ftp://user:password@URL
mysql -u root -p password
```

Поскольку эти команды сохранятся в файле истории вместе с другими, кто-то потом сможет прочесть их и узнать какие-то из ваших паро-

[1] **Pdksh** написан группой энтузиастов во главе с Чарльзом Форситом (Charles Forsyth).

лей. Размер файла истории команд ограничен. По умолчанию **bash** хранит 500 команд, **tcsh** – 1000, **csh** – 100, **sh** – зависит от версии. Часто **sh** не имеет файла `.history` вовсе и дает доступ только к командам, которые были введены в течение текущего сеанса работы с ним. Можно изменить количество хранимых команд, указав иное значение переменной, в которой оно хранится.

Таблица 8.4. Переменные, определяющие количество хранимых команд

shell	Переменная	Значение по умолчанию
sh	HISTSIZE	не определено
csh	history	100
ksh	HISTSIZE	128
bash	HISTFILESIZE	500

Псевдонимы команд

Псевдонимы – это сокращения для часто используемых последовательностей команд или длинных команд. Тот, кто при вводе команд часто допускает однотипные опечатки, может сделать команду с опечаткой псевдонимом правильной команды, например

```
alias telent telnet
```

Создать псевдоним очень просто:

```
alias псевдоним команда
```

Команда может быть последовательностью команд с аргументами, тогда ее лучше заключить в апострофы:

```
alias ls 'ls -lt | more'
```

Приведенный пример заставит командный процессор каждый раз при вводе **ls** выполнять команду

```
ls -lt | more
```

Синтаксис определения псевдонимов в разных командных процессорах разный (табл. 8.5). Псевдонимы обычно определяются в стартовых файлах командных процессоров.

Таблица 8.5. Синтаксис команд alias

shell	команда alias	ключ для экспорта
sh	alias qq=ls	
csh	alias qq ls	
ksh	alias qq=ls	-x
bash	alias qq=ls	

Псевдонимы могут экспортироваться. Для этого используются ключи команды **alias**. Команда **alias** является встроенной командой командного процессора.

Можно отменить назначенный псевдоним командой **unalias** *псевдоним*.

Подстановка имен файлов. Шаблоны

Многие программы принимают в качестве аргументов имена файлов. Любой командный процессор умеет подставлять в командную строку имена файлов, соответствующие шаблону: **ls *.c** выводит информацию обо всех файлах, имена которых заканчиваются на **.c**. Символ «*» трактуется командным процессором как шаблон, которому соответствует любая подстрока, в том числе и пустая.

Внимание! Командный процессор не умеет работать с регулярными выражениями. Шаблоны и регулярные выражения похожи, но не имеют между собой ничего общего. Регулярные выражения обрабатываются стандартной функцией regex(), в то время как функции обработки шаблонов имен файлов у каждого командного процессора свои.

Ниже приведены шаблоны имен файлов, которые понимает shell:

- * – любая подстрока, в том числе пустая;
- ? – один любой символ;
- [...] – один любой символ из списка (например, [avgf]) или диапазона (например, [a-z]).

Подстановка имен файлов по шаблону выполняется так: shell вначале отыскивает все файлы, имена которых соответствуют шаблону, а потом подставляет имена всех этих файлов в строчку в то место, где вы задали шаблон. Таким образом, **ls *.c** раскроется в `ls a.c b.c qq.c`, если в текущем каталоге есть файлы с именем, заканчивающимся на `.c`.

Подстановка результата выполнения

В любом командном процессоре UNIX есть уникальная возможность подстановки результата выполнения команды в командную строку. Например, вы хотите найти все файлы в файловой системе своей машины, которые называются `userio.h`, и получить о них подробную информацию. Используем **find** для поиска и **ls** для получения сведений:

```
ls -l `find / -name "userio.h"`
```

Команда, заключенная в ``` (обратные апострофы), выполняется первой, а результат ее выполнения подставляется вместо конструкции «обратные апострофы» в командную строку. Затем выполняется вся

строка. Конструкция «обратные апострофы» и обеспечивает подстановку результата выполнения.

Подсказка имен файлов

Функция подсказки (завершения) имен файлов (file name completion) позволяет набрать на клавиатуре часть имени существующего файла и нажать клавишу <Tab>. При этом командный процессор закончит за вас имя файла. Например, желая обратиться к файлу `/usr/local/apache/bin/apachectl`, вы можете набрать

```
/usr/loc<Tab>al/a<Tab>pache/bin/a<Tab>pachectl
```

Все, что в этом примере выделено жирным шрифтом, набираете вы, все, что не выделено, подставляет командный процессор. При наличии файлов с длинными и уникальными именами эта функция здорово экономит время.

UNIX'исты вообще любят экономить время. Например, чтобы набирать имя пользователя побыстрее, его делают короче: bob вместо robert, iv вместо ivan, fil вместо philippe и т. д.

При двойном нажатии клавиши <Tab> выводятся все возможные варианты. Например, в каталоге есть два файла, начинающиеся с буквы q: `q14.txt` и `q34.txt`. Я хочу отредактировать один из них и набираю **vi q<Tab><Tab>**.

Bash мне подскажет:

```
q14.txt  q34.txt
```

Мне достаточно набрать еще цифру 3, чтобы указать, какой именно файл мне нужен, и еще раз нажать <Tab> для окончательного завершения имени. Если после двойного нажатия <Tab> **bash** видит слишком много возможных вариантов имен, он спросит

```
There are 1102 possibilities.  Do you really
wish to see them all? (y or n)
```

Имеет смысл ответить «n», ведь столько и на экран не поместится. Завершение имен файлов реализовано в **bash** и **tcsh**.

Средства редактирования командной строки

Удобнее всего редактировать командную строку в **bash**. В **sh** средств редактирования нет вовсе. Точнее говоря, есть минимальные, которые есть в любом командном процессоре, а именно: стирание последнего символа (клавиша) <Backspace> и стирание строки целиком <Ctrl>+<U>.

В **bash** можно двигаться вперед и назад по уже введенной строке с помощью «стрелок», вводить символы не только в конец строки, но также в середину и в начало. **Bash** воспримет всю строку целиком независимо от того, в какой позиции строки находится курсор в момент нажатия клавиши <Enter>. Например, вы можете вернуть курсор к середине строки, если заметили ошибку ввода, исправить ее и тут же нажать <Enter> для ввода команды. В более старых версиях командных процессоров пришлось бы исправить строку и перейти к ее концу для ввода команды полностью.

В **ksh** редактирование по умолчанию выключено, его надо включить нажатием <Esc> . После этого вы сможете редактировать строку командами редактора, имя которого задано в переменной среды окружения VISUAL или EDITOR. По умолчанию это **vi**. Но так редактировать строки не слишком удобно. К тому же приходится помнить о включении режима редактирования строки и переходе в режим ввода!

В **csh** редактирование командной строки выполняется так же, как и в **bash**.

Командный процессор с ограничениями возможностей

В жизни каждого системного администратора бывают такие минуты, когда ему хочется загнать всех пользователей в клетку. Причем, каждого – в свою, чтобы было легче запрещать каждому что-то свое. Кроме модных изобретений типа виртуальных серверов, есть и более простой выход, чтобы создать пользователю «ограниченную видимость» в системе. Это так называемый «restricted shell» – командный процессор с ограничением возможностей пользователя. Существует несколько реализаций такой программы; мы рассмотрим это на примере Korn shell.

Если запустить его с ключом **-r** или сделать символическую ссылку на него и запускать его под именем, отвечающим шаблону ***r*sh** (например, **rksh**, **rpdksh** и т. п.), то командный процессор после выполнения стартовых файлов не разрешит пользователю:

- выполнять команду **cd**;
- откажется изменять значения переменных SHELL, ENV и PATH;
- запускать команду, если ее имя указано как полное или относительное имя файла (т. е. passwd запустить можно, а /bin/passwd или ../../bin/passwd – нельзя);
- перенаправлять потоки в файл, если это требует создания файла.

Фактически это ограничит деятельность пользователя текущим каталогом (очевидно, это будет его домашний каталог, если **rksh** будет указан в качестве login shell'а пользователя).

Такое ограничение может пригодиться, например, при разрешении интерактивного доступа пользователям, получающим на вашем сервере услугу хостинга.

Повторение ранее введенных команд (с модификацией и без)

В **csh** введены средства модификации командной строки. Например, **!n** (n – целое положительное число) означает повтор n-й команды из истории команд. История команд доступна по команде **history**. Для пользователя с именем root эта команда выводит полную историю команд с момента установки системы. Для просмотра последних команд применяется **tail**:

```
history | tail
```

!! в **csh** означает повтор последней команды, **^ls^cat** означает повтор последней команды с заменой первого вхождения подстроки «ls» на подстроку «cat».

!*cmd* вызывает выполнение самой недавней команды, начинающейся на «cmd», а **!***n* – выполнение команды, номер которой в файле истории равен *n*. Однако надо иметь в виду, что **csh** и **tcsh** по-разному интерпретируют конструкцию, например !3d. Для **csh** это команда номер 3, в конец которой надо приписать d и выполнить то, что получится, а для **tcsh** – последняя команда, начинающаяся на «3d». Такой синтаксис введен в **tsch** для того, чтобы можно было повторять команды, начинающиеся с цифр. Чтобы заставить **tcsh** вести себя подобно **csh**, надо перед тройкой поставить обратный слэш, экранирующий специальное значение тройки: !\3d.

В **bash** повторение команд достигается так же, как и в **csh**. Кроме того, можно вызывать ранее введенные команды при помощи клавиш «стрелка вверх» и «стрелка вниз».

В **ksh** для повторения команд предназначена встроенная команда **r**, а в **sh** – команда **fc**, при этом в **sh** можно вызвать только команду, которая была введена в текущем сеансе командного процессора.

Переменные среды окружения

Среда окружения (environment) есть у каждого процесса UNIX. В ней хранятся переменные, у каждой из которых есть имя и значение. В командном процессоре значение переменной можно получить при помощи конструкции **$***имя_переменной*.

Например, **echo $HOME** выведет на экран значение переменной HOME. Пользователь может определить свои собственные переменные. Новая переменная появляется, как только вы даете команду присвоить зна-

чение еще не существующей переменной. Переменная не имеет типа, поскольку считается, что все переменные имеют тип «строка». Присвоение переменной значения в разных командных процессорах выполняется по-разному.

Таблица 8.6. Присвоение переменной в разных командных процессорах

shell	присвоение значения переменной
sh	VAR=*value*
csh	set VAR=*value*
ksh	VAR=*value*
bash	VAR=*value*

Переменные со специальными названиями типа `http_proxy` используются разными программами, которые проверяют свою среду окружения на наличие таких переменных и их значений. Поведение программы меняется в зависимости от значения переменной.

Переменные могут экспортироваться, то есть передаваться от процессов-родителей процессам-потомкам. Для экспорта переменной надо дать команду

export *имя_переменной*

Для того чтобы удалить переменную среды окружения, можно присвоить ей пустое значение (это не равнозначно удалению, но иногда этого достаточно), а можно удалить ее вовсе командой **unset**.

Команда **unset** *pattern* удаляет все переменные, имена которых удовлетворяют шаблону, если только эти переменные не имеют статус «только для чтения». Таким образом, команда **unset *** удаляет все переменные, кроме переменных только для чтения. Команда **unset** без параметров попытается удалить «ничто», то есть не выполнит никаких действий. Проверьте это в вашей системе, чтобы быть уверенным, что это работает!

В **tcsh** есть возможность создавать переменные только для чтения при помощи команды **set -r** *имя=значение*. Кроме этого в **tcsh** можно присвоить переменной значение в виде списка слов, обращаться к которым впоследствии можно по номеру слова, начиная с 1:

```
set var=(hello world)
echo {$var[1]}
hello
echo {$var[2]}
world
```

Практическое применение этой редкой возможности нам неизвестно, хотя можно предположить, что оно удобно в скриптах. Правда, мало кто пишет скрипты на **tcsh**, потому что это крайне неуниверсально.

Предопределенные переменные

Имена всех предопределенных переменных среды окружения во всех командных процессорах, кроме csh, записываются прописными буквами: HOME, USER, TERM, PS1, PS2 и т. д. Многие программы (веб-сервер apache, perl и другие) при запуске проверяют значения некоторых специфичных для них переменных. Присваивая этим переменным определенные значения, можно сообщить что-то новое программе. Большинство программ проверяют значения переменных окружения, имена которых тоже состоят из прописных букв латинского алфавита. Принято пользовательские переменные в скриптах называть именами из символов нижнего регистра (строчных букв). Это делается для того, чтобы отличать переменные, придуманные разработчиком системы или программы, от тех, что вводит пользователь.

В разных командных процессорах заранее определены разные переменные, хотя есть те, что присутствуют в любом из них. Вот эти обязательные переменные (табл. 8.7).

Таблица 8.7. Переменные командных процессоров

Переменная	Смысл
HOME	домашний каталог пользователя
SHELL	текущий командный процессор
PS1	вид приглашения командного процессора
PS2	вид приглашения «продолжение команды»
TERM	тип терминала пользователя
USER или LOGNAME	имя пользователя
PATH	список каталогов, где надо искать программу, когда пользователь запускает ее, указывая только имя, но не путь. Каталоги разделяются двоеточиями

Некоторые переменные определяются только в одном командном процессоре.

В **csh** все имена предопределенных переменных пишутся маленькими буквами.

Любой скрипт может быть вызван с аргументами, которые в скрипте называются позиционными параметрами. К аргументам с первого по девятый можно обращаться как к $1..$9. Значение $0 – это имя самого скрипта, по которому его вызвали. Один и тот же скрипт может быть вызван с разными именами, например, если на него установлена символическая ссылка (ярлык, symbolic link).

Команда **shift** *n* (*n* – положительное или отрицательное смещение) позволяет двигать «окошко» $1...$9 по всему массиву аргументов скрипта вперед и назад, если аргументов больше девяти. **Shift** без параметров эквивалентен **shift 1**. В **csh** эта команда умеет сдвигать «окно» только на одну позицию вперед.

Кроме вышеописанных переменных, обычно определены:

- $ – номер процесса текущего командного процессора;
- * – все позиционные параметры;
- @ – то же, что и *;[1]
- ! – идентификатор последнего процесса, запущенного из текущей сессии командного процессора в фоновом режиме; для конвейеров – идентификатор последнего в конвейере процесса;
- _ – последняя вызванная команда;
- 1,2,..9 – один позиционный параметр;
- ? – количество позиционных параметров.

Язык написания скриптов

Есть два типа языков, которые применяются в различных командных процессорах. Это sh-подобный и csh-подобный языки. Первый поддерживается в **sh**, **ksh**, **bash**, а второй – в **csh** и **tcsh**. Языки первого типа происходят от первоначального языка sh, разработанного Баурном. В новых версиях командных процессоров иногда появляются дополнения к старому доброму языку. Однако из соображений совместимости все стараются писать скрипты, соблюдая стандартный синтаксис. Мы будем в этом разделе называть sh-подобный язык языком **sh**, а csh-подобный – языком **csh**. Между собой скрипты на этих языках несовместимы. Абсолютное большинство скриптов, приходящих к нам в систему вместе с дистрибутивами, написано для **sh**.

Первой строкой скрипта должна быть строка вида #!*полное_имя_shell*, указывающая, в каком командном процессоре надо выполнять этот скрипт. Если строки нет, то система запускает /bin/sh для его выполнения.

Те, кому программирование совершенно чуждо, могут перейти к следующему разделу. Все, что вы прочтете ниже в этом разделе, предполагает, что вы знакомы с основами программирования и собираетесь писать скрипты сами. Или хотя бы разбираться в чужих.

В операторах циклов можно использовать команды **break** (прерывание цикла, управление передается за конец цикла) и **continue** (пере-

[1] Разница между * и @ в том, что если есть два позиционных параметра $1 и $2, где $1 – это «abc», $2 – «qaz wsx», то если переменные * и @ встречаются в кавычках, то они раскрываются по-разному: $* даст «abc qaz wsx», а $@ даст две строки: «abc» и «qaz wsx».

дача управления на начало цикла, переход к следующей итерации). Команды действуют одинаково в **sh** и **csh**.

Язык sh

В языке **sh** определены управляющие конструкции ветвления и циклов. В описании операторов жирным шрифтом выделены ключевые слова. Конструкция «команды» подразумевает одну или несколько команд командного процессора, которые отделяются друг от друга точкой с запятой или символом перевода строки. При вводе многострочных операторов в интерактивном диалоге с командным процессором последний будет начинать новую строку приглашением «продолжение команды» (его вид определяется значением предопределенной переменной PS2).

Оператор ветвления **if**:

```
if условие then
   команды
else
   команды
fi
```

В **bash** и **ksh**, а также в новых версиях **sh** допустима конструкция **elseif**:

```
if условие then
   команды
     elseif условие then
команды
     fi
else
   команды
fi
```

Оператор множественного ветвления **case**:

```
case значение_переменной of
Значение1) команды; break;
значение2) команды; break;
Значение3) команды; break;
*) команды;
esac
```

* означает «все остальное», то есть если значение переменной не равно ни *значению1*, ни *значению2*, ни *значению3*.

Операторы цикла **while** (выполнять, пока условие истинно) и **until** (выполнять до тех пор, пока условие не станет истинным):

```
while условие do
     команды
```

```
done

until условие do
     команды
done
```

Что такое *условие*? Это любая команда (в том числе скрипт, конвейер и т. п.), возвращающая некий код завершения. Строго говоря, любая команда в UNIX возвращает код завершения, но не всегда этому коду можно потом найти осмысленное применение. Условие считается истинным, если код завершения равен нулю, и ложным, если код завершения не равен нулю.

Для того чтобы можно было проверять всякие интересные условия, существует программа **test**. В некоторых системах **test** – это не отдельная программа, а встроенная команда командного процессора. С ее помощью обычно проверяется наличие файла или каталога в файловой системе:

```
if test -e имя_файла then echo "имя_файла существует!"
```

Можно еще проверить, является ли объект каталогом:

```
if test -d имя then echo "имя - каталог!"
```

Некоторые программы всегда возвращают ноль. Этим можно воспользоваться для организации бесконечных циклов. Может пригодиться для перезапуска процессов, имеющих тенденцию к неожиданному и нежелательному завершению. Например, так можно запускать **pppd** для соединения с провайдером по выделенной или коммутируемой линии:

```
while sleep 10
do
  /usr/sbin/pppd
done
```

Программа **sleep** всегда будет возвращать ноль, так что цикл никогда не прервется и будет исправно запускать **pppd** всякий раз, как только он завершится, например из-за ошибки связи. Десять секунд ожидания добавлены просто для того, чтобы было время переждать неблагоприятную ситуацию, из-за которой произошла ошибка.

Оператор **for** (цикл повторяется для каждого значения из списка):

```
for имя_переменной in список do
     команды
done
```

Последний оператор цикла применяется особенно часто, например для однотипной модификации нескольких файлов сразу. Предположим, надо во всех файлах `*.html` в текущем каталоге изменить название файла с логотипом:

```
for i in *.html do
```

```
sed 's/old_logo.gif/new_logo.gif/g' $i >tmp
mv -f tmp $i
done
```

Поскольку **sed** не меняет файл, а лишь выдает измененный текст в stdout, мы вынуждены перенаправить вывод во временный файл tmp (название может быть любым) и затем переименовать tmp в файл с тем же именем, что у исходного файла. В момент переименования старый файл исчезнет.

Обратите внимание на **$i** в вызове **sed**: необходимо написать именно **$i**, чтобы командный процессор подставил в командную строку **значение** переменной i, а не символ «i».

Перенаправить вывод в файл **$i** нельзя, так как перенаправление вывода в существующий файл вызовет уничтожение его прежнего содержимого еще до того, как **sed** попытается его открыть для чтения.

Ключ **-f** нужен **mv** для того, чтобы она выполняла работу молча, не задавая вопросов: «А вы действительно хотите уничтожить старый файл?»

НИКОГДА не запускайте скрипт, который что-либо уничтожает, если не уверены в том, что он написан совершенно правильно и без ошибок. Сомневаетесь в результате – поставьте команду **echo** вместо команды, безвозвратно уничтожающей файл, посмотрите, нужные ли имена выведет **echo**. Потом, если все оказалось в порядке, меняйте **echo** на **mv**, **rm** или что-нибудь еще опасное.

Для вывода информации в **sh** принято использовать **echo**. Команда **echo -n** выводит строку и не добавляет перевод строки в конце. Для ввода применяется оператор **read**:

```
read name
echo "Name is $name"
```

Язык csh

К описанию языка **csh** можно отнести все, что выше говорилось о языке **sh**. Однако надо помнить, что:

- в **csh** системные переменные среды окружения записываются строчными буквами;
- в **csh** нет оператора **until** и вообще операторы называются по-другому.

В **csh**, а также в **bash** есть свои правила вычисления выражений для того, чтобы выяснить истинность или ложность условия в операторах цикла и ветвления. Подробности лучше почерпнуть из man: их довольно много, здесь мы приведем лишь некоторые детали. Команда **test** работает обычно в любом командном процессоре.

Везде, где написано «команда» в единственном числе, подразумевается одна команда (не конвейер, не псевдоним (alias) и не набор команд в скобках).

```
if (условие)команда
```

или

```
if (условие) then
      команды
       else if (условие) then
 команды
       else
            команды
endif
```

Допускается любое количество **elseif**, при этом нужен всего один оператор **endif**.

Операторы **else** и **endif** должны начинаться с новой строки, **if** должен следовать за **else** или начинаться с новой строки. Оператор множественного ветвления в **csh** называется **switch**:

```
switch (строка)
       case строка1:
 команды
            breaksw
команды
       default:
команды
            breaksw
       endsw
```

Оператор работает подобно **switch** из языка C, команда **breaksw** передает управление за оператор **endsw**.

Оператор **while** действует так же, как **while** в **sh**, но имеет другой синтаксис:

```
while (условие)
 команды
end
```

Оператор цикла **for** в **csh** назвали **foreach** – для доходчивости, наверное.

```
foreach имя_переменной (список)
команды
end
```

Когда оператор **foreach** вводится с терминала в интерактивном режиме, **csh** предваряет каждую новую строку цикла приглашением `foreach?`.

Установка приглашения командного процессора

В командном процессоре могут быть определены разные приглашения для разных случаев. Приглашением мы называем фразу, которая означает «я готов, вводите команду». По умолчанию в **sh** – это $, в **csh** – %. Пользователя с именем root принято приветствовать символом «#». Часто эта фраза не столь коротка. Многим хочется, чтобы в ней содержалось имя текущего каталога.

В любом командном процессоре определены переменные PS1 и PS2. Их значения соответствуют обычному приглашению командного процессора и приглашению к продолжению команды. Например, если вы вводите длинную команду и набираете ее в две строки, то можете в конце первой поместить символ «\», нажать клавишу <Enter> и продолжить ввод на следующей строке. Обратный слэш экранирует следующий за ним символ перевода строки от интерпретации в качестве конца команды. При переходе к новой строке вы увидите символ «>», который по умолчанию является приглашением к продолжению команды.

В **csh** вместо PS1 и PS2 установлены prompt и prompt2. В некоторых командных процессорах (например, в **ksh**) определены и другие приглашения для специальных случаев. Они хранятся в переменных PS3, PS4 и т. д., а в **csh** – в prompt3 и, возможно, в prompt4.

Bash, **ksh**, **csh**, **tcsh** можно научить показывать весьма изощренные приглашения, в которых будут содержаться динамически подставляемые части (например, имя текущего каталога). В этих командных процессорах есть свои собственные сокращения для таких подстановок. Например, имя текущего каталога в **bash** подставляется макросом \w, в **csh** – %/. Помните, что назначать имя переменной можно только указывая макрос внутри апострофов, чтобы экранировать его от интерпретации командным процессором:

```
$PS1='\w$' # так можно делать в bash
```

Более подробно макросы для установки приглашений описаны в man по соответствующим командным процессорам.

Перечень спецсимволов

В каждом командном процессоре есть свой набор спецсимволов. Командный процессор интерпретирует спецсимволы особым образом, многие из них интерпретируются одинаково в разных командных процессорах. Выше были рассмотрены многие случаи употребления спецсимволов: шаблоны, модификаторы командной строки, обращение к переменным среды окружения, управляющие конструкции в скриптах. Большинство спецсимволов перечислено в табл. 8.8. Для ввода са-

мих этих символов в командной строке предваряйте их символом экранирования «\» или заключайте в кавычки или апострофы. С точки зрения экранирования кавычки отличаются от апострофов тем, что кавычки экранируют все символы, за исключением `, $ и \, а апострофы экранируют любые символы.

Таблица 8.8. Специальные символы командных процессоров

shell	спецсимволы
sh	$ * ? \ ` ‘ “ [ ]
csh	$ * ? \ ` ‘ “ ! ^ { } [ ]
ksh	$ * ? \ ` ‘ “ ! ^ { } [ ]
bash	$ * ? \ ` ‘ “ ! ^ { } [ ]

Разбор командной строки

Любой командный процессор выполняет разбор командной строки в следующем порядке:

- выделение команд и их аргументов;
- подстановка псевдонимов (aliases);
- подстановка имен файлов по шаблонам;
- подстановка результата выполнения команды, если есть конструкция ``;
- запуск программ и передача им среды окружения и аргументов.

Из любой программы или скрипта всегда можно обратиться к любой переменной среды окружения этой программы или аргументу программы.

Надо также иметь в виду, что **csh** и **tcsh** кэшируют при запуске список выполняемых файлов из каталогов, указанных в переменной path, поэтому при изменении последней и при добавлении новых файлов в системные каталоги полезно давать команду **rehash**, чтобы **csh** «увидел» новые каталоги и команды.

Дополнительная литература

[5] Ellie Quigley «UNIX Shells by Example». – Prentice Hall, 2000 (Квигли Э. «Командные процессоры UNIX в примерах»).

9

Резервное копирование

Расчет на авось против резервного копирования

Десятки миллионов людей ежедневно теряют свои вещи. Им приходится мириться с потерей мобильных телефонов, ножниц, ключей, очков и даже с утоплением пейджеров в унитазе. Однако пользователи компьютеров терпеть не могут терять информацию, которая создавалась в течение долгих лет работы. Разрушение почтового ящика в подъезде никогда не вызывает такого отчаяния у человека, как разрушение его электронного почтового ящика. Последнее, правда, связано только с тем, что мы не имеем привычки хранить старые письма в почтовом ящике в подъезде. Вместо этого мы предпочитаем держать архив в кладовке или в гараже. Электронный почтовый ящик для большинства пользователей играет роль почтового архива. По сравнению с гаражом это хранилище не более надежно, чем пороховая бочка с воткнутым в нее горящим факелом.

В отношении любых данных не может быть вопроса, будут ли они потеряны без резервного копирования, есть лишь вопрос «когда?».

Резервное копирование – одна из самых нелюбимых системными администраторами задач. Наверное, беда в том, что его сложнее всего автоматизировать, а само копирование – задача совершенно не творческая. Самое же противное в резервном копировании – это его неотвратимость.

Для резервного копирования можно использовать множество различных устройств. Традиционно оно производилось на магнитные ленты. Этот способ плох тем, что существует слишком много разных форматов и форм-факторов лент, и довольно часто прочесть ленту удается

только на том же устройстве, на котором она была записана. Кроме того, быстрые ленточные накопители (стримеры) и кассеты к ним довольно дорого стоят.

Например, стример HP StorageWorks SDLT 160/320 стоит порядка $5600. На его кассету умещается от 160 Гбайт несжатых данных и до 320 Гбайт сжатых, но аппаратное сжатие обычно не позволяет на самом деле сжимать произвольные данные вдвое. Коэффициент сжатия в действительности зависит от характера данных, поэтому серьезно рассчитывать на запись более чем 220–230 Гбайт на такую кассету не следует. Стоимость одной кассеты – $70, и не рассчитывайте на большие скидки при покупке двух десятков кассет: слово «опт» в этом бизнесе обозначает куда большие объемы закупок.

Самые емкие кассеты в мире по состоянию на май 2005 года – у стримера HP StorageWorks SDLT 600. Они вмещают 300 Гбайт несжатых файлов и 600 Гбайт сжатых, согласно информации производителя. Стоит такой стример $6500, а кассеты к нему – около $150.

Как и любые магнитные носители, ленты необходимо бережно хранить вдали от источников тепла и сильного электромагнитного поля. Через год-другой хранения ленту желательно переписывать, потому что ее «память» слабеет. Прочесть совсем старую ленту просто не удастся, так как качество магнитной записи со временем ухудшается. Кто-то, возможно, помнит этот эффект со времен аудиокассет. Однако поставщики коммерческих систем UNIX предлагают именно этот способ резервного копирования как стандартный. Ниже мы рассмотрим программы `dump` и `restore`, традиционно применяемые для резервного копирования на ленту.

Резервное копирование следует всегда применять там, где цена информации высока – в информационных системах медицинских и финансовых организаций, других компаний с большой интенсивностью накопления важных данных. Кроме этого резервное копирование позволяет быстро восстановить случайно удаленные данные, причем сделать это даже тогда, когда невольный виновник вспомнил о своей ошибке через несколько дней или даже недель после удаления файлов. Разумеется, резервное копирование страхует нас еще и от сбоев накопителей – даже RAID-массив не спасет от сбоя, случившегося при записи данных по вине неисправного контроллера.

Хранить резервные копии надо вдали от того места, где находится диск-оригинал, потому что при пожаре может погибнуть и диск, и его резервные копии, если они будут находиться в том же здании.

Другой способ надежного хранения данных – их зеркалирование. При зеркалировании диска или его части можно копировать данные по сети сколь угодно далеко. Ограничениями в этой ситуации являются скорость и стоимость канала связи. Зеркалирование может быть непрерывным, когда запись данных вызывает их мгновенное копирова-

ние на другой носитель (локальный диск или диск удаленного компьютера), а может быть регулярным, когда с некоторой периодичностью файлы на разных носителях синхронизируются, например, происходит переписывание одного диска на другой раз в сутки.

Разумеется, можно еще делать резервные копии на ZIP-дисках (100 и 250 Мбайт), дискетах, магнитооптических дисках (120 Мбайт), JAZZ-дисках, DVD, CD-ROM и CD-RW. Ни один из этих способов не лучше простого копирования информации на соседний сервер. Современные жесткие диски обычно сильно превосходят по емкости и надежности все вышеперечисленные носители, к тому же они в несколько раз дешевле при расчете цены за гигабайт объема.[1]

При резервном копировании очень важно определить необходимые для копирования данные и разумную периодичность копирования. Например, всегда следует хранить резервную копию всех текущих конфигурационных файлов ОС и приложений. Однако делать ее следует не ежедневно, а в случае изменений. С другой стороны, файлы пользователей надо копировать регулярно, исходя из того, как часто они принципиально изменяются. Имеет смысл делать резервную копию каталога `/var/mail` (в некоторых системах `/usr/mail`), чтобы иметь копию почтовых ящиков всех пользователей.

Копировать файлы на другой компьютер можно с помощью **`ftp`**, **`rdist`** или весьма совершенной утилиты **`rsync`**. Последняя хороша тем, что передает не весь файл, а лишь ту его часть, которая отличается от предыдущей копии. Копирование по **`ftp`** можно делать либо вручную интерактивно, либо с помощью короткого скрипта на языке Tcl (tools command language) – простом языке, позволяющем запрограммировать имитацию интерактивного общения с любой программой в стиле «командуем-ожидаем». Это значит, что вы указываете, что сообщить программе (например, **`ftp`**) и что она должна ответить. Например, в случае любой ошибки скрипт может послать вам письмо с указанием места, где его выполнение было прервано, и завершиться.

Окончательный выбор способа резервного копирования за вами: никто лучше вас не знает особенностей вашего бизнеса. В одном случае будет достаточно раз в неделю копировать основные файлы конфигурации на дискету, в другом – перекачивать весь диск целиком на другой компьютер в конце дня.

Для тех, кто вынужден использовать **`dump`** и **`restore`** из-за того, что стример в их распоряжении есть, а иного способа делать резервные копии нет, ниже коротко описаны эти программы.

---

[1] Насколько нам известно, из всех жестких дисков IDE для PC, которые продавались в России после 2001 года, наиболее надежными оказались диски Samsung. Эта сноска не оплачена фирмой Samsung, я просто делюсь опытом.

Как использовать dump и restore

По умолчанию **dump** и **restore** работают с первым по счету стримером в системе. Однако не стоит полагаться на их выбор. Указывайте явно, на какое устройство вы хотите выполнить резервное копирование. Программа **dump** применяется только к целым файловым системам, копировать два-три каталога с ее помощью просто нецелесообразно. Как стрелять из пушки по воробьям.

Программа **dump** может выполнять копирование не только на ленту, но и на диск, а также на любое другое устройство. Однако при копировании информации на жесткий диск разумнее использовать обычные утилиты копирования типа **cp** или специализированные типа **rsync**, если надо выполнить копирование на удаленный компьютер.

При копировании на ленту следует обязательно указать уровень копирования. Программа **dump** позволяет задать 10 уровней – с 0 до 9. Уровень 0 обозначает копирование всей файловой системы целиком. При копировании уровня n копируются все файлы, которые были изменены с момента последнего копирования уровня n или меньше. Уровни копирования введены для облегчения инкрементного копирования, то есть копирования только измененных файлов. Это экономит время при копировании и занимаемый данными объем. С другой стороны, если время позволяет, а свободное место на ленте заведомо превышает объем копируемых данных, удобнее делать полное копирование выбранной файловой системы (раздела UNIX). Это позволяет всегда иметь копию всего раздела и быстро его восстановить после сбоя. Намного быстрее, чем при инкрементном копировании. Потому что при инкрементном копировании для восстановления придется последовательно считывать все ленты начиная с последнего дампа нулевого уровня. Кроме роста затрат времени и кассет на такое копирование, после восстановления появятся уже удаленные файлы, которые удалялись после дампа нулевого уровня.

При нехватке места на ленте или диске, на которые производится резервное копирование, **dump** предупредит об этом и попросит вставить в то же самое устройство новый носитель (кассету с лентой). Конец носителя детектируется по сигналу накопителя. Если ваш стример не умеет корректно выдавать этот сигнал, укажите явно в команде **dump** размер носителя в килобайтах. Если вы просто не доверяете стримеру, сделайте то же самое. Мне не встречались стримеры, на которые я мог бы положиться.

Программа **dump** позволяет также задать плотность записи на ленту. Помните, что эта информация полностью зависит от типа и модели стримера, не забудьте внимательно прочесть его описание и проверить соответствие кассет спецификации.

Существует много красивых схем выполнения резервных копий разных уровней, в том числе оптимизированных для уменьшения числа задействованных лент, уменьшения времени восстановления, умень-

шения занимаемого объема. Одна из схем носит романтическое название «ханойские башни» (по имени реализованного в ней математического алгоритма). На практике обычно применяется регулярное полное (уровня 0) резервное копирование раздела с важной информацией.

Синтаксис команды **dump** отличается от синтаксиса большинства команд UNIX:

dump *[[ключи] [аргументы]] файловая_система*

Сначала в командной строке указываются все ключи подряд, но без минуса перед ними. Затем через разделитель в том же порядке, что и ключи, указываются параметры (аргументы), относящиеся к этим ключам, например:

```
/sbin/dump 0ufBds /dev/nst0 4194304 62000 1500 /var
```

Это означает дамп уровня 0 (ключ 0), с обновлением файла /etc/dumpdates (**u**), на устройство (**f**, файл) /dev/nst0, размер тома (то есть картриджа, кассеты – **B**) – 4 194 304 Кбайт (то есть 4 Гбайт), плотность записи (**d**) 62 000 бит на дюйм (BPI), длина ленты (**s**) – 1500 футов. Аргумент *файловая_система* обязателен, так как остальные могут быть приняты по умолчанию (весьма странному, надо сказать), а какую именно файловую систему копировать, надо решать вам.

Приведенный пример справедлив для ленты Tandberg емкостью 4 Гбайт (8 при сжатии). Конкретные параметры кассет и стримера вам придется выяснить в документации на стример: разновидностей слишком много, чтобы их все можно было привести здесь.

Существуют версии **dump** с более привычным администратору синтаксисом и аналогичными ключами. Например, таков **dump** из FreeBSD 4.x и выше. Однако описанный выше синтаксис применим в любом диалекте UNIX, так как новый **dump** из соображений совместимости принимает команды и в современном, и в традиционном синтаксисе.

Важно, что каждый ленточный накопитель представлен в системе двумя файлами устройств – с перемоткой ленты по концу записи и без перемотки (табл. 9.1). Можно записать на одну кассету несколько разных файловых систем или несколько последовательных резервных копий одной системы. Для этого надо в качестве устройства для записи указывать файл устройства без перемотки ленты в конце записи. По умолчанию выбирается устройство с перемоткой.

Таблица 9.1. Название файлов устройств для ленточных накопителей

ОС	С перемоткой	Без перемотки
FreeBSD	/dev/rsa#	/dev/nrsa#
Linux	/dev/st#	/dev/nst#
Solaris	/dev/rmt/#l /dev/rmt/#n	/dev/nst#

Названия устройств могут быть другими в зависимости от типа стримера и его интерфейса (обычно SCSI). Символ «#» обозначает номер стримера, начиная с нуля. Обычно вместо решетки стоит ноль, так как в компьютере редко бывает установлено несколько стримеров.

Очень рекомендуется иметь достаточное количество кассет (их еще называют картриджами, но, по сути дела, это кассеты с магнитной лентой) для создания нужных резервных копий. Запись нескольких копий на одну ленту «от бедности» ведет к снижению надежности в несколько раз. Тем более что можно забыть перемотать ленту при восстановлении данных и в результате стереть нужные данные или потратить очень много времени на исправление своих ошибок.

Кассеты следует обязательно маркировать; к каждой из них прилагается специальный стикер. Забыть о том, на какой кассете что записано – пара пустяков, а вот вспомнить, если понадобится, не шутка!

Пользуйтесь тем, что **dump** умеет обновлять `/etc/dumpdates`, задавайте ключ **u**. В этом файле в случае успешного завершения копирования появится новая строка, описывающая, что, с какими параметрами и куда было скопировано. Потом эти сведения могут пригодиться, если придется восстанавливать данные.

Восстановление данных, записанных программой **dump**, производится командой **restore**. Ей можно указать, в какой каталог восстанавливать данные, задать новый каталог вместо прежнего, например при крахе файловой системы.

Не забывайте своевременно удалять восстановленные файлы из временных каталогов, если вы их уже перенесли по месту постоянной дислокации.

При записи нескольких файловых систем на одну кассету на ней образуется несколько томов (volumes). Поскольку ни **dump**, ни **restore** не умеют искать нужный том, вам придется озаботиться самостоятельным поиском. На кассете должно быть написано ручкой или фломастером, в каком порядке и что именно на ней записано. Необходимо перемотать ленту до нужного тома командой **mt**.

Программа **mt** оперирует понятием файла (перемотать ленту на файл вперед, на два файла назад). При записи на ленту с помощью программы **dump** таким «файлом» будет целая файловая система, пусть слово «файл» не вводит вас в заблуждение.

Кроме **dump**, на ленту можно писать командой **tar**, она записывает группу файлов в один. Программа **tar** часто применяется для упаковки дистрибутивов. Она широко распространена и годится не только для записи на ленту. С помощью **tar** можно записать группу файлов в один большой файл, который можно сжать. Как правило, сжатие выполняется с помощью программы **gzip**, и **tar** даже имеет ключ **z** для автоматического вызова этой программы. Программа **tar** более подробно описана в главе 27.

Копирование файлов на удаленный компьютер: программа rsync

Для того чтобы синхронизировать данные на дисках разных компьютеров, равно и для создания резервных копий информации, отлично подходит программа **rsync**. Эта программа служит для копирования файлов из каталога в каталог, причем один из каталогов, задействованных в операции копирования, может находиться на локальном компьютере, а другой – на удаленном. В качестве транспорта для передачи файлов **rsync** использует какой-нибудь командный процессор (в прошлом – **rsh**, в последних версиях по умолчанию – **ssh**). Выбор **ssh** в качестве транспорта позволяет передавать файл по каналу с шифрацией, и это без создания отдельного туннельного соединения!

Синтаксис вызова **rsync** очень прост:

```
rsync -avz -e ssh /home/user/localfile \
user@remote.host:/directory/on/remote/host
```

По этой команде файл `/home/user/localfile` будет копироваться на компьютер `remote.host` в каталог `/directory/on/remote/host`. Причем для успешного копирования надо будет ввести пароль пользователя user на машине `remote.host` в ответ на запрос, который не заставит себя ждать. Если параметр user не указывать, т. е. дать команду

```
rsync -avz -e ssh /home/user/localfile \
remote.host:/directory/on/remote/host
```

аутентификация будет проводиться на компьютере `remote.host` с таким же именем, как имя запустившего **rsync** пользователя. Подобным образом можно инициировать и передачу файла или целого каталога с удаленного компьютера на локальный. С точки зрения rsync файл находится на удаленном компьютере, если в пути к файлу есть символ двоеточия «:», и на локальном, если такого символа нет.

Чаще других применяются ключи **rsync**:

-a – при копировании каталогов выполнять рекурсивное копирование и всегда сохранять все свойства файла (владельца, группу, копировать символические ссылки и т. п.)

-v – выводить подробную информацию, в нее входит список передаваемых файлов, скорость передачи и т. п.

-z – сжимать файлы при передаче, чтобы передавалось быстрее

-e – указать командный процессор (remote shell), который надо использовать в качестве транспорта; категорически рекомендуется **ssh**, ибо сервер **sshd** распространен, прост в настройке и входит во все современные дистрибутивы UNIX.

Можно обойтись и вовсе без внешней транспортной программы, и тогда на одном из компьютеров, вовлеченных в обмен файлами, надо запустить **rsync** в режиме сервера:

```
rsync --daemon
```

Для успеха этого мероприятия следует обязательно создать и наполнить разумным содержанием файл /etc/rsyncd.conf. Вот пример минимального файла конфигурации для сервера **rsync**:

```
uid = root
gid = daemon
syslog facility = local5

[home]

    path = /home
    auth users = syncer
    hosts allow = 192.168.1.12
    hosts deny = 0.0.0.0/0.0.0.0
    read only = yes
    secrets file = /etc/rsyncd.secrets
```

Файл /etc/rsyncd.secrets должен содержать имя и пароль (открытым текстом), например, быть вот таким:

```
syncer:00016783249000812910283 74
```

Обратите внимание на ограничения: в нашем примере на компьютере открывается доступ к ресурсу home, но только компьютеру с адресом 192.168.1.12 и только для чтения. Предположим, мы настроили **rsync** в режиме демона на компьютере с именем homeserver.aix.com. Чтобы синхронизировать каталог /home на компьютере backup.aix.com с каталогом /home на homeserver.aix.com, нам потребуется ввести такую команду (на компьютере backup.aix.com):

```
export RSYNC_PASSWORD='00016783249000812910283 74'
rsync -e ssh -avz syncer@homeserver.aix.com::home /home
```

Переменной RSYNC_PASSWORD следует присвоить значение, идентичное значению пароля из файла /etc/rsyncd.secrets на **rsync**-сервере.

10

Работа UNIX в сети

Эта глава самая большая в книге. В ней рассмотрена работа UNIX в сети с самого начала – с основ протокола TCP/IP до сравнительно глубоких деталей. Термины, которые используются в главе, обычно поясняются в тексте. Исключение составляют общеупотребительные термины. По поводу одного из них – **firewall** – прочтите, пожалуйста, нижеследующее примечание.

Термин **firewall** обозначает ПО, применяемое для ограничения транзитного трафика через тот компьютер, на котором установлен и запущен firewall. Термин трафик обозначает все передаваемые по сети данные. **Транзитный трафик** – это данные, которые передаются через компьютер с одного интерфейса на другой и не предназначаются данному компьютеру.

В этой книге слово «firewall» переводится не так, как обычно. Его традиционно переводят с английского на русский прекрасным русским словом «брандмауэр». Это замечательное, столь привычно звучащее для русского уха слово означало в относительной древности (век этак восемнадцатый-девятнадцатый) каменную стену между домами, которая не дает распространяться огню при пожаре. Мне кажется, что: (1) английский firewall ничуть не хуже звучит и пишется, чем в прошлом немецкий brandmauer, (2) аналог с каменной стеной в любом случае плох, ибо сетевой firewall технически похож на мелкое сито или мембрану, которые пропускают только те пакеты данных, которым разрешено пройти. В самой современной литературе переводчики стали предпочитать термин «межсетевой экран», что несколько лучше, но опять ассоциируется с чем-то непроницаемым. Фактически firewall выполняет функции сита или, лучше сказать, фильтра. Так тому и быть – назовем firewall фильтром пакетов.

Сетевые стандарты и организации

Ключ к всечеловеческому языку, потерянный в Вавилонской башне, должен быть вновь искусственно выкован при помощи эсперанто.
Жюль Верн

Со времен Вавилонской башни люди помнят, что невозможно работать вместе, если не понимаешь друг друга. Чтобы заставить творения своих рук работать согласованно, люди выдумали десятки тысяч стандартов. Хорошо, что благодаря разделению труда не все должны задумываться о стандартах на водопроводные краны или хлебобулочные изделия.

Существует несколько групп стандартов, которые применяются в компьютерных сетях. Прежде всего, это стандарты ITU-T (ранее CCITT). Затем стандарты IEEE, которые обычно определяют правила работы сетевого оборудования. Кроме того, есть RFC (Request For Comments). Последние стандарты открытые, их можно раздобыть в Интернете (например, на сайте *www.ripn.net*). Они создаются группами специалистов, заинтересованных в развитии того или иного протокола или сетевой службы. После создания они публикуются и становятся открытыми для замечаний и предложений. По прошествии некоторого времени группа создателей стандарта изучает собранные предложения, улучшает стандарт и наконец принимает его в качестве окончательного. Довольно часто через год-другой выходит новый документ, заменяющий старый.

Каждый стандарт RFC имеет свой номер и определяет какой-нибудь один протокол или принцип. Кстати, протокол IP тоже введен одним из документов RFC.

Справка. Кто есть кто

CCITT (Consultative Committee on International Telegraphy and Telephony). В оригинале Comite Consultatif Internationale de Telegraphie et Telephonie, потому что находится в Женеве, где принято говорить по-французски. Группа специалистов по электронике, компьютерным системам и сетям, занимающаяся стандартизацией и составлением спецификаций, публиковала рекомендации по широкому кругу вопросов, включая определения терминов, основные принципы и характеристики, конструкции протоколов, описания моделей и другие спецификации. Группа создала ряд очень известных стандартов, таких как стандарты сжатия звука и знаменитые стандарты V., описывающие модемные соединения (V.34 и т. п.). В настоящее время называется ITU-T.

ITU-T (International Telecommunications Union-Telecommunications Standards Sector). Официальный международный комитет, ранее называвшийся CCITT. ITU-T является частью Международного телекоммуникационного союза (*www.itu.int*, International Telecommunications Union – ITU), основанного в 1948 году и спонсируемого ООН для стандартизации средств связи (изначально телефонии и телеграфной связи).

IEEE (Institute of Electrical and Electronics Engineers, *www.ieee.org*). Американское некоммерческое объединение технических специалистов, которое разрабатывает, определяет и пересматривает стандарты в области электроники и компьютерной индустрии.

ICANN (Internet Corporation for Assigned Names and Numbers, *www.icann.org*). Некоммерческая организация, ответственная за распределение IP-адресов в Интернете, управление системой имен доменов, обслуживание корневых серверов этой системы. До образования ICANN эти функции исполняла IANA по контракту с правительством США. Пять из девяти членов Совета директоров ICANN избраны голосованием пользователей Интернета.

Модель межсетевого взаимодействия

В 1980 году Международный комитет по стандартам (ISO – International Organization for Standardization) разработал модель межсетевого взаимодействия открытых систем (OSI – Open System Interconnection). Как и всякая модель, она описывает идеал, а не реальную жизнь. Модель состоит из семи явных уровней и одного неявного (рис. 10.1). Каждый уровень описывает свой тип протокола взаимодействия. Протокол, реально используемый в существующих сетях, обычно не может быть строго сопоставлен одному определенному уровню модели. Как правило, каждый реальный протокол определяет правила передачи и упаковки данных, которые с точки зрения модели надо отнести к разным уровням.

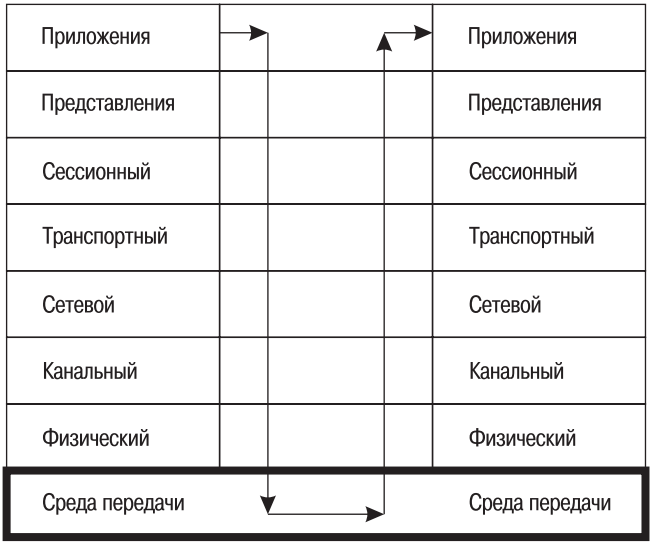


Рис. 10.1. *Модель OSI/ISO*

Модель межсетевого взаимодействия предполагает, что существует семь уровней протоколов.

Протоколы физического уровня определяют правила передачи бит данных (например, «после установления соединения устройства должны договориться о количестве стартовых и стоповых битов при передаче байта и о битах четности в передаваемом байте»).

Протоколы канального уровня определяют правила доступа к каналу передачи данных и правила использования этого канала. Грубо говоря, протокол Ethernet (IEEE 802.3, широко применяемый в локальных сетях) представляет собой пример протокола канального уровня. Протокол сетевого уровня – это, например, протокол IP (Internet Protocol, то есть «межсетевой протокол»).

Число стандартов, относящихся к передаче данных, очень велико. Стандарты семейства 802, разработанные комитетом IEEE, описывают наиболее популярные протоколы передачи данных канального уровня модели OSI.

Вот основные стандарты семейства IEEE 802:

- EEE 802.1: описание управления сетью (network management)
- IEEE 802.2: общая часть протоколов канального уровня модели OSI. IEEE разделил этот уровень на два подуровня: управления логической связью (Logical Link Control – LLC) и контроля доступа к среде (Media Access Control – MAC). Уровень контроля доступа к среде различается для сетей разных типов и определяется стандартами IEEE 802.3 – IEEE 802.5
- IEEE 802.3: определяет контроль доступа к среде в сетях типа «общая шина», использующих алгоритм доступа CSMA/CD. На этом стандарте основан Ethernet
- IEEE 802.4: определяет контроль доступа к среде в сетях типа «общая шина», использующих алгоритм доступа с передачей маркера[1] **(token bus networks)**
- IEEE 802.5: определяет контроль доступа к среде в сетях token-ring
- IEEE 802.6: стандарт для сетей Metropolitan Area Networks (MANs). MAN-сети – это сети размером больше локальных (LAN) и меньше глобальных (WAN)

Когда компьютеры стали объединять в разнородные сети, понадобилось создать протокол передачи данных, независимый от конкретной среды и оборудования передачи (модемы, сетевые адаптеры разных типов). Был придуман протокол IP, который предполагает формат пакета данных, универсальный по отношению к среде и каналу передачи. Во всем, что касается канала и среды передачи, программа, форми-

[1] Также их называют сетями с маркерным доступом.

рующая IP-пакеты, полагается на драйверы протоколов более низкого уровня. На сетевом уровне обязательно определяются сетевые адреса.

Сетевой адрес каждого сетевого интерфейса должен быть уникальным в пределах одной сети. Например, все компьютеры, подключенные к сети Интернет, имеют уникальные адреса. Правда, многие из них подключены через шлюзы, которые обращаются в Интернет от их имени. А компьютер, подключенный через шлюз, на самом деле не имеет прямого подключения к Интернету, он просто имеет туда доступ. В этом случае уникальный в пределах Интернета адрес должен иметь шлюз.

Транспортный уровень модели определяет протоколы, которые обеспечивают надежную (или ненадежную) транспортировку пакетов данных по сети посредством протокола сетевого уровня. Транспортный протокол TCP (Transmission Control Protocol – протокол передачи данных), разработанный вместе с IP, настолько тесно интегрирован с протоколом IP, что часто их упоминают вместе как протокол TCP/IP.

Сеансовый уровень[1] определяет то, как должны быть описаны правила установления сессии. Сеанс (или сессия) – это процесс, включающий в себя установление соединения, управление им и завершение соединения между приложениями. Примером сеанса может быть сеанс связи с сервером с помощью программы `telnet` или http-запрос к веб-серверу.

Протокол уровня представления описывает, как трактуются передаваемые и получаемые данные. Например, если в письме есть заголовок, говорящий о наличии приложенного к письму файла, то определенный блок текста письма будет интерпретирован как закодированный приложенный к письму файл. Протокол уровня представления, например, может определять формат такого заголовка.

Наконец, протокол уровня приложения определяет, как приложение намерено общаться с другим подобным приложением через сеть. Здесь могут быть определены правила аутентификации, синтаксис передаваемых конструкций и т. п.

Приложение (программа), работающее с сетью, может само реализовывать один или несколько протоколов, относящихся к нескольким уровням. Например, программа `telnet` фактически реализует протоколы, относящиеся к сеансовому уровню, а также уровню представления и приложения.

При передаче от одного компьютера к другому через сеть данные претерпевают изменения, определяемые протоколами всех перечисленных уровней. Протокол или группа протоколов реализуются подпрограммами приложения и операционной системы. При путешествии от высокоуровневых подпрограмм (уровень приложения) к низкоуровневым (канальный уровень) данные дробятся на пакеты и обрастают служебными заголовками. Когда компьютер-адресат получит данные, они

[1] Называемый также сессионным.

проделают обратный путь снизу вверх по стеку протоколов. Вначале драйверы и подпрограммы операционной системы соберут разрозненные пакеты данных, удалят из них служебные заголовки, затем данными займутся подпрограммы конкретного приложения-получателя.

В модели межсетевого взаимодействия неявно присутствует еще один уровень – нулевой. Это уровень среды передачи, так как физические параметры среды передачи тоже должны быть строго определены. Это может быть частотный диапазон радио- или видеосигнала, характеристики передающих лазеров, мощность сигнала и т. п.

Протокол TCP/IP

Аббревиатурой TCP/IP называют семейство протоколов, включающее TCP, IP, UDP (User Datagram Protocol), ICMP (Internet Control Message Protocol) и ряд других протоколов. Семейство протоколов TCP/IP весьма условно соотносится с моделью межсетевого взаимодействия открытых систем, поэтому в дальнейшем мы постараемся не сопоставлять протоколам семейства TCP/IP определенные уровни этой модели.

Абсолютное большинство современных операционных систем имеют встроенную поддержку TCP/IP. Многие операционные системы сейчас считают протокол TCP/IP своим основным сетевым протоколом по умолчанию. UNIX традиционно включала поддержку TCP/IP начиная с появления этого протокола. Он отлично вписался в идеологию UNIX, потому что эта операционная система изначально разрабатывалась как сетевая. Все самые передовые идеи, связанные с IP-сетями, тестировались и реализовывались в первую очередь на UNIX-платформе.

Поэтому все серьезные сетевые приложения, особенно те, в которых важна скорость реакции системы и скорость передачи данных, как правило, реализуются в среде UNIX.

Например, в Интернете можно найти такое показательное свидетельство преимуществ UNIX-систем при большой нагрузке, особенно в коммерческих Интернет-приложениях. Текст приводится в переводе и сокращении, оригинал доступен по адресу *http://kirch.net/unix-nt/hotmail.html*. Эта статья была впервые опубликована 22 апреля 1998 года компанией VNU Business Publications Ltd.

«Компания Microsoft попыталась заменить на Windows NT операционную систему, которая управляет бесплатной почтовой службой Hotmail. Служба работала на компьютерах Sun под управлением Solaris. Раньше Hotmail работала под управлением FreeBSD на нескольких многопроцессорных компьютерах с процессорами Pentuim, но было принято решение перейти на Sun ради поддержки потоков (threads). Однако выяснилось, что Windows NT не может справиться с нагрузкой, которую создают 10 миллионов пользователей службы. Одна из основных проблем в том, что веб-сервер

Apache 1.2.1, который используется на Hotmail, не может быть установлен на NT. Специалисты Hotmail сделали все, что могли, чтобы запустить службу на NT, но потерпели поражение и оставили ее работать под Solaris.»

Справедливости ради надо отметить, что сейчас, насколько можно судить по ответам почтовой системы Hotmail, она работает на веб-сервере Microsoft IIS 5.0, скорее всего, под Windows 2000. А новые версии Apache могут работать и под NT.

Подпрограмма-разборщик IP-пакетов получает пакеты, доставлемые нижними уровнями, например драйвером сетевого адаптера, и передает их выше подпрограммам уровня TCP. И наоборот, IP передает пакеты, полученные от уровней TCP и UDP, к нижележащим уровням.

Пакеты IP представляют собой датаграммы[1], протокол IP ничего не делает для обеспечения гарантии доставки IP-пакетов по порядку и без ошибок. Пакеты IP содержат адрес компьютера, с которого был послан пакет (адрес отправителя), и адрес компьютера, который должен получить пакет (адрес получателя).

Некоторые сетевые службы принимают решение о предоставлении доступа к информации на основе адреса отправителя пакета. Например, так можно настроить веб-сервер. Однако в TCP/IP есть возможность отправить пакет так, что в поле отправителя пакета будет адрес последнего маршрутизатора, через который прошел пакет. Эту возможность предоставляет режим маршрутизации от источника – так называемый source routing, режим передачи пакета, когда путь к получателю и обратный путь ответа от получателя задается отправителем пакета. Данный режим полезен в ряде случаев, например при сбое канала или невозможности передать пакет определенного типа иным путем. Изначально он предназначался для отладки сетей, а сейчас обычно используется внутри корпоративных сетей. Провайдерам Интернета настоятельно рекомендуется не принимать и не передавать пакеты, отправленные с source routing. Если внутри вашей сети используется режим маршрутизации от источника, запретите пакетам такого типа проходить через маршрутизатор. Команда зависит от маршрутизатора. В Cisco это делается командой `no ip source-route`.

Кроме адресов отправителя и получателя IP-пакет содержит номер порта отправителя и номер порта получателя. Порт в данном случае – это просто целое число в диапазоне от 0 до 65 535, которое идентифици-

[1] Датаграмма – это пакет, передаваемый через сеть независимо от других пакетов без установления логического соединения и подтверждения приема. Датаграмма сама содержит всю необходимую для ее передачи информацию. Датаграммы не содержат средств обнаружения и исправления ошибок передачи, поэтому при передаче данных с их помощью необходимо принимать меры по обеспечению надежности пересылки информации.

рует приложение, участвующее в передаче данных. Например, веб-сервер при старте сообщает ядру, что намерен «слушать» порт номер 80. Ядро заносит номер процесса веб-сервера и номер порта в таблицу. Затем, когда по сети придет пакет со значением 80 в поле порта получателя, подпрограмма разборки IP-пакетов отправит этот пакет веб-серверу в соответствии с таблицей.

Каждый сетевой интерфейс в IP-сети имеет уникальный IP-адрес (плюс, возможно, несколько адресов-псевдонимов). В мире существует много IP-сетей. Адресация в частных сетях (например, во внутренней сети лаборатории или коттеджного поселка) практически не регламентируется. Ограничения на фантазию администратора при назначении IP-адресов налагает только программное обеспечение, которое используется в такой сети. Однако если частную сеть предполагается подключить к общей сети, то назначение адресов в ней должно подчиняться вполне определенным правилам. Даже если вы планируете подключить частную сеть к Интернету через месяц, полгода или год.

Сейчас мы рассмотрим адресацию в «классическом» протоколе IP (IPv4), а IPv6 рассматривать не будем, поскольку последний не распространен еще в достаточной степени.

Все адресное пространство Интернета (всемирной IP-сети) разделено на диапазоны адресов, которые называются «сетями». Существует несколько классов сетей. За исключением специальных случаев, адреса сетевых интерфейсов компьютеров в сети должны относиться к классам A, B или C.

IP-адрес состоит из четырех байт, которые принято записывать в виде 4-х десятичных чисел, разделенных точками, например 193.124. 5.98.

Случайное совпадение адресов из примеров с реальными адресами чьих-то компьютеров действительно случайное. В самом деле, не проверять же всякий раз, когда хочешь привести пример, не совпадает ли он с каким-то реальным адресом!

Сети упомянутых классов отличаются значением первого байта IP-адреса. Значение первого байта сети класса A находится в диапазоне от 0 до 126, класса B – от 128 до 191, класса C – от 192 до 223. Классы сетей, в диапазонах адресов которых первый байт имеет значение от 224 до 254, имеют названия от D до F . Они предназначены для служебных целей, и их адреса не используются для реальных сетевых интерфейсов.

IP-адрес состоит из двух частей – номера сети и номера компьютера в сети.

Предполагается, что в сетях класса A номер сети занимает первый байт, а остальные три – это номер компьютера. Таким образом, сеть класса A может объединять до 16 777 216 компьютеров (теоретически). В сети класса B номер сети занимает два байта адреса, максимальное

количество компьютеров в такой сети – 65 536. В сети класса C номер сети занимает три байта, номер компьютера – один, максимальное количество компьютеров в такой сети – 256.

На самом деле указанное число ограничивает не количество компьютеров в сети, а количество возможных адресов сетевых интерфейсов, потому что один компьютер может иметь несколько интерфейсов, и каждый интерфейс может иметь несколько адресов. Помимо этого в сети есть зарезервированные адреса.

Это, прежде всего, адрес 127.0.0.1, всегда указывающий на локальный внутренний интерфейс. 127.0.0.1 – это слово «я» для компьютера в сети. Как для каждого человека слово «я» обозначает его самого, так и 127.0.0.1 всегда обозначает собственный интерфейс для любого компьютера. Локальный интерфейс нужен для того, чтобы одна программа (клиент) могла обратиться к другой программе (серверу), работающей на том же компьютере, стандартным образом. То есть если на вашем компьютере запущен веб-сервер, то вы должны иметь возможность на этом же компьютере обратиться к нему при помощи броузера. Локальный интерфейс обычно называется lo или lo0 (от слова loopback – петля).

IP-адрес, в котором поле номера компьютера заполнено нулями, выступает в качестве номера всей данной сети. Например, адрес 129.54.0.0 обозначает целую сеть класса B.

IP-адрес, в котором поле номера компьютера заполнено единицами (имеется в виду двоичное представление байта), называется широковещательным (broadcast) адресом сети и используется для рассылки пакетов всем компьютерам данной сети одновременно. Получив пакет с адресом получателя 213.222.1.255, каждый компьютер сети 213.222.1.0 (класс C) воспримет этот пакет как предназначенный себе.

Адрес 0.0.0.0 зарезервирован для указания «всех остальных адресатов». Пакеты, предназначенные для отправки «всем остальным», направляются в шлюз. Шлюз – это такое место, куда любой компьютер сети отправляет пакет, если не знает, в какую сторону его лучше отправить. Действие такого шлюза подобно действию почтальона. Если вы хотите передать письмо своей соседке по дому, то можете просто положить конверт в ее почтовый ящик. Если же отправляете письмо маме, которая живет в другом городе, то положите конверт в другой почтовый ящик – в тот, из которого почтальон вынимает почту для отправки. Шлюз имеет сетевой интерфейс, который работает таким «почтовым ящиком» для пакетов, адресованных за пределы локального сегмента сети.

Теперь представим себе небольшую сеть, изображенную на рис. 10.2. Сеть разбита на два сегмента, имеющих доступ друг к другу и в Интернет через маршрутизатор. Таким образом, маршрутизатор имеет как минимум три сетевых интерфейса.[1] Те интерфейсы, к которым под-

[1] На самом деле четыре, потому что всегда есть локальный интерфейс lo0.

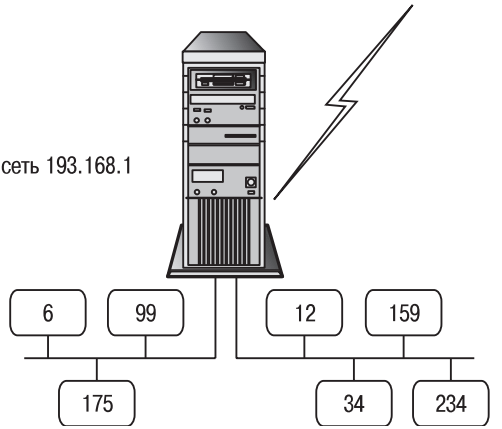


Рис. 10.2. *Пример локальной сети*

ключены локальные сегменты сети, имеют соответственно адреса 193.168.1.1 и 193.168.1.129. Предположим, из Интернета пришел пакет, адресованный компьютеру с адресом 193.168.1.12. Как маршрутизатор решает, через какой сетевой интерфейс передать этот пакет? Он смотрит в свою таблицу маршрутизации, где написано примерно следующее:

куда	**через что**
193.168.1.12	193.168.1.1
193.168.1.159	193.168.1.1
193.168.1.34	193.168.1.1
193.168.1.6	193.168.1.129
193.168.1.75	193.168.1.129
193.168.1.199	193.168.1.129

Так пакет попадает в нужный сегмент локальной сети через сетевую карту с IP-адресом 193.168.1.1. Теперь представим себе, что компьютеров в локальной сети не семь, а двести восемь, и пакеты приходят со скоростью один мегабайт в секунду. Каждый пакет требует одинакового внимания: маршрутизатор решает, куда отправить каждый из пришедших пакетов. Размер пакетов при ethernet-соединениях обычно равен 1500 байт, а это значит, что при скорости один мегабайт в секунду в маршрутизатор за секунду приходит 699 пакетов. Стало быть, надо 699 раз в секунду просмотреть всю таблицу маршрутизации. Если она состоит из 208 записей и в среднем искомая запись обнаруживается после просмотра половины таблицы (позволим себе такое разумное допущение), то в секунду маршрутизатор должен будет просмотреть

699 × 208/2=72 696 строк таблицы. Маршрутизатор будет, что и говорить, занят делом.

В то время, когда разрабатывали TCP/IP, больше думали об эффективности сети, а не о том, как заработать побольше денег, заставив доверчивого покупателя все время покупать все более мощные системы. Поэтому было решено минимизировать затраты на маршрутизацию довольно простым способом.

Для мира UNIX вообще характерно стремление найти эффективное и красивое решение, пусть даже оно будет недорогим. Именно поэтому энтузиасты пишут тысячи серьезных и полезных программ, на которых держится сеть Интернет и многие ее коммерческие службы, поэтому есть бесплатные версии UNIX и из мира UNIX вырос проект бесплатного программного обеспечения GNU. Вы заметили, что бесплатной версии Windows, какой-нибудь Lindows XP, пока что нет?

Вернемся к локальной сети, описанной чуть выше, и представим себе, что договорились назначать компьютерам в одном сегменте адреса с 193.168.1.0 по 193.168.1.128, а в другом с 193.168.1.129 по 193.168.1.255. Таблица маршрутизации сжимается до двух строк:

193.168.1.0128	193.168.1.1
193.168.1.129255	193.168.1.129

Правда, теперь надо объяснить маршрутизатору, что такое диапазон адресов.

Это было предложено делать так: каждому IP-адресу была сопоставлена так называемая маска сети. Маска сети (netmask) состоит из четырех байтов. В тех битах маски, которые в адресе соответствуют номеру сети, стоят единицы, а в тех, что в адресе отвечают за номер компьютера, в маске стоят нули. Пусть адрес равен 193.168.1.5, а маска – 255.255.255.0, тогда 193.168.1 – номер сети, 5 – номер компьютера в ней. В двоичном представлении маска 255.255.255.0 выглядит так:

11111111 11111111 11111111 00000000

Такая маска по умолчанию задана для любого компьютера сети класса C. Однако если мы хотим разделить нашу сеть на две части, как предложено выше, то необходимо указать, что номер сети будет длиннее: он займет еще один (старший бит) младшего байта IP-адреса. Фактически этот бит будет указывать, к какой подсети всей нашей сети относится компьютер. Маска становится такой (255.255.255.128):

11111111 11111111 11111111 10000000

Теперь, если IP-адрес будет попадать в диапазон с 193.168.1.0 по 193.168.1.128, сразу будет видно, что он из подсети номер 0, так как

числа от 0 до 128 имеют двоичный ноль в старшем бите. Аналогично по адресу из диапазона с 193.168.1.129 по 193.168.1.255 будет видно, что он из подсети номер 1.

Видно, что теоретически можно указать маску, где номер сети будет расположен не подряд, что-нибудь вроде

11111111 11111111 11111101 10101000

Но исторически сложилось так, что это признано неудобным и не принято. Поэтому маска всегда представляет собой последовательность единиц, записанную, начиная со старшего байта и заканчивая тем, который соответствует классу сети и количеству подсетей в ней. Поэтому маску сейчас часто записывают после IP-адреса через дробь в виде количества единиц в маске. Например, адрес 193.168.1.5, которому соответствует маска 255.255.255.0, может быть записан как 193.168.1.5/24.

CIDR

В связи с быстрым ростом Интернета в конце 80-х – начале 90-х годов XX века стали проявляться серьезные недостатки в организации распределения адресного пространства:

- Нехватка адресов. Она заключается в том, что размеры существующих классов сетей не отражают требований средних организаций. Количество компьютеров в сети организации часто больше, чем адресов в сети класса C, но сильно меньше, чем в сети класса B.
- Сложности обработки таблиц маршрутизации. Рост размеров таблиц в интернет-маршрутизаторах приводит к тому, что их становится сложно администрировать, а сами маршрутизаторы работают на пределе своих возможностей.

Поэтому в июне 1992 года IETF (InternetEngineering Task Force) принял решение о применении технологии бесклассовой междоменной маршрутизации CIDR (Classless Inter-Domain Routing) для решения этих проблем. В 1994–1995 годах технология была внедрена интернет-провайдерами для маршрутизации между их сетями. Термин «междоменная» относится к так называемым доменам маршрутизации, то есть совокупностям сетей, объединенных общим администрированием и политикой маршрутизации. Домен маршрутизации обычно ограничен сетью крупного провайдера и его субпровайдеров. CIDR может применяться в любой группе доменов Интернета, работающих как с IPv4, так и с IPv6, и успешно взаимодействовать со старыми технологиями адресации. В основе CIDR лежит принцип использования маски сети переменной длины (VLSM – Variable Length Subnet Masks) и отказ от деления Интернета на сети классов A, B и C. Тогда все организации, предоставляющие услуги Интернета, будут разделяться не по классам своих сетей, а по маске предоставленного им адреса.

Согласно идеологии CIDR провайдеры должны объединять в одну запись информацию о блоках адресов своих клиентов и анонсировать в Интернете свой блок адресов в целом. Раньше (до 1994 года) анонсировалась информация о каждой из сетей каждого класса в отдельности.

Например, интернет-провайдер «Быстрое ухо», которому NIC делегировал адреса с 198.24.0.0 по 198.31.255.255 (маска 255.248.0.0), может назначить своему клиенту фирме «Мускул» группу адресов с 198.24.8.0 до 198.24.11.255 (маска 255.255.252.0). Ясно, что сеть «Мускула» не принадлежит ни к какому классу и, тем не менее, может успешно работать в Интернете. Очевидно, что новая структура адресного пространства обеспечит значительную экономию адресов для тех организаций, которым они действительно нужны, и взять лишние у других. Если организация желает получить пул адресов для работы в Интернете, она заказывает диапазон адресов `<адрес, маска>`, исходя из своих потребностей. Если, к примеру, требуется подсоединить к Интернету 999 компьютеров, то не надо заказывать четыре сети класса С или одну класса В, достаточно заказать адрес и маску, позволяющие работать такому количеству компьютеров, например < 198.24.8.0, 255.255.252.0> – 1024 сетевых адреса.

Во-первых, это удобно из экономических соображений: не приходится платить за неиспользуемые адреса. Во-вторых, сетевым администраторам не надо хлопотать о настройке маршрутизации между группами интерфейсов, принадлежащих разным классам. В-третьих, значительно упрощается управление единой сетью. Как всякий переход на новую технологию, изменение иерархии Интернета, видимо, будет осуществляться одновременно с переходом на новый тип адресации (128-битной) и новый сетевой протокол IPv6. К тому времени значительно изменятся и стандарты, и принципы управления сетями. В настоящее время CIDR поддерживается некоторыми протоколами внутридоменной (OSPF, RIP-2, E-IGRP) и междоменной маршрутизации (BGP-4).

CIDR описывается в RFC 1519. Одно из лучших в сети русскоязычных описаний протокола CIDR находится по адресу *http://www.acnet.ge/networking/net_l/routing/internet/cidr.htm*. Оно было использовано при подготовке этого раздела.

Настройка TCP/IP

В большинстве диалектов UNIX есть административные утилиты, которые позволяют быстро и с удобствами назначить все параметры сетевого интерфейса в одном окне (текстовом или графическом – неважно). Как минимум должны быть указаны адрес и маска сети интерфейса. Если предполагается, что надо будет работать не только со своим сегментом локальной сети, следует указать также шлюз (gateway), а если хочется обращаться к компьютерам не по IP-адресу, а по имени, придется настроить **`resolver(3)`**.

Назначение адреса и маски интерфейсу

Основная программа настройки сетевого интерфейса в UNIX – **ifconfig**. При старте системы эта команда запускается для настройки сетевых интерфейсов, указанных в файлах конфигурации системы. Чтобы посмотреть, какие есть интерфейсы, надо дать команду **ifconfig -a** (ключ **-a** требует показать все (all) интерфейсы, которые есть в системе):

```
ifconfig -a
rl0: flags=8843<UP,BROADCAST,RUNNING,SIMPLEX,MULTICAST> mtu
1500 inet 195.77.192.166 netmask 0xfffffffc broadcast
195.77.192.167 inet6 fe80::2e0:29ff:fe62:af22%rl0 prefixlen
64 scopeid 0x1 ether 00:e0:29:62:af:22 media: 100baseTX
<full-duplex> status: active supported media: autoselect
100baseTX <full-duplex> 100baseTX 10baseT/UTP <full-duplex>
10baseT/UTP 100baseTX <hw-loopback>
ed0: flags=8843<UP,BROADCAST,RUNNING,SIMPLEX,MULTICAST> mtu
1500 inet 193.114.38.33 netmask 0xffffffe0 broadcast
193.114.38.63 inet6 fe80::200:1ff:fe39:96%ed0 prefixlen 64
scopeid 0x2 ether 00:00:01:39:00:96
ppp0: flags=8010<POINTOPOINT,MULTICAST> mtu 1500
sl0: flags=c010<POINTOPOINT,LINK2,MULTICAST> mtu 552
lo0: flags=8049<UP,LOOPBACK,RUNNING,MULTICAST> mtu 16384
inet6 fe80::1%lo0 prefixlen 64 scopeid 0xb inet6 ::1
prefixlen 128
inet 127.0.0.1 netmask 0xff000000
inet 193.124.85.15 netmask 0xffffffff
inet 193.114.38.7 netmask 0xffffffff
```

Каждый блок информации сообщает об одном интерфейсе, имена интерфейсов расположены в начале блоков и совпадают с именами файлов устройств в каталоге /dev. Устройство rl0 – это сетевой адаптер на чипе, подобном Realtek (например, SMC9432), ed0 – NE-2000-совместимая сетевая карта, ppp0 и sl0 – интерфейсы для последовательных соединений, например через модем, lo0 – «петля», локальный интерфейс. Интерфейс может быть в одном из двух состояний – UP и DOWN. UP – интерфейс работает, DOWN – нет. Наличие интерфейсов, связанных с сетевыми картами, но не работающих, – нехороший признак. Возможно, неисправна сетевая карта или произошел обрыв кабеля. То, что интерфейсы ppp0 и sl0 не находятся в состоянии UP, нормально. Это лишь значит, что связь через последовательный порт ни с кем не установлена. Например, если мы рассматриваем работу сервера удаленного доступа, это значит, что интерфейс готов к работе, но пока никто не дозвонился до сервера.

Программа **ifconfig** сообщает нам не только состояние интерфейса, но и все параметры его настройки: IP-адрес, маска сети, максимальный размер пакета (MTU – Maximum Transmission Unit) на интерфейсе, широковещательный адрес. Новые версии **ifconfig**, как в нашем

примере, сообщают также адрес в формате IPv6. Пока такие адреса поддерживаются новыми операционными системами, но не используются в общих сетях.[1]

Кроме информации о настройках, **ifconfig** выдает сведения о сетевом адаптере (MAC-адрес и в зависимости от драйвера еще может сообщать о поддерживаемой среде передачи данных, например 100 BaseTX fullduplex).

Каждый интерфейс может иметь адрес-псевдоним – alias. В нашем примере у интерфейса lo0 даже два псевдонима – 193.124.85.15 и 193.114. 38.7. Псевдоним может потребоваться для локального интерфейса, если важно сохранить старый адрес машины, но нет возможности назначить его ни одному физическому интерфейсу. Такое может случиться, если внутренние адреса локальной сети изменились, но надо продолжать делать вид, что существуют и прежние адреса, и нынешние.

Физический интерфейс тоже может иметь псевдонимы. Это способ заставить компьютеры из одного физического сегмента локальной сети общаться друг с другом только через маршрутизатор. В этом случае маршрутизатором может быть даже компьютер с одной сетевой картой, просто у нее будет два IP-адреса. Так, например, можно обезопасить компьютеры бухгалтерии, работающие в одном физическом сегменте с компьютерами менеджеров или посетителей компании. Назначение компьютерам бухгалтерии IP-адресов из другой сети (другого адресного пространства – не того, что используется всеми остальными сотрудниками) сделает работу безопаснее. Правда лишь ненамного: это спасет только от случайного подключения к бухгалтерскому компьютеру, а не от взлома сети с прослушиванием сообщений на уровне сетевой карты. Последнее только звучит солидно, зато реализуется легко, например с помощью программы **tcpdump**.

Программа **ifconfig** может использоваться для того, чтобы назначить интерфейсу адрес и маску:

ifconfig *имя_интерфейса IP-адрес* **netmask** *маска_сети*

Если маска стандартная (то есть 255.0.0.0 для сети класса A, 255.255. 0.0 для сети класса B и 255.255.255.0 для сети класса C), то netmask можно не указывать: **ifconfig** назначит правильную по умолчанию.

---

[1] Протокол IPv6 уже довольно давно стандартизирован, однако применяется далеко не везде, потому что во многих коммерческих системах и приложениях пока не реализована его поддержка. В США правительство и 190 учебных заведений и частных компаний развернули проект Internet2, в рамках которого разрабатываются и тестируются новые приложения, в частности связанные с внедрением IPv6. В Европе тоже есть подобный проект – Geant. Подробности о проектах есть на их сайтах – *www.internet2.edu* и *www.geant.net*.

Если же вы разделили сеть на подсети, то маску следует указать явно. Помните, что на всех компьютерах в одной подстети должна быть одинаковая маска, иначе они друг друга не увидят. Неаккуратно написанное программное обеспечение некоторых операционных систем (обычно не UNIX) иногда позволяет обращаться к компьютерам в своей сети, невзирая на разницу масок. Это ошибка: такого быть не должно.

Для добавления псевдонима используйте **ifconfig** с параметром **alias**:

```
ifconfig имя_интерфейса alias IP-адрес
```

Иногда **ifconfig** из своих соображений устанавливает неверный широковещательный адрес при настройке интерфейса. В таком случае надо дать команду **ifconfig** с параметром **broadcast**, указав верный широковещательный адрес:

```
ifconfig имя_интерфейса broadcast широковещательный_IP-адрес
```

В разных диалектах **ifconfig** может иметь разный синтаксис и ключи, если приведенный вариант не сработал, обратитесь к **man**.

Указание адреса шлюза

Адрес шлюза обычно устанавливается при старте системы из стартового скрипта. Если при установке системы он не был задан или вы хотите изменить этот адрес, то можно его задать в стартовом скрипте (подробности можно найти в разделе «Где находятся базовые настройки сети в UNIX» далее в этой главе) либо выполнить команду:

```
route delete default
route add default адрес_шлюза
```

Для некоторых диалектов UNIX (в частности, Linux) команда будет немного иной:

```
route delete default
route add default gw адрес_шлюза
```

Обратите внимание на ключевой параметр **gw** – *адрес_шлюза*.

Проверка соединения

Для того чтобы проверить, есть ли связь между двумя сетевыми интерфейсами, выполняют команду **ping**. Программа **ping** посылает маленькие (обычно 56 или 64 байт) пакеты специального типа по протоколу ICMP (Internet Control Messages Protocol). Отвечая на такой пакет, компьютер-получатель обязан прислать точно такой же пакет. Программа заработала себе такое имя благодаря сходству процесса с игрой в ping-pong.

По приходе ответа отмечается время, которое понадобилось пакету на путешествие туда и обратно. Поэтому **ping** применяют не только для проверки наличия соединения, но и для оценки его скорости. Последнее не совсем корректно, потому что пакеты разных протоколов и разной длины могут передаваться по сети с разной скоростью. Более того, в зависимости от настроек маршрутизаторов по пути к получателю пакеты разных протоколов могут следовать по разным траекториям. Из-за этого измерение скорости соединения между компьютерами с помощью **ping** позволяет лишь примерно оценить, как быстро между этими компьютерами будут, например, пересылаться файлы.

В UNIX **ping** посылает пакеты до тех пор, пока ее работа не будет прервана по <Ctrl>+<C>. Программа **ping** измеряет среднее время передачи пакета и сообщает также минимальное и максимальное время передачи за сеанс своей работы.

Для проверки корректности работы подпрограмм, связанных с передачей IP-пакетов на вашем компьютере, скомандуйте:

```
ping 127.0.0.1
PING 127.0.0.1 (127.0.0.1): 56 data bytes
64 bytes from 127.0.0.1: icmp_seq=0 ttl=255 time=0.189 ms
64 bytes from 127.0.0.1: icmp_seq=1 ttl=255 time=0.190 ms
64 bytes from 127.0.0.1: icmp_seq=2 ttl=255 time=0.177 ms
^C
--- 127.0.0.1 ping statistics ---
3 packets transmitted, 3 packets received, 0% packet loss
round-trip min/avg/max/stddev = 0.177/0.185/0.190/0.006 ms
```

Ответ должен приходить практически мгновенно. Также почти мгновенно должен отвечать физический сетевой интерфейс вашего компьютера.

```
ifconfig -a
ed0: flags=8843<UP,BROADCAST,RUNNING,SIMPLEX,MULTICAST> mtu
1500 inet 193.114.38.33 netmask 0xffffffe0 broadcast
ping 193.114.38.33
PING 193.114.38.33 (193.114.38.33): 56 data bytes
64 bytes from 193.114.38.33:icmp_seq=0 ttl=255 time=0.212ms
64 bytes from 193.114.38.33:icmp_seq=1 ttl=255 time=0.175ms
64 bytes from 193.114.38.33:icmp_seq=2 ttl=255 time=0.172ms
64 bytes from 193.114.38.33:icmp_seq=3 ttl=255 time=0.169ms
64 bytes from 193.114.38.33:icmp_seq=4 ttl=255 time=0.172ms
64 bytes from 193.114.38.33:icmp_seq=5 ttl=255 time=0.173ms
^C
--- 193.114.38.33 ping statistics ---
6 packets transmitted, 6 packets received, 0% packet loss
round-trip min/avg/max/stddev = 0.169/0.179/0.212/0.015 ms
```

Для проверки соединения с удаленным компьютером наберите:

ping *адрес_удаленного_компьютера*

Для проверки предельных возможностей сети используйте команду

```
ping -f адрес
```

Это развлечет тех, кто работает в сети. Команда **ping -f** посылает пакеты с высокой частотой, а именно сто раз в секунду или как только придет ответ на предыдущий пакет; выбирается вариант, позволяющий послать больше пакетов. Сеть при этом забивается насмерть, и все остальные пользователи не могут послать свои данные. Эта команда дозволена только пользователю root. Наверное, в порядке льготы за трудную работу.

Настройка обращения к DNS

Те, кому приходилось хоть раз настраивать любую Windows-систему для работы с Интернетом, помнят, что кроме адреса и маски нужно было указывать шлюз (gateway) и адрес DNS-сервера. Настройки интерфейса в UNIX те же самые, только находятся в другом месте. За обращение к DNS-серверу (также называемому сервером имен) отвечают два файла: /etc/host.conf и /etc/resolv.conf. В первом указывается порядок обращения к источникам информации о соответствии IP-адреса имени. Файл /etc/host.conf краток, как сама сестра таланта:

```
cat /etc/host.conf
order hosts, bind
```

В Solaris вместо host.conf используется файл /etc/inet/nsswitch.conf.

В нашем случае там сказано, что сначала надо обращаться к файлу /etc/hosts, а если в нем ответ не нашелся, то к серверу имен. Ключевое слово order говорит само за себя, hosts означает /etc/hosts, а bind – сервер имен. Подчеркнем, слово hosts в host.conf не имя файла с информацией, а указание, что смотреть надо в /etc/hosts. Если вместо hosts написать, скажем, hostnames, то в файл hostnames система и не подумает смотреть.

Если дело дошло до обращения к серверу имен, то его адрес берется из файла /etc/resolv.conf.

```
cat /etc/resolv.conf
domain     slava.ru
nameserver 193.114.38.65
```

Ключевое слово domain указывает, какое имя домена надо добавлять к именам компьютеров, набранных без точек (то есть zeus, а не zeus.slava.ru). Допускается до трех строк с ключевым словом nameserver.

Файл hosts

Чтобы помнить не адреса компьютеров в локальной сети, а лишь их имена, в файле /etc/hosts указывают соответствия IP-адресов именам компьютеров. Это забота системного администратора – обновлять по мере необходимости этот файл.

```
cat /etc/hosts
# Host Database
# This file should contain the addresses and aliases
# for local hosts that share this file.
127.0.0.1   localhost localhost.my.domain myname.my.domain
#
# Imaginary network.
#10.0.0.2               myname.my.domain myname
#10.0.0.3               myfriend.my.domain myfriend
```

Первый столбец в файле – IP-адрес, второй – имя компьютера, остальные – псевдонимы. Файл hosts обычно не используется в сетях, подключенных к Интернету, так как удобнее получать всю необходимую информацию от сервера имен (см. главу 11). Имена и IP-адреса самых важных компьютеров, доступ к которым следует иметь и тогда, когда сервер имен недоступен, надо указать в файле /etc/hosts. Например, чтобы сетевой файловый сервер был доступен для монтирования с него файловых систем при неработающем сервере имен, надо адрес файлового сервера внести в /etc/hosts.

Имя localhost используется для интерфейса с адресом 127.0.0.1.

Маршрутизация

Маршрутизация – это процесс передачи пакетов между сетевыми интерфейсами. Маршрутизация предполагает существование маршрутов. Если компьютер подключен по единственному каналу к одному провайдеру, то от этого компьютера до провайдера маршрут один – по этому каналу. Но в Интернете пакет, как правило, может добраться от одного компьютера до другого разными путями (рис. 10.3).

Для выбора оптимального пути разработали протоколы маршрутизации. Любая система, включенная в сеть, хранит таблицу маршрутизации. В UNIX ее можно посмотреть командой **netstat -rn**.

```
netstat -rn
Routing tables
Internet:

Destination       Gateway           Flags Refs Use       Netif Expire
default           195.70.192.165    UGSc  63   100657856 rl0
127.0.0.1         127.0.0.1         UH    0    2105768   lo0
```

```
193.124.85.15     193.124.85.15     UH    0    0         lo0
193.114.38.6      193.114.38.6      UH    0    0         lo0
193.114.38.7      193.114.38.7      UH    0    0         lo0
193.114.38.8      193.114.38.8      UH    0    0         lo0
193.114.38.9      193.114.38.9      UH    0    0         lo0
193.114.38.10     193.114.38.10     UH    0    0         lo0
193.114.38.11     193.114.38.11     UH    0    0         lo0
193.114.38.32     ff:ff:ff:ff:ff:ff UHLWb 0    1093      ed0   =>
193.114.38.32/27  link#2            UC    0    0         ed0   =>
193.114.38.33     0:0:1:39:0:96     UHLW  0    220       lo0
193.114.38.64     ff:ff:ff:ff:ff:ff UHLWb 0    1075      ed1   =>
193.114.38.64/27  link#3            UC    0    0         ed1   =>
193.114.38.65     0:20:4c:1:c5:6    UHLW  2    380950    lo0
195.70.192.164/30 link#1            UC    0    0         rl0   =>
195.70.192.165    0:80:c8:c9:34:63  UHLW  51   0         rl0   138
195.70.192.166    0:e0:29:62:af:22  UHLW  0    6120      lo0
```

Это пример относительно сложной конфигурации, когда есть несколько сетевых интерфейсов и один из них имеет несколько псевдонимов.

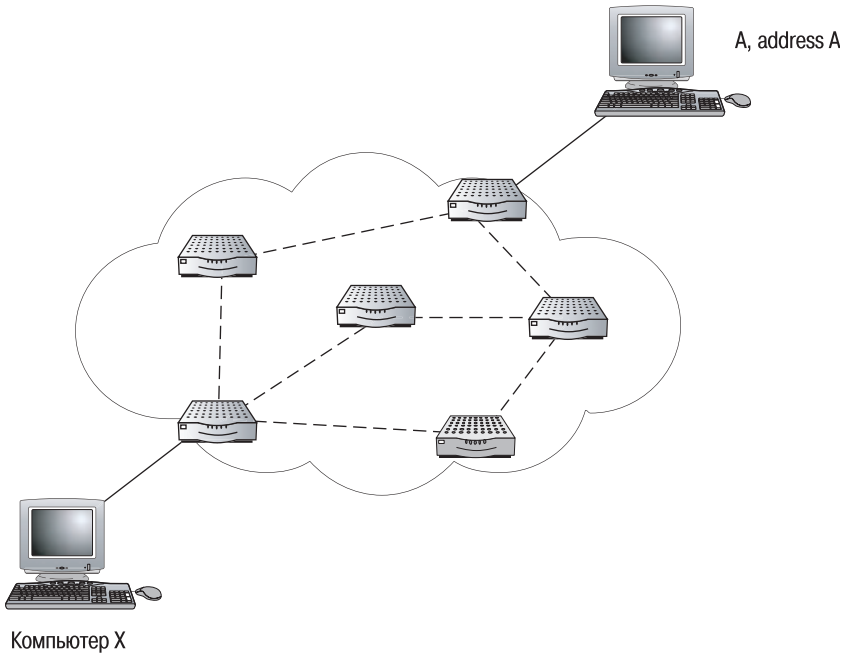


Рис. 10.3. *Возможные маршруты пакета*

Простейшая конфигурация, например, такова (один интерфейс, псевдонимов нет):

```
netstat -rn
Kernel IP routing table
Destination Gateway     Genmask         Flags MSS  Window irtt  Iface
192.168.5.18 0.0.0.0    255.255.255.255 UH    40   0      0     eth0
192.168.5.0  0.0.0.0    255.255.255.0   U     40   0      0     eth0
127.0.0.0    0.0.0.0    255.0.0.0       U     40   0      0     lo
0.0.0.0      192.168.5.1 0.0.0.0        UG    40   0      0     eth0
```

Формат вывода одной и той же команды в приведенных примерах разный, потому что в первом примере компьютер работает под FreeBSD, а во втором – под Linux. Однако смысл выводимой информации схож. Выводится информация о том, в какие сети через какие интерфейсы отправлять пакеты. В поле `Destination` (получатель) может быть указана и сеть, и конкретный компьютер. В поле `Gateway` (шлюз) указывается, через какой сетевой интерфейс (возможно, удаленный) надо переслать пакет данному получателю. Во FreeBSD **`netstat`** выводит в поле **`Gateway`** MAC-адрес интерфейса, если получателем числится непосредственно присоединенная к компьютеру сеть. *Непосредственно присоединенной сетью* (*directly connected network*) в терминах маршрутизации называется сеть, для достижения которой пакет не должен проходить через маршрутизатор. Получатель 0.0.0.0 – это обозначение для всех интерфейсов вообще. В шлюз для этого получателя посылаются все пакеты, фактические получатели которых **не** обозначены явным образом в таблице маршрутизации.

Любой компьютер, подключенный к сети TCP/IP, всегда имеет свою собственную таблицу маршрутизации. В UNIX маршрутизация является функцией ядра системы.

Флаги, соответствующие каждому получателю, означают его статус. U (up) означает работоспособность маршрута, G (gateway) означает возможность пересылать пакеты через получателя дальше.

Некоторые другие флаги указаны в табл. 10.1.

Таблица 10.1. Некоторые флаги состояний сетевых интерфейсов

Флаг	Мнемоническое обозначение	Значение
b	RTF_BROADCAST	маршрут представляет собой широковещательный адрес
C	RTF_CLONING	требуется построить новый маршрут
G	RTF_GATEWAY	получатель достижим через промежуточный маршрутизатор
H	RTF_HOST	получатель – компьютер, а не сеть

Флаг	Мнемоническое обозначение	Значение
M	RTF_MODIFIED	маршрут добавлен динамически
R	RTF_REJECT	получатель недоступен
S	RTF_STATIC	маршрут добавлен статически (вручную)
U	RTF_UP	получатель работоспособен

Построение нового маршрута указывается для получателей, представляющих собой сеть в целом.

Таблица маршрутизации строится при старте системы из стартовых скриптов на основе информации файлов конфигурации сети (см. раздел «Где находятся базовые настройки сети в UNIX»). При необходимости ее корректировки вручную используется команда **route**.

Программа **route** изменяет таблицу маршрутизации в ядре. С ее помощью можно удалять записи из таблицы, вносить новые, изменять старые. Синтаксис команды:

```
route [-dnqtv] команда [[модификаторы] аргументы]
```

- **-n** – выводить на экран IP-адреса, а не имена компьютеров/сетей. Для вывода имени его еще надо определить по адресу, это занимает время и может не сработать при сбое настроек или сети;
- **-v** – печатать детали;
- **-q** – делать дело молча.

Команды, которые выполняет программа **route**:

- **add** – добавить маршрут;
- **flush** – удалить все маршруты;
- **delete** – удалить один указанный маршрут;
- **change** – изменить маршрут (например, шлюз для сети);
- **get** – найти и показать маршрут для получателя;
- **monitor** – следить за маршрутом и сообщать об изменении информации в таблице. Имеет смысл для динамически обновляемых маршрутов. Команда **monitor** выглядит так:

```
route [-n] monitor
```

Обычно **route** вызывается для изменения или добавления маршрута следующим образом:

```
route [-n] команда [-net | -host] куда через_что [netmask]
```

Для указания всех интерфейсов вообще предназначено ключевое слово `default`, синоним

```
-net 0.0.0.0.
```

Если для указания маршрута недостаточно указать IP-адрес назначения или шлюза, можно дополнительно указать интерфейс с помощью модификатора **-ifp** или **-ifa** (имя устройства или адрес интерфейса соответственно). Дело в том, что в специальных случаях один адрес может принадлежать нескольким интерфейсам.

Возможные ответы программы **route** и их смысл поясняются ниже.

```
Network is unreachable
```

Маршрут не добавить, шлюз находится вне сети, присоединенной непосредственно. Похоже, вы ошиблись адресом.

```
not in table
```

Удалить или изменить эту строку в таблице нельзя, потому что ее там нет. Похоже, снова ошиблись адресом.

```
routing table overflow
```

Нет места в памяти, чтобы добавить строку в таблицу. Проверьте, кто украл так много памяти.

Команда **route** может иметь несколько отличный от вышеприведенного синтаксис в разных диалектах UNIX. Например

route add -net *сеть_получателя* **netmask** *маска* **gw** *шлюз*
route delete -net *сеть_получателя*

Слово default вместо адреса сети назначения всегда обозначает маршрут по умолчанию, и в этом случае маска не требуется. В остальных случаях, если маска не указана, **route** выберет ее из своих соображений.

Если устанавливается маршрут до сети, к которой данный компьютер подсоединен непосредственно, в поле *шлюз* указывается адрес IP-интерфейса компьютера, через который эта сеть достижима; если же адрес шлюза не указан, то **route** сама выберет наиболее подходящий IP-интерфейс для связи с этой сетью.

В маршрутизации используется понятие метрики. Метрика – это количество маршрутизаторов, через которые должен пройти пакет до получателя. В Solaris метрику можно указать непосредственно в таблице маршрутизации:

route add|delete host|net *получатель шлюз метрика*

Обычно метрика устанавливается протоколом динамической маршрутизации. При динамической маршрутизации информация о маршрутах в сети распространяется по сети. Службы маршрутизации (демоны **routed** или **gated**) принимают эту информацию и обновляют локальные таблицы маршрутизации.

Нулевая метрика говорит о том, что пункт назначения доступен непосредственно, ненулевая – что маршрут до пункта назначения проходит через промежуточный узел (шлюз).

Более подробное описание **route** выводится командой **man route** (по крайней мере, в Linux и Solaris).

При маршрутизации могут быть задействованы фильтры пакетов (firewalls) – программное обеспечение, разрешающее или запрещающее определенные виды передачи данных в сети. Оно также предоставляет возможности трансляции адресов (см. раздел «NAT» данной главы) и ограничение скорости передачи данных определенного типа через заданный интерфейс (см. раздел «Защита сетей с помощью фильтров пакетов»).

Где находятся базовые настройки сети в UNIX

Сейчас пойдет речь о том, где записаны настройки сетевых интерфейсов системы. Присвоение IP-адресов выполняется каждый раз при старте системы из стартовых скриптов. Эти скрипты пользуются файлами конфигурации, из которых они считывают информацию. В скриптах так или иначе выполняются команды **ifconfig** и, возможно, **route**.

Файлы, в которых содержатся основные настройки сетевых интерфейсов, в разных диалектах UNIX находятся в разных местах и по-разному называются.

В этом разделе книги указано, где искать упомянутые настройки во FreeBSD, Linux и Solaris. В других системах вам придется их находить экспериментальным путем или путем внимательного чтения сопроводительной документации.

FreeBSD

Многие настройки системы, в том числе адреса интерфейсов, находятся в **/etc/rc.conf**, ниже приведен фрагмент файла из работающей системы:

```
cat /etc/rc.conf
…
hostname="imc.example.ru"# Hostname
# List of network interfaces (lo0 is loopback)
network_interfaces="rl0 lo0 ed0 ed1"
# default loopback device configuration."
ifconfig_lo0="inet 127.0.0.1"
ifconfig_lo0_alias1="193.114.38.7 netmask 255.255.255.255 \
193.124.85.193"
ifconfig_lo0_alias2="193.114.38.8 netmask 255.255.255.255 \
193.124.85.193"
ifconfig_ed0="inet 193.114.38.33 netmask 255.255.255.224"
ifconfig_ed1="inet 193.114.38.65 netmask 255.255.255.224"
…
```

Linux

В классическом Linux настройки сетевых интерфейсов выполняются командами **ifconfig** прямо из стартовых скриптов. Скрипты вызываются из главного стартового скрипта /etc/rc.d/rc. Главный стартовый скрипт может иметь другое имя, это зависит от того, что записано в /etc/inittab. Как правило, /etc/rc.d/rc – это именно то, что нужно. В этом скрипте после проверки дисков перед запуском большинства демонов выполняется настройка сети.

В некоторых диалектах Linux (RedHat, Mandrake) настройки сетевых интерфейсов и часть других настроек сети прячутся в нескольких файлах в каталоге /etc/sysconfig:

```
cd /etc/sysconfig
cat network
NETWORKING=yes
FORWARD_IPV4=false
HOSTNAME=ixy.co.spb.ru
DOMAINNAME=co.spb.ru
GATEWAY=192.168.5.1
cd /etc/sysconfig/network-scripts
ls ifcfg-*
ifcfg-eth0
ifcfg-ippp0
ifcfg-lo
cat ifcfg-eth0
DEVICE=eth0
BOOTPROTO=static
IPADDR=192.168.5.18
NETMASK=255.255.255.0
NETWORK=192.168.5.0
BROADCAST=192.168.5.255
ONBOOT=yes
```

Здесь в /etc/sysconfig/network FORWARD_IPV4=false означает отключение маршрутизации через данный компьютер, это обычная установка для компьютера с одним сетевым интерфейсом, так как маршрутизация обычно происходит между разными интерфейсами. Обратите внимание, что в /etc/sysconfig/network-scripts имена файлов конфигурации интерфейсов соответствуют названиям файлов устройств – сетевых адаптеров. На каждый интерфейс приходится по файлу конфигурации. Обязательная фраза для автоматической инициализации интерфейса при загрузке системы – ONBOOT=yes.

Если в системе необходимы дополнительные статические маршруты (например, указание соединяться с локальной сетью филиала через конкретный сетевой интерфейс), то их следует вписать в файл /etc/sysconfig/static-routes. Формат строки в файле:

ethx **net** *ip-address netmask 255.netmask* **gw** *gateway-ipaddr*

В Debian Linux настройки сети хранятся в /etc/network, главными файлами являются файлы interfaces и options:

```
cat /etc/network/interfaces
# The loopback interface
auto lo
iface lo inet loopback
# The first network card - this entry was created during the
# Debian installation
# (network, broadcast and gateway are optional)
auto eth0
iface eth0 inet static
        address 192.168.110.4
        netmask 255.255.255.0
        network 192.168.110.0
        broadcast 192.168.110.255
        gateway 192.168.110.19
auto eth1
iface eth1 inet static
        address 192.168.150.1
        netmask 255.255.255.0
        network 192.168.150.0
        broadcast 192.168.150.255
```

```
cat /etc/network/options
ip_forward=yes
spoofprotect=yes
syncookies=no
```

Solaris

Параметры сети определяются в:

- /etc/defaultdomain – полное имя домена (например, **macro.su**)
- /etc/inet/hosts – соответствие имен и адресов
- /etc/hostname.le0 – краткое имя интерфейса le0. Остальные интерфейсы описаны соответствующими файлами. Должно совпадать с тем, что указано для адреса этого интерфейса в /etc/inet/hosts:

```
cat /etc/hostname.le0
king
cat /etc/inet/hosts | grep king
192.243.78.11     king.macro.su     king
```

- /etc/hosts – символьная ссылка на /etc/inet/hosts
- /etc/inet/nsswitch.conf – то же, что host.conf во FreeBSD, указан порядок опроса сервера имен и файла hosts

```
cat /etc/inet/nsswitch.conf
hosts:  files, DNS
```

- /etc/bootptab – адреса принт-сервера и принтера

- /etc/resolv.conf – настройки DNS, включая адреса DNS-серверов
- /etc/inet/netmasks – в каждой строчке пара: адрес сети, маска сети

```
cat /etc/netmasks
163.239     255.255.255.0
```

- /etc/defaultrouter – адрес шлюза
- /etc/nodename – имя компьютера

Настройки сети также могут выполняться из файлов /etc/init.d/rootusr и /etc/rcS.d/S30rootusr.sh.

Планирование адресации в сети

Если блок адресов для вашей локальной сети выделил провайдер и вам не интересно, как происходит регистрация IP-адресов для новой сети, не читайте этот раздел.

В последнее время назначение IP-адресов при планировании сети в офисах компаний происходит так: решается, что внутренняя (локальная) сеть организации будет иметь свою независимую от Интернета адресацию. Выход в Интернет будет осуществляться через proxy-сервер, подключенный к интернет-провайдеру, который сам сообщит компании внешний адрес proxy-сервера – корпоративного шлюза. Подробнее о proxy-серверах рассказано в разделе «proxy-серверы». Это решение имеет и плюсы, и минусы.

Плюсы

Подключение через proxy-сервер безопаснее, оно сильно снижает риск проникновения взломщика в корпоративную сеть извне.[1]

Не надо регистрировать свой блок IP-адресов, адреса в локальной сети можно придумать на свой вкус.

Количество компьютеров в локальной сети не ограничено количеством адресов в выделенном блоке, всегда можно переделать схему внутренней адресации, если адресов перестало хватать (например, была сеть класса C, решили перейти к сети класса B).

Не надо тратить деньги на поддержку вторичных серверов обратной зоны у провайдера, так как обратная зона внутренняя, регистрировать ее тоже нет необходимости.[2]

---

[1] Если только proxy-сервер надежен.

[2] Если слова «обратная зона» вы видите впервые и вам интересно, что это такое, обратитесь к главе 11 или к рекомендованной в ней литературе.

Минусы

Хороший proxy-сервер с функцией трансляции адресов требует внимательной настройки (если устанавливается на основе FreeBSD или Linux) или, плюс к тому, стоит немалых денег, если это маршрутизатор Cisco.[1]

Что бы вы ни применяли в качестве proxy-сервера, это устройство будет довольно сильно загружено, причем скорость передачи данных может снизиться по сравнению с сетью, в которой proxy-сервера нет.

Для настройки доступа извне к каким-нибудь корпоративным ресурсам (веб-сайту, например) придется дополнительно потрудиться. Не все proxy-серверы приспособлены для предоставления доступа к внутренней сети по инициативе снаружи. Не исключено, что надо будет официально зарегистрировать столько адресов, сколько у вас в локальной сети интерфейсов, к которым потребуется доступ снаружи.

При регистрации дополнительных адресов придется заплатить дополнительные деньги.

Все компьютеры, напрямую подключенные к Интернету, должны иметь «публичные», официально зарегистрированные IP-адреса. Те, что подключены к сети через proxy-сервер, могут иметь «приватные», локальные адреса, которые нигде регистрировать не надо.

Предположим, вы решили пойти по трудному пути регистрации собственного блока IP-адресов. Прежде всего надо сказать, что координирующие работу Интернета административные структуры не выделяют блоки адресов, меньшие сети класса С. Поэтому тем, кому потребуется пять, десять или даже двадцать адресов, прямая дорога к провайдеру. Он их уже заранее зарегистрировал и охотно продаст по сходной цене.

Если вы готовы обосновать необходимость заказа сети класса С, читайте дальше. Предполагается, что вы живете в России, по крайней мере, собираетесь зарегистрировать сеть класса С на ее территории. Если вы – житель ближнего или дальнего зарубежья и собираетесь регистрировать сеть вне России, обратитесь к описанию процедуры регистрации на сайте *www.internic.net* или напишите письмо своему интернет-провайдеру – пусть он вам объяснит местные правила. Еще один полезный источник информации – сайт международной организации, координирующей распределение IP-адресов в Интернете, – *www.iana.org* (IANA – Internet Assigned Numbers Authority).

В России национальный регистратор IP-адресов – РосНИИРОС (Российский НИИ развития общественных сетей, *www.ripn.net*).

Регистрация собственной IP-сети предполагает, что вы договоритесь со своим интернет-провайдером об анонсировании вашей сети. *Анонси-*

---

[1] Маршрутизаторы Cisco делает Cisco Systems Inc., зарегистрированная в San Jose, CA, USA. Самый крупный дилер этой компании в России – фирма АМТ.

рование – это процесс оповещения маршрутизаторов соседних сетей о вашей сети. То есть о том, что вот такая-то IP-сеть подключена именно к этому провайдеру, в эту точку сети. И пакеты для нее надо слать ему. Для последующей маршрутизации в вашу сеть.

Процедура регистрации предполагает написание заявки и получение блока адресов от регистратора или провайдера. После получения адресов необходимо назначить их компьютерам и проверить, доходят ли пакеты из Интернета к вам и от вас. Проверку можно осуществлять в соответствии с разделом «Мониторинг сети и борьба со сбоями сети» этой главы.

Теперь предположим, что администратор решил во внутренней сети организации выдать всем компьютерам локальные «приватные» адреса. Тогда ему надо следовать **RFC1918**, документу, определяющему порядок использования приватных адресов. Этот документ указывает диапазоны адресов, которые закреплены за «приватными» сетями. Пакеты с адресом отправителя или получателя из таких сетей не должны передаваться маршрутизаторами в Интернет. Иногда при сбое маршрутизатора пакеты все же начинают передаваться, и это приводит к перегрузке и сбою сети.

Рекомендуется не изобретать свою схему адресации локальной сети, а следовать правилам: назначать приватным сетям приватные адреса.

Определены следующие диапазоны адресов для приватных сетей:

- 10.0.0.0 – 10.255.255.255 (10/8). Обозначение 10/8 соответствует сети 10.0.0.0 с маской, в которой единицы стоят только в старших восьми битах, то есть 255.0.0.0;
- 172.16.0.0 – 172.31.255.255 (172.16/12), маска 255.240.0.0;
- 192.168.0.0 – 192.168.255.255 (192.168/16), маска 255.255.0.0;
- можно выбрать для своей сети любой непрерывный блок адресов внутри диапазона. Часто в небольших сетях ограничиваются сетью 192.168.1.0/24.

Адреса из этих диапазонов разрешено назначать только приватным интерфейсам, через которые данные в Интернет напрямую передаваться не будут. Слово «напрямую» означает, что передача пакетов идет без промежуточного устройства (или программного обеспечения), подставляющего другой адрес в поле адреса отправителя.

Маршрутизатор следует настроить так, чтобы исключить передачу во внешнюю сеть пакетов с адресами отправителя или получателя, входящими в диапазоны приватных адресов.

Компьютерам сети следует назначать уникальные приватные адреса из выбранного блока. Не надо координировать назначение адресов ни с кем, у кого есть свои сети с приватными адресами.

Планируя адресацию в сети, учтите, что сеть со временем может вырасти. Обычным делом является рост сети организации вдвое от запла-

нированного в течение года-двух. Если в сети уже есть две сотни компьютеров, планируйте адресацию так, чтобы адресов хватило минимум на 512 интерфейсов.

DHCP

Когда количество машин в сети превышает одну, дальновидный системный администратор начинает догадываться, что это только начало. И во избежание чрезмерных затрат усилий при настройке новых компьютеров готовит систему автоматической настройки.

Протокол DHCP (Dynamic Host Configuration Protocol) был придуман для того, чтобы свойства сетевого интерфейса настраивались автоматически, т. е. такие параметры, как IP-адрес, маска сети, адрес шлюза и многое другое, назначались интерфейсу при запуске системы. Протокол помогает, когда операционная система, настроенная как DHCP-клиент, при загрузке обращается к серверу DHCP в сети, чтобы получить от него все параметры базовой настройки интерфейса TCP/IP (адрес, маску, адрес шлюза, адрес сервера имен). В UNIX в роли сервера DHCP с успехом может выступить небольшая и простая в настройке программа `dhcpd`.

Эта программа очень удобна в сетях, где часто подключаются новые компьютеры и отключаются от сети старые, а также там, где компьютеры сильно разнесены географически. В первом случае системный администратор избавляется от необходимости вести учет занятых и освобожденных IP-адресов, во втором – бегать сломя голову между корпусами и этажами, чтобы назначить новые адреса. Кроме этого, если компьютеров в сети немало, она облегчает жизнь не только системному администратору (Боже мой, мне надо обойти еще сто шестнадцать компьютеров в четырнадцати комнатах!), но и неискушенному пользователю (опять ничего не работает, что только делают эти проклятые компьютерщики!).

С другой стороны, DHCP усложняет учет событий в сети, потому что в протоколы заносятся IP-адреса компьютеров. А при использовании DHCP адреса компьютерам выдаются динамически, и один и тот же адрес может сегодня принадлежать одному компьютеру, а завтра – другому. Кроме этого, если один из компьютеров стащат, системному администратору будет нелегко определить, какой именно: это следствие того, что IP-адрес не привязан к компьютеру. По той же причине будет труднее определить, какой именно компьютер вызывает сбой сети или генерирует сумасшедший трафик. Однако идентифицировать компьютер поможет программа `arp`, которая показывает соответствие IP-адреса компьютера и MAC-адреса сетевой карты.

Вообще говоря, системному администратору компании следует вести учет оборудования, работающего в сети. В частности знать MAC-адреса сетевых карт в компьютерах и иметь возможность определить ме-

стонахождение компьютера по сетевой карте. На практике редко в каких компаниях налажен такой учет. Прежде всего потому, что он требует внесения изменений в однажды составленный регистр компьютерного оборудования при ремонтах и заменах. Вы видели сотрудника техотдела, который бы знал, где лежит этот регистр? А имел право на внесение изменений в него? И при этом не забывал бы вносить эти изменения? Если на все вопросы вы ответили «да», то вы работаете в иностранной компании или в военном ведомстве и ваш доход заметно превышает 500 евро. Признайтесь, вы – майор? Или читаете эту книгу, сидя в своем офисе в Чикаго?

Применение DHCP делает сеть немного более уязвимой к взлому там, где к ней есть свободный доступ и любой желающий может подключить свой ноутбук к общей сети незаметно для администратора. В таких сетях неоходимы повышенные меры безопасности, и безоглядная раздача адресов по DHCP может привести к беде. Впрочем, в умелых руках и отвертка – оружие, так что если боитесь хакеров, просто избегайте ставить сетевое оборудование (концентраторы, коммутаторы, розетки) в местах, куда есть неконтролируемый доступ.

Для настройки UNIX-компьютера в качетсве DHCP-клиента следует внести имена интерфейсов, которые должны получать настройки по DHCP, в файл `/etc/dhclient.conf`. При установке многих версий UNIX эту работу за вас выполнит программа-установщик, опираясь на ваш ответ о необходимости настройки DHCP. Во FreeBSD можно вызвать программу-установщик командой **`/stand/sysinstall`** даже после установки системы и, в частности, настроить с ее помощью параметры сетевых интерфейсов.

Для настройки компьютера в качестве DHCP-сервера надо установить программу **`dhcpd`** и грамотно написать файл `/etc/dhcpd.conf`. Этот файл должен содержать настройки сервера, включающие перечень интерфейсов, через которые следует раздавать настройки IP, а также сами эти настройки и, прежде всего, диапазоны адресов, из которых следует назначать адреса DHCP-клиентам по их запросу. Вот пример минимального файла `/etc/dhcpd.conf`:

```
ddns-update-style none;

option domain-name "macro.ru";
option domain-name-servers ns.macro.ru;

default-lease-time 60000;
max-lease-time 720000;

# If this DHCP server is the official DHCP server for
# the local network, the authoritative directive should
# be uncommented.
authoritative;

log-facility local7;
subnet 195.70.215.64 netmask 255.255.255.224 {
```

```
    range 195.70.215.66 195.70.215.73;
    range 195.70.215.75 195.70.215.94;
    option domain-name-servers ns.macro.ru;
    option domain-name "macro.ru";
    option routers 195.70.215.65;
    option broadcast-address 195.70.215.95;
    default-lease-time 600;
    max-lease-time 7200;
  }
```

В конфигурации DHCP-сервера обязательно следует указать, какой стиль отношений с DNS-сервером ему следут избрать, и если не требуется выполнять динамическое обновление зоны, за которую отвечает DNS-сервер по мере выдачи адресов, то надо указать

```
ddns-update-style none;
```

Кроме IP-адресов DHCP-сервер может сообщать и другую полезную информацию, что определяют параметры options:

```
option domain-name "macro.ru";
option domain-name-servers ns.macro.ru;
```

Чтобы выданные настройки своевременно обновлялись, укажите соответствующие временные значения в секундах в полях `default-lease-time` и `max-lease-time`.

Разумеется, следует обязательно описать по крайней мере один диапазон адресов, из которого и будут назначаться адреса DHCP-клиентам, причем размера этого диапазона должно хватить на всех: не жадничайте, запланируйте диапазон с запасом. Если из диапазона надо сделать исключения для компьютеров, имеющих статически назначенные IP-адреса, обязательно укажите это в настройках DHCP-сервера.

Сетевые службы в UNIX

Многие программные решения предполагают работу в рамках архитектуры «клиент-сервер». Это значит, что одна программа, представляющая ту или иную службу, отвечает на внешние запросы (эта программа – сервер), а многие другие программы обращаются к ней с запросами (это клиенты). Клиент и сервер в общем случае могут меняться ролями, но, как правило, этого не происходит.

Практически все демоны, работающие с сетевыми приложениями в UNIX, – это серверы. Например, **`telnetd`**, **`ftpd`**, **`sendmail`**.

Для изучения работы в сети важно понимать, что сервером называется любой объект, способный обслужить внешний запрос, то есть послать какой-либо ответ.

Часто используемые термины «веб-сервер», «почтовый сервер», «ftp-сервер» надо понимать как обозначение программ, обслуживающих определенный вид запросов по сети.

Процесс inetd

Одни сетевые демоны запускаются при старте системы и ждут запросов по сети постоянно, другие мирно лежат на диске и запускаются по запросу, когда придет пакет, предназначенный для них. К первым относятся, например, почтовый сервер **sendmail** и разные веб-серверы (скажем, Apache). Ко вторым – **telnetd**, **ftpd** (демоны, обрабатывающие входящие соединения по протоколам telnet и ftp).

Возможность запускать демоны по запросу была добавлена в UNIX потому, что некоторые типы соединений (скажем, telnet) устанавливаются довольно редко. К веб-серверу могут приходить сотни и тысячи запросов в секунду. Одновременно работающих через telnet пользователей даже на европейских университетских серверах не может быть больше ста-ста пятидесяти человек. Обычно при среднероссийской нагрузке на UNIX-сервер на нем интерактивно работают через telnet два-три, редко десять человек одновременно. Поэтому нет никакого смысла держать в памяти постоянно загруженный **telnetd**. Такая же ситуация и с ftp: только специализированные ftp-серверы принимают сотни входящих соединений одновременно, обычный UNIX-компьютер не привлекает такого количества желающих скачать что-нибудь по ftp.

Не открывайте анонимный ftp-вход к себе на сервер без ограничения числа соединений, если у вас достаточно быстрый канал и вы не хотите оплачивать сотни мегабайт игр, музыки и порнографии, которые охотно «зальют» на сервер ушлые сетевые жители. Мест, через которые можно бесплатно обмениваться большими файлами произвольного содержания, в мире становится меньше, и каждое случайно появившееся новое привлекает внимание. Напротив, если вас не смутит счет на сотни или тысячи долларов в месяц от провайдера за входящий трафик, откройте ворота ftp-сервера пошире: вы сделаете доброе дело для общества, хотя благодарности от него все равно не дождетесь.

По запросу демоны запускает программа **inetd** (не путайте с identd, демоном протокола идентификации!), которая запускается при старте системы. Некоторые добрые программы-установщики (например, из Mandrake Linux) при выборе высокого уровня секретности при установке системы не настраивают систему для запуска **inetd** при старте. Помните: без **inetd** вы не сможете работать с UNIX-машиной через сеть, только с консоли или физического терминала!

Демон **inetd** при старте считывает файл конфигурации `/etc/inetd.conf` и ведет себя соответственно тому, что там написано.

```
cat /etc/inetd.conf
# Internet server configuration database
#
#       @(#)inetd.conf  5.4 (Berkeley) 6/30/90
#
ftp stream tcp nowait root /usr/local/libexec/in.ftpd \
in.ftpd -ai
telnet stream tcp nowait root /usr/libexec/telnetd telnetd
shell stream tcp  nowait root /usr/libexec/rshd rshd
login stream tcp  nowait root /usr/libexec/rlogind rlogind
#finger stream tcp nowait/3/10 nobody /usr/libexec/fingerd \
#fingerd -s
comsat dgram udp wait tty:tty /usr/libexec/comsat comsat
#ntalk dgram udp wait tty:tty /usr/libexec/ntalkd ntalkd
ntalk dgram udp  wait root    /usr/libexec/ntalkd ntalkd
#tftp dgram udp  wait nobody  /usr/libexec/tftpd tftpd \
#/tftpboot
#echo   stream  tcp     nowait  root    internal
#echo   dgram   udp     wait    root    internal
pop3 stream  tcp  nowait root /usr/local/sbin/ipop3d ipop3d
imap4 stream tcp  nowait root /usr/local/sbin/imapd  imapd
# Entry for an external ident server
auth stream tcp wait root /usr/local/sbin/identd identd -w \
-t120
```

Символ «\» (обратный слэш) в конце строки говорит о том, что на следующей строке не начинается новая запись, а продолжается текущая.

Все, что в /etc/inetd.conf начинается с символа «#», представляет собой комментарий. Каждая строка описывает одну сетевую службу – telnet, ftp и т. д.

Вначале идет имя службы. Соответствие имен служб и номеров используемых службой портов определяется в файле /etc/services, имя службы однозначно соответствует номеру порта. Затем – тип соединения и протокол, используемый для передачи данных (tcp, udp, icmp). Протоколы перечислены в /etc/protocols.

Поле wait/nowait относится только к службам, работающим через сокеты датаграмм, потому что все остальные должны содержать nowait в этом поле. Если такая служба (демон) устанавливает свое соединение с программой-клиентом и освобождает сокет для поступления следующих запросов, такой демон представляет собой многопоточный сервер и должен содержать nowait в этом поле. Те демоны (службы, серверы), которые обрабатывают входящие запросы через один сокет и один процесс, называются однопоточными и должны содержать в этом поле wait.

Демон **talkd** – отличный пример демонов последнего типа, а **tftpd** – исключение: он устанавливает псевдосоединения, ему соответствует wait. Фактически параметр wait/nowait определяет, должен ли

inetd ждать завершения запущенного демона перед обработкой нового запроса на тот же самый порт. Необязательный параметр max, отделенный от wait или nowait точкой, обозначает максимальное количество демонов такого типа, которые **inetd** имеет право запустить в течение 60 секунд. По умолчанию он равен 40.

Затем идет имя пользователя, от имени которого следует запустить демон. После указывается путь к демону, который надо вызвать, как только соответствующей службе придет пакет. Наконец, последняя строка – команда, вызывающая демон (здесь можно указать ключи).

Длина строки в **inetd.conf** ограничена 1022 символами.

Для того чтобы изменения в файле /etc/inetd.conf возымели ожидаемый эффект, надо или заставить **inetd** перечитать файл конфигурации, или перезапустить **inetd**. В Linux и FreeBSD можно потребовать от **inetd** перечитать файл конфигурации, послав ему сигнал HUP с номером 1:

```
ps ax | grep inetd
144      …     inetd
kill -HUP 144            # или kill -1 144
```

Страж tcp-wrapper

Бывает, что доступ к определенным службам (демонам, отвечающим на запросы по сети) надо ограничить. Например, разрешить им обрабатывать запросы только из собственного домена. Если демон запускается через **inetd**, это можно сделать с помощью tcp-wrapper. Это не название программы, а класс программ. Представителем этого класса является программа **tcpd**. Во FreeBSD tcp-wrapper встроен в **inetd**, хотя в старых версиях он существовал отдельно.

Tcp-wrapper перехватывает пакет, направленный компьютеру. Пакет анализируется на соответствие правилам, определенным в /etc/hosts.allow и /etc/hosts.deny. Если пакет можно пропустить, он передается демону, которому предназначался. Если правила требуют отбросить пакет, он не передается демону.

В новых версиях FreeBSD /etc/hosts.allow и /etc/hosts.deny объединены в файл /etc/hosts.allow. Его формат отличается от стандартного hosts.allow тем, что теперь в нем для каждого правила надо указывать тип – allow или deny.

Для включения функции tcp-wrapper в **inetd** во FreeBSD надо **inetd** запускать с ключами **wW**:[1]

[1] Это и так делается по умолчанию. Вам придется отыскать в /etc/rc.conf строку, где указаны ключи для inetd, чтобы убрать их и таким образом отключить встроенный tcp-wrapper.

```
inetd -wW
```

В тех системах, где **tcpd** – отдельная программа, для активации tcp-wrapper надо изменить вызовы демонов в /etc/inetd.conf:

```
ftp       stream  tcp     nowait  root    /usr/sbin/tcpd
                                    /usr/ local/sbin/proftpd
telnet  stream  tcp     nowait  root    /usr/sbin/tcpd
in.telnetd
```

Формат файла /etc/hosts.allow:

```
# hosts.allow  This file describes the names of the hosts
#                  which are allowed to use the local INET
#                 services, as decided
#              by the '/usr/sbin/tcpd' server.
#
telnet: :   .example.ru  : ALLOW
     telnet: : .co.spb.ru : ALLOW
     telnet: :192.168.5.10: ALLOW
```

Файл /etc/hosts.deny содержит список сетей и компьютеров, не имеющих доступа к некоторым или ко всем сетевым службам на вашем компьютере.

```
# hosts.deny    This file describes the names of the hosts
#          which are
#              *not* allowed to use the local INET services,
#         as decided by the '/usr/sbin/tcpd' server.
ALL:ALL EXCEPT localhost:DENY
```

Proxy-серверы

Темином «proxy-сервер» сейчас называют широкий спектр программ. Некоторые фирмы подлили масла в огонь, назвав свои программные продукты «proxy server» или «что_то_proxy». И эти продукты выполняют разные функции. У системных администраторов разбегаются глаза при их виде, как у советского туриста во французском супермаркете. Между тем, есть две концептуально разные функции, обычно присущие всем этим продуктам.

Первая – это кэширование http-запросов. Кэширующий сервер не умеет делать ничего, кроме кэширования запросов. Http-запрос от клиентского компьютера (из броузера Opera или Internet Explorer, например) вначале попадает к кэширующему серверу. Тот его переправляет веб-серверу, которому запрос предназначен, и сохраняет у себя ответ сервера. Если кто-то из пользователей обратится к кэширующему серверу с таким же запросом, он получит страницу быстрее – прямо из кэша. Это не только ускоряет работу конечных пользователей с сетью, но

и уменьшает входящий трафик организации. Следовательно, экономятся и время, и деньги.

Такая схема дает и побочный эффект: пользователи, которые обращаются к кэширующему серверу, могут иметь «приватные» IP-адреса, поскольку им не нужен прямой доступ к Интернету. Для сети такой организации достаточно одного официально зарегистрированного, «публичного» IP-адреса, и этот адрес им охотно предоставит провайдер. Этот адрес надо будет присвоить внешнему интерфейсу компьютера, на котором работает кэширующий сервер.

Таким образом, кэширующий сервер решает две задачи: собственно кэширование и подстановка своего адреса в качестве адреса отправителя запроса. Это позволяет даже компьютерам, не имеющим публичного адреса в Интернете, получать оттуда информацию через кэширующий сервер.

Вторая функция, которую может выполнять программа, называемая proxy-сервером, – это трансляция адреса отправителя (NAT – Network Address Translation). При трансляции адреса в каждом пакете, проходящем через NAT, адрес отправителя подменяется адресом компьютера, выполняющего трансляцию. Подробнее о NAT рассказано в следующем разделе – «NAT». Существуют программы, выполняющие трансляцию адреса только для пакетов определенных служб. Например, **telnet**, **ftp** и **http**. Часто такая программа совмещает функции NAT и кэширования http-запросов. Некоторые программы выполняют эти функции или одну из них после авторизации пользователя, что еще сильнее усложняет схему их работы. Нельзя же требовать авторизации на передачу каждого пакета, приходится запрашивать пароль при начале работы в сети и затем хранить его или пользоваться существующей в сети аутентификацией.

Из-за этого часто возникает терминологическая путаница и бывает неясно, что собеседник думает, когда говорит «proxy-сервер». Правильнее всегда уточнять, какая функция имеется в виду – кэширование запросов или подстановка адресов.

До сих пор мы говорили о proxy-серверах вообще, не упоминая операционную систему, в которой они работают. Многие proxy-серверы работают под управлением Windows. Однако наиболее популярный, особенно у провайдеров, http-кэш **squid** работает под UNIX. Он распространяется в исходных текстах и может быть установлен практически на любой диалект UNIX. Подробнее об установке, настройке и возможностях **squid** см. главу 21.

Однако на практике обычно требуется доступ не только к веб-страницам и ftp-серверам, но и к почте. В случае если сеть организации использует NAT, это не вызывает никаких дополнительных проблем. Даже если почтовый сервер организации находится во внешней сети,

пользователи будут обращаться к нему через NAT. Так же, как и к любому другому внешнему серверу.

Если же NAT не используется, адреса во внутренней сети назначены из блока приватных адресов, то эта задача решается установкой почтового сервера на ту же машину с «публичным» внешним интерфейсом, на которой работает и кэширующий сервер. Вы уже догадались, что маршрутизатором будет эта же машина, то есть она будет подключена и к внутренней сети организации, и к интернет-провайдеру.

NAT

Правила работы NAT описаны в RFC2663. Рассмотрим схему работы NAT на примере (рис. 10.4).

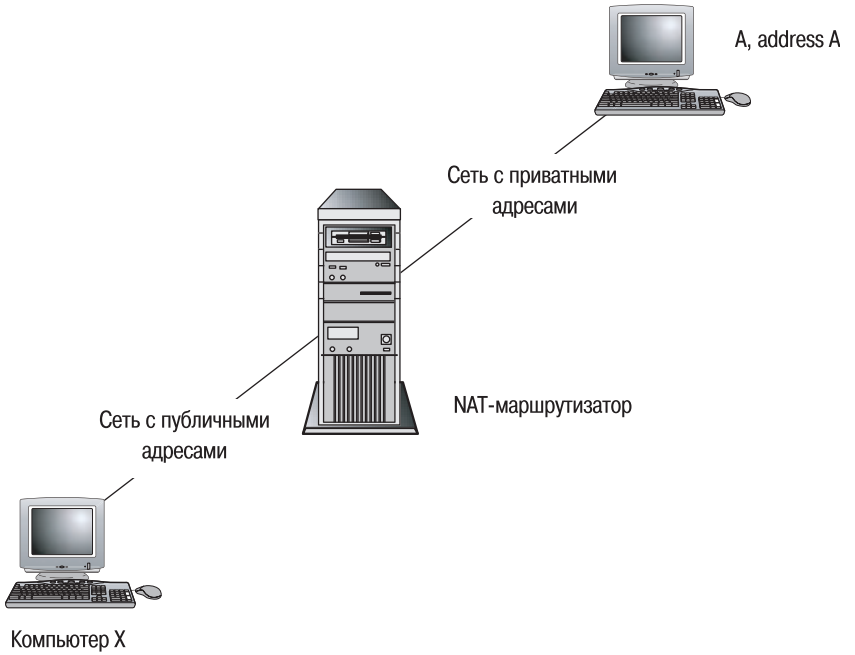


Рис. 10.4. *Схема работы NAT*

Пакеты с компьютера A отправляются компьютеру X. Сеть X – это сеть с публичными адресами, следовательно, мы можем смело отправить пакет с адресом получателя из этой сети. Однако в пути пакет попадает на NAT-маршрутизатор, стоящий на границе приватной сети. Он подставляет вместо приватного адреса отправителя свой внешний адрес из публичной сети и отправляет пакет компьютеру X. Есть несколько вариантов выбора подставляемого адреса.

NAT-маршрутизатор может заменять приватный IP-адрес на публичный из пула адресов. Пул должен быть заранее заказан, но адреса из этого пула не должны назначаться сетевым интерфейсам. Они будут предназначены только для того, чтобы соответствовать определенным адресам приватной сети. Соответствие может быть взаимно однозначным, если количество адресов пула совпадает с числом компьютеров приватной сети. В таком случае никакой «экономии» на IP-адресах не происходит, а смысл NAT сводится к повышению безопасности сети с приватными адресами. Соответствие не будет взаимно однозначным, если число компьютеров в приватной сети больше, чем число адресов в пуле. Тогда одновременно устанавливать соединения с внешней публичной сетью смогут ровно столько компьютеров с приватными адресами, сколько IP-адресов в пуле. На практике такая реализация NAT применяется не очень часто.

NAT-маршрутизатор может заменять приватный IP-адрес на свой собственный публичный IP-адрес, принадлежащий его внешнему интерфейсу. В этом случае в передаваемом пакете заменяется не только IP-адрес отправителя, но и порт отправителя. Диапазон портов однозначно ассоциируется с реальным компьютером-отправителем из приватной сети.

Вместо создания таблицы соответствий из адресов приватной сети и номеров портов со строками вида

```
<оригинальный IP, оригинальный порт>     <IP NAT-сервера,
подставленный порт>
```

может применяться таблица соответствий вида

```
<оригинальный IP, номер TCP-сессии>     <IP NAT-сервера,
подст. номер TCP-сессии>
```

Проще говоря, NAT-маршрутизатор, выше для краткости названный NAT-сервером, запоминает настоящего отправителя по номеру TCP-сессии, а не по номеру порта.

Трансляцию адресов может выполнять и специализированный маршрутизатор, и компьютер под управлением UNIX. Наиболее распространены трансляторы адресов на базе Linux и FreeBSD. Для поддержки трансляции адресов требуется, чтобы ядро имело встроенный механизм такой поддержки. Нам пока не известно, есть ли коммерческие версии UNIX с такой поддержкой. Нижеследующие примеры будут касаться только Linux и FreeBSD. В этих системах трансляция адресов реализована по-разному и управляется разными программами. Общим является только то, что трансляция адресов основывается на том же механизме, что и фильтр пакетов (firewall). Подробнее о последнем рассказано в разделе «Защита сетей с помощью фильтров пакетов».

Изучая NAT, важно представлять, каков путь пакета через любой маршрутизатор.

Во-первых, пакет по отношению к маршрутизатору может быть входящим (inbound), исходящим (outbound) и транзитным (being forwarded). Входящий пакет предназначен самому маршрутизатору, исходящий отправлен самим маршрутизатором, транзитный проходит через маршрутизатор, однако маршрутизатор не является ни отправителем, ни получателем транзитного пакета. Мы понимаем, конечно, что маршрутизатор в данном случае – это компьютер с двумя или более интерфейсами, который, возможно, выполняет еще какие-то функции кроме маршрутизации. Впрочем, излагаемая концепция применима и к выделенным маршрутизаторам.

Во-вторых, к пакету, пришедшему в маршрутизатор, сразу применяются правила маршрутизации. Он может быть направлен в другой сетевой интерфейс, уничтожен или направлен службе NAT для преобразования и дальнейшей отправки по назначению.

Маршрутизация пакета в соответствии с таблицей маршрутизации производится до того, как к нему будут применены правила фильтра пакетов (возможно, они его уничтожат) и трансляция адреса.

Вообще говоря, в Linux предусмотрена возможность выполнения операций с пакетом на следующих этапах: INPUT (пакет вошел в интерфейс), PREROUTING (пакет еще не прошел таблицы маршрутизации), POSTROUTING (пакет уже прошел таблицы маршрутизации, идет на выход), OUTPUT (пакет готовится к выходу с интерфейса).

Реализация NAT в Linux

Для организации фильтров пакетов и, следовательно, поддержки NAT в разных версиях ядра Linux использовали разные средства. Сначала это была утилита **ipfwadm** (ядра версий 2.0.x), затем – **ipchains** (ядра версий 2.2.x) и теперь – **iptables** (ядра версий 2.4.x). Кроме этого, к настоящему времени разработано ПО iproute2, которое реализует возможности source routing, маршрутизации пакетов в зависимости от их отправителя (т. е. пакеты, отличающиеся лишь адресом отправителя, могут маршрутизироваться по-разному). Все эти средства требуют поддержки в ядре, т. к. маршрутизация сетевых пакетов является функцией ядра в UNIX. Мы сразу рассмотрим **iptables**,[1] чтобы иметь дело с самым современным ПО. Технология реализации firewall и NAT похожа при использовании любых средств, различия есть лишь в деталях. Впрочем, это довольно значительные детали. Например, **ipfwadm** не поддерживал Quality of Service, имел менее гибкую и менее настраиваемую логику, позволял фильтровать пакеты только по очень грубым признакам (например, «все пакеты TCP»), тогда как **iptables**

[1] Программа **iptables** является частью проекта Netfilter (подробности можно узнать на сайте *http://www.netfilter.org*).

можно сказать что-нибудь вроде «все пакеты TCP, направленные на 53 порт с интерфейса ppp0».

Итак, для настройки NAT с помощью **iptables** понадобится:

- ядро Linux версии 2.4.x;[1]
- программа iptables версии 1.2.4 или свежее, доступная для скачивания с *http://www.netfilter.org/*.

Версии **iptables** до 1.2.4 имеют уязвимости в реализации трансляции протокола ftp, и работать с ними не рекомендуется. Со всеми современными дистрибутивами Linux поставляется свежая версия **iptables**.

В Linux функцию NAT называли «маскарадинг» (IP Masquerading), однако имели в виду ту же функцию, которая в других системах носит название NAT.

Сейчас под маскарадингом в Linux (в пакете Netfilter) подразумевается трансляция с подстановкой не определенного заранее адреса, тогда как NAT является более широким термином. Маскарадинг обычно применяется для трансляции адресов локальной сети в динамически выделяемый адрес внешнего интерфейса маршрутизатора.

Поставщики коммерческих систем часто не включают поддержку функции NAT в свои системы. Хочется верить, что в будущем они придут к этой поддержке.

Здесь мы говорим лишь о настройке NAT в Linux. При этом используются команды настройки фильтра пакетов, строго необходимые для настройки NAT. Можно (и часто нужно) сделать настройки первого более строгими, ограничив или запретив определенные виды трафика. Настройка фильтров пакетов обсуждается ниже в разделе «Защита сетей с помощью фильтров пакетов». И фильтр пакетов, и NAT реализуются посредством задания правил передачи пакетов. А именно посредством добавления правил в соответствующую таблицу в ядро с помощью одной из сервисных программ: **ipfwadm**, **ipchains**, **iptables**. Как вы помните, мы рассматриваем последний вариант.

Скрипт `/etc/rc.d/rc.local` должен автоматически запускать на выполнение `/etc/rc.d/rc.firewall` сразу после перезагрузки. Последний скрипт будет как загружать модули IPMASQ, так и включать их в работу. В этом же файле могут быть определены правила ограничения трафика с помощью фильтра пакетов.

---

[1] Подробнее о том, чем отличается **iptables** от старых реализаций фильтра пакетов и NAT, а также о том, когда не следует переходить с **ipchains** на **iptables** (бывают редкие случаи, в российской практике маловероятные), можно прочесть по адресу *http://www.tldp.org/HOWTO/IP-Masquerade-HOWTO/kernel-2.4.x-requirements.html*.

В общем случае должен существовать файл /etc/rc.d/rc.firewall (текст файла приводится чуть ниже). Если файла нет, то перед его созданием узнайте точно и запишите, где находится программа **iptables** (можно найти командами **locate**, **whereis** или **find**) и как называются ваш внутренний (локальный, приватный) и внешний (публичный) сетевые интерфейсы. Вероятнее всего, **iptables** лежит либо в /usr/local/sbin, либо в /sbin. Убедитесь, что **iptables** не может запустить никто, кроме root! В пакете **iptables** есть довольно много модулей, которые выполняют разные функции. Например, ограничение какого-либо специфического трафика (IRC, FTP) или протоколирование всех пакетов заданного типа. Последнее полезно для сбора статистики. Однако в нашем примере загружаются лишь самые основные модули, обеспечивающие работоспособность NAT.

```
cat /etc/rc.d/rc.firewall-2.4
#!/bin/sh
# rc.firewall-2.4          0.63 - версия конфигурации
FWVER=0.63
#
# Initial SIMPLE IP Masquerade test for 2.4.x kernels
# using IPTABLES.
#
# Log:
# 0.63 - Added support for the IRC IPTABLES module
# 0.62 - Fixed a typo on the MASQ enable line that used eth0
#        instead of $EXTIF
echo -e "\n\nLoading simple rc.firewall version $FWVER..\n"
# Путь к 'iptables'
IPTABLES=/usr/local/sbin/iptables
#  Имена интерфейсов - EXTERNAL и INTERNAL
#  For this example, "eth0" is external and "eth1"
#  is internal"
EXTIF="eth0"
INTIF="eth1"
echo "   External Interface:  $EXTIF"
echo "   Internal Interface:  $INTIF"
# Проверим, все ли необходимые модули загружены
echo "  - Verifying that all kernel modules are ok"
/sbin/depmod -a
#Load the main body of the IPTABLES module - "iptable"
#  - Loaded automatically when the "iptables" command is
#invoked
#  - Loaded manually to clean up kernel auto-loading timing
#issues
echo -en "ip_tables, "
/sbin/insmod ip_tables
#Load the general IPTABLES NAT code - "iptable_nat"
#  - Loaded automatically when MASQ functionality is turned
#on
```

```
#  - Loaded manually to clean up kernel auto-loading timing
#issues
echo -en "iptable_nat, "
/sbin/insmod iptable_nat
#Loads the FTP NAT functionality into the core IPTABLES code
# Required to support non-PASV FTP.
# Enabled by default -- insert a "#" on the next line to
#deactivate
echo -en "ip_nat_ftp, "
/sbin/insmod ip_nat_ftp

#          Redhat Users:  you may try changing the options in
#                           /etc/sysconfig/network from:
#                       FORWARD_IPV4=false
#                             to
#                       FORWARD_IPV4=true
echo "   enabling forwarding.."
echo "1" > /proc/sys/net/ipv4/ip_forward
# /proc - это отражение таблиц ядра на структуру виртуальной
# файловой системы, команды типа вышеприведенной вызывают
# изменение значений в таблицах загруженного в оперативную
# память ядра (не в файл ядра)
# Dynamic IP users:
#   If you get your IP address dynamically from SLIP, PPP, or
#   DHCP, enable this following option.  This enables
#   dynamic-address hacking which makes the life with Diald
#   and similar programs much easier.
echo "   enabling DynamicAddr.."
echo "1" > /proc/sys/net/ipv4/ip_dynaddr
#   Unless specified, the defaults for INPUT and OUTPUT is
#ACCEPT
#    The default for FORWARD is DROP
#
echo "   clearing any existing rules and setting default
policy.."
$IPTABLES -P INPUT ACCEPT
$IPTABLES -F INPUT
$IPTABLES -P OUTPUT ACCEPT
$IPTABLES -F OUTPUT
$IPTABLES -P FORWARD DROP
$IPTABLES -F FORWARD
$IPTABLES -t nat -F
echo "   FWD: Allow all connections OUT and only existing and
related ones IN"
$IPTABLES -A FORWARD -i $EXTIF -o $INTIF -m state --state
ESTABLISHED,RELATED -j ACCEPT
$IPTABLES -A FORWARD -i $INTIF -o $EXTIF -j ACCEPT
$IPTABLES -A FORWARD -j LOG
echo "   Enabling SNAT (MASQUERADE) functionality on $EXTIF"
$IPTABLES -t nat -A POSTROUTING -o $EXTIF -j MASQUERADE
echo -e "\nrc.firewall-2.4 v$FWVER done.\n"
```

В RedHat и основанных на RedHat системах файл `rc.firewall` можно выполнить двумя способами. Можно просто вызвать его в конце скрипта `/etc/rc.d/rc.local` либо поместить его вызов в файл в каталоге `/etc/rc.d/init.d/`. Первый способ плох тем, что `rc.local` выполняется самым последним из стартовых скриптов и во время загрузки фильтра пакетов (firewall) не будет работать. Это потенциально опасно. С другой стороны, второй способ лишен этого недостатка: при его использовании `rc.firewall` запустится сразу после старта подсистемы, поддерживающей работу с сетью. Минус второго способа в том, что он требует написания дополнительного скрипта для выполнения `rc.firewall`. Пример такого скрипта есть по адресу

http://www.tldp.org/HOWTO/IP-Masquerade-HOWTO/firewall-examples.html

Его несложно написать самому, надо лишь следовать синтаксису скриптов из каталога `init.d`, чтобы в них были варианты выполнения при вызове с аргументом `start` или `stop`.

Когда скрипт готов, его нужно поместить в надлежащую позицию в иерархии подкаталогов `/etc/rc.d/`. Это можно сделать командой

```
chkconfig:chkconfig - level=345 firewall-2.4 on
```

После этого скрипт будет выполняться при загрузке системы автоматически.

Распространенной ошибкой начинающих администрировать NAT является его включение командой:

```
iptables -t nat -A POSTROUTING -j MASQUERADE
```

Эта команда делает NAT общей политикой для любого интерфейса и позволит внешним пользователям входить в вашу сеть под видом вашего же маршрутизатора! Включайте NAT только для пакетов, исходящих вовне через внешний интерфейс, как в примере:

```
iptables -t nat -A POSTROUTING -o имя_внешнего_интерфейса -j
MASQUERADE
```

Реализация NAT во FreeBSD

Во FreeBSD для поддержки NAT требуется, чтобы:

- ядро было скомпилировано с включенными функциями `IPDIVERT` и `IPFIREWALL` в файле конфигурации ядра:

```
options IPDIVERT
options IPFIREWALL
```

- была включена поддержка передачи пакетов между интерфейсами **gateway_enable=YES** в файле /etc/rc.conf. В файле /etc/rc.conf надо также проверить наличие следующих установок:

```
firewall_enable="YES"
firewall_flags="/etc/rc.firewall"
natd_enable="YES"
natd_interface="ed0"
```

- правила для фильтра пакетов заворачивали бы все пакеты в сокет повторной обработки:[1]

```
natd -interface ed0
/sbin/ipfw -f flush
/sbin/ipfw add divert natd all from any to any via ed0
/sbin/ipfw add pass all from any to any
```

- был запущен **natd**, работающий с сокетами повторной обработки.

Подставьте вместо ed0 реальное имя внешнего интерфейса, который используется в вашем компьютере.

Строка **firewall_enable=YES** вызовет выполнение файла /etc/rc.firewall (в нем будут содержаться правила обработки пакетов firewall, в частности NAT).

Пакет, посланный согласно правилу в таблице правил **ipfw** в сокет повторной обработки, попадает на вход программы **natd**. После трансляции адреса этой программой пакет снова направляется в фильтр пакетов к правилу, номер которого следует за номером правила, отправившего пакет к **natd**. Именно к правилу со следующим номером, а не к следующему правилу, т. к. правила в фильтре могут иметь одинаковые номера.

Демон **natd** применяется для поддержки NAT только с интерфейсами, которые связаны с сетевыми адаптерами. Для поддержки NAT на ppp-интерфейсах следует предпочесть встроенный NAT программы pppd.

Интерфейсы, упомянутые в правилах **ipfw**, должны существовать и быть работоспособными до запуска **natd**. Если внешний интерфейс системы, например ppp0, создается при старте pppd или иного ПО, надо создавать и настраивать этот интерфейс до старта **natd**.

---

[1] Мы рискнули перевести таким образом термин «divert socket». Так называют сокет, в который для дополнительной обработки перенаправляются пакеты, уже принятые сетевым интерфейсом и прошедшие маршрутизацию. Дополнительная обработка может представлять собой подмену адреса отправителя или учет пакета программой мониторинга трафика. Насколько нам известно, в русскоязычной литературе по UNIX этот термин до сих пор не переводили. Приживется ли наш вариант, покажет время.

Вышеприведенный пример правил **ipfw** предполагает, что вы включаете только NAT. Если вы используете еще и другие правила **ipfw** для ограничения или запрета трафика, добавьте их к правилам перенаправления пакетов NAT.

Обратите внимание, как FreeBSD при выполнении NAT определяет, какой интерфейс у системы внешний, а какой – внутренний. Внешним считается интерфейс, указанный в правиле **ipfw add divert**, а все остальные считаются внутренними. Повторим для памяти: в нашем примере внешний адрес, который смотрит на провайдера (во внешнюю сеть), – `ed0`!

Никогда не выполняйте вышеприведенный пример, зайдя на UNIX-машину по сети! Если вы запустите **ipfw -f flush** не с консоли, а как-то иначе, то заблокируете передачу данных самому себе, как и всю остальную передачу данных тоже. Если вы все-таки запускаете эту команду удаленно, не с консоли, запускайте ее в фоновом режиме – разработчики FreeBSD обещают, что это поможет:

```
ipfw -f flush &
```

Это же относится и к настройке фильтра пакетов, описанной в следующем разделе.

Защита сетей с помощью фильтров пакетов

Если tcp-wrapper помогает ограничить входящий трафик компьютера, то есть предназначенный самому компьютеру, на котором запущен tcp-wrapper, то фильтры пакетов предназначены для ограничений транзитного трафика. Фильтры пакетов позволяют запретить передачу пакетов определенного типа с заданных или заданным компьютерам и сетям, а также вести подсчет числа проходящих пакетов, отвечающих заданным условиям. Идеология фильтра пакетов одинакова в любом клоне UNIX. Мы рассмотрим реализацию фильтра пакетов на примере FreeBSD. Для его реализации в Linux надо применять не **ipfw**, а **iptables** (или **ipchains** в ядре 2.2.x, **ipfwadm** в ядре 2.0.x).

Любой фильтр пакетов действует согласно таблице правил, которую ему задают. Таблица правил формируется программой **ipfw**, принимающей определенные команды (**add**, **delete** и т. п.). Каждая строка таблицы имеет номер. Если не задавать в команде номер строки, то строка будет вставлена перед последней строкой таблицы и пронумерована автоматически.

Номер строки играет важную роль, так как правила применяются в порядке возрастания номеров. Правила, задающие NAT, рекомендуется вносить в начало таблицы. В конец таблицы логично вписывать правило для тех пакетов, которые не подошли по условиям фильтрации ни к одной из строк таблицы. Поэтому таблица правил всегда за-

канчивается строкой с завершающим правилом номер 65 535, которое не может быть изменено. Оно применяется ко всем пакетам, не нашедшим соответствия раньше. Выполняемое действие (**deny** или **allow**) зависит от конфигурации ядра. Вообще же с пакетами могут быть совершены следующие операции:

- **reject** (устар.) – не пропускать пакет, послать отправителю сообщение «host unreachable». Сопоставление пакета с правилами на этом заканчивается;
- **deny** – не пропускать пакет. Сопоставление пакета с правилами на этом заканчивается;
- **drop** – синоним для **deny**;
- **allow** – пропустить пакет. Сопоставление пакета с правилами на этом заканчивается;
- **pass** – то же, что **allow**;
- **count** – сосчитать пакет (добавить единицу к счетчику пакетов, отвечающих этому правилу). Сопоставление пакета с правилами продолжается со следующего правила;
- **divert port** – направить пакет в сокет повторной обработки через указанный порт. Для **natd** по умолчанию – 8668. Сопоставление пакета с правилами на этом заканчивается (пакет может вернуться после обработки программой, читающей сокет повторной обработки).

Могут быть допустимы и некоторые другие операции, но это уже зависит от конкретной реализации фильтра пакетов. Подробности можно получить из man по **divert**, **ipfw**, **ipfirewall**.

Предположим, к маршрутизатору через интерфейсы ed0 и ed1 подключены две локальные сети, 193.114.38.32/27 и 193.114.38.64/27. Внешний интерфейс маршрутизатора rl0 имеет адрес 195.166.12.1.

Требуется настроить свободную маршрутизацию пакетов между локальными сетями, за исключением UDP-пакетов с номерами портов 137 и 139 (SMB-протокол, Microsoft sharing). Выход в Интернет должен быть открыт всем, за исключением компьютера с адресом 193.114.38.73. Передача пакетов с номерами TCP-портов получателя 23 и 25 во внутреннюю сеть должна быть запрещена. Смысл в том, чтобы во внутренней сети никто не устанавливал свои собственные telnet- и smtp-серверы (рис. 10.5).

При формировании таблицы для начала следует дать команду:

```
ipfw -f flush          # удалить все правила из таблицы
```

Затем добавить правила:

```
ipfw add 100 deny udp  from any to any 137 via ed0
ipfw add 110 deny udp from any to any 137 via ed1
ipfw add 120 deny udp from any to any 139 via ed0
ipfw add 130 deny udp from any to any 139 via ed1
```

```
ipfw add 140 deny all from 193.114.38.73 to any via rl0
ipfw add 150 deny tcp from any to any 23 via ed0
ipfw add 160 deny tcp from any to any 25 via ed0
ipfw add 170 deny tcp from any to any 23 via ed1
ipfw add 180 deny tcp from any to any 25 via ed1
ipfw add 200 allow all from any to any
```

Правила вступают в действие в момент добавления. Редко бывает необходимо модифицировать таблицу по частям. Обычно ее меняют при изменении политики работы в сети в целом и поэтому используют скрипты, начинающиеся с

```
ipfw -f flush
```

В нашем примере запреты базировались главным образом на интерфейсах, через которые проходят пакеты. Однако на практике могут понадобиться более избирательные правила, например с указанием протоколов, которые запрещены для использования определенными компьютерами. Например:

```
ipfw add 300 deny tcp from 193.114.38.16 to 193.114.38.1
```

Если интерфейс не указывается, то правило применяется к пакету независимо от того, через какой интерфейс он проходит.

Правила применяются к пакетам по порядку номеров. Как только найдено соответствие, выполняется действие, определенное правилом. Большинство действий прерывают проверку пакета, но некоторые позволяют применить к пакету оставшиеся правила. На практике из таких «непрерывающих» правил используют только count.

Каждый пакет, который соответствует какому-нибудь правилу, может быть посчитан. Счетчики пакетов можно вывести в файл протокола.

Функции фильтра пакетов может выполнять не только пакет **ipfw** (он состоит из подпрограмм ядра и программы **ipfw**), но и другие пакеты.

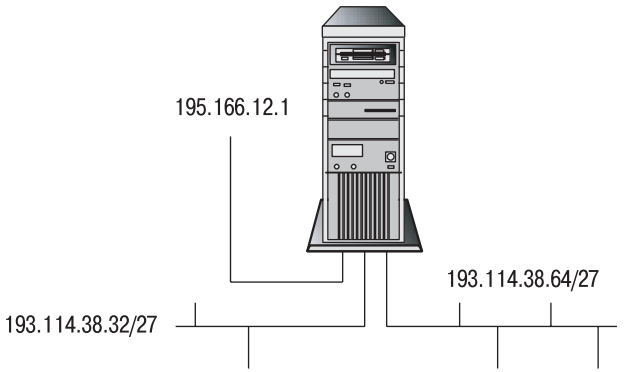


Рис. 10.5. *Схема работы NAT*

Например, широко применяется пакет **ipfilter**, который тоже имеет встраиваемую в ядро часть. Принципы работы **ipfilter** такие же, как у **ipfw**. Отличается синтаксис правил и некоторые возможности. Например, подсчет трафика удобнее выполнять в **ipfilter**.

Виртуальные частные сети

Виртуальные частные сети (Virtual Private Networks – VPN) стали стандартом де-факто для организации распределенных сетей организаций, а также для подключения сотрудников, работающих удаленно, к офисным сетям по шифрованному каналу связи. Виртуальные частные сети хороши тем, что сравнительно легко настраиваются и позволяют объединить в одну логическую сеть компьютеры, распределенные географически и подключенные к общей сети самыми различными способами – от модемного соединения до включения в высокоскоростную локальную сеть.

Есть несколько протоколов и соответствующих программ, которые позволяют объединять компьютеры в виртульные сети. Здесь мы рассмотрим только один из них – OpenVPN. Этот продукт существует в версиях для Windows и для UNIX, что позволяет организовать сервер удаленного доступа так, чтобы клиенты (удаленно работающие сотрудники, обычно использующие Windows-компьютеры) могли соединяться с сервером из любой точки Интернета так, чтобы работать с полным ощущением работы в локальной сети своей организации.

Многие протоколы VPN (и OpenVPN в их числе) предполагают установление шифрованных соединений типа точка-точка (туннелей) между клиентом и сервером. Клиент инициирует соединение, а сервер дожидается запроса от клиента для того, чтобы ответить и установить шифрованный канал связи. Сервер может выдать клиенту настроечную информацию подобно тому, как сервер **pppd** выдает такую информацию клиенту, подсоединившемуся через последовательный порт.

Шифрация соединения требуется для того, чтобы конфиденциальная информация (и просто частная – скажем, ничего конфиденциального в имени ваших разделяемых каталогов нет, но зачем их знать интернет-провайдерам?) не попадала в чужие руки. Шифрация в OpenVPN довольно надежна: при организации туннеля применяются современные алгоритмы шифрования с открытым ключом, длина ключа может быть установлена в 2048 бит и, кроме того, ключ динамически изменяется во время работы туннеля.

Туннель удобен для организации связи нескольких площадок, а также для соединения удаленно работающих сотрудников (так называемых road warriors – бойцов трудового фронта, работающих вне офиса – агентов, контролеров, службу безопасности и просто командированных) с сетью родной организации. При соединении между площадками можно, например, разделить адресное пространство IP-сети на две час-

ти, а созданный туннель и организовавшие его маршрутизаторы обеспечат передачу данных между этими двумя половинками сети так, как будто их разделяет не вся сеть Интернет (и полгорода или полпланеты географически), а всего лишь маршрутизатор. Перейдем к примеру.

Требуется настроить туннель и обеспечить маршрутизацию через него для общения компьютеров двух сетей: 192.168.10.0/24 и 192.168.15.0/24. Каждая из этих сетей представляет собой сеть одного офиса и имеет один выход во внешний мир – через свой маршрутизатор, подключенный к провайдеру. Адреса внешних интерфейсов этих сетей – 195.71.210.1 и 179.11.22.3 соответственно (рис. 10.6).

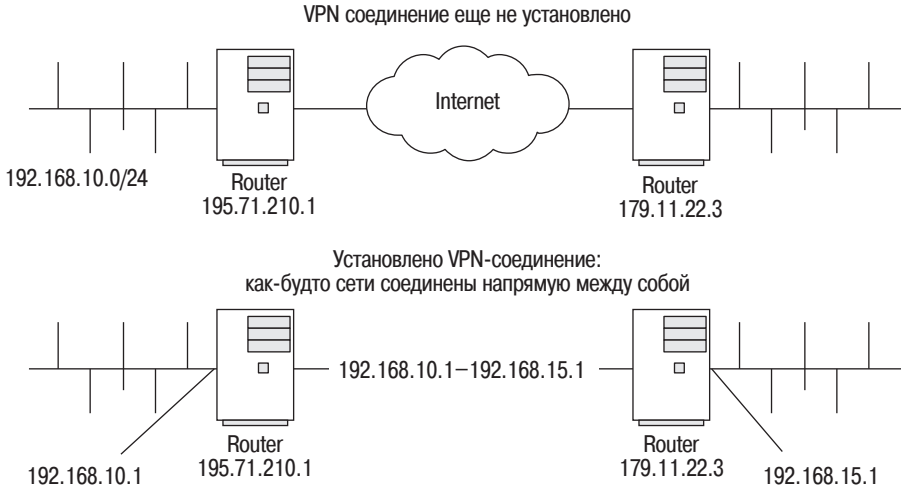


Рис. 10.6. *VPN-соединение*

Для настройки туннеля нам надо заранее решить, какой из маршрутизаторов будет сервером, а какой – клиентом.[1] Целесообразно клиентом делать тот маршрутизатор, который перезагружается чаще либо к которому труднее доступ администратора. Это, в частности, связано с тем, что сервер VPN может назначать параметры канала и передавать их для исполнения клиенту, а клиент не может. Поэтому выгоднее вносить изменения в настройку того компьютера, который управляет связью и работает более стабильно. Предположим, что из этих соображений сервером VPN мы делаем компьютер с внешним интерфейсом 195.71.210.1, а клиентом будет 179.11.22.3.

[1] Строго говоря, это не обязательно; программы пакета OpenVPN умеют создавать туннели в режиме «равноправного» функционирования, но в качестве пары TLS-сервера и TLS-клиента они обеспечивают большую безопасность канала связи за счет динамического обновления ключей. См. подробности на *http://openvpn.net/man.html*.

Теперь решим вопрос с шифрованием. Наиболее надежным на данный момент является шифрование с использованием динамически меняющихся ключей. Для того чтобы его организовать, нам потребуется создать мастер-ключ доверенной организации, а также приватный и публичный ключи клиента и сервера VPN. Мастер-ключ доверенной организации (Certification Authority – CA) нужен для того, чтобы подписать им ключи клиента и сервера. Неподписанный ключ нельзя использовать согласно общепринятому алгоритму шифрования с открытым ключом. Строго говоря, можно не создавать мастер-ключ, а подписать свои ключи в доверенной организации либо просто купить такие ключи у нее.

Какой организации доверять подписание сертификата, решает системный администратор или, если решение принимается на более высоком уровне, менеджер компании или правительственный чиновник. У каждой организации может быть свой список доверенных организаций. Более подробно о признанных во всем мире доверенных организациях можно узнать из документов, разбросанных по Интернету. Интересная попытка собрать ссылки на многие из них сделана Хуаном Авелланом (Juan A. Avellan) на странице *http://www.qmw.ac.uk/~tl6345/ca.htm*.

Предположим для простоты, что вы решили сделать доверенной саму вашу организацию, так как по смыслу алгоритма шифрования с открытым ключом подпись внешней доверенной организации требуется в том случае, когда стороны, использующие ключи, – это разные организации, которые могут сомневаться в подлинности ключей друг друга. Это не наш случай: мы сейчас сами создадим обе пары ключей и для клиента, и для сервера и тут же их сами и подпишем. Делается это тремя командами, для выполнения которых требуется установленный пакет OpenSSL (*http://www.openssl.org/*):

```
openssl req -nodes -new -x509 -keyout master.key \
     -out master.crt -days 3650
```

Мастер-ключ готов (мастер-ключ часто называют сертификатом CA). Теперь дело за ключами клиента:

```
openssl req -nodes -new -keyout client.key -out client.csr
openssl ca -out client.crt -in client.csr
```

и сервера:

```
openssl req -nodes -new -new -keyout server.key \
     -out server.csr -days 3650
openssl ca -out server.crt -in server.csr
```

Кроме того, для сервера следует создать файл с параметрами для динамического обмена ключами по протоколу Диффи-Хеллмана (Diffie-Hellman):

```
openssl dhparam -out dh2048.pem 2048
```

Получившиеся файлы `*.key` – это приватные ключи, а `*.crt` – публичные ключи. Теперь остается их разложить по каталогам, запретить доступ к приватным ключам всем, кроме пользователя root, даже для чтения, и на сервере и клиенте создать файлы конфигурации VPN-соединения. На сервере это будет:

```
dev tun0
tls-server
user nobody
group nogroup
route-up "route delete -net 192.168.15.0/24"
route-up "route add -net 192.168.15.0/24 tun0"
port 1194
ifconfig 192.168.10.1 192.168.15.1
verb 5
comp-lzo
comp-noadapt
ca /etc/ssl/certs/master.crt
cert /etc/ssl/certs/server.crt
key /etc/ssl/certs/server.key
dh /etc/openvpn/dh2048.pem
persist-key
persist-tun
ping 30
ping-restart 60
```

На стороне клиента:

```
dev tun0
tls-client
port 1194
remote 195.71.210.1
ca /etc/ssl/certs/master.crt
cert /etc/ssl/certs/client.crt
key /etc/ssl/certs/client.key
ifconfig 192.168.15.1 192.168.10.1
route-up "route delete -net 192.168.10.0/24"
route-up "route add -net 192.168.10.0/24 tun0"
comp-lzo
user nobody
group nogroup
verb 5
mute 10
persist-key
persist-tun
ping 30
ping-restart 60
```

При настройке **openvpn** следует обращать внимание, чтобы:

- файлы мастер-сертификата (мастер-ключа) были одинаковыми с обеих сторон;
- клиент знал фактический (внешний, видный из Интернета) IP-адрес сервера;
- параметры `ping` и `ping-restart` были одинаковыми с обеих сторон;
- ключ и интерфейс не терялись при перезагрузке сервиса (параметры `persist-key` и `persist-tun`).

Запуск программы **openvpn** обычно осуществляется из стартовых скриптов с указанием файла конфигурации, которым ей следует пользоваться, например, так:

```
/usr/local/sbin/openvpn --config /etc/openvpn/server-config \
        --writepid /var/run/openvpn.pid --daemon
```

Мониторинг сети и борьба со сбоями сети

Проверка соединений – ping

Проверка правильности работы сетевых соединений всегда начинается с команды **ping**. Программа **ping** посылает короткие (56 или 64 байта по умолчанию) пакеты по заданному адресу. Оттуда ей должно прийти в ответ «echo» – точно такой же пакет. Программа **ping** измеряет время, необходимое пакету на преодоление пути туда и обратно, и выдает его в конце работы. В UNIX программа ping завершает работу по сигналу прерывания (например, от нажатия <Ctrl>+<C>).

Если сетевой интерфейс компьютера, с которого производится проверка, сам настроен и работает нормально, надо проверять соединение с другими компьютерами. Проверка настроек локального интерфейса выполняется командой **ifconfig**.

Предположим, нам надо выяснить, может ли быть установлено соединение между компьютером gw.example.ru в нашей сети с компьютером book.spb.ru из удаленной сети.

Для проверки соединений в сети необходимо сначала убедиться, что на локальном компьютере подсистема TCP/IP работает корректно:

```
ifconfig                    # посмотреть все живые интерфейсы
eth0 Link encap:Ethernet  HWaddr 00:60:B0:3C:99:05
inet addr:192.168.5.18  Bcast:192.168.5.255
Mask:255.255.255.0
UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
RX packets:87406688 errors:0 dropped:0 overruns:0 frame:0
TX packets:104592793 errors:0 dropped:0 overruns:0 carrier:5
collisions:10141010 txqueuelen:100
Interrupt:9
```

```
lo Link encap:Local Loopback
inet addr:127.0.0.1  Mask:255.0.0.0
UP LOOPBACK RUNNING  MTU:16436  Metric:1
RX packets:13131479 errors:0 dropped:0 overruns:0 frame:0
TX packets:13131479 errors:0 dropped:0 overruns:0 carrier:0
collisions:0 txqueuelen:0
```

```
ping 127.0.0.1        # должно быть всегда, иначе - сбой
ping 192.168.5.18     # локальный интерфейс, должно работать
ping book.spb.ru      # удаленный интерфейс
```

Все в порядке, если все три команды привели к ожидаемому результату, примерно такому:

```
ping book.spb.ru
PING imc.example.ru (193.114.38.33): 56 data bytes
64 bytes from 193.114.38.33: icmp_seq=0 ttl=248 time=6.4 ms
64 bytes from 193.114.38.33: icmp_seq=1 ttl=248 time=24.3 ms
64 bytes from 193.114.38.33: icmp_seq=2 ttl=248 time=6.2 ms
64 bytes from 193.114.38.33: icmp_seq=3 ttl=248 time=7.4 ms
64 bytes from 193.114.38.33: icmp_seq=4 ttl=248 time=6.3 ms
64 bytes from 193.114.38.33: icmp_seq=5 ttl=248 time=6.0 ms
^C
--- book.spb.ru ping statistics ---
6 packets transmitted, 6 packets received, 0% packet loss
round-trip min/avg/max = 6.0/9.4/24.3 ms
```

Если программа **ping** выдает ошибки, значит, что-то не в порядке с соединением. Следует проверить:

- вставлен ли кабель в сетевую карту;
- подключен ли кабель куда-нибудь с другой стороны;
- включено ли устройство, к которому он подключен;
- не зависло ли оно;
- горит ли лампочка «соединение»[1] на **сетевой карте**;
- горит ли лампочка «соединение» там, куда подключен кабель с другой стороны;
- не перегнут ли кабель под слишком большим углом по дороге (на нем стоит стул, шкаф, по нему ребром протащили сейф, какой-то добрый парень вбил в него гвоздь);
- исправен ли кабель в принципе (выясняется его подключением к заведомо работающему в сети компьютеру вместо заведомо работоспособного кабеля; второй вариант – взять заведомо работоспособный кабель и заменить им подозрительный).

[1] На практике около лампочки написано «LINK», «LN» или что-то подобное. Для слова «соединение» там просто не хватит места.

Если кабельное хозяйство и ближайший коммутатор/концентратор в порядке, надо искать неисправность дальше. Дайте команду **ping** *адрес_шлюза*. Если шлюз отвечает (специалисты любят выражение «пингуется»), стало быть, неисправность дальше – по дороге к провайдеру или у провайдера.

Полезно знать адрес глядящего на вашу сеть интерфейса провайдера – можно «пропинговать» этот интерфейс. Если и он отвечает, можно смело попробовать сделать то же самое с каким-нибудь известным компьютером. Мне больше всего нравится *ftp.funet.fi* за рубежом и *www.rbc.ru* в России, но я надеюсь, что вы выберете другие серверы, иначе владельцы моих любимых серверов на меня обидятся: им же не нужен лишний паразитный трафик, верно?

Если известный компьютер не отвечает, похоже, проблемы у провайдера – можно ему звонить и спрашивать, в чем дело. Если еще работает электронная почта, напишите провайдеру письмо. Вы не поверите, но служба технической поддержки нежно любит тех, кто пишет письма. Ведь письма не отрывают ее сотрудников от срочного ремонта канала звонком и голосом секретарши, на них можно ответить чуть позже.

Если известный компьютер извне отвечает, а тот сервер, на который вы хотите попасть, – нет, то проблемы, похоже, именно там. Вряд ли вы способны их решить, если только нужный вам сайт не обслуживает ваш школьный друг.

Программу **ping** хорошо использовать для проверки наличия соединения. Для локализации места, где возникла проблема, следует предпочесть **traceroute** (см. ниже). К сожалению, ни одна из этих программ не идеальна. Они бесполезны, если где-то по дороге запрещена трансляция пакетов ICMP (ими пользуется **ping**) или передача пакетов с нестандартными номерами порта назначения (ими пользуется **traceroute**). Более того, пакеты разных типов (например, http и telnet или udp и tcp – уровень абстракции здесь не важен) могут передаваться по сети разными путями. При этом наличие соединения, установленного программой **ping**, не гарантирует корректной работы любого сетевого приложения.

Состояние сети – netstat

Для диагностики состояния сети и наблюдения за сетью удобно использовать команду **netstat**:

```
netstat -i
Name  Mtu   Network     Address           Ipkts     Ierrs Opkts   Oerrs Coll
rl0   1500  <Link#1>    00:e0:29:62:af:22 276054891 0     1060796 0     0
                                                      935
rl0   1500  macro.pu.ru 276054891         0         1060796 0       0
                                                      935
```

```
rl0   1500  fe80:1::2e0 fe80:1::2e0:29ff: 276054891 0      1060796 0     0
                                                          935
ed0   1500  <Link#2>    00:00:01:39:00:96 2745641   102    2748157 0     18910
ed0   1500  net1/27     imc               2745641   102    2748157 0     18910
ed0   1500  fe80:2::200 fe80:2::200:1ff:f 2745641   102    2748157 0     18910
ed1   1500  <Link#3>    00:20:4c:01:c5:06 2249232   0      2777424 7     486786
ed1   1500  net1/27     imc               2249232   0      2777424 7     486786
ed1   1500  fe80:3::220 fe80:3::220:4cff: 2249232   0      2777424 7     486786
ppp0* 1500  <Link#4>                      148823    62     152994  0     0
sl0*  552   <Link#5>                      0         0      0       0     0
lo0   16384 <Link#11>                     2461171   0      2461171 0     0
lo0   16384 fe80:b::1   fe80:b::1          2461171   0      2461171 0     0
lo0   16384 localhost   ::1               2461171   0      2461171 0     0
lo0   16384 127         localhost         2461171   0      2461171 0     0
lo0   16384 www.liternet.www.liternet.sp  2461171   0      2461171 0     0
```

- `Name` – имя интерфейса
- `Mtu` – максимальный размер пакета на интерфейсе
- `Network` – сеть
- `Address` – адрес в сети
- `Ipkts` – входящих пакетов всего
- `Ierrs` – из них входящих пакетов с ошибками
- `Opkts` – исходящих пакетов всего
- `Oerrs` – из них исходящих пакетов с ошибками
- `Coll` – коллизий на интерфейсе (имеет смысл для Ethernet)

Количество пакетов может быть довольно большим: считаются пакеты с момента последнего рестарта системы или последнего обнуления статистики. Число коллизий не всегда прямо связано с работой того компьютера, где вы запустили netstat, потому что коллизии могут возникать между другими компьютерами в сегменте сети. Число ошибок не должно превышать 5–10% от переданных пакетов. Иначе это будет говорить о явной неисправности сети. Например, плохо обжаты разъемы на кабеле, кабель имеет недопустимо маленький радиус закругления (перегиба) и т. п.

С помощью **`netstat`** можно просмотреть таблицу маршрутизации системы:

```
netstat -rn
Routing tables
Internet:
Destination       Gateway           Flags Refs Use      Netif  Expire
default           195.70.192.165    UGSc  55   99199900 rl0
```

```
127.0.0.1          127.0.0.1          UH     0    2104914  lo0
193.124.85.15      193.124.85.15      UH     0    0        lo0
193.114.38.6       193.114.38.6       UH     0    0        lo0
193.114.38.7       193.114.38.7       UH     0    0        lo0
193.114.38.32      ff:ff:ff:ff:ff:ff UHLWb 0    840      ed0   =>
193.114.38.32/27   link#2             UC     0    0        ed0   =>
193.114.38.33      0:0:1:39:0:96      UHLW   0    220      lo0
193.114.38.64      ff:ff:ff:ff:ff:ff UHLWb 0    855      ed1   =>
193.114.38.64/27   link#3             UC     0    0        ed1   =>
193.114.38.65      0:20:4c:1:c5:6     UHLW   2    349963   lo0
193.114.38.73      0:80:c8:6e:ca:f8   UHLW   1    1562     ed1   318
193.114.38.74      0:80:c8:6f:fd:85   UHLW   0    6105     ed1   955
195.70.192.164/30 link#1              UC     0    0        rl0   =>
195.70.192.165     0:80:c8:c9:34:63   UHLW   45   0        rl0   566
195.70.192.166     0:e0:29:62:af:22   UHLW   0    6120     lo0
```

Таблицы маршрутизации подробно обсуждаются в разделе «Маршрутизация», где в табл. 10.1 рассказано о смысле флагов на интерфейсах.

Функции **netstat** этим не ограничиваются. Это очень мощная программа для сбора информации о состоянии сети.

Изучение пути – traceroute

Можно проследить путь, по которому пакет данных идет от одного сетевого интерфейса к другому. Для этого предназначена программа **traceroute**. Она по умолчанию прослеживает путь, состоящий из тридцати или менее «транзитных участков (hops)». **Hop** (читается «хоп») – это один переход от одного сетевого интерфейса до другого. Если говорят, что до получателя один hop, это значит, что в дороге пакет не встретит ни одного маршрутизатора и выполнит только один переход – от отправителя к получателю. В некотором смысле их можно уподобить прыжкам пакета с одного маршрутизатора на другой.

Программа **traceroute** выдает примерно такую информацию:

```
traceroute www.funet.fi
traceroute to nic.funet.fi (193.166.3.1), 30 hops max, 40
byte packets
1  yard.pu.ru (195.70.192.165)  0.782 ms  2.591 ms  0.666 ms
2  efac.pu.ru (195.70.192.65)  1.241 ms  3.320 ms  1.187 ms
3  end.pu.ru (193.124.85.149)  1.490 ms  211.693 ms  1.829 ms
4  loof.pu.ru (193.124.85.153) 1.836 ms  124.784 ms  2.088 ms
5  chest.pu.ru (194.58.104.206) 2.724 ms  66.524 ms  2.887 ms
6  StPetersburg-LE-4.Relcom.ru (193.125.189.110) 3.178 ms
98.573 ms  3.010 ms
```

```
7  fa0-0-1.SPB-4.Relcom.net (193.125.15.134)  378.433 ms
67.703 ms  3.914 ms
8  stp-i1-feth0-0.telia.net (193.45.192.13)  18.308 ms
61.931 ms  18.140 ms
9  sap-b1-feth2-0-0.telia.net (213.248.67.157)  22.704 ms
127.302 ms  26.977 ms
10  ov-i10-serial2-0-1.telia.net (193.45.4.197)  18.660 ms
243.148 ms  18.272 ms
11  s-b1-geth8-2.telia.net (213.248.78.25)  16.977 ms
516.309 ms  18.700 ms
12  sto-b2-pos4-0.telia.net (194.17.1.89)  16.932 ms  329.523
ms  17.772 ms
13  sto-b3-pos1-0.telia.net (213.248.66.62)  17.891 ms
73.876 ms  18.061 ms
14  nordunet-01202-s-b3.c.telia.net (213.248.67.174)  17.165
ms  211.110 ms  18.258 ms
15  fi-gw.nordu.net (193.10.68.42)  16.980 ms  213.069 ms
20.362 ms
16  funet1-rtr.nordu.net (193.10.252.50)  18.047 ms  138.563
ms  19.330 ms
17  helsinki0-p000-csc0.funet.fi (193.166.255.154)  18.667 ms
128.621 ms  22.899 ms
18  csc3-g0000-helsinki0.funet.fi (193.166.187.174)  22.701
ms  291.379 ms  17.887 ms
19  nic.funet.fi (193.166.3.1)  21.104 ms  422.794 ms *
```

В Windows-системах есть своя **traceroute**, но там она называется tracert.

Работает **traceroute** так. Она посылает пакеты с адресом получателя, переданным ей в качестве аргумента. Однако в поле TTL (Time To Live – время жизни пакета, измеряется в пересылках от маршрутизатора к маршрутизатору, то есть в hop) для начала ставится число 1. В поле номера порта ставится какое-нибудь большое число, заведомо неиспользуемый порт. Естественно, ближайший же маршрутизатор присылает сообщение о том, что пакет умер по причине истечения TTL. Каждый маршрутизатор вычитает единицу из поля TTL каждого транзитного пакета.

Ответ маршрутизатора протоколируется, посылается пакет с TTL, равным двум. Ответ по истечении времени присылает следующий маршрутизатор по пути следования пакета.

Если пакет дойдет до получателя, то сообщение об ошибке будет иным: «порт не обслуживается» (ведь порт специально задан таким, чтобы он не обслуживался).

По умолчанию задается базовый порт 33 434, а фактический порт получателя в пакете задается равным базовому плюс количество переходов (hop) минус один. При помощи ключа **P** можно задать другое базовое число вместо 33 434.

Для более точного подсчета времени, которое занимает путь пакета до каждого промежуточного маршрутизатора, программа `traceroute` посылает три пакета на каждый маршрутизатор. Кроме этого, она помогает обнаружить тайм-ауты в сети (символ «*» выводится вместо времени ответа маршрутизатора, если ответ не пришел за пять секунд).

Некоторые маршрутизаторы разрешают пересылку только определенных транзитных пакетов через них, и пакеты `traceroute` часто оказываются «за бортом». Потому что пакеты с номерами портов, не относящихся к основным службам, просто не пропускаются. На всякий случай, по соображениям безопасности. Внутрь сетей, защищенных таким образом, пакетам `traceroute` не проникнуть, что для анализа соединений в сети неприятно. Старайтесь скрывать истинные пути пакетов в сети только там, где этого действительно требует безопасность.

Информация `traceroute` о маршруте из одной точки сети в другую может оказаться неточной, поскольку различные типы пакетов могут передаваться в сети разными путями. Настройки маршрутизаторов могут динамически изменяться при изменении ситуации в сети (отказы линий и оборудования, изменение нагрузки). Не следует ожидать, что `traceroute` всегда выдает абсолютно надежную информацию.

Чтобы сделать `traceroute` более достоверной, можно изменить длину посылаемых ею пакетов и другие их параметры ключами `traceroute`.

Наблюдение за соединениями в сети

Когда мы чувствуем, что передача данных по сети тормозится, нам надо выяснить, что стало тому причиной. В каждый конкретный момент любой канал связи может быть задействован полностью, наполовину или вообще свободен. Если мы пытаемся открыть новое соединение в полностью занятом канале, то его скорость будет намного меньше привычной нам. Например, в telnet буквы будут появляться вчетверо медленнее, чем вы будете их набирать.

Для того чтобы посмотреть, кто и почему так загрузил канал связи, используют программы `tcpdump` и `trafshow`.

Программа `tcpdump` показывает все пакеты, которые идут в сегменте сети, куда подключен сетевой интерфейс компьютера. Технически возможно заставить сетевой адаптер принимать и передавать подпрограммам более высокого уровня все пакеты, которые в принципе проходят «мимо» сетевого адаптера. В обычном режиме сетевой адаптер принимает только те пакеты, которые предназначены ему лично (это видно по MAC-адресу пакета). В режиме, который называется беспорядочным (promiscous mode), сетевой адаптер принимает все пакеты, которые попадают в данный сегмент сети. Разумеется, до своих адресатов они тоже доходят. Сетевой адаптер в таком режиме похож на бабушку у подъезда, которая внимательно слушает разговоры соседей. Понятно, что романтическую историю девушки из пятнадцатой квар-

тиры услышит и давняя подруга девушки – ее собеседница, и «бабушка» – наш сетевой адаптер.

Перевести сетевой адаптер в режим прослушивания может только root, никто больше доступа к настройкам драйверов не имеет. Перевод в этот режим произойдет автоматически, как только root запустит программу **tcpdump** или **trafshow**.

Встречаются, и часто, такие конфигурации сети, при которых сетевой интерфейс не в состоянии получить пакет, предназначенный другому сетевому интерфейсу. Например, если UNIX-машина включена в коммутатор, она никак не сможет принять пакеты, предназначенные компьютерам, подключенным в другие порты коммутатора. Из-за того, что коммутатор так устроен. Аналогично если интерфейс подключен к каналу «точка-точка» (например, последовательный канал связи с интернет-провайдером через модем), то никакие пакеты, кроме как уходящие к провайдеру и приходящие от него, на этом интерфейсе не увидеть.

Для работы **tcpdump** и **trafshow** нужна поддержка механизма bpf (Berkeley Packet Filter) в ядре. FreeBSD имеет встроенную поддержку, при компиляции нового ядра нужно включить

```
options bpfilter число_фильтров
```

в файл конфигурации ядра. Количество фильтров равно максимальному числу одновременно просматриваемых интерфейсов (четырех обычно хватает).

Программа **tcpdump** выдает на стандартный вывод заголовки пакетов, идущих в сети. Разными ключами можно менять формат вывода и даже выводить содержимое пакетов (вот раздолье для ловцов паролей!).

Рекомендуется вывод **tcpdump** перенаправлять в файл, ибо уследить за выводом на экран очень трудно. Каждый пакет обычно описывается минимум одной длинной строчкой сведений, а на практике по сети проходят десятки пакетов в секунду.

При анализе загрузки сети надо обращать внимание на однотипный трафик, особенно высокоскоростной.

Программа **trafshow** отличается от **tcpdump** тем, что работает в полноэкранном режиме. Она выдает сведения обо всех текущих соединениях, проходящих через заданный интерфейс. Интерфейс в командах **tcpdump** и **trafshow** задается одинаково ключом **-i**:

```
tcpdump -i ed0
```

Для каждого соединения указывается инициатор (отправитель данных) и их получатель (имя или, если невозможно его выяснить, просто IP-адрес), а также порты получателя и отправителя. Отображается также объем переданных в этом соединении данных и скорость их передачи.

Программа **trafshow** может выдавать данные в странном формате или не выдавать ничего, кроме заголовка. Это, скорее всего, говорит о неправильных установках терминала. Проверьте значение переменной среды окружения TERM.

Клавишами <Ctrl>+<F>, <Ctrl>+<B> в большинстве реализаций **trafshow** можно листать список соединений.

В начале нового тысячелетия в Италии в университете Пизы (University of Pisa) Люка Дери (Luca Deri) и Стефано Суин (Stefano Suin) разработали еще одну программу для наблюдения за сетью – **ntop**. Это программа, выполняющая учет трафика и протоколирование соединений. Она немного похожа на **trafshow**. Может работать в текстовом полноэкранном режиме, как **trafshow**, а может – в режиме веб-сервера, тогда для просмотра статистики понадобится броузер. Программа может вести протокол в текстовом файле, а может передавать информацию SQL-серверу. В дистрибутиве **ntop** есть клиентские части интерфейса с некоторыми СУБД. Подробности о **ntop** и ее исходные тексты доступны по адресу *www.ntop.org*.

Истории из жизни

Однажды я заметил чудовищное замедление связи с одним из наших серверов. Незадолго до этого к серверу была подключена высокоскоростная линия связи, и скорость резко выросла. И неожиданно резко упала. Через несколько минут после падения скорости сервер вообще перестал отвечать, и его пришлось перезагрузить. После перезагрузки он несколько секунд работал нормально, но уже через полминуты перестал отвечать снова.

После очередной перезагрузки я быстро запустил **trafshow** и заметил, что с этим сервером уже установлен десяток соединений по ftp, причем скорость передачи данных была довольно высокой: кто-то «заливал» файлы со скоростью до 200 Кбайт/с! Число этих соединений росло лавинообразно, и когда их стало около пятидесяти (на это ушло всего несколько секунд), сервер перестал справляться с нагрузкой и отказался отвечать.

Дело явно было в том, что медленный компьютер просто не успевал записывать на диск столько информации. Если бы я находился рядом с ним, то просто выдернул бы кабель компьютерной сети, а потом разобрался, что случилось. Но я соединялся с этим компьютером через сеть, с другого конца города, поэтому мне пришлось соревноваться в скорости с неизвестными файлокачателями.

Подождав, пока сервер перезагрузится, я быстро вошел в систему и отключил **ftpd**. Теперь сервер тратил время только на сообщения «connection refused» в ответ на попытки соединиться по ftp. Изучение ситуации показало, что на сервере после переустановки системы забыли выключить анонимный доступ по ftp. Этим не преминули воспользо-

ваться люди, которым высокоскоростной канал и достаточное пространство на диске пришлись очень кстати, чтобы организовать временное хранилище музыкальных файлов и краденых игрушек под Windows.

В другой раз замедление отклика сервера было вызвано иной причиной. Взгляд на сеть с помощью **trafshow** показал, что открыто несколько соединений с портами 2000 и 2001 на сервере. Разумеется, администратор и не думал открывать какие-то службы на этих портах. Программа **top** показала незнакомую программу, которая съедала 90% времени CPU и была запущена от имени nobody.

Поскольку ни один процесс в системе, кроме веб-сервера Apache, не запускался от имени nobody, стало ясно, что кто-то взломал веб-сервер и запустил вредоносную программу. Программа выполняла какую-то работу. Возможно, читала /etc/passwd и пыталась взломать пароли пользователей. А несколько процессов с удаленных систем общались с ней через порты 2000 и 2001. Естественно, для начала была убита чужая программа и поставлена свежая версия Apache, в которой уже исправили очередную уязвимость. Затем появилось время отследить с помощью **trafshow**, откуда приходят запросы на порты 2000 и 2001, и написать письма администраторам соответствующих сетей. Эти письма сообщат своим адресатам, что их сети заражены вирусом. Скоординированная атака из множества разных сетей часто говорит о том, что ранее компьютеры в этих сетях заразили и теперь заставляют их плясать под чужую дудку.

Анализ таблиц arp

Наряду с таблицей маршрутизации (через какие интерфейсы в какие IP-сети отправлять пакеты) есть таблица соответствий IP-адресов MAC-адресам сетевых адаптеров. Эта таблица используется при пересылках пакетов в локальной сети.

Посмотреть таблицу arp (Address Resolution Protocol) можно одноименной командой (ключ **-a** предназначен для вывода всей таблицы):

```
arp -a
imc.example.ru (193.114.38.33) at 0:0:1:39:0:96 permanent
[ethernet]
imc.example.ru (193.114.38.65) at 0:20:4c:1:c5:6 permanent
[ethernet]
herakl.example.ru (193.114.38.74) at 0:80:c8:6f:fd:85
[ethernet]
yara.ku.ru (193.10.212.165) at 0:80:c8:c9:34:63 [ethernet]
```

Таблица arp динамически изменяется и содержит обычно только те адреса, к которым приходилось обращаться в последние минуты. Данные этой таблицы иногда помогают обнаружить несанкционированные подключения к сети или неожиданные замены сетевых карт

в компьютерах. Разумеется, только если вы ведете учет MAC-адресов карт, а также учитываете, что любой MAC-адрес в большинство новых сетевых адаптеров легко записать программным путем.

В случаях непонятного поведения компьютеров в сети просмотр arp-таблицы – одно из первых действий.

Например, однажды в нашей сети мы столкнулись с удивительным феноменом. Ни один компьютер сети не мог соединиться с двумя нашими UNIX-серверами. Ни с одним, ни с другим. При этом вся остальная сеть работала и два этих компьютера отлично видели друг друга. Дикость ситуации была в том, что при попытке проследить путь пакета от локального компьютера к UNIX-серверу мы видели, как пакет уходит в наш маршрутизатор так, как если бы UNIX-серверы стояли вне локальной сети. Но они были внутри!

Анализ показал, что единственной проблемой был сбой таблицы маршрутизации в маршрутизаторе. Это устройство фирмы Cisco при определенной ситуации не справлялось с нагрузкой и портило свою таблицу маршрутизации. Маршрутизатор Cisco не был виноват: после того, как ему снизили нагрузку, он успешно справляется с работой уже не первый месяц подряд, как раньше.

Из-за сбоя таблицы маршрутизации маршрутизатор решил, что он сам владеет IP-адресами, которые в реальности принадлежали UNIX-серверам. И он записал эту информацию в свою таблицу arp. И более никуда! Но этого хватило: при попытке любой машины в сети узнать, какой MAC-адрес у интерфейса с таким IP-адресом, маршрутизатор реагировал быстрее серверов и утверждал, что это MAC-адрес его сетевого интерфейса. Естественно, получив адресованный ему пакет с чужим IP-адресом, он негодовал и считал пакет ошибочным. После перезагрузки маршрутизатора и уменьшения нагрузки на него проблема решилась.

Дополнительная литература

[6] Ravi Malhotra «IP Routing». – O'Reilly Ass., 2002 (Малхотра Р. «IP-маршрутизация»).

[7] Craig Hunt «TCP/IP Network Administration, 3rd Edition». – O'Reilly Ass., 2002.[1]

[1] Крейг Хант «TCP/IP. Сетевое администрирование», 3-е издание. – СПб: Символ-Плюс, 2004.

11

Основы системы имен доменов (DNS)

Вы помните наизусть номер мобильного телефона своего друга? А телефон случайного знакомого, что год назад помогал вам менять колесо на пустынном шоссе, когда у вас не было домкрата? А рабочий телефон подружки своей мамы? Скорее всего, телефоны вы храните в записной книжке. Если это записная книжка вашего мобильника, вы извлекаете их оттуда, находя имя нужного человека.

Было бы удивительно видеть человека, который помнит наизусть IP-адреса всех нужных ему сайтов. И даже такая феноменальная память была бы бесполезна для поиска сайта после того, как сайт поменяет IP-адрес. А это случается не так уж редко. Поэтому для IP-адресов был придуман аналог записной книжки.

Вначале, пока компьютеров было немного, а их общение часто ограничивалось локальной сетью, такой записной книжкой служил файл `/etc/hosts`. Он есть не только в UNIX-системах, но в других ОС он может находиться в каком-нибудь ином каталоге. Фактически файл `hosts` должен быть в любой системе, которая поддерживает работу с TCP/IP.

Системный администратор записывал в `/etc/hosts` имена всех компьютеров в локальной сети и соответствующие им адреса. Когда кто-то обращался к соседнему компьютеру в сети по имени, а не по адресу, система смотрела в `/etc/hosts` и брала оттуда адрес компьютера с указанным именем. Подпрограмма ядра, которая отвечала за получение адреса по имени, называлась resolver (to resolve – разрешать, то есть находить решение).

В сети, где для получения адреса по имени используется `/etc/hosts`, необходимо постоянно поддерживать идентичность файлов `/etc/hosts` на всех машинах и обновлять их, как только произойдут какие-

то изменения. Поэтому системный администратор должен был озаботиться регулярным распространением файла `hosts` по компьютерам сети.

Со временем число компьютеров выросло и часто возникала необходимость соединения с удаленным компьютером, расположенным за пределами локальной сети. Естественно, в такой ситуации решение с распространением файлов `hosts` не годилось. Разве вы захотите, чтобы файлы в вашей системе обновлялись по команде какого-то незнакомого вам коллеги? Да вы лучше вовсе не будете обращаться к его сети!

Было найдено весьма элегантное решение, которое позволило объединить многие сети в одну большую IP-сеть Интернет, но при этом сохранить возможность обращаться к компьютерам уважительно, по именам. Системные администраторы при этом по-прежнему отвечают только за имена в собственной сети. Все компьютеры объединили в разные **домены**. Домены, о которых сейчас пойдет речь, **не** имеют ничего общего с доменами в сетях Microsoft-систем (Microsoft network). Компьютеры, входящие в один домен Microsoft network, могут принадлежать к одному или нескольким доменам в смысле IP-сетей и наоборот. Имеется в виду тип сети, а не организационная принадлежность: сеть типа Microsoft network, как вы понимаете, может быть организована и на складе запчастей в Урюпинске.[1]

Домен – это совокупность компьютеров, объединенных общим именем – именем домена. Например, все компьютеры компании «Росбизнесконсалтинг» могут относиться к домену rbc.ru. Веб-сервер этой компании имеет адрес *www.rbc.ru*, а какой-нибудь компьютер, хранящий архивы, может называться archive.rbc.ru.

Система имен доменов не привязана именно к Интернету, вы можете создать свою собственную сеть, не имеющую соединения с Интернетом, и там организовать любые домены. Однако систему имен доменов имеет смысл рассмотреть в приложении к Интернету, ибо именно здесь она нашла самое широкое применение.

Система имен доменов (Domain Name System – DNS) организована иерархически и древовидно. Есть корневой домен, обозначаемый символом «.» (точка). В него входят домены верхнего уровня. Не правда ли, похоже на организацию файловой системы? Это еще одно дерево в мире UNIX, которое опять растет корнем вверх. И за что только мы так любим это положение?

Домены верхнего уровня делятся на географические и организационные. Географические – это .uk, .de, .ru и т. д. Они распределяются по странам или территориям. Каждая страна имеет свой собственный до-

[1] Пусть читатели из Урюпинска подтвердят это: наверняка в городе есть не один склад запчастей, а бухгалтерия и учет почти везде уже ведутся на компьютерах.

мен. Чтобы получить в нем поддомен (например, rbc.ru), надо обратиться к национальному регистратору этого государства. Организационные домены – это те, в которых регистрируют свои поддомены компании и организации разных направлений деятельности. Домен .com – для коммерческих организаций, .org – для некоммерческих, .edu – для учебных, .net – для связанных с сетью. Домен .mil был зарезервирован для североамериканских военных организаций. Эти наименования доменов верхнего уровня сохранялись с начала 1980-х годов до 2000 года. В ноябре 2000 года после широкого обсуждения Совет ICANN (подробнее об ICANN рассказано в главе 10) добавил в число доменов верхнего уровня семь новых имен: .aero (авиатранспорт), .biz (бизнес-организации), .coop (кооперативы), .info (для всех), .museum (музеи), .name (для частных лиц) и .pro (определенные профессии). По состоянию на лето 2002 года во всех новых доменах, кроме .pro, можно было регистрировать свои домены. Ниже перечислены сайты компаний, ответственных за регистрацию поддоменов в новых доменах. Там можно получить полную информацию о правилах регистрации. Многие интернет-провайдеры в России оказывают посреднические услуги при регистрации имен доменов в любых доменах верхнего уровня. Собственно регистрацию поддомена в любом домене верхнего уровня выполняют компании-регистраторы. Свежую информацию о доменах верхнего уровня и регистрации можно получить на одном из сайтов ICANN – *www.internic.net*.

- .aero – Societe Internationale de Telecommunications Aeronautiques SC (SITA), *http://www.nic.aero*
- .biz – NeuLevel, *http://www.nic.biz/*
- .info – Afilias Limited, *http://www.nic.info/*
- .name – Global Name Registry, *http://www.nic.name/*
- .museum – Museum Domain Management Association (MuseDoma), *http://www.nic.museum*
- .coop – National Cooperative Business Association (NCBA), *http://www.nic.coop/*

Зарегистрировать свой домен в организационном домене верхнего уровня может любое частное лицо или организация. Вообще говоря, если вы хотите зарегистрировать домен в домене .com, компания-регистратор могла бы и проверить, действительно ли вы – коммерческая организация. Но зачем? Ведь вы заплатите за регистрацию деньги (пусть и небольшие – обычно около 20 долларов в год, в зависимости от аппетита регистратора в данный момент). А если вы при этом нарушите авторские права, зарегистрировав домен с именем чужого товарного знака (типа cocacola.ru), то суд привлечет к ответственности вас, а не регистратора.

Каждый домен обязательно должен обслуживаться постоянно работающим сервером, хранящим список соответствий имен компьютеров

этого домена их IP-адресам. Такой сервер представляет собой программное обеспечение, отвечающее на внешние запросы об этом домене. Этот сервер называется сервером имен (NS, Name Server). Термин «DNS-сервер» означает то же самое.

Серверов имен, обслуживающих домен, должно быть не менее двух – это главный, или первичный (primary), и вторичный (secondary). Эти серверы имен могут сами относиться к домену, который они обслуживают, а могут и не относиться.

Самый распространенный в Интернете сервер имен – это BIND (Berkeley Internet Name Domain). Именно этот сервер поставляется вместе со всеми диалектами UNIX.

В DNS есть еще одно понятие, которое иногда путают с понятием домена, – **зона**. Зона – это часть домена, администрирование которой делегировано организации или подразделению, отличным от администратора домена в целом.

Домен – это поддерево дерева доменных имен. Зона – это часть дерева, за которую отвечает тот или иной NS. Например, в домене ru есть поддомен example. Администрация домена ru делегировала домен example некой организации. После этого в домене ru образовалась еще одна зона: example.ru.

Термин «зона ru» обозначает «домен ru» без всех его поддоменов, то есть без всех делегированных зон этого домена.

Недавно в России вышла книга «DNS и BIND», перевод четвертого издания лучшей в мире книги по системе имен доменов и программе bind.[1] Этот 700-страничный фолиант представляет собой отличную энциклопедию по DNS и BIND. Поэтому здесь мы будем говорить только об основных принципах работы DNS и основных правилах настройки bind. Этого вполне хватит, чтобы верно настроить корпоративный сервер. Детальная информация о DNS содержится в книге [4].

Как работает DNS

Представим себе, что некий пользователь в Австралии набрал в адресной строке броузера «*http://www.eu.spb.ru*» для того, чтобы познакомиться с правилами приема в Европейский университет.[2] Естественно, его локальный компьютер не представляет себе, какой IP-адрес может быть у компьютера с именем www.eu.spb.ru. Локальный компьютер любознательного австралийца посылает запрос серверу имен его интернет-провайдера (помните, при настройке TCP/IP вы указывали адрес DNS-сервера? Он тоже указывал). Сервер имен австралийского провай-

---

[1] Альбитц П., Ли К. «DNS и BIND», 4-е издание. – СПб: Символ-Плюс, 2002.

[2] *www.eu.spb.ru* – это адрес веб-сайта Европейского университета в Санкт-Петербурге.

дера, как и любой сервер имен в мире, выполняет следующую процедуру: сначала он смотрит в собственный кэш. DNS-запросы кэшируются, и если кто-нибудь недавно прислал ему запрос о том же компьютере *www.eu.spb.ru*, он достанет ответ из кэша. Предположим, в кэше нет ответа. Тогда сервер имен посмотрит в свою локальную базу записей о компьютерах своего домена: нет ли там адреса *www.eu.spb.ru*? Естественно, нет: австралийский интернет-провайдер отвечает за свой домен, а не за домен Европейского университета. Тогда сервер имен отправит запрос серверу имен корневого домена. Последующая процедура изображена на рис. 11.1.

Сервер корневого домена не знает, какой адрес у www.eu.spb.ru, зато он знает, какой сервер имен отвечает за домен .ru. Он сообщает этот адрес серверу имен интернет-провайдера. Тот видит, что ему сообщили не ответ на вопрос, а только адрес места, где его могут знать, и отправляет новый запрос серверу имен домена .ru. Тот тоже не знает нужного адреса, но сообщает адрес сервера имен домена .spb.ru. Последний является поддоменом домена .ru и поэтому его адрес известен серверу, отвечающему за домен .ru. Сервер имен провайдера получает адрес сервера имен домена spb.ru и отправляет туда запрос. Адрес *www.eu.spb.ru* там неизвестен, зато известен адрес сервера имен домена eu.spb.ru, этот адрес и отправляется серверу имен провайдера. Наконец, сервер интернет-провайдера отправляет запрос серверу имен eu.spb.ru. Поскольку www.eu.spb.ru – это компьютер из его домена, он тут же отправляет в ответ адрес этого компьютера. Сервер интернет-провайдера наконец получает долгожданный адрес и переправляет его компьютеру пользователя.

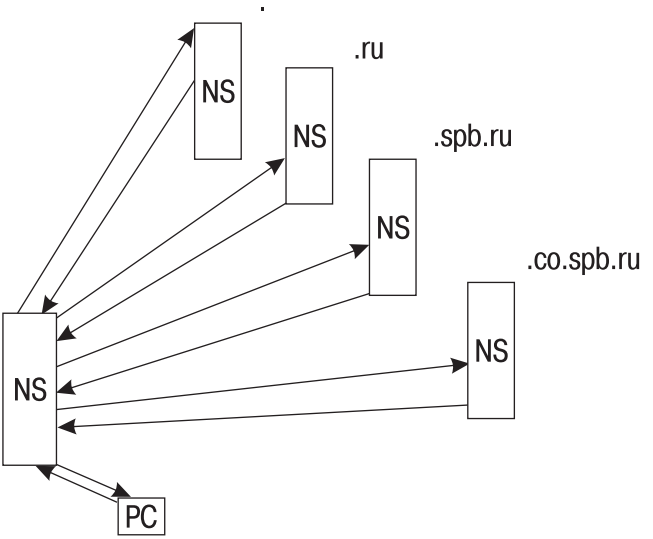


Рис. 11.1. *Процедура выполнения запроса сервером имен*

Спрашивается, как долго ждал пользователь? Как ни странно, недолго. Во-первых, дело в том, что DNS-запросы – это высокоприоритетный трафик, любые устройства в сети стараются передавать их побыстрее, вне очереди. Во-вторых, все вышеупомянутые серверы имен связаны достаточно быстрыми каналами связи. В-третьих, высока вероятность, что на каком-то этапе ответ на запрос окажется в кэше промежуточного сервера имен, и тогда процесс пройдет быстрее. Например, если в кэше сервера имен австралийского интернет-провайдера есть адрес сервера имен домена spb.ru, то запрос отправится прямо туда, минуя серверы корневого домена и географического домена России.

Из этого примера видно, что любой сервер имен знает адрес сервера корневого домена. Более того, этих серверов несколько, и в каждой стране могут быть свои серверы корневого домена. Они регулярно обмениваются информацией, несмотря на то, что список доменов верхнего уровня и ответственных за них серверов имен меняется очень редко. Этот список входит в дистрибутив любого сервера имен.

Полностью определенное доменное имя

В DNS принято называть полное имя компьютера (*имя.домен.*) полностью определенным доменным именем (FQDN – Fully Qualified Domain Name). Пример такого имени – *imc.example.ru*. Полностью определенное доменное имя – это аналог полного имени файла, только для DNS. Точка в конце полностью определенного доменного имени обязательна: она символизирует корневой домен. Компьютеры в локальной сети обычно называют сокращенными именами – imc, post и т. п. Если имя указывается без точки в конце при обращении к программам или функциям BIND (в настройках сервера имен, например), то к имени будет автоматически добавлен суффикс по умолчанию.

Суффикс по умолчанию – это второе поле инструкции `primary` или `secondary` в файле `named.boot` (для BIND версии 4) или второе поле оператора `zone` в файле `named.conf` (BIND 8 или 9). Фактически суффикс по умолчанию – это имя домена, автоматически добавляемое ко всем именам, не заканчивающимся точкой.

Настройка сервера имен

Существует несколько распространенных серверов имен. Самый известный – это bind из поставки любой UNIX-системы. Он отличается простой настройкой, высокой скоростью и надежностью работы. Однако несколько лет назад в bind была найдена ошибка, которая позволяла взломать систему. После этого bind был полностью переписан. Последней версией «старого» типа была версия 4.9.3. Затем перешли на версии 8.x, а вскоре и на 9.x. Последняя версия к осени 2002 года – 9.2.1.

При переходе на «новый» bind изменился не только код программы, но и формат файлов конфигурации. В дистрибутиве новых версий **bind** есть программа **named-bootconf**, которая конвертирует старые файлы конфигурации в новые без потери функциональности. Если вы пользуетесь старым bind, переходите скорее на новую версию: хакеры не дремлют!

Сердцем bind является программа **named** – собственно сервер имен. Также в пакет bind входят примеры файлов конфигурации и упомянутая утилита превращения старых файлов конфигурации в новые, программы для проверки правильности работы DNS-сервера, а также библиотека клиента DNS[1] для формирования DNS-запросов.

Конфигурационный файл named – /etc/named.conf. По правилам хорошего тона файлы, связанные с сервером имен, следует помещать в каталог /etc/namedb/ или /var/namedb/. Обычно их хранят в /etc/namedb/. Namedb означает «name database» – база данных имен доменов.

В /etc/named.conf указывается, где располагаются файлы с описаниями доменов, а также то, за какие домены отвечает данный сервер имен. Один сервер имен может хранить информацию о многих доменах. Кроме того, правила регистрации доменов обычно требуют, чтобы за каждый домен отвечали как минимум два сервера имен в разных IP-сетях класса С. Это сделано для надежности: если одна сеть потеряла связь с Интернетом, есть надежда, что вторая работает нормально. Рассмотрим структуру named.conf, а затем перейдем к файлу описания домена.

```
options {
// это комментарий, раз строка начинается с //
// сначала определяем, где лежат описания доменов, за которые
// отвечает наш сервер имен
        directory "/etc/namedb";

// Вместе с определением "forwarders" можно потребовать,
// чтобы наш сервер имен никогда сам не опрашивал серверы
// имен по обычному алгоритму, а лишь переправлял запрос
// нашему провайдеру.
// Для этого надо раскомментировать следующую строку
//      forward only;
//      forwarders {
//                193.124.15.219;
//      };
};
// Если мы создаем не сервер forward-only, а хотим,
```

[1] Клиент DNS – это подпрограмма, посылающая запросы серверу имен. Запрос может быть инициирован любой программой в системе – броузером, почтовым сервером и т. п. В UNIX клиент DNS – это подпрограмма **resolver(3)**.

```
// чтобы он работал по обычному алгоритму,
// надо не забыть вписать 127.0.0.1 в /etc/resolv.conf
// в качестве первого сервера имен.

// Дальше идет описание зон.
// Эти - корневая и обратная локальная - обязательны
zone "." {
        type hint;
        file "named.root";
};
zone "0.0.127.IN-ADDR.ARPA" {
        type master;
        file "127.0.0";
};

// Следующая - для сетей IPv6 - пока не актуальна,
// хотя и есть по умолчанию.

// Слово zone и имя зоны пишется всегда в одну строчку!
// в книге это невозможно изобразить, так как строка
// не может быть продолжена за пределами страницы книги,
// но может быть продолжена за пределами экрана.
zone "
0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.
0.I P6.INT" {
        type master;
        file "localhost.rev";
 };
// обратная зона нашей IP-сети
zone "38.114.193.in-addr.arpa" {
        type master;
        file "193.114.38";
};

// наш домен

zone "example.ru" {
        type master;
        file "example.ru";
};

// еще один наш домен

zone "carwash.spb.ru" {
        type master;
        file "carwash.spb.ru";
};

// третий домен на нашем ns

zone "glory.spb.ru" {
        type master;
        file "glory.spb.ru";
};
```

```
// а это - те зоны, для которых наш сервер - secondary name
server

zone "co.spb.ru" {
        type slave;
        file "sec/co.spb.ru";
        masters {
                213.193.6.19;
        };
};
zone "kir.spb.ru" {
        type slave;
        file "sec/kir.spb.ru";
        masters {
                213.193.6.19;
        };
};
```

Каталог, в котором будут храниться зоны, для которых наш сервер имен – вторичный, должен быть доступен **named** для записи. Для этого рекомендуется выполнить следующие команды (предполагается, что вторичные зоны лежат в `/etc/namedb/sec`):

```
mkdir /etc/namedb/sec
chown bind.bind /etc/namedb/sec
chmod 750 /etc/namedb/sec
```

В файле `named.root` содержится список серверов имен корневого домена. С ним ничего не надо делать, разве что обновить, если он старый. Однако со всеми новыми версиями bind приходит свежий список. На свежую версию bind все равно придется перейти.

Еще раз: старые версии bind содержали серьезные уязвимости, пора переходить на новые, пока не поздно!

Файл описания домена (зоны) представляет собой набор записей, называемых записями о ресурсах (RR – Resource Record). Ресурсы имеют класс и тип. В Интернете широко используется только класс IN, хотя DNS позволяет описывать ресурсы иных классов тоже. Тип ресурса (о нем часто говорят как о типе записи, например «запись типа MX») может быть одним из следующих (мы указываем только часто используемые типы):

- A – строка соответствия адреса имени (для поиска адреса по имени)
- PTR – строка соответствия имени адресу (для поиска имени по адресу)
- MX – указание почтового сервера для ресурса (домена или компьютера)

- CNAME – псевдоним
- HINFO – описание ресурса
- SOA (Start Of Authority) – важная служебная информация о домене в целом

Каждая запись имеет определенный формат. Для большинства записей он таков:

```
имя ресурса     класс     тип     информация
```

Например, для описания соответствия имени qwerty адресу 193.11.11.2 используется запись типа A:

```
qwerty     IN     A     193.11.11.2
```

Здесь IN – класс ресурса, A – тип записи. Записи типа SOA и MX имеют другой, более сложный формат. Запись SOA располагается на нескольких строках и описывает соответственно имя домена, имя авторитетного сервера имен домена, адрес лица, ответственного за домен, временные параметры, связанные с обновлением сведений о домене:

```
@       IN      SOA     imc.example.ru.
hostmaster.imc.example.ru. (
                                  2001021501      ; Serial
                                  21600   ; Refresh 3 hours
                                  7200    ; Retry 1 hour
                                  3600000 ; Expire 1000
hours
                                  86400 ) ; Minimum 24 hours
        IN      NS      imc.example.ru.
        IN      NS      ns.ussr.eu.net.
        IN      NS      ns.spb.su.
        IN      NS      hq.example.ru.
```

- Если доменное имя совпадает с суффиксом по умолчанию, его можно указывать сокращенно, заменяя символом @. Этот символ широко известен под названием «собака», хотя его настоящее имя – «коммерческое эт». Такая запись доменного имени обычна для записей типа SOA.
- imc.example.ru – это имя авторитетного сервера имен, то есть источника, информация которого не подвергается сомнению; этим источником является главный (primary) сервер имен домена.
- hostmaster.imc.example.ru. – это e-mail ответственного за домен, в отором символ @ заменен на точку. В письмах, адресованных ответственному, надо мысленно делать обратную замену и писать по адресу hostmaster@imc.example.ru.
- После открывающей скобки идет серийный номер версии описания домена. Его надо менять каждый раз, когда меняется хоть что-то в описании домена. Если его не менять, информация на вторичных

серверах имен не обновится. Принято серийный номер указывать в формате YYYYMMDDVV, где YYYY – год, MM – месяц, DD – число месяца, VV – версия описания домена за этот день. В течение суток, стало быть, не получится изменить описание домена больше 100 раз (версии нумеруются с нуля).

- Следующие строки – это период, через который вторичные серверы запрашивают главный сервер об обновлениях (refresh), время, через которое вторичный сервер должен предпринять новую попытку опроса главного сервера, если тот пока недоступен (retry), время, через которое вторичный сервер, не имеющий возможности соединиться с главным, должен считать, что информация устарела (expire). В конце указывается отрицательное TTL (Time To Live – время жизни). Оно относится ко всем отрицательным ответам серверов имен.

Записи MX описывают правила передачи почты. Каждая запись может указывать на один из почтовых серверов, принимающих почту для данного домена (или компьютера).

```
example.ru.        IN      MX      3       relay.example.ru.
example.ru.        IN      MX      10      relay.pt.ru.
host1.example.ru.  IN      MX      5       relay1.example.ru.
```

В этом примере почта для домена example.ru принимается одним из двух почтовых серверов – relay.example.ru или relay.pt.ru. При этом почта, направленная компьютеру host1.example.ru, направляется почтовому серверу relay1.example.ru.

Число, следующее за ключевым словом MX в записях о маршрутизации почты, называется preference. Оно показывает, насколько предпочтительно посылать почту на указанный почтовый сервер по сравнению с другими почтовыми серверами. Если для домена (компьютера) указано несколько записей MX с разными значениями preference, то почта направляется на первый доступный почтовый сервер с **минимальным** значением preference.

Отправка почты происходит так:

1. Почтовый сервер отправителя запрашивает информацию о том, куда надо отправить письмо для указанного в нем адресата.
2. Сервер имен выдает ему список записей MX.
3. Почтовый сервер выбирает запись с наименьшим значением preference и пытается отправить почту указанному в записи почтовому серверу.
4. Если это не удается, выбирается следующая запись с минимальным preference и делается попытка отправить почту через указанный в ней почтовый сервер.

Попытки повторяются до успеха. Если ни один из почтовых серверов не откликнулся, письмо помещается в очередь. Когда при обработке очереди снова придет пора отправить его, процедура повторится.

На любом компьютере, где запущен демон **named**, обязательно должен быть файл, описывающий обратную зону самого компьютера (то есть соответствие имени localhost адресу 127.0.0.1). Этот файл часто носит имя `localhost.rev`.

Файл `localhost.rev` генерируется скриптом **make-localhost** на основании имени компьютера и домена. Имеет смысл воспользоваться скриптом, если он есть.

```
cat localhost.rev
; This file is automatically edited by the `make-localhost'
script in
; the /etc/namedb directory.
;
@       IN      SOA     ns.example.ru.
hostmaster.example.ru.  (
                                1.2     ; Serial
                                3600    ; Refresh
                                300     ; Retry
                                3600000 ; Expire
                                3600 )  ; Minimum
        IN      NS      ns.example.ru.
1       IN      PTR     localhost.
```

Каждая IP-сеть может иметь так называемую обратную зону. Она представляет собой подмножество компьютеров, объединенных общим номером IP-сети и общим администрированием. Описание обратной зоны содержит записи о соответствии имен компьютеров их адресам, то есть с помощью описания обратной зоны можно найти имя, которое соответствует заданному IP-адресу. Такая зона называется обратной, так как направление поиска (имя по известному адресу) обратно «традиционному» поиску адресов в DNS.

Файл обратной зоны не является копией файла прямой зоны с «переставленными» столбцами, так как в домене могут быть компьютеры из разных IP-сетей, а одна IP-сеть может объединять компьютеры разных доменов. Однако в качестве простого примера можно рассмотреть ситуацию, когда все компьютеры домена входят в одну IP-сеть.

Если вам выделена IP-сеть, это не значит, что вам автоматически делегировали обратную зону, соответствующую этой IP-сети. Надо написать заявку о делегировании обратной зоны по принятой в Интернете форме. Проконсультируйтесь в службе технической поддержки провайдера, как это сделать.

Имя обратной зоны всегда пишется как адрес IP-сети в обратном порядке, а за ним – in-addr.arpa. Домен in-addr.arpa был создан специально для размещения в нем обратных зон IP-сетей. Такая запись была придумана потому, что система DNS построена иерархически: левая часть имени обозначает компьютер, правая – домен. В случае с IP-адресами все наоборот. В адресе имя сети находится слева, а адрес ком-

пьютера в сети – справа. Для целей DNS это не годилось, и адреса сетей «подогнали» под стандарт DNS.

Рассмотрим для примера файл обратной зоны сети 193.114.38.0:

```
; $ORIGIN 38.220.194.IN-ADDR.ARPA.
; 38.220.194.IN-ADDR.ARPA. - это имя обратной зоны
@       IN      SOA     imc.example.ru.
hostmaster.imc.example.ru.        (
                                  2001021501      ; Serial
                                  21600   ; Refresh 3 hours
                                  7200    ; Retry 1 hour
                                  3600000 ; Expire 1000 hours
                                  86400 ) ; Minimum 24 hours
        IN      NS      imc.example.ru.
        IN      NS      ns.ussr.eu.net.
        IN      NS      ns.spb.su.
        IN      NS      hq.example.ru.
;
;
33              IN      PTR     imc.example.ru.
65              IN      PTR     imc.example.ru.
66              IN      PTR     post.example.ru.
67              IN      PTR     magix.example.ru.
```

В этом файле есть запись SOA, описывающая обратную зону в целом, записи NS, говорящие о том, какие серверы имен хранят эту зону, и записи PTR, собственно описывающие соответствие имен компьютеров из домена example.ru адресам IP-сети.

Файл `example.ru` описывает домен example.ru:

```
; example.ru
;
$ORIGIN ru.
macro           IN      SOA     ns.example.ru.
hostmaster.imc.example.ru. (
                    2002072401      ; Serial
                    21600   ; Refresh 3 hour
                    7200    ; Retry 1 hour
                    3600000 ; Expire 100 hours
                    86400)  ; Minimum 1 hours
                IN      NS      ns.example.ru.
; Secondary servers
                IN      NS      ns.olly.ru.
                IN      NS      ns.pu.ru.
                IN      NS      hq.example.ru.
; Domain mail address
               IN      MX      3       relay.example.ru.
               IN      MX      10      relay.pu.ru.
$ORIGIN example.ru.
;
```

```
relay           IN      A       195.70.192.166
ns              IN      A       193.114.38.65
imc             IN      A       193.114.38.65
imc             IN      A       195.70.192.166
imc             IN      A       193.114.38.33
imc             MX      10      relay.example.ru.
imc             MX      15      relay.pt.ru.
proxy           IN      A       193.114.38.65
;
; Aliases
;
ftp             IN      CNAME   imc.example.ru.
www             IN      CNAME   imc.example.ru.
mail            IN      CNAME   imc.example.ru.
; Computers
;
post            IN      A       193.114.38.66
magix           IN      A       193.114.38.67
                IN      HINFO   AT-486, WGR
```

Обратите внимание на записи CNAME. Они создают удобные псевдонимы. Если роль, скажем, почтового сервера, будет передана от imc.example.ru другому компьютеру, будет достаточно всего лишь изменить имя в записи CNAME. Думать о том, какой у нового почтового сервера IP-адрес, не придется.

Один компьютер может иметь несколько IP-адресов. Так бывает, если у него несколько сетевых интерфейсов. Один IP-адрес может соответствовать нескольким именам компьютеров.

Если в файле описания домена встретятся имена доменов или компьютеров, не заканчивающиеся точкой, то они будут восприняты как неполностью определенные, к ним будет дописан суффикс по умолчанию. Так, если в описании зоны вместо

```
magix           IN      A       193.114.38.67
```

написать

```
magix.example.ru  IN      A       193.114.38.67
```

то name-сервер воспримет эту запись как

```
magix.example.ru.example.ru.     IN      A
193.114.38.67
```

Если же мы хотим указать полное имя, то должны написать так (обратите внимание на точку после ru):

```
magix.example.ru.               IN      A       193.114.38.67
```

В BIND 8.x введена директива **GENERATE**, позволяющая создавать имена для динамически раздаваемых IP-адресов вида dhcp43.company.com.

Вот как ее можно применить (эта строка пишется прямо в описании домена company.com):

```
$GENERATE 40-128 dhcp$ A 192.168.120.$
```

Получение информации от DNS

Для проверки правильности настройки NS можно использовать несколько утилит, опрашивающих серверы имен.

Прежде всего, это программа **nslookup**. Вот типичный сеанс проверки корректности свежесозданного домена:

```
nslookup
Default Server:  localhost
Address:  127.0.0.1

> set typ=soa
> kir.spb.ru.
Server:  localhost
Address:  127.0.0.1

kir.spb.ru
        origin = gate.co.spb.ru
        mail addr = milu.co.spb.ru
        serial = 2001012301
        refresh = 21600 (6H)
        retry   = 3600 (1H)
        expire  = 864000 (1w3d)
        minimum ttl = 3600 (1H)
kir.spb.ru      nameserver = gate.co.spb.ru
kir.spb.ru      nameserver = imc.example.ru
gate.co.spb.ru  internet address = 192.168.5.19
imc.example.ru  internet address = 193.114.38.33
imc.example.ru  internet address = 193.114.38.65
imc.example.ru  internet address = 195.70.192.166
```

А так можно вытащить всю информацию о конкретной зоне или компьютере:

```
> set typ=any
> www.rbc.ru
Server:  localhost
Address:  127.0.0.1

Non-authoritative answer:
www.rbc.ru      internet address = 62.118.249.16
www.rbc.ru      internet address = 62.118.249.66
www.rbc.ru      internet address = 194.186.36.138
www.rbc.ru      internet address = 194.186.36.175
www.rbc.ru      preference = 10, mail exchanger = mail.rbc.ru
```

```
www.rbc.ru      preference = 20, mail exchanger =
relay.rbc.ru
www.rbc.ru      preference = 30, mail exchanger =
relay2.rbc.ru

Authoritative answers can be found from:
rbc.ru  nameserver = ns2.rbc.ru
rbc.ru  nameserver = ns3.rbc.ru
mail.rbc.ru     internet address = 80.68.240.91
relay.rbc.ru    internet address = 80.68.240.103
relay2.rbc.ru   internet address = 194.186.36.142
ns2.rbc.ru      internet address = 62.118.249.100
ns3.rbc.ru      internet address = 194.186.36.186
```

Выход из программы **nslookup** осуществляется по нажатию клавиш <Ctrl>+<D>.

С программой **host** работать еще проще:

```
host -t any www.funet.fi
www.funet.fi is a nickname for nic.funet.fi
nic.funet.fi responsible person hks@funet.fi nic.funet.fi
nic.funet.fi descriptive text "FTP (and www) servers in the
Internet."
nic.funet.fi descriptive text "NIC.FUNET.fi is one of the
oldest and biggest Anonymous"
nic.funet.fi mail is handled (pri=20) by mailbackup.funet.fi
nic.funet.fi mail is handled (pri=10) by nic.funet.fi
nic.funet.fi has address 193.166.3.1
```

Программа **dig** – это более мощное средство получения информации от DNS. С ее помощью, например, можно посчитать, сколько зарегистрировано зон в домене ru.

Программа **whois** применяется для поиска имен в международных регистрах. Ее можно использовать для поиска хозяев и регистраторов определенного домена, контактных данных зарегистрировавшего домен лица и т. п. Альтернативой **whois** являются многочисленные веб-интерфейсы whois (*www.internic.net/whois.html*, *www.ripn.net/nic/whois/ index.html*).

Правовые аспекты регистрации доменов

Сейчас почти каждая компания желает иметь свой сайт в Интернете. Сайт стал элементом имиджа, знаком солидности. Это дало повод регистрировать доменные имена в целях наживы,[1] намеренно идя на возможные нарушения прав на торговую марку. Например, регистра-

[1] «С целью наживы» в данном случае означает «с целью поживиться за чужой счет, несправедливо, нечестно».

ция домена kommersant.com неким Стенли Тобиасоном (Stanley Tobiason) из США нарушила права издательского дома «Коммерсант», который 30 июля 2002 года выиграл дело, после чего регистратора домена .com обязали передать домен «Коммерсанту».

До 1999 года решение вопросов о правах организации на то или иное имя домена основывались на законах об авторском праве, торговых марках, патентном праве и прецендентах. Последние не всегда были в пользу истцов – компаний, владевших торговыми марками. В начале 90-х годов была распространена практика перепродажи имен доменов и некоторые ловкие персоны погрели руки на интернет-буме, заранее зарегистрировав домены с популярными именами.

В 1999 году ICANN был принят документ Uniform Domain-Name Dispute-Resolution Policy (UDRP). С этого момента все споры о доменных именах должны решать специальные организации, которым интернет-сообщество в лице ICANN делегировало такие полномочия. Решения этих организаций (фактически судов по вопросам, связанным с именами доменов – approved dispute-resolution service providers) обязательны для исполнения всеми международными регистраторами доменов верхнего уровня. Национальные регистраторы могут жить по своим правилам. Список организаций, ведающих решением спорных вопросов в организационных доменах, есть на сайте ICANN: *http://www.icann.org/dndr/udrp/approved-providers.htm*.

О вопросах, связанных с регистрацией доменов в домене .ru, можно прочесть на сайте РосНИИРОС *www.ripn.net*. РосНИИРОС (Российский НИИ развития общественных сетей) – это организация, которая осуществляет администрирование домена .ru и до недавних пор была монопольным регистратором доменов в нем. Обращайтесь в административные органы сети, если вас оскорбили, назвав именем вашей компании порносайт. Скорее всего, дело выиграете вы.

По договору между издательством «Символ-Плюс» и Интернет-магазином «Books.Ru-Книги России» единственный легальный способ получения данного файла с книгой ISBN 5-93286-084-7 «UNIX. Практическое пособие администратора. 2-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» (www.symbol.ru), где именно Вы получили данный файл.

12

Процедуры запуска и останова системы

Запуск системы происходит после включения питания компьютера или после рестарта по команде оператора. При включении компьютера требуется:

1. Проверить конфигурацию и состояние оборудования компьютера.
2. Выбрать операционную систему для запуска.
3. Считать в память ядро этой операционной системы.
4. Передать управление этому ядру.
5. Загрузить необходимые программы.

В компьютерах с архитектурой, отличной от i386, задачи 1–4 обычно выполняет так называемое firmware – программа, записанная в ПЗУ производителем компьютера. Например, так происходит запуск RISC-компьютеров компаний Hewlett Packard, Sun Microsystems, Silicon Graphics. Firmware – весьма интеллектуальное программное обеспечение, потому что оно не только умеет инициализировать оборудование компьютера, но и имеет представление о файловой системе UNIX. Это необходимо ему для считывания ядра ОС в память.

Пятую задачу независимо от архитектуры компьютера выполняет ядро ОС.

В нашей книге мы не будем останавливаться на том, как повлиять на работу firmware, потому что обычно это описывается в документации к компьютеру.

В компьютерах i386 дело обстоит сложнее. Во-первых, в отличие от вышеупомянутых компьютеров, компьютеры i386 изначально не предназначены для работы с какой-нибудь одной строго определенной ОС.

На них можно установить широкий спектр систем – от DOS и Windows до UNIX. Во-вторых, функции firmware здесь выполняют аж три разных программы (иногда и больше).

Начальную проверку конфигурации и состояния оборудования производит программа, записанная в ПЗУ компьютера (это часть BIOS, BIOS – Basic Input/Output System). Эта программа в конце своей работы считывает в память начальный загрузчик с жесткого диска из MBR (Master Boot Record).

Как правило, во время ее работы можно нажать какую-нибудь клавишу или комбинацию клавиш для получения доступа к меню настроек оборудования (BIOS Setup). Это обычно клавиша <Del> или клавиши <Ctrl>+<F1>, <Ctrl>+<F2>, <Shift>+<F1> или что-то подобное. Обычно фраза «Press DEL to enter Setup» отображается в нижней строке экрана во время работы этой программы инициализации оборудования.

Начальный загрузчик из MBR может включать в себя менеджер загрузки (boot-manager), а может и не включать. Менеджер загрузки – это программа, которая позволяет оператору выбрать, какую операционную систему из имеющихся на диске надо загрузить. Если менеджер загрузки присутствует, то он спросит, какую операционную систему загружать. В общем случае главная функция начального загрузчика из MBR состоит в том, чтобы определить, какой из разделов диска является активным, считать из первого сектора этого раздела начальный загрузчик операционной системы и передать ему управление. Начальный загрузчик из MBR занимает 512 байт в MBR. Поскольку он такой маленький, его функции весьма ограниченны. Никакие файловые системы он не понимает и сам никакой файл считать не может.

Загрузчик операционной системы из первого сектора раздела жесткого диска вызывается менеджером загрузки или начальным загрузчиком. Он уже имеет представление о формате файловой системы той ОС, которую он будет загружать. Он считывает в память ядро ОС и передает ему управление.

По умолчанию FreeBSD грузит ядро с именем `/kernel`, Solaris – `/vmunix`, а Linux – `/boot/bzImage` (то, что записано как загрузка по умолчанию в файле `/etc/lilo.conf`). Подробности, стало быть, доступны в **`man kernel`**, **`man vmunix`** и **`man lilo.conf`** соответственно.

Ядро выполняет инициализацию драйверов оборудования. На этом этапе могут появиться ошибки, связанные с неверной конфигурацией драйверов. Например, инициализация драйвера сетевой карты не пройдет, если драйвер настроен для работы с IRQ 7, а фактический IRQ сетевой карты иной.

При инициализации драйверов ядро выдает на консоль и в файл протокола (обычно `/var/log/messages`) информацию о том, какие драйверы инициализируются. Эта информация потом может пригодиться при поиске неисправности.

В конце загрузки ядра загружаются модули (Loadable Kernel Modules – LKM). После загрузки модулей стартует процесс, отвечающий за свопинг. Этот процесс – тоже часть ядра.

Затем ядро находит файл `init` и порождает процесс init. Поведение процесса init зависит от того, какая это система – System V или BSD. В разных диалектах UNIX могут быть реализованы разные версии init. Принципиально отличаются init для BSD и System V.

Собственно, все отличия загрузки систем BSD и System V сводятся к различным действиям init при загрузке. Эти различия и описаны ниже.

Несмотря на то что загружать ОС с дискеты или CD-ROM приходится редко, несколько слов об этом сказать надо.

UNIX обычно грузится с двух дискет. Исключение составляют специально минимизированные варианты загрузки (например, демонстрационная версия QNX или «аварийные» версии Linux – 1 дискета).

Загрузка с дискет может понадобиться при установке ОС без загрузки с компакт-диска. Или для того чтобы загрузить ядро и минимальную виртуальную файловую систему при сбое загрузки с жесткого диска. Последний вариант предполагает, что после исправления ошибок на жестком диске можно будет воспользоваться программами, записанными на нем.

В настоящее время принято устанавливать ОС с компакт-дисков. Загружать систему с компакт-диска приходится тогда же, когда и с дискет. Разница состоит в удобстве. И, конечно, в объеме того, что вы можете загрузить: на компакт-диске легко умещаются все нужные утилиты UNIX.

Режимы работы UNIX: runlevels

Принципиальное отличие BSD-систем от систем System V состоит в том, что в System V принята концепция уровней выполнения (runlevels). В SD-системах различаются лишь многопользовательский и однопользовательский режимы работы системы. В System V система может выполняться на одном из уровней. Уровень номер 1 соответствует однопользовательскому режиму, уровень 0 – остановке системы, а остальные – различным вариантам многопользовательского режима. Однако можно настроить систему так, чтобы вариантов однопользовательского режима было несколько.

При запуске в BSD-системе процесс init считывает и выполняет главный стартовый скрипт `/etc/rc`. Из этого скрипта запускается множество других скриптов `/etc/rc*`, каждый из которых настраивает или запускает что-то однотипное: сетевые демоны, локальные демоны, дополнительное оборудование типа карт PCMCIA и т. п. Во FreeBSD (начиная с версии 2.2.5) конфигурация системы сведена в один файл

/etc/rc.conf; этот файл считывается скриптом /etc/rc перед запуском других скриптов. В файле фактически определяются значения переменных, которые будут использованы в дальнейшем стартовыми скриптами.

При запуске в System V демон **init** считывает файл /etc/inittab и выполняет команды, определенные в нем для заданного уровня выполнения. Как правило, первой такой командой будет команда запуска стартового скрипта /etc/rc.d/rc с параметром – номером уровня выполнения.

Скрипт /etc/rc.d/rc в зависимости от переданного ему параметра будет запускать те или иные команды.

В RedHat-подобных системах принято размещать в каталогах /etc/rc.d/rc*N*.d (N – номер уровня выполнения) так называемые start- и kill-скрипты. Start-скрипты запускаются при переходе системы на этот уровень выполнения, а kill-скрипты – при переходе с этого уровня выполнения на другой. Имена start-скриптов начинаются с буквы S, а kill-скриптов – с буквы K. Имя такого скрипта выглядит примерно так: K88httpd. Из этого имени видно, что это kill-скрипт процесса httpd. Выполняются эти скрипты по порядку номеров, следующих в имени соответственно за буквой S или K.

Таким образом, процесс init выполняет стартовые скрипты, которые инициализируют основные параметры системы (имя компьютера, адреса сетевых интерфейсов) и запускают демоны.

Похожая идеология start- и kill-скриптов используется в некоторых других системах System V, но структура каталогов может быть иной.

Общие функции стартовых скриптов состоят в следующем:

1. Смонтировать корневую файловую систему только для чтения.
2. Проверить все файловые системы на предмет ошибок, исправить их при необходимости (запускается программа **fsck** для каждого раздела диска).
3. Смонтировать все файловые системы так, как указано в /etc/fstab (/etc/vfstab в Solaris и некоторых других ОС).
4. Запустить все нужные демоны.
5. Запустить процессы getty (в некоторых системах agetty) для каждого терминала.

Загрузка систем BSD

Конфигурация BSD UNIX записана в файлах /etc/rc*. Во FreeBSD начиная с версии 2.2.5 основной файл конфигурации – /etc/rc.conf. Рассмотрим его пример:

```
#!/bin/sh
#
```

```
# This is rc.conf - a file full of useful variables that you
# can set to change the default startup behavior of your
# system.
# All arguments must be in double or single quotes.
#
#$Id: rc.conf,v 1.1.2.9 1997/05/13 08:27:49 jkh Exp $

##############################################################
### Important initial Boot-time options ####################
##############################################################

swapfile="NO"          # Имя дополнительного swap-файла,
                       # если есть
pccard_enable="NO"     # YES, если требуется разрешить
                       # устройства PCCARD
pccard_mem="DEFAULT"   # Если pccard_enable=YES, здесь
                       # указывается адрес PCCARD
pccard_ifconfig="NO"   # Параметры настройки pccard ethernet
                       # или NO

##############################################################
###  Network configuration sub-section #####################
##############################################################
### Basic network options: ###
hostname="imc.example.ru"  # устанавливать всегда: имя
                           # компьютера
firewall="NO"        # Тип firewall или NO, если он не нужен
network_interfaces="rl0 lo0 ed0 ed1"  # Список интерфейсов
                                      # (lo0 is loopback)
# Все интерфейсы из списка должны получить свои параметры
ifconfig_lo0="inet 127.0.0.1"
ifconfig_lo0_alias0="193.124.85.15 netmask 255.255.255.255
193.124.85.193"
ifconfig_lo0_alias1="193.114.38.7 netmask 255.255.255.255
193.124.85.193"
# У интерфейса могут быть дополнительные адреса-псевдонимы
ifconfig_ed0="inet 193.114.38.33 netmask 255.255.255.224"
ifconfig_ed1="inet 193.114.38.65 netmask 255.255.255.224"
# backbone
ifconfig_rl0="inet 195.70.192.166 netmask 255.255.255.252
media 100BaseTX mediaopt full-duplex"

### Настройки сетевых демонов и NFS ###
syslogd_enable="YES"   # Запускать syslogd (или NO).
syslogd_flags=""   # Какие флаги передать syslogd при запуске
inetd_flags="-wW"      # Флаги для inetd
named_enable="YES"     # Запускать DNS server (или NO).
named_flags="-b /etc/namedb/boot"     # Флаги для named
nfs_client_enable="NO" # Этот компьютер - клиент NFS (или NO)
nfs_server_enable="NO" # Этот компьютер - сервер NFS (или NO)
# Нижеследующих параметров NFS может не быть
# В большинстве случаев их смысл интуитивно ясен
```

```
weak_mountd_authentication="NO"  # Running PCNFSD / other
                                 # non-root nfsd (or NO)
nfs_reserved_port_only="NO"    # Provide NFS only on secure
                               # port (or NO)
rpc_lockd_enable="NO"      # Run NFS rpc.lockd (*broken!*)
                           # if nfs_server.
rpc_statd_enable="YES"     # Run NFS rpc.statd if nfs_server
                           # (or NO)
portmap_enable="YES"       # Run the portmapper service
                           # (or NO)
### Настройки демонов времени: ###
timed_enabled="NO"         # Run the time daemon (or NO)
timed_flags=""             # Flags to timed (if enabled)
ntpdate_enable="NO"   # Run the ntpdate to sync time (or NO)
ntpdate_flags=""      # Flags to ntpdate (if enabled)
xntpd_enable="NO"   # Run xntpd Network Time Protocol (or NO)
xntpd_flags=""        # Flags to xntpd (if enabled)
tickadj_enable="NO"   # Run tickadj (or NO)
tickadj_flags="-Aq"   # Flags to tickadj (if enabled)
### Настройки маршрутизации: ###
defaultrouter="195.70.192.165"     # Адрес шлюза
                                   # (NO, если его нет)
static_routes=""           # Статические маршруты (или пусто)
gateway_enable="YES"       # YES, если этот компьютер -
                           # маршрутизатор"
router_enable="NO"         # YES, если надо запускать
                           # демон маршрутизации
# Обычно демон маршрутизации (routed или gated)
# запускать не надо.
# Статическую маршрутизацию выполняет ядро
router="routed"         # Как вызывать демон маршрутизации
router_flags="-q"       # Флаги для routed
# Настройки multicast и маршрутизации IPX
mrouted_enable="NO"          # Do multicast routing
                             # (see /etc/ mrouted.conf)
ipxgateway_enable="NO"       # Set to YES to enable
                             # IPX routing
ipxrouted_enable="NO"        # Set to YES to run the IPX
                             # routing daemon
ipxrouted_flags=""  # Flags for IPX routing daemon
arpproxy_all=""     # эквивалент устаревшей настройки
                    # ядра ARP_PROXY_ALL

##############################################################
###  System console options ##################################
##############################################################

# Настройки консоли, включая русификацию
keymap="ru.koi8-r" # keymap in /usr/share/syscons/keymaps/*
                   # (or NO)
keyrate="normal"   # keyboard rate to: slow, normal, fast
                   # (or NO)
```

```
keybell="normal"   # bell to duration.pitch or normal
                   # or visual (or NO)
keychange="NO"     # function keys default values (or NO)
cursor="NO"        # cursor type {normal|blink|destructive}
                   # (or NO)
scrnmap="NO"            # screen map in /usr/share/syscons/
                        # scrnmaps/* (or NO)
font8x16="koi8-r-8x16"  # font 8x16 from /usr/share/syscons/
                        # fonts/* (or NO)
font8x14="koi8-r-8x14"  # font 8x14 from /usr/share/syscons/
                        # fonts/* (or NO)
font8x8="koi8-r-8x8"    # font 8x8 from /usr/share/syscons/
                        # fonts/* (or NO)
blanktime="600"         # blank time (in seconds) or "NO"
                        # to turn it off
saver="snake"           # screen saver desired: blank/green/
                        # snake/star/NO
mousedtype="NO"         # See man page for rc.conf(8) for
                        # available settings
#mousedport="/dev/cuaa0"
mousedflags=""          # Any additional flags to moused
##############################################################
###  Miscellaneous administrative options ##################
##############################################################
lpd_enable="NO"             # Запускать lpd
sendmail_enable="YES"       # Запускать sendmail
sendmail_flags="-bd -q30m"  # Флаги sendmail, -bd нужно,
                            # чтобы он был демоном
savecore_enable="NO"        # Save kernel crashdumps
                            # for debugging (or NO)
dumpdev="NO"                # Device name to crashdump to
                            # (if enabled)
check_quotas="NO"           # Проверять дисковые квоты
# -- добавления от sysinstall -- #
sendmail_enable="YES"
sshd_enable="YES"
inetd_enable="YES"
```

Можно видеть, что содержимое файла `/etc/rc.conf` довольно прозрачно. В свежеустановленных системах этот файл может быть много меньше, ибо все отрицательные строки (НЕ запускать что-либо) можно смело удалять. В этом примере они приведены только для того, чтобы вы могли потом изменить значение NO того или иного параметра.

Начиная с FreeBSD 4.2 существует файл `/etc/defaults/rc.conf`, который содержит настройки системы по умолчанию. Файл `/etc/rc.conf` содержит только дополнения и изменения по отношению к этим стандартным настройкам.

Загрузка систем System V

В System V нет какого-то одного файла, в котором определяются все параметры системы. Для управления загрузкой системы надо начать с файла /etc/inittab и проследить, какие стартовые скрипты запускаются. В одном из них вы наверняка найдете, откуда берется тот или иной параметр. К счастью, менять порядок загрузки приходится довольно редко. Если возникает необходимость запускать конкретную программу на каком-либо уровне выполнения, то самый простой путь – включить ее запуск в /etc/inittab. Более корректный способ состоит в том, чтобы найти файл, в который принято помещать вызовы специфичных для данной системы программ. Это может быть, например, файл rc.local в каталоге /etc/rc.d/ или в каталоге с похожим именем. Пример файла inittab рассмотрен ниже:

```
#
# inittab This file describes how the INIT process should set
#         up      the system in a certain run-level.
#
# Default runlevel. The runlevels used by RHS are:
# 0 - halt (Do NOT set initdefault to this)
# 1 - Single user mode
# 2 - Multiuser, without NFS (The same as 3, if you do not
#     have networking)
#   3 - Full multiuser mode
#   4 - unused
#   5 - X11
#   6 - reboot (Do NOT set initdefault to this)
#
id:3:initdefault: # это уровень выполнения по умолчанию

# System initialization.
si::sysinit:/etc/rc.d/rc.sysinit
# Запуск rc-скриптов
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

# Things to run in every runlevel.
ud::once:/sbin/update

# Trap CTRL-ALT-DELETE
ca::ctrlaltdel:/sbin/shutdown -a -t3 -r now

# Запускать getty
1:2345:respawn:/sbin/mingetty tty1
2:2345:respawn:/sbin/mingetty tty2
```

```
3:2345:respawn:/sbin/mingetty tty3
4:2345:respawn:/sbin/mingetty tty4
5:2345:respawn:/sbin/mingetty tty5
6:2345:respawn:/sbin/mingetty tty6

# Запускать графическую подсистему xdm на уровне 5
x:5:respawn:/etc/X11/prefdm -nodaemon
```

Стартовые скрипты при выполнении могут обращаться к файлам настроек. Эти файлы в разных диалектах UNIX могут быть разными. Например, в Solaris есть несколько файлов, содержимое которых важно для запуска системы:

- `/etc/default` – настройки некоторых программ по умолчанию
- `/etc/default/login` – указание терминалов, с которых можно входить под именем root
- `/etc/system` – параметры ядра
- `/etc/dfs` – параметры NFS
- `/etc/vfstab` – список автоматически монтируемых файловых систем

В других диалектах тоже есть некоторые из этих файлов, но они по-другому называются. Например, файл `/etc/vfstab` обычно называется `/etc/fstab` и содержит важную информацию о разделах UNIX. Эту информацию используют программы **`fsck`** и **`mount`**. В частности тогда, когда они запускаются из стартовых скриптов.

Обычно стартовые скрипты выполняются без сучка, без задоринки. Однако если при проверке диска **`fsck`** нашла ошибку, которую не смогла исправить сама, она по умолчанию спросит совета оператора. При невозможности завершить стартовый скрипт система прекратит загрузку. Исправить положение в такой ситуации вы сможете, только работая на консоли. Поэтому не вносите никаких серьезных изменений в систему, находясь от нее далеко и работая по сети.

13

Установка новых программ в UNIX

Компиляторы и компоновщики

Для того чтобы получить из исходного текста на каком-нибудь языке программирования выполняемую программу, ее надо скомпилировать и затем скомпоновать с библиотекой стандартных системных вызовов и, возможно, другими библиотеками и другими объектными модулями. В UNIX есть компиляторы самых разных языков программирования.

Некоторые из них бесплатные, так как рапространяются в рамках проекта GNU. Эти компиляторы не всегда генерируют эффективный машинный код. С другой стороны, новые версии UNIX и новые процессоры появляются достаточно часто, и если вы купили коммерческий компилятор, оптимизированный для конкретной аппаратуры и версии UNIX, то не исключено, что деньги были потрачены зря: через полгода ваша аппаратура и версия UNIX окажутся позавчерашним днем, в них найдут кучу ошибок и заменят более свежими версиями.

Самый известный и самый используемый компилятор в UNIX – это компилятор GNU C (gcc). Часто его можно вызвать командой **cc** (это дань традиции: первый компилятор языка C в UNIX назывался cc).

Практически все программное обеспечения для UNIX пишется на C. Сложные серверы приложений иногда пишут на C++. Энтузиасты ФОРТРАНА пишут свои, обычно научно-расчетные, программы на ФОРТРАНЕ. Совершенное и документированное API в UNIX есть только для функций языка C.

Компилятор вызывается командой **cc** *имя_файла*.**c**.

По умолчанию компилятор **cc** создает файл с именем `a.out`, содержащий выполняемый код. Это дань старой традиции. Если вы хотите от-

личать свои программы друг от друга, запускайте их компиляцию с ключом **-o**:

```
cc -o имя_результата имя_исходного_файла
```

Например:

```
cc -o a.exe a.c
```

Программа make

Для компиляции и компоновки программ из большого числа исходных файлов принято использовать программу **make**. Более того, это единственный общепринятый способ установки программ из исходных текстов.

Программа **make** относится к программам управления проектами. Это означает, что до ее запуска разработчик (программист, администратор) создает файл описания проекта. В этом файле указывается, какие файлы должны стать результатом проекта, а какие являются исходными для него.

Когда придумывали программу **make**, предполагалось, что исходные файлы – это тексты программ, а результат представляет собой выполняемую программу (или несколько таковых). Однако, например, исходным файлом вполне может быть текстовый файл конфигурации sendmail, а результатом – двоичная база, которая строится по нему программой **makemap**.

Программе **make** безразлично, что представляют собой исходный файл и результат, ей важнее всего правила преобразования первого во второе. Эти правила и записываются в файле описания проекта. По умолчанию этот файл носит имя `Makefile`.

Владеть искусством составления make-файлов системный администратор не обязан, но это умение часто бывает полезным. Мы рассмотрим стандартный формат make-файла. Существуют различные расширения, с которыми можно ознакомиться в оригинальной документации или дополнительной литературе.

Включение раздела о программе **make** в главу об установке программ оправданно, но не до конца. Программе **make** следовало бы уделить большую главу. Возможно, это будет сделано впоследствии.

Формат make-файлов

Программа **make** существует очень давно. В современных системах встречаются либо ее классические версии (в основном в коммерческих UNIX), либо GNU-версии. В последних синтаксис make-файлов заметно расширен по сравнению с классическим, добавлены многие новые возможности, при этом сохранена обратная совместимость с классиче-

скими версиями. Это значит, что make-файл, написанный много лет назад для классической версии **make**, будет верно интерпретирован GNU make.

Мы рассмотрим только основные конструкции make-файлов, относящиеся к классической версии.

В make-файле могут содержаться комментарии (строки, начинающиеся с «#») и команды. Команды бывают двух типов: определения переменных и правила.

Определение переменных – это конструкция типа

```
OBJECTS = main.o too.o far.o gsm3d.o graphics.o
```

Встречаются и более сложные определения, например

```
OBJECTS = main.o too.o far.o gsm3d.o graphics.o
FILES = $(OBJECTS) add.c add.o
```

Переменные могут использоваться в любых командах, но только после определения. Правило определяет, что надо получить, из чего и как именно:

```
main.o : main.c
gcc -c main.c
```

ВАЖНО! Перед **gcc** стоит знак табуляции. Настоящий знак табуляции, а не восемь пробелов. Если при обработке make-файла вы получаете сообщение `"missing separator"`, это значит, что ваш текстовый редактор тихо заменил табуляцию пробелами. Это бывает при редактировании файлов UNIX в Windows- и DOS-редакторах.

В этом примере есть три части: результат (target), источники, от которых зависит результат (dependencies), и команда. Команд может быть несколько.

Результат – это то, что мы хотим получить. В данном случае мы хотим получить файл `main.o`. Результат отделяется от источников двоеточием. Затем следует список источников. В данном случае для создания файла `main.o` нужен всего один файл-источник – `main.c`. Обычно их бывает много больше.

Термин «dependencies» (дословно – зависимости) указывает на то, что, если хотя бы один из файлов-источников будет изменен, необходимо будет заново переделать файл результата.

Последняя часть правила – это команда, которую должна запустить программа **make**, чтобы из источников получить результат. В нашем случае команда **gcc** нужна для компиляции `main.o` из `main.c`.

Если команд несколько, то каждая располагается в отдельной строке, пустые строки между ними не допускаются, каждая строка с командой начинается с символа табуляции.

Обычно make-файл содержит несколько (и даже много!) переменных и правил. Например, для создания исполняемого файла `main.exe` из двух файлов исходных текстов потребуется такой make-файл:

```
main.exe: main.o too.o
gcc main.o too.o -o main.exe

main.o: main.c
gcc -c main.c

too.o: too.c
gcc -c too.c
```

Этот make-файл будет использован, как только вы дадите команду **make** в каталоге, где он находится. Программа **make** при запуске ищет файл с именем `Makefile`, если его нет, то `makefile`. Можно указывать программе **make**, какой именно файл следует искать:

```
make -f makefile
```

Программа **make** всегда начинает работу с первого встреченного в make-файле правила. Она пытается получить указанный в нем результат. Когда она находит в правиле источник, она выясняет, не надо ли освежить результат, зависящий от этого источника.

Например, в нашем случае с `main.exe` программа **make** выполнит следующие команды, когда мы ее запустим в первый раз:

```
gcc -c main.c
gcc -c too.c
gcc main.o too.o -o main.exe
```

Если мы затем изменим `too.c` и запустим **make** снова, она выполнит только две команды:

```
gcc -c too.c
gcc main.o too.o -o main.exe
```

Make не станет перекомпилировать файл `main.c`, так как по дате его последней модификации увидит, что он не менялся с момента последней компиляции.

Реальные make-файлы обычно содержат не только правила, как наш пример, но и определения переменных. Последние размещаются в начале make-файла, что позволяет легко изменять набор компилируемых файлов.

Добавим в наш пример определения переменных, чтобы сделать его более похожим на самый настоящий make-файл:

```
CC = gcc
CFLAGS = -g -O2
OBJECTS = main.o too.o
```

```
main.exe: $(OBJECTS)
$(CC) $(CFLAGS) $(OBJECTS) -o main.exe

main.o: main.c
$(CC) $(CFLAGS) -c main.c

too.o: too.c
$(CC) $(CFLAGS) -c too.c
```

Этот make-файл выглядит весьма похоже на предыдущий, но правила написаны с использованием подстановки значений переменных. Обратите внимание, что переменные могут иметь значение как имен команд, так и аргументов.

Такая организация make-файла помогает быстрее вносить изменения в наборы используемых файлов или легко менять имя используемого компилятора.

В GNU make допустимо использовать не длинные списки файлов, а шаблоны:

```
%.o : %.c
$(CC) $(CFLAGS) -c $<
```

Здесь на совпадающие части имени результата и источника указывает знак процента «%». Специальная переменная `$<` означает «все, от чего зависит результат». Еще одна полезная переменная `$@` означает «результат».

Теперь наш make-файл приобретает такой вид:

```
CC = gcc
CFLAGS = -g -O2
OBJECTS = main.o too.o

main.exe : $(OBJECTS)
$(CC) $(CFLAGS) $(OBJECTS) -o main.exe

%.o : %.c
$(CC) $(CFLAGS) -c $<
```

Теперь для добавления новых файлов с исходными текстами нам достаточно изменить переменную `OBJECTS`.

В этом примере мы могли опустить последнюю строку, так как **make** имеет свои встроенные предопределенные правила. Например, **make** знает, что создавать объектные файлы надо с помощью команды **$(CC) $(CFLAGS) -c $<**, так что достаточно просто использовать предопределенные имена переменных, чтобы **make** сама догадалась о необходимых командах.

Для получения полного списка предопределенных команд можно обратиться к документации. Используйте также команду **info make** для получения дополнительной информации.

Несколько способов установить программу

Новую программу можно получить через Интернет или иным способом в виде исходных текстов программы, готовых для компиляции, или двоичного файла, представляющего собой выполняемую программу, готовую к запуску. Установка программы из исходных текстов может быть осуществлена единственным разумным способом, а именно с использованием программы **make**. Остальные способы (запуск компилятора вручную и т. п.) применяются в случае некомплектных или неправильно укомплектованных дистрибутивов, ошибок в конфигурации, совершенных разработчиком, отсутствия необходимых системных библиотек и прочих редких событий.

Установка программы, полученной в виде двоичного файла, готового к запуску, выполняется по-разному. Программа может представлять собой один файл или набор файлов (пакет). Для установки может потребоваться наличие определенных каталогов или определенных программ в системе. Наконец, двоичные файлы могут быть упакованы в архивы разных форматов. Это разнообразие вариантов приводит к тому, что пакеты программ, поставляемых не в исходных текстах, могут быть установлены разными способами. Наиболее распространенные будут рассмотрены ниже.

Лучше всего, если системный администратор имеет полный контроль над процессом установки пакета программ: от компиляции до копирования получившихся программ в каталоги типа `/bin` и `/usr/bin`. Поэтому рекомендуется всегда ставить пакеты программ из исходных текстов. Оправданием установки двоичных файлов, готовых к запуску, может служить невозможность скомпилировать пакет на локальной машине. Например, это может иметь место при установке компилятора языка C: если его еще нет, то и компилировать программу нечем. Лень или неграмотность системного администратора ни в коем случае не может оправдать этого!

Грамотный системный администратор всегда ведет учет того, что он устанавливает в систему. Идеально, если вы записываете дату установки, версию пакета, каталог, куда пакет ставится, расположение его файлов конфигурации и любые комментарии, связанные с пакетом, в тетрадь, которую держите рядом с сервером. Роль тетради может играть и файл, но обязательна его печатная копия. Не дай Бог, придется восстанавливать умершую файловую систему!

FreeBSD: ports collection, команда pkg_add

Ports collection

Во FreeBSD есть технология, позволяющая устанавливать программное обеспечение через сеть автоматически. Это так называемые «пор-

тированные» (ports), то есть адаптированные к использованию в системе FreeBSD программы. В ports collection содержится масса очень полезных программ. Нужный программный пакет будет скачан с ftp-сервера, где располагается его дистрибутив, распакован и установлен в системе, если применять эту технологию.

По идее схема применения такова:

1. При установке системы на жесткий диск переписывается часть ports collection – установочные make-файлы и структура каталогов (для этого создается специальный каталог /usr/ports).
2. При желании установить какую-либо программу вы входите в один из подкаталогов каталога `/usr/ports` и даете там команду **`make`**. У подкаталогов «говорящие» названия, скажем, каталог `bash` соответствует программе **`bash`**.
3. Архив с дистрибутивом программы, которой соответствует этот подкаталог, скачивается через сеть оттуда, где он лежит, файл архива распаковывается и пакет устанавливается в системе.

Таким образом устанавливаются готовые к запуску пограммы. Исходные тексты при таком способе установки обычно НЕ скачиваются!

На практике эта технология весьма неудобна. Дело в том, что make-файлы в каталоге `/usr/ports` содержат фиксированные имена файлов архивов. Эти имена часто включают в себя номер версии программы. Более того, в make-файле явно указывается потребная версия. Однако свежие версии пакетов выходят очень часто, и если ваша ports collection установлена из дистрибутива месячной давности, многие файлы на сайтах производителя ПО успели поменять местоположение и название.

Поэтому пользоваться ports collection так, как предполагалось изначально, не слишком удобно. С другой стороны, сама коллекция программ на *www.freebsd.org* организована почти безукоризненно. По адресу *www.freebsd.org/ports* можно обратиться за дистрибутивом практически любой программы или пакета программ. Там же вы найдете и ссылки на сайты производителей ПО, а также на исходные тексты программ.

Команда pkg_add

Во FreeBSD можно добавить пакет программ. Для этого надо скачать архивный файл, который имеет формат пакета программ FreeBSD. Фактически это tgz-архив, в котором есть готовые к запуску программы и необходимые им файлы конфигурации, а также файлы, описывающие этот архив (назначение файлов и т. п.). Подробнее об архивах tgz можно прочесть в главе 27.

Если такой архив у вас уже есть, то соответствующий пакет программ устанавливается командой

`pkg_add` *`имя_пакета`*

После установки пакета **pkg_add** копирует файл со списком содержимого этого пакета в каталог /var/db/pkg/<*имя_пакета*>.

Туда же будут скопированы скрипт удаления пакета из системы (deinstall script) и файл с описанием пакета. Если пакет требует наличия каких-то иных пакетов, установленных в системе, в их каталогах /var/ db/pkg/<*other-pkg*> будет размещен файл +REQUIRED_BY, в отором будет указано, какие пакеты от них зависят.

Удаление пакета выполняется командой **pkg_delete** *имя_пакета*.

Linux: команда rpm

Аналогом **pkg_add** в Linux является команда **rpm**. Она устанавливает пакеты, которые доступны в Интернете в формате файлов rpm (RedHat Package Module). Пусть такая аббревиатура вас не смущает: формат rpm давно перешагнул рамки RedHat Linux и используется во многих Linux.

Команда установки пакета из rpm-файла выглядит так:

```
rpm -i имя_пакета
```

Удаление пакета также выполняется командой **rpm**, но с другим ключом:

```
rpm -e имя_пакета
```

У команды **rpm** есть и другие ключи, подробнее о них можно узнать в **man rpm**.

Формат **rpm** бывает нескольких версий, программы одних версий не понимают файлы других. Современная версия **rpm** – четвертая.

Debian Linux: команда dselect

В Debian Linux установкой и настройкой пакетов занимается менеджер пакетов – программа **dselect**. Она предоставляет текстовое упрощенное меню для удобства работы со списком пакетов. Для ее успешной работы следует задать источник получения списка пакетов и самих пакетов (это будет одно из зеркал официального сайта debian, но можно указать и сайт по умолчанию). Если UNIX-машина должна выполнять обращение к Сети через HTTP proxy, то перед запуском **dselect** следует выполнить команды

```
export http_proxy="http://your.proxy.address:port"
export ftp_proxy="http://your.proxy.address:port"
```

Дополнительная литература

[8] Richard Stallman «Gnu Make Ver 3.79». – Free Software Foundation; ASIN: 188211481.

14

Конфигурация ядра системы

Ядро UNIX – это файл, который загружается при старте системы. Он содержит системные таблицы и подпрограммы работы с ними, а также драйверы устройств. Драйверы могут быть вкомпилированы в ядро статически, а могут быть выполнены в виде отдельных модулей, загружаемых в ядро.

Реконфигурация ядра требует его последующей перекомпиляции. В коммерческих системах UNIX исходный текст ядра не поставляется, поэтому вместо перекомпиляции происходит перекомпоновка ядра – сборка его из объектных модулей, указанных в файле конфигурации или взятых по умолчанию.

Многие параметры ядра, которые в давних версиях UNIX можно было изменить только путем перекомпиляции ядра, в современных версиях меняются динамически, с помощью простых команд. Например, количество семафоров в Linux и FreeBSD можно задать в файле `/etc/sysctl.conf`, а добавить функциональность маршрутизатора UNIX-компьютеру, который пока не выполнял такую роль, можно командой

```
sysctl -w net.inet.ip.forwarding=1
```

или

```
echo "1" > /proc/sys/net/ipv4/ip_forward
```

Модули и статические драйверы

Модули, загружаемые в ядро, были придуманы для поддержки технологии plug-and-play и для легкого добавления устройства в систему. Последнее важно, ибо до появления модулей добавление физического устройства в компьютер требовало перекомпилировать ядро (если только поддержка этого устройства не была добавлена в ядро заранее провидцем-администратором).

Достоинства модулей состоят в том, что:

- они загружаются только тогда, когда нужна поддержка соответствующих устройств или функций системы, а в остальное время они не занимают место в памяти;
- в качестве модуля можно оформить драйвер устройства, драйвер новой файловой системы;
- с помощью модуля можно переопределить системный вызов;
- можно оформить драйвер нового сетевого протокола как модуль (например, модуль поддержки IPX или Apple Talk);
- эмулятор для запуска двоичных файлов чужого формата можно оформить в виде модуля;
- модули легки для отладки и поиска проблем. Если вы пишете новый драйвер и компилируете его в ядро статически, то при каждом изменении кода драйвера вы должны перекомпилировать ядро. В случае модуля надо выгрузить старую версию модуля, перекомпилировать модуль и загрузить в ядро его новую версию.

Недостатки сводятся к тому, что:

- нестабильно работающий непроверенный модуль может дестабилизировать работу ядра;
- подмена модуля хакером-взломщиком приведет к тому, что он получит полный контроль над системой;
- случайная подмена модуля старой или нефункциональной версией (например, при восстановлении с резервной копии диска) приведет к неожиданному отказу системы;
- о модулях легко забыть, обновляя ядро, и возможна неприятная ситуация, когда работающее ядро фактически будет состоять из свежей статической части и старой необновленной загружаемой части. Это благо, если совместимость старых модулей и нового ядра сохранена, и вред – если нет.

Конфигурация основного ядра

Под основным ядром мы понимаем ту его часть, которая скомпилирована статически.

FreeBSD: изменение конфигурации

Для изменения конфигурации ядра надо отредактировать файл, описывающий эту конфигурацию. Файл расположен в `/sys/i386/conf/`. Точнее говоря, там расположено несколько файлов. После установки системы там будет файл `GENERIC`, описывающий конфигурацию ядра по умолчанию. **В него не будут** внесены никакие изменения, если вы при установке меняли конфигурацию устройств по умолчанию, напри-

мер, удалили поддержку IDE-контроллеров. Файл GENERIC описывает самый общий случай: неоптимизированное ядро, которое обычно годится для любой системы. Как и любое общее решение, это ядро может быть малоэффективно для вашего конкретного случая. Если вы собираетесь внести какие-нибудь небольшие изменения в ядро, просто создайте копию файла GENERIC и сохраните ее под любым симпатичным вам именем, скажем, TEST. И уж этот файл изменяйте по своему усмотрению. Довольно часто лишь удалением ненужных драйверов из ядра можно уменьшить его размер на несколько сотен килобайт.

Кроме файла GENERIC в /sys/i386/conf/ лежит файл LINT. Он содержит полный перечень всех возможных драйверов и настроек ядра. Компилировать ядро на основе этого файла не надо: таких конфигураций аппаратуры не бывает, тем более что одни настройки исключают некоторые другие. Однако LINT удобен как справочник по синтаксису и смыслу настроек. Все строки файлов конфигурации понятно откомментированы в самих файлах.

При создании своих файлов конфигурации можно вставлять в них нужные строки из файла LINT.

При редактировании файла конфигурации ядра соблюдайте осторожность и будьте последовательны. Удаляя поддержку SCSI-контроллеров, не оставляйте поддержку SCSI-устройств. Это нелогично и оттого приведет к конфузу: части ядра взаимосвязаны и отсутствие одной логически связанной с остальными части может повлечь невозможность скомпилировать ядро в целом. Не удаляйте те строки, смысл которых вам неясен, в надежде, что их непонятность говорит об их ненужности. Бывает, какой-нибудь мистически названный контроллер отвечает за работу с клавиатурой и его удаление даст замечательное ядро, но система будет глухой и не откликнется на ваши попытки с ней пообщаться.

Сам по себе файл конфигурации никакого влияния на работающее ядро не оказывает. Он нужен только для того, чтобы программа **config**, глядя в него, собрала все необходимые файлы, из которых будет компилироваться новое ядро, в один каталог.

FreeBSD: процедура

Для компиляции ядра на основе только что созданного файла конфигурации надо в каталоге /sys/i386/conf/ дать команду

config *имя_файла_описания_ядра*

затем перейти в каталог, который **config** укажет по окончании своей работы:

cd ../../compile/*имя_файла_описания_ядра*

И выполнить компиляцию и установку нового ядра:

```
make depend
make
make install
```

Старое ядро будет сохранено в корневом каталоге в файле с именем /kernel.old.

Linux: процедура

Для создания конфигурации ядра Linux применяются команды[1] **make menuconfig** или **make config** в каталоге с дистрибутивом Linux. Программа **menuconfig** более удобна, чем просто **config**, так как в последнем случае вы будете долго отвечать на несколько десятков вопросов о желаемых параметрах конфигурации ядра. А в первом сможете сделать то же самое из сравнительно удобного текстового меню. Для использования **make menuconfig** в системе должна быть уже установлена библиотека ncurses. Результатом выполнения команд **make config** и **make menuconfig** является текстовый файл .config, который содержит конфигурацию ядра в виде ответов на вопросы, заданные вам программой **make**.

```
cd /usr/src/linux
make menuconfig     # или make config, # создает файл .config
make bzImage        # КОМПИЛЯЦИЯ ЯДРА
make install
make modules              # когда нужны модули
make modules_install      # вслед за make modules
```

После создания ядра надо копировать файл ядра в /boot или другое удобное вам местоположение ядра и указать в /etc/lilo.conf путь к новому ядру. Не забудьте после этого запустить **lilo**, чтобы внести изменения в загрузчик! Если вы компилируете достаточно свежее ядро Linux (скажем, версию 2.4.30), то по команде **make install** будет автоматически сохранена резервная копия старого ядра, выполнено копирование нового, модифицирован /etc/lilo.conf и запущено **lilo**.

В структуре конфигурационного файла ядра Linux бывают интересные странности. Например, в версии ядра 2.4.30 новые драйверы поддержки устройств Serial ATA (жесткие диски SATA) включены в подсистему SCSI, а драйверы сетевых интерфейсов TUN/TAP, необходимые для создания виртуальной частной сети с использованием продукта OpenVPN, – в подсистему драйверов ARCNet. Внимательнее изучайте все разделы меню конфигурации ядра – ищите то, без чего ваша система не может работать так, как надо!

[1] В среде X Window можно обратиться к **menu xconfig** – тогда для создания конфигурации будет предоставлен графический интерфейс.

Linux: изменение конфигурации

В Linux есть возможность скомпилировать новое ядро, используя конфигурацию старого (последняя хранится в файле `.config`). Для этого перепишите файл `.config` из старого дерева дистрибутива ядра в новое и дайте команду

```
make oldconfig
```

В этом случае надо будет ответить только на вопросы о новых настройках, появившихся в свежем ядре. При переходе с очень древней версии ядра на новую вопросов будет очень много, проще выполнить те же действия, что и для создания нового ядра с нуля.

Другие диалекты UNIX

Для добавления драйверов в ядро Solaris надо в момент загрузки дать команду **`boot -r`**. Это вызовет распознавание нового оборудования и установку драйверов для него. Для изменения параметров ядра (числа семафоров и т. п.) достаточно изменить соответствующую изменяемому параметру строку в файле `/etc/system` и перезагрузиться.

Во всех коммерческих диалектах UNIX предусмотрена некая процедура изменения параметров ядра. Нельзя дать общий совет, как это делать в любой системе, надежнее всего поискать документацию по ключевым словам «kernel tuning», «kernel configuration». Идеология изменения ядра одинакова в Solaris, HP-UX и других коммерческих системах.

Загружаемые модули ядра

FreeBSD

Во FreeBSD загружаемые модули появились позже, чем в других системах. Не исключено, что из-за этого в ранних версиях системы они были немного менее отработаны и несвободны от ошибок. Модуль компилируется подобно любой программе. Во FreeBSD до версии 3.0 модули имели интерфейс **`lkm`**. Начиная с версии 3.0 интерфейс изменился и был назван **`kld`**. По умолчанию модули лежат в каталоге `/modules`, ими занимается семейство программ **`kld`**. Загрузка модуля – **`kldload`**, выгрузка – **`kldunload`**. Имя загружаемого модуля должно заканчиваться на `.ko`. Подробности доступны по команде **`man kld`**. Статус уже загруженных модулей можно посмотреть командой **`kldstat`**:

```
kldstat
Id Refs Address    Size     Name
 1    2 0xc0100000 30cfb4   kernel
 2    1 0xc0c06000 2000     snake_saver.ko
```

Linux

В Linux модульная идеология появилась с рождением ядра. Лучше всего перед тем, как компилировать и настраивать модули ядра, изучить файл `Documentation/modules.txt`, поставляемый с дистрибутивом. Для нетерпеливых и тех, кто любит читать по-русски больше, чем по-английски, ниже приводится его сокращенный литературный перевод, выполненный мною.

«Во-первых, всегда надо использовать наиболее свежее ядро Linux. Не ради красоты, а для безопасности. В старых версиях уже найдены многочисленные лазейки и дыры. Выбрав старую версию ядра, вы обрекаете себя на риск взлома системы. В новом ядре дыр может быть и не меньше, но они пока еще никому не известны.

Во-вторых, утилиты работы с модулями ядра (пакет modules-x.y.z.tar.gz) также должны быть самыми свежими. Здесь «x.y.z» обозначает версию ядра, например 2.4.18. Используйте с ядром версии x.y.z утилиты той же версии, а не более старой или более новой. Создатели ядра Linux могут менять (и делают это!) состав и принципы работы утилит, не надейтесь, что здесь есть сквозная совместимость. В Linux принято жертвовать (в разумных пределах) совместимостью в угоду безопасности и производительности.

Если вы работаете с ядром версии 2.4.x или более новым, имеет смысл изучить также файл `Documentation/kmod.txt`, потому что в этой версии изменили идеологию работы демона `kerneld`. Его переименовали в `kmod` и обучили работать с модулями без вызова внешних утилит. Подробнее об изменениях в ядре см. файл `Documentation/Changes`.

Какие именно элементы ядра будут скомпилированы как модули, вы решаете на этапе выполнения команды **`make config`**. Там можно указать, что элемент будет статической частью ядра (y), не будет включен в ядро вообще (n) и будет скомпилирован в виде модуля (m). Создатели Linux рекомендуют все, кроме самых главных частей ядра, компилировать в виде модулей.

Чтобы избежать проблемы с загрузкой, нельзя компилировать в виде модулей следующее: драйвер файловой системы вашего корневого UNIX-раздела диска, поддержку жестких дисков (normal HDD support), поддержку сети (CONFIG_NET), поддержку TCP/IP (CONFIG_INET). Остальное, в том числе драйверы сетевых карт, рекомендуют компилировать в виде модулей.

Серьезным основанием для такой конфигурации (почти все в модулях) является меньший объем памяти, занимаемой ядром. Поставщикам разнородного оборудования и тем, кому надо управлять тучей этого разнородного оборудования, идеология модулей может пригодиться для поставки и настройки Linux-компьютеров с различной аппаратной конфигурацией. Проще говоря, вы собираете базовое ядро, которое будет работать на чем угодно, а в за-

висимости от аппаратуры конкретной машины настраиваете на ней загрузку нужных модулей.

Для компиляции модулей надо после компиляции статической части ядра дать команды

```
make modules              # когда нужны модули
make modules_install      # вслед за make modules
```

Первая команда вызывает компиляцию всех модулей и обновление каталога `linux/modules` дистрибутива. Вторая команда копирует все свежие модули в подкаталоги `/lib/modules/kernel_release/`, где «kernel_release» – это нечто вроде 2.4.18. Добавление готового модуля в выполняющееся ядро осуществляется командой `insmod`.

Удаление модуля из ядра – **rmmod**. Модулями можно управлять и посредством утилиты **modprobe**. Конфигурационный файл `modprobe - /etc/conf.modules`.

Linux предоставляет уникальную возможность динамической загрузки модуля ядра по запросу к этому модулю. Это значит, например, что если ваш SCSI-драйвер оформлен в виде загружаемого модуля ядра, то пока к нему нет обращений, он не загружен. Как только происходит обращение к нему, выполняется следующая процедура:

1. Ядро обнаруживает, что запрашиваемая функция не находится в ядре.
2. Ядро посылает запрос в **kerneld** с символьным описанием запрошенной функции.
3. **kerneld** просит **modprobe** загрузить модуль, отвечающий этому символьному описанию.
4. **modprobe** изучает свою внутреннюю таблицу псевдонимов. Она может быть изменена или дополнена строками alias в файле `/etc/conf.modules`.
5. **modprobe** просит **insmod** вставить в ядро модуль (или несколько), который **modprobe** решила загрузить. Каждый модуль может быть настроен командой **options** в файле `/etc/conf.modules`.
6. **modprobe** завершает работу, и **kerneld** сообщает ядру, что запрос на загрузку модуля выполнен (или не выполнен).
7. Ядро адресует свежезагруженному модулю пришедший запрос.

В новых версиях ядра набор программ **kerneld - modprobe-*mod** может быть заменен одним демоном kmod.

Загруженный по запросу модуль, если к нему нет обращений, автоматически выгружается через некоторое время. Обычно через одну минуту».

Другие диалекты UNIX

Многие диалекты UNIX используют идею модулей ядра. Скорее всего, вы сможете найти документацию по ним, используя команду **man**. Весьма вероятно, что ключ к дальнейшему изучению модулей отыщется в **man insmod** или **man modules**.

15

Настройка удаленного доступа pppd

Для установки соединений с использованием последовательных портов последние надо настраивать, указывая как минимум скорость на интерфейсе и тип управления потоком данных (аппаратный или программный; для работы с модемом всегда требуется устанавливать аппаратный). Для выполнения этих настроек служит команда **stty**, обсуждавшаяся в главе 3 в отношении терминалов. Обычный набор настроек для FreeBSD представлен ниже:

```
stty -f /dev/ttyld0 57600 crtscts cs8
stty -f /dev/ttyid0 57600 crtscts cs8
stty -f /dev/ttyd0 57600 crtscts cs8
```

Эти строки несколько расходятся с обычно рекомендуемыми в том, что для устройств `/dev/ttyid0` и `/dev/ttyld0` выполняется несколько большая настройка, чем требуется, но практика показывает, что лишней она не будет. Параметры, указанные для COM1 (`/dev/ttyd0`), в этом примере означают:

- `57600` – скорость порта
- `crtscts` – задействовать аппаратное управление сигналами RTS/CTS
- `cs8` – использовать 8-битные символы при передаче данных.

В середине 1990-х годов был весьма популярен PPP-сервер iijppp, по настройкам довольно сильно похожий на современные ppp-серверы.

Сейчас используют сервер **pppd**, причем он есть в поставке как новых коммерческих, так и некоммерческих UNIX. В Solaris до версии 8 включительно он называется **aspppd** (as – от asyncronous). Начиная с версии Solaris 9 и в эту систему устанавливается более удобная программа **pppd**.

Программа **pppd** может работать как сервер удаленного доступа, то есть принимать входящие звонки, а может играть роль клиента, дозваниваясь до сервера удаленного доступа.

При соединении двух компьютеров по протоколу PPP каждый из них рассматривается как равноправный участник соединения. Любой участник может предъявить к соединению свои требования: запросить определенный IP-адрес для себя или другого участника, потребовать выполнения аутентификации и т. д. Если второй участник соглашается с требованием или его собственные требования не противоречат запрошенным, соединение устанавливается. Если требования участников противоречивы, то соединение не устанавливается.

Настройки программы **pppd** обычно находятся в каталоге `/etc/ppp` (FreeBSD и Linux). В других диалектах надо отыскать какой-нибудь файл настроек pppd (например, `options` или `ppp.conf`) – рядом найдутся все остальные.

Синтаксис вызова **pppd**:

pppd *параметры*

Параметры должны включать *имя_устройства* и *скорость*.

Имя устройства – `/dev/tty01`, например, для соединений через COM2 (`tty00` – COM1, `tty01` – COM2). В диалектах UNIX, где приняты разные имена устройств для исходящих и входящих последовательных соединений, сервер удаленного доступа будет запускаться с параметром `/dev/tty01`, а дозвонка до сервера удаленного доступа – с параметром `/dev/cua01`, при этом физически будет задействован один и тот же модем. Иногда вместо `cua01` устройство носит имя `cuaa1`.

Если вы настраиваете сервер удаленного доступа, то надо заранее записать в конфигурацию модема (то есть в NVRAM самого модема) значения S0=1 и S1=1 для того, чтобы модем снимал трубку после первого звонка.

Для непосредственного общения с модемом удобна простейшая терминальная программа **cu**:

```
cu -l /dev/cua01
```

В Linux обычно применяют **minicom** вместо **cu** – первая имеет более понятный интерфейс и работать с ней проще.

Если никакая программа не общается с `/dev/cua01` в данный момент, то по этой команде вы подключитесь к порту COM2. Теперь можно набирать команды модему: все, что вы наберете, будет передано в последовательный порт. Для выхода из **cu** надо набрать ~. (тильда, затем точка) и подождать секунду-другую.

Обычно право обращаться к устройствам напрямую через **cu** имеет только root. Это право почти всегда определяется только правами доступа к файлам последовательных устройств `/dev/cua*` и `/dev/tty*`.

Параметры **pppd** удобно записать в файл /etc/ppp/options. Если все ppp-сессии на вашем сервере будут однотипными (одна скорость, соединение через один и тот же модем, звонки всегда только входящие), в этот файл можно записать все параметры **pppd**. Саму программу **pppd** можно прописать в качестве командого процессора в /etc/passwd всем пользователям, соединяющимся с сервером по телефонной линии.

```
cat /etc/ppp/options
/dev/cuaa0                  # устройство
57600                       # скорость
crtscts                     # управл. сигн. RTS/CTS
modem                       # модем, использовать DTR
debug                       # протоколирование сессии
passive                     # устанавливать соед. и ждать[1]
dns1 193.114.38.65          # установить DNS
193.114.38.5:193.114.38.4   # назначить адреса local:dialup[2]
-detach                     # не уходить в background
lock                        # блокировка порта по типу UUCP
```

Параметр debug включает протоколирование сессии. Программа **pppd** расценит этот параметр как требование записывать через **syslogd** все управляющие входящие и исходящие пакеты в удобочитаемой форме. Запись происходит от имени источника daemon с уровнем debug. Подробнее об источниках и уровнях записей в **syslogd** см. главу 31.

Параметр modem указывает, что надо использовать сигналы управления модемом. Программа **pppd** ждет сигнала CD (carrier detect) от модема, если только не указан параметр connect script (сценарий соединения, script – это имя файла сценария), и «передергивает» (коротко выключает, затем включает) сигнал DTR (data terminal ready), как только соединение завершается и перед тем, как запустить сценарий соединения.

Для каждого пользователя можно установить индивидуальные настройки **pppd**, положив в их домашние каталоги файлы .ppprc, которые будут прочитаны демоном **pppd** после /etc/ppp/options и могут содержать дополнительные сведения для него.

Выдача динамических адресов выполняется путем создания файлов /etc/options.ttyS0 и т. п. Кроме этого можно вообще ничего не пи-

---

[1] Этот параметр влияет на установление соединения в фазе LCP – Line Control Protocol. Демон **pppd** посылает пакет LCP и ждет ответа с другого конца. Если не дожидается, остается ждать приемлемого пакета LCP, а не разрывает связь, как по умолчанию. Приемлемость определяется требованиями к соединению, описанными в файле options.

[2] local – адрес интерфейса сервера, на котором запускается **pppd**, dialup – адрес, который будет выдан пользователю, дозвонившемуся до сервера. Если модемов несколько, то этой строчки в /etc/ppp/options быть не должно, адреса должны выдаваться динамически.

сать в options о назначении адресов. По умолчанию всем позвонившим на ваш сервер удаленного доступа **pppd** выдаст адрес ethernet-интерфейса этого сервера. Различать клиентов он будет не по IP-адресу, а по интерфейсу – ppp0, ppp1 и т. д.

Приведенный пример предполагает, что пользователи при дозвоне на сервер удаленного доступа избегают стандартного протокола аутентификации в Windows-системах – так называемого PAP (Password Authentication Protocol). В данном примере предполагалось, что они выбирают конфигурацию «generic login» из Windows XP или вводят имя и пароль вручную, открытым текстом.

Если требуется настроить аутентификацию с применением PAP и при этом использовать для аутентификации файл /etc/passwd, то надо записать в файл /etc/ppp/options параметр login. При этом пользователь, который соединяется с сервером удаленного доступа по протоколу PAP, должен иметь учетную запись и в /etc/passwd, и в /etc/ppp/pap-secrets. Формат записей в последнем таков:

имя_клиента имя_сервера пароль IP-адрес

В каждом из полей может стоять символ «*», говорящий о допустимости любого значения. Как правило, строка аутентификации в pap-secrets выглядит так:

```
ivan * KqZXV5-u *
```

Здесь определены имя и пароль пользователя ivan, которому разрешено соединяться с этим сервером.

Если нет параметра login, некоторые версии **pppd** ограничиваются аутентификацией с использованием только /etc/ppp/pap-secrets.

В некоторых версиях **pppd** при установленном параметре login пароль, записанный в /etc/ppp/pap-secrets, сначала считается незашифрованным. Поэтому любой, прочтя этот файл, может ввести в строку пароля именно то, что записано в этом файле, и будет аутентифицирован. Только во вторую очередь предполагается, что пароль там зашифрован. В таких **pppd** надо выяснить, как указывать в /etc/ppp/options, что требуется считать пароль, записанный в pap-secrets, зашифрованным.

Есть, таким образом, две схемы организации сервера удаленного доступа:

1. Пользователь дозванивается на модем сервера, где его встречает программа **getty** (стандартная программа, обслуживающая вход на любой терминал, кроме псевдотерминала). Она запрашивает регистрационное имя (login) и пароль (password), которые пользователь сообщает открытым текстом вручную или запуская скрипт на своей машине. Затем, уже после входа в систему, для пользователя запускается **pppd** в качестве оболочки.

2. При запуске системы вместо getty[1] на тот последовательный порт, где находится модем, запускается **pppd**, который ждет входящего звонка. Когда кто-то дозванивается, **pppd** сам проводит аутентификацию с использованием /etc/ppp/pap-secrets.

Запуск **getty** для определенных терминалов контролируется /etc/ttys во FreeBSD и в /etc/inittab в системах System V.

Файл /etc/ppp/pap-secrets может содержать как строки для аутентификации удаленных клиентов, так и строки, которые соответствуют аутентификации самого **pppd** на сервере удаленного провайдера. Имя пользователя, по которому **pppd** будет идентифицировать самого себя, задается параметром user.

При настройке клиента удаленного доступа надо добавить несколько строк в файл параметров, чтобы заставить **pppd** посылать удаленному серверу провайдера имя и пароль по запросу.

Если мы звоним на сервер удаленного доступа, организованный по типу 1, то файл options выглядит примерно так:

```
/dev/cuaa0
57600
crtscts
modem
debug
defaultroute
passive
-detach
lock
connect "chat -f /etc/ppp/chat.x"
```

Файл /etc/ppp/chat.x при этом должен быть примерно таким:

```
'' ATDT3258752 CONNECT \r name:-BREAK-name: pppuser ssword:
e.67FJq1
```

Если сервер, куда мы звоним, организован по типу 2, то options выглядит примерно так:

```
/dev/cuaa0
57600
crtscts
modem
debug
defaultroute
passive
-detach
lock
user myname
```

[1] Вместо **getty** может запускаться **mgetty**, **agetty** и т. п.

И в /etc/ppp/pap-secrets должна быть строка

```
myname * mypassword *
```

Если соединение с удаленной системой должно быть постоянным (например, нам необходимо постоянное соединение с провайдером по модему), можно написать простой скрипт, который будет запускать **pppd** снова, если он почему-то завершится:

```
#!/bin/sh
while sleep 3
do
/usr/sbin/pppd
done
```

«Засыпание» на три секунды нужно на всякий случай. Чтобы дать проблеме, из-за которой завершился **pppd**, время на то, чтобы самоустраниться. Цикл бесконечный, прервать его можно только посылкой сигнала kill. Такой цикл удобно оформить в виде скрипта, запускающегося при старте системы.

Вместо такого «вечного» цикла можно вписать вызов **pppd** в /etc/inittab (для System V) с параметром respawn (вызвать заново, если завершится).

Для дозвона по запросу (dial on demand) следует указать параметр demand в /etc/options. Он запускает дозвон **pppd** по факту появления пакета, который надо отправить по коммутируемому (телефонному) каналу.

16

Сетевая файловая система NFS

Общие соображения

Иногда необходимо работать с файлами, находящимися в файловой системе удаленного компьютера. В Windows-сетях для этого применяют механизм разделяемых каталогов (shared folders).

Файловые серверы под управлением UNIX получили достаточное распространение в мире и несколько меньшее – в России. Однако из-за того что клиентские машины на рабочих местах в России обычно управляются Windows-системами, в качестве файловых серверов выступают UNIX-компьютеры с установленным пакетом Samba. Он позволяет UNIX-компьютеру «прикинуться» сервером Windows NT так, чтобы все клиентские компьютеры воспринимали его как самый обычный файл- и (или) принт-сервер Windows-сети.

В тех случаях, когда в сети работают несколько UNIX-компьютеров и им надо обращаться к одному файл-серверу, имеет смысл обратиться к механизму NFS (Network File System), реализованному в UNIX в давние времена. Этот механизм, вообще говоря, не нашел большого признания в UNIX-сетях в России. NFS не позволяет шифровать данные при передаче по сети, а надежные UNIX-машины часто служат именно для хранения очень важных или конфиденциальных данных. Кроме того, NFS не очень устойчив к сбоям сети, он требует ее стабильной работы.

Здесь мы обсудим, как настроить работу NFS, однако я бы советовал применять NFS только там, где без него не обойтись. Особое внимание надо уделить жесткому распределению прав доступа к разделяемым каталогам, чтобы избежать взлома NFS-сервера.

NFS предполагает быстрое соединение между клиентом и сервером. Применение NFS для монтирования файловых систем иначе, чем в локальной сети, например через Интернет, технически возможно, но нерационально. Передавать файлы между удаленными компьютерами в Интернете удобнее с помощью протокола ftp.

Принцип работы NFS

После настройки в качестве NFS-сервера наша UNIX-машина будет предоставлять доступ внешним пользователям к определенным нами каталогам своей файловой системы. Такое предоставление доступа в NFS называется «экспортом»: говорят, что UNIX-машина экспортирует свои каталоги.

NFS работает через RPC (Remote Procedure Calls – удаленный вызов процедур).

Клиент NFS посылает запрос на монтирование демону **mountd**. Тот выдает клиенту NFS секретный ключ. Этот ключ представляет собой идентификатор, который однозначно определяет каталог, смонтированный по сети.

Через NFS можно монтировать как целые файловые системы, так и отдельные каталоги. Однако из соображений безопасности монтировать каталоги «через раздел» нельзя. Это значит, что если каталог `/alpha/var` расположен на одном UNIX-разделе, а каталог `/alpha` – на другом, то при монтировании каталога `/alpha` каталог `/alpha/var` не будет смонтирован автоматически. Если его тоже надо монтировать, то это следует потребовать у NFS-сервера отдельно.

Секретный ключ сохраняется при перезагрузке NFS-сервера, чтобы после восстановления его работы клиенты, повисшие в ожидании, могли продолжить работу с ним как ни в чем не бывало. Однако при демонтировании и новом монтировании файловой системы через NFS секретный ключ, как правило, меняется.

После монтирования файловой системы через NFS клиент посылает серверу NFS файловые запросы, которые удовлетворяет демон **nfsd**.

Демонтирование файловой системы выполняется командой **umount**.

Как устроен RPC

Идеология RPC очень проста и привлекательна для программиста. Как обычно работает сетевое приложение? Оно следует некоему протоколу (например, HTTP): формирует пакет с запросом, вызывает системную функцию установления соединения, затем функцию отправки пакета, затем ждет ответного пакета и вызывает функцию закрытия соединения. Это значит, что вся работа с сетью является заботой программиста, который пишет приложение: он должен помнить о вызове

функций сетевого API системы, предусмотреть действия в случае сбоя сети или недоставки пакетов.

RPC предполагает иной способ обмена данными между клиентом и сервером. С точки зрения программиста приложение клиента, работающее посредством RPC, вызывает функцию на сервере, она выполняется и возвращает результат. Пересылка запроса на выполнение функции через сеть и возврат результатов от сервера клиенту происходит незаметно для приложения, поэтому последнее не должно беспокоиться ни о сбоях сети, ни о деталях реализации транспортного протокола.

Для того чтобы обеспечить прозрачность пересылки запросов RPC и данных через сеть, придумана двухступенчатая процедура.

Каждое приложение на сервере, способное отвечать на запросы RPC, при запуске регистрируется в специальном списке. Этот список обслуживает программа, которая называется транслятором портов (port mapper). Функция этой программы – устанавливать соответствие между номером процедуры RPC, которую запросил клиент, и номером TCP или UDP порта, на котором приложение сервера ждет запросов.

RPC способен работать не только с протоколами TCP и UDP, например, в Solaris реализована работа на базе механизма TI (Transport-Independent), поэтому в Solaris транслятор портов называется **rpcbind**, а не **portmap**, как в Linux или FreeBSD.

Приложение, которое регистрируется у транслятора портов, сообщает ему номер программы, номер версии и номера процедур, которые будут обрабатываться данной программой. Эти процедуры будут впоследствие вызываться клиентом по номеру. Кроме этого, приложение сообщает номера портов TCP и UDP, которые будут использоваться для приема запросов на выполнение процедур.

Клиент, желающий вызвать выполнение процедуры на сервере, отправляет запрос транслятору портов на сервер, чтобы узнать, на какой TCP или UDP порт надо отправить запрос. Транслятор портов запускается при старте системы и всегда работает на порту 111. Получив ответ от него, клиент отправляет запрос на тот порт, который соответствует требуемому приложению. Например, сервер NFS работает на порту 2049.

Настройка NFS-сервера

Для настройки NFS-сервера нам потребуется настроить несколько программ.

portmap: объявление служб RPC

Прежде всего, надо запустить программу `portmap`. В разных системах она может называться по-разному: `portmap`, `rpc.portmap`, `rpcbind`.

Эти программы являются *трансляторами портов*, и поэтому в дальнейшем мы так и будем их называть. Транслятор портов преобразует номера вызовов RPC в номера портов.

Транслятор портов должен быть запущен до того, как будет запущен любой из RPC-серверов. При аварийном завершении транслятора портов требуется перезапустить его, а затем перезапустить все RPC-серверы, чтобы они по новой зарегистрировались у транслятора портов.

Доступ к последнему контролируется файлами `/etc/hosts.allow` и `/etc/hosts.deny` (в новых версиях FreeBSD используется только первый из них, см. **man hosts_access** или **man hosts.allow**). Подробнее об этих файлах рассказано в разделе «Страж tcp-wrapper» главы 10.

Программа **portmap** по умолчанию при запуске становится демоном, разрывая связь с терминалом, с которого она была запущена, и начинает выполняться в фоне, подобно всем демонам UNIX. Она может быть запущена с ключами **-d** и **-v**. Ключ **-d** меняет ее поведение по умолчанию: **portmap -d** не запускается в режиме демона в фоне и печатает в стандартный вывод (на экран) сообщения об ошибках и отладочную информацию. Ключ **-v** вызывает детальное протоколирование проверок прав доступа при обращении к RPC. Программа **portmap** записывает протокол через **syslog(3)**.

Когда транслятор портов запущен, можно проверить, все ли с ним в порядке, с помощью команды **rpcinfo -p**. Если **rpcinfo -p** не работает, но транслятор портов запущен, то надо проверить права доступа, определенные в файле(ах) `/etc/hosts.allow` и `/etc/hosts.deny`.

```
rpcinfo -p
program vers proto   port
100000    2   tcp    111  portmapper
100000    2   udp    111  portmapper
```

Программа **portmap** должна вызываться из стартовых скриптов, если в системе должны работать какие-либо RPC-службы, в том числе NFS-сервер.

mountd, nfsd: экспорт файловых систем

Сервис NFS предоставляется двумя программами, которые обрабатывают соответствующие RPC-запросы. Это программы **mountd** и **nfsd**.

Программа **mountd** обрабатывает запросы на удаленное монтирование файловых систем. Она также отвечает на запрос **showmount** о доступных для монтирования каталогах. Команда **showmount -e** представляет собой аналог команды **net view** в сетях Windows.

Программа **nfsd** – это обработчик запроса удаленного клиента NFS к файловой системе. Необходимо запустить достаточное количество

демонов **nfsd** – не меньше максимального числа одновременных обращений к NFS. Слишком маленькое количество демонов **nfsd** будет вызывать задержки в работе клиентов NFS: им придется становиться в очередь на обработку своих запросов. Слишком большое количество **nfsd** приведет к нерациональному расходу памяти NFS-сервера. Оптимальное значение определяется эмпирически, но оно точно не должно превышать общее количество компьютеров вашей сети, которые настроены для работы через NFS. Если при загрузке некоторого числа **nfsd** загрузка процессора возрастет до 100%, имеет смысл уменьшить число демонов **nfsd**.

Перед запуском **mountd** и **nfsd** надо указать в файле `/etc/exports`, какие именно каталоги предполагается экспортировать. В Solaris этот файл называется `/etc/dfs/dfstab`.

Вопросы безопасности рассмотрены ниже, в разделе «Безопасность сервера NFS». В частности там рассматривается доступ к файлам на сервере NFS от имени определенных на сервере пользователей. Обычно предполагается идентичность учетных записей на клиенте и сервере NFS, но удобно потребовать, чтобы все пользователи или только некоторые из них работали от имени какого-то конкретного пользователя. В последнем случае этот пользователь ограничивается в правах. Особенно это важно в отношении пользователя root, которому обычно сопоставляется какая-то иная учетная запись. Дело в том, что под именем root на клиентском компьютере работает не тот же самый человек, что под именем root на сервере NFS.

В `/etc/exports` надо указывать, каким компьютерам будет доступна по сети каждая экспортируемая файловая система. Допустим, на компьютере zeus мы экспортируем файловую систему `/space/nfs` и хотим, чтобы она была доступна для компьютера apollon. Тогда на компьютере zeus в файле `/etc/exports` мы должны написать:

```
/space/nfs  apollon(rw)
```

Этой строкой мы разрешаем компьютеру apollon монтировать файловую систему `/space/nfs` на компьютере zeus средствами NFS в режиме чтения и записи. По умолчанию устанавливается режим ro (read-only, только чтение).

Полный список доступных режимов и настроек при указании экспортируемой файловой системы доступен в **man exports**.

Файл `/etc/exports` в разных диалектах UNIX может иметь разный синтаксис.

Есть и более привлекательные способы определения прав доступа к экспортируемым файловым системам, чем указание каждой машины поименно в файле `/etc/exports`. Мы можем указывать имена сетевых

групп, если используем систему NIS[1] (известна еще как YP – yellow pages). Также можно использовать шаблоны, причем в шаблонах допустимы IP-адреса и сетей, и доменов.

Применяя шаблоны, будьте осторожны: указание слишком широкого шаблона приведет к излишней доступности NFS-сервера. Давать доступ к нему следует только тем компьютерам локальной сети, которые известны как надежные (в смысле безопасности).

Примеры файлов `/etc/exports` для разных систем приведены ниже. Для изучения полного набора ключей и деталей синтаксиса обратитесь к **`man exports`** или **`man dfstab`** в вашей системе.

Solaris: `/etc/dfs/dfstab`:

```
share -F nfs -o rw=@193.250.160, ro=@193.250.161 /export/home
share -F nfs -o ro=193.250.160@, root=193.250.161.132 /
export/usr/man
```

Объяснение

Файловая система `/export/home` экспортируется с возможностью чтения и записи для компьютеров сети 193.250.160 и только чтения – для компьютеров сети 193.250.161. Файловая система `/export/usr/man` экспортируется для чтения для компьютеров сети 193.250.160. Суперпользователю машины 193.250.161.132 разрешен доступ с правами root к этой файловой системе.

FreeBSD: `/etc/exports`:

```
/usr /usr/local -maproot=0:10 friends
/usr -maproot=daemon grumpy.cis.uoguelph.ca 131.104.48.16
/usr -ro -mapall=nobody
/u -maproot=bin -network 131.104.48 -mask 255.255.255.0
/u2 -maproot=root friends
/u2 -alldirs -kerb -network cis-net -mask cis-mask
```

Объяснение

`/usr`, `/u` и `/u2` – файловые системы сервера NFS. Тогда вышеприведенный `/etc/exports` описывает следующую ситуацию.

`/usr` экспортируется для компьютеров группы friends (friends описана в файле `netgroup`); пользователи получают доступ в соответствии с правами, определенными для их учетных записей на сервере NFS, root получает доступ от имени пользователя с UID 0 и GID 10. `/usr`

[1] В большинстве систем в России NIS не применяется. Это связано с несовершенством NIS и с тем, что она может быть полезна только в больших сетях UNIX. Пока сети UNIX в России редко включают более десятка UNIX-машин, хотя некоторые университеты уже обзавелись UNIX-сетями, в которых работают сотни Linux-компьютеров. Существует также NIS+ – более удобный механизм.

экспортируется с правом чтения и записи, и компьютеры, входящие в группу friends, могут монтировать как /usr, так и /usr/local.

Следующая строка говорит о том, что /usr экспортируется компьютерам grumpy.cis.uoguelph.ca и 131.104.48.16, пользователь root получает права доступа учетной записи daemon, ассоциируясь с UID и GID этой учетной записи. Обычные пользователи с этих компьютеров получают доступ в соответствии с правами, определенными для их учетных записей на сервере NFS.

Остальные компьютеры имеют доступ к /usr в режиме read-only, причем пользователи таких компьютеров получают доступ с правами учетной записи nobody.

При доступе к /u пользователь root удаленной машины получит права пользователя bin, доступ к /u разрешен только из сети 131.104.48.

Во FreeBSD допустимо появление каталога в нескольких строках файла /etc/exports. Поэтому к /u2 применяются все настройки, указанные в двух последних строках нашего примера. Во FreeBSD доступ к NFS реализован немного не так, как в других системах, потому что по умолчанию нельзя монтировать подкаталог разделяемого ресурса. Для того чтобы можно было отдельно монтировать не только каталог, указанный в /etc/exports, но и любой его подкаталог, надо указать параметр alldir.

Таким образом, /u2 монтируется для группы компьютеров friends с выдачей прав локального суперпользователя удаленному пользователю root и для компьютеров из сети cis-net с маской cis-mask с аутентификацией типа Kerberos. Соответствие имени сети ее IP-адресам определяется в файле /etc/networks (или /etc/netgroup).

Linux: /etc/exports:

```
/galant/homes          mars(rw,no_root_squash)
jupiter(ro)
/usr/man               *.company.spb.ru(ro)
```

Объяснение

Файловая система /galant/homes экспортируется компьютеру mars для чтения и записи, с трансляцией вызова от пользователя root в вызов от имени nobody, компьютеру jupiter – для чтения.

Файловая система /usr/man экспортируется компьютерам из домена company.spb.ru только для чтения.

Сервер NFS и бездисковые станции

Если организуется доступ к NFS с бездисковых станций, надо удостовериться, что сервер NFS экспортирует корневую файловую систему (и, если необходимо, файловую систему, где расположен swap) для сетевого клиента (бездисковой станции), а клиент имеет доступ с права-

ми администратора к этим файловым системам. В случае FreeBSD типичный файл /etc/exports может иметь такой вид:

```
/rootfs/client -maproot=0:0 client.domain
/swapfs -maproot=0:0 client.domain
```

Здесь под корневой файловой системой подразумевается специальный каталог на сервере, который играет роль корневой файловой системы для бездисковой станции. Собственный корневой каталог сервера экспортировать с правами администратора, конечно же, не надо.

В HP-UX тот же файл /etc/exports в том же случае будет иметь вид:

```
/rootfs/myclient -root=myclient.mydomain
/swapfs -root=myclient.mydomain
```

Запуск mountd и nfsd

После подобающей модификации файла /etc/exports можно запускать **mountd** и **nfsd**. В некоторых системах они могут называться **rpc.mountd** и **rpc.nfsd**. Программы **mountd** и **nfsd** всегда читают файл /etc/expotrs при запуске. Если они уже были запущены, а вы модифицировали /etc/exports, то должны сообщить им об этом.

Для того чтобы **mountd** и **nfsd** перечитали файл /etc/exports, в разных системах нужно давать разные команды. Однако посылка сигнала HUP каждому из этих демонов представляет собой наиболее общий способ:

```
ps -x | grep mountd
122 ?        SW     0:01 rpc.mountd
kill -HUP 122
ps -x | grep nfsd
125 ?        SW     0:01 [nfsd]
126 ?        SW     0:01 [nfsd]
127 ?        SW     0:01 [nfsd]
128 ?        SW     0:01 [nfsd]
129 ?        SW     0:01 [nfsd]
130 ?        SW     0:01 [nfsd]
131 ?        SW     0:01 [nfsd]
kill -HUP 125
```

Есть более короткий вариант:

```
killall -HUP /usr/sbin/rpc.mountd
killall -HUP /usr/sbin/nfsd
```

Вместо посылки сигнала HUP в HP-UX и некоторых Linux можно дать команду **exportfs -a**.

В Solaris NFS организована несколько иначе, а именно:

- Файл /etc/exports называется /etc/dfs/dfstab.

- Этот файл является не файлом конфигурации, а скриптом, который вызывает программу **share**; программа **share** управляет содержимым двоичной (не текстовой) базы данных экспортируемых файловых систем.
- После изменения параметров монтирования в файле /etc/dfs/dfstab надо дать команду **shareall** для вступления изменений в силу. Однако **shareall** просто выполняет все команды **share**, содержащиеся в /etc/dfs/dfstab. Поэтому если вы удалили оттуда команды экспорта нескольких файловых систем, действие старых команд **share**, запущенных до модификации файла, не будет отменено командой **shareall**. Вам придется просто перезапустить **mountd** для того, чтобы изменения возымели эффект. Однако не следует перезапускать **mountd** тогда, когда кто-то работает с файловой системой компьютера, где запущен **mountd**. Это может привести к потерям данных и зависанию клиентов NFS в сети.
- Solaris автоматически становится NFS-сервером при загрузке, если в файле /etc/dfs/dfstab указаны экспортированные каталоги. Например, для того чтобы экспортировать каталог /dfold/users, добавьте следующую строчку в файл /etc/dfs/dfstab:

  ```
  share -F nfs /dfold/users
  ```

  В этом же файле можно указать, что каталог экспортируется только для определенных машин:

  ```
  share -F nfs -o rw=mashine1:mashine2 /home1/dummy1
  ```

  Ну и конечно, можно экспортировать каталог только для чтения:

  ```
  share -F nfs -o ro=mashine3:mashine4 /home1/dummy2
  ```

Теперь необходимо убедиться, что **mountd** и **nfsd** запущены правильно. Сначала это делается с помощью команды **rpcinfo -p**. Вывод программы должен быть примерно таким:

```
100000    2   tcp    111  portmapper
100000    2   udp    111  portmapper
100003    2   udp   2049  nfs
100003    3   udp   2049  nfs
100005    3   udp   1023  mountd
100005    3   tcp   1023  mountd
100005    1   udp   1023  mountd
100005    1   tcp   1023  mountd
```

Как видно, транслятор портов анонсировал свои службы.

Если мы получили сообщение

```
rpcinfo: can't contact portmapper: RPC: Remote system error -
Connection refused
```

или

```
RPC_PROG_NOT_REGISTERED
```

или что-то подобное вместо этого, значит, транслятор портов не запущен. Возможно, в файлах /etc/hosts.allow или /etc/hosts.deny есть настройки, запрещающие транслятору портов отвечать. Завершите выполнение **nfsd**, **mountd** и **portmapper** и запустите их снова: сначала **portmapper**, затем **mountd** и потом **nfsd**. Программе **nfsd** может быть передан аргумент – число копий ее самой, которые следует запустить при старте **nfsd**. Программа размножится в нужном количестве копий сразу после запуска.

Программы **mountd** и **nfsd** должны запускаться на NFS-сервере из стартовых скриптов при загрузке системы. Например, во FreeBSD для того, чтобы компьютер был NFS-сервером, в файле /etc/rc.conf должны быть верно установлены параметры

```
nfs_enable="YES"
nfs_flags="…"   # необходимые флаги
```

Убедившись, что транслятор портов объявил службы, имеет смысл проверить, запущены ли соответствующие программы. Это позволяет сделать команда **ps**:

```
ps auxw | grep mountd # ключи auxw для BSD-систем
ps auxw | grep nfsd
```

или

```
ps -ef | grep mountd # ключи ef для SVR-систем
ps -ef | grep nfsd
```

Транслятор портов объявляет свои службы независимо от того, работают ли в данный момент программы, которые должны выполнять работу этих служб. Даже если эти программы (например, **mountd**) уже закончили свою работу. Так что проверка с помощью **ps** совершенно необходима, если что-то не работает.

Вам придется исправить системные файлы rc для запуска **mountd** и **nfsd** при загрузке. Очень вероятно, что эти скрипты уже существуют на вашей машине и потребуется только раскомментировать нужный раздел или активизировать скрипт на требуемом уровне выполнения. Справочные страницы, которые полезно изучить перед настройкой NFS, – **portmap**, **mountd**, **nfsd** и **exports**.

Блокировка файлов на NFS-сервере

Для того чтобы несколько процессов не конфликтовали при доступе к одному и тому же файлу, обычно применяется механизм блокировки файла. В UNIX этот механизм работает не идеально даже в локальных

файловых системах. Подробнее о нем можно прочесть в **man lockf** или **man flock**. В NFS блокировка реализована с помощью демонов **lockd** и **statd**, запускаемых на сервере NFS. Демон **lockd** устанавливает и снимает блокировку по запросу, а **statd** следит за состоянием блокировок и NFS-сервера в целом.

Демон **statd** (во многих системах **rpc.statd**) общается с демонами **statd** на других компьютерах. Он также принимает запросы от демона **lockd**, запущенного на той же (локальной) машине для отслеживания статуса указанных ему компьютеров. Если компьютер, за которым следит **statd**, перестает отвечать и перезапускается, удаленный **statd** сообщает об этом локальному, следящему за ним, и локальный информирует об этом программы, которым это важно. Аналогично, если локальная служба умирает и затем перезапускается, то **statd** информирует об этом соседей по сети.

Дисковые квоты в NFS

Кроме вышеупомянутых демонов на сервере NFS может быть запущен еще и демон, отвечающий за дисковые квоты при работе с NFS, – **rpc.rquotad**.

Таким образом, полная коллекция демонов на NFS-сервере будет выглядеть примерно так:

```
119 ?    SW    0:00 portmapper
120 ?    S     0:00 /usr/sbin/rpc.rquotad
122 ?    SW    0:01 [nfsd]
123 ?    SW    0:00 [lockd]
124 ?    SW    0:00 [rpciod]
125 ?    SW    0:01 [nfsd]
126 ?    SW    0:01 [nfsd]
127 ?    SW    0:01 [nfsd]
128 ?    SW    0:01 [nfsd]
129 ?    SW    0:01 [nfsd]
130 ?    SW    0:01 [nfsd]
131 ?    SW    0:01 [nfsd]
134 ?    S     0:00 /usr/sbin/rpc.mountd --no-nfs-version 3
137 ?    S     0:00 /usr/sbin/rpc.statd
```

Безопасность сервера NFS

К сожалению, настройки безопасности NFS разные в разных диалектах UNIX. В целом современные реализации NFS объединены следующим общим подходом:

- root с клиентской машины не должен получать по умолчанию права пользователя root сервера NFS;
- можно устроить все так, чтобы пользователи с клиентских машин могли работать от имени единственного пользователя сервера NFS.

Несколько слов о конкретных реализациях этого подхода.

В HP-UX пользователи root клиентов NFS по умолчанию получают доступ от имени nobody, остальные – от имени своих собственных учетных записей.

В Linux запросы от root транслируются в запросы от пользователя с UID и GID, указанными параметрами `anonuid` и `anongid` в `/etc/exports`, причем это делается по умолчанию, а для явного указания именно такой практики можно добавить параметр `root_squash`. Параметр `no_root_squash` разрешает выполнять запросы от root от имени пользователя root сервера NFS. Разрешается подменять учетные записи не только пользователю root, но и другим пользователям.

Во FreeBSD учетная запись, от имени которой обрабатываются запросы от root, назначается параметром `maproot`; по умолчанию это учетная запись nobody. Учетная запись, от имени которой будут выполняться запросы всех пользователей, задается параметром `mapall`.

Разберемся подробнее в назначении прав доступа к NFS и работе системы безопасности на примере Linux.

Подробности на примере Linux

Мы можем решить, что на NFS-сервере мы не хотим доверять администратору клиентских машин. Поэтому мы указываем параметр `root_squash` в файле `exports`:

```
/externals/usr/local apollo(rw,root_squash)
```

Теперь, если пользователь с UID 0 на стороне клиента попытается получить доступ на чтение и запись к файловой системе `/externals/usr/local`, то файловый сервер выполнит подстановку UID пользователя nobody на сервере. Это означает, что root с клиентской машины не сможет получить доступ с правами суперпользователя к файлам сервера, изменять которые или иметь доступ к которым может только администратор сервера. Это хорошо. Именно так и было задумано. Параметр `root_squash` следует указывать для всех экспортируемых файловых систем.

Несмотря на это администратор клиентской машины все равно может выполнять команду **su** на клиентской машине и обратиться на NFS-сервер под любым другим именем, которое есть на клиентской машине. Это не очень хорошо. Но если на сервере доступ ко всем важным файлам открыт только местному, серверному суперпользователю, то вход от имени bin или любого другого пользователя не даст таких возможностей, как обращение от имени root.

Если вы полагаете, что на некой машине в сети работает администратор, которому вы не доверяете, запретите доступ к NFS с той машины вообще. Если на сервере хранятся важные данные, которые не должны

быть доступны всем желающим, запишите их в каталог, который не экспортируется через NFS. Поступайте с важными данными так же, как со своими важными документами. Вы же не оставляете портфель с ними где попало только потому, что кроме нужных бумаг в нем лежат старые проездные билеты и разный хлам?

С помощью разных параметров, описанных в **man nfsd**, можно ограничить доступ к NFS-серверу пользователям с определенными UID и GID.

Параметр `root_squash` установлен по умолчанию для **nfsd** в Linux, поэтому если вы желаете передать администраторские полномочия для доступа к файловой системе, установите параметр `no_root_squash`. Но помните, что лучше этого не делать. Если кто-то взломает сервер, которому вы доверяете, это будет автоматически означать взлом вашего сервера, если этот параметр установлен.

Одна из широко известных уязвимостей **nfsd** состоит в согласии последнего принимать запросы из любых портов клиентского компьютера. Номера портов до 1024 считаются привилегированными, поэтому только root может послать запрос из такого порта. Если **nfsd** принимает запросы из непривилегированных портов, его можно взломать, используя программу-взломщик, доступную в Интернете.

В Linux **nfsd** по умолчанию выполняет проверку порта отправителя. Поинтересуйтесь в **man nfsd**, как включить в **nfsd** разрешение работать только с привилегированными портами.

Кроме обеспечения безопасности **nfsd** надо подумать о защите транслятора портов. Старые его версии содержали ошибку, позволявшую получать файлы с серверов NFS без всяких привилегий.

Старайтесь иметь дело с новыми, официально выпущенными версиями старых добрых программ. Избегайте запускать бета-версии ПО из праздного любопытства на компьютерах, от которых зависит чья-то работа. Обращение к программам, находящимся в стадии разработки, может быть оправдано только крайней необходимостью или желанием провести эксперимент. Крайняя необходимость – это, например, потребность срочно заставить работать новое оборудование, для которого нет стабильных, проверенных драйверов. На свой страх и риск вы можете попробовать бета-версию драйвера. Что до экспериментов, проводите их на специально выделенных компьютерах. Научите этому ваших младших (или старших) коллег!

Достаточно давно был открыт способ взломать систему с помощью транслятора портов. При этом удаленный пользователь мог прикинуться локальным так, что транслятор портов считал его монтирующим локальную файловую систему с локальной же машины. В частности поэтому нельзя экспортировать файловую систему для машины с именем 'localhost' или 127.0.0.1. Впрочем, экспорт локальной файло-

вой системы для локального компьютера является, как нам кажется, разновидностью шизофрении.

Не забудьте отрегулировать доступ к транслятору портов с помощью /etc/ hosts.deny и /etc/hosts.allow.

Помните, что не все дистрибутивы Linux одинаковы. Некоторые дистрибутивы, которые кажутся современными (и даже имеют соответствующие годы выпуска!), не содержат безопасный вариант программы **portmapper** даже сейчас, через много лет после того, как стало известно о ее уязвимостях. В Linux NFS-HOWTO указано, что «по крайней мере один дистрибутив содержит справочную страницу более безопасного транслятора портов, но на самом деле его транслятор портов не является безопасным».

Можно проверить, хорош ли ваш транслятор портов, запустив программу **strings**. Эта программа выдает текстовые строки, содержащиеся в двоичном файле, если длина текста больше или равна 4 символам. Так можно выяснить, есть ли в portmap обращение к файлу hosts.allow и hosts.deny.

```
strings /usr/sbin/portmap | grep hosts
hosts_ctl
```

В моем **portmap** оказалось обращение к функции **hosts_ctl()**, которая обеспечивает правильное восприятие файлов /etc/hosts.deny и /etc/ hosts.allow.

По идее это значит, что можно контролировать доступ к **portmap** через файлы /etc/hosts.allow и /etc/hosts.deny. Однако это надо проверять для каждой конкретной системы. Например, между версиями FreeBSD в какой-то момент состоялся переход от файлов /etc/hosts.deny и /etc/hosts.allow к единственному файлу /etc/hosts.allow. Не исключено, что в результате старые версии portmap в каких-то случаях могут не работать с новым механизмом во FreeBSD. Хотя собственно взаимодействие обеспечивается функцией системной библиотеки **hosts_ctl()**, и главное, чтобы эта функция правильно понимала значение и формат файлов /etc/hosts.deny и /etc/hosts.allow.

Linux NFS-HOWTO предлагает такой порядок редактирования /etc/hosts.deny:

Вначале пишем в нем

```
portmap: ALL
```

Эта команда запретит доступ всем. Затем запускаем **rpcinfo -p**, чтобы проверить, действительно ли транслятор портов читает этот файл и выполняет данные инструкции. Команда **rpcinfo** не должна ничего вывести или должна выдать сообщение об ошибке. Перезапуск транслятора портов не обязателен. Не удивляйтесь, если ничего не получится.

Внимательно изучите документацию по `hosts.allow`. Кроме того, возможно, ваш транслятор портов следует заменить более свежим.

Закрытие доступа к транслятору портов для всех может быть слишком кардинальной мерой, поэтому мы снова откроем доступ (не всем), изменив файл `/etc/hosts.allow`. Там следует перечислить все машины, которые имеют право получить доступ к вашему транслятору портов. Среди множества систем лишь некоторым машинам для нормальной работы нужен полный доступ к транслятору портов. Он обслуживает **nfsd**, **mountd**, **ypbind**/**ypserv** (NIS), **pcnfsd** и 'r'-сервисы, такие как **ruptime** и **rusers**. Из них только **nfsd**, **mountd**, **ypbind**/**ypserv** и, возможно, **pcnfsd** имеют какое-либо важное значение.

Всем машинам, которым необходим доступ к RPC-службам на вашей машине, должно быть разрешено обращаться к транслятору портов. Скажем, адрес машины – 129.240.223.254, она находится в подсети 129.240.223.0 и ей нужен доступ к RPC-службам на вашей машине.

Чтобы разрешить ей этот доступ, пишем в файле `hosts.allow`:

```
portmap: 129.240.223.0/255.255.255.0
```

Это те же самые значения, что и сетевой адрес, который вы даете командой **route**, и маска подсети, которую вы передаете команде **ifconfig**. Для устройства eth0 на этой машине **ifconfig** должна показывать:

```
eth0  Link encap:10Mbps Ethernet  HWaddr 00:60:8C:96:D5:56
  inet addr:129.240.223.254  Bcast:129.240.223.255
                                 Mask:255.255.255.0
  UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
  RX packets:360315 errors:0 dropped:0 overruns:0
  TX packets:179274 errors:0 dropped:0 overruns:0
  Interrupt:10 Base address:0x320
```

а программа **netstat -rn** должна показывать:

```
Kernel routing table
Destination     Gateway Genmask   Flags Metric    Ref  Iface
129.240.223.0   0.0.0.0 255.255.255.0   U 0  0   174412 eth0
```

Адрес сети находится в первой колонке.

В строках для настройки **portmap** в файлах `/etc/hosts.deny` и `/etc/hosts.allow` не следует помещать ничего, кроме IP-адресов. Если туда вписать имена компьютеров, то поиск их имен приведет к получению новых пакетов с ответами программой **portmap**, которая снова станет проверять источники этих пакетов, что вызовет новый поиск имен машин, а это вызовет еще большую активность **portmap** и т. д.

При настройке NFS совместно с фильтром пакетов надо знать, что **nfsd** работает на порту 2049 по протоколам udp и tcp. Транслятор пор-

тов работает на порту 111 по протоколам tcp и udp, а **mountd** работает на портах 745 и 747, протоколы tcp и udp. Это по умолчанию. Номера портов можно проверить командой **rpcinfo -p**.

Для тех, кто хочет использовать NFS сквозь фильтр пакетов, есть в новых версиях **nfsd** и **mountd** ключи, которые заставляют их работать с нестандартными портами. Последние можно открыть в фильтре пакетов.

Окончательный вердикт: безопасность NFS- сервера

Использование файлов `hosts.allow` и `hosts.deny`, параметра `root_squash` и привилегированных портов в `portmapper/nfsd` уменьшает риск взлома через NFS. Настраивая NFS, помните:

- не доверяйте настройкам по умолчанию, проверяйте все;
- не экспортируйте файловые системы с важными данными, если этого можно избежать;
- читайте список рассылки CERT (*www.cert.org*) и новости в группах USENET.

Главное – как всегда, запускайте NFS-сервер только там, где это действительно необходимо!

Настройка NFS-клиента

Для работы в качестве NFS-клиента система должна иметь ядро с поддержкой NFS. Не исключено, что поддержка NFS встроена в ядро, если только вы пользуетесь сравнительно новой системой и не модифицировали ядро после установки. Монтирование удаленной системы выполняется командой **mount**:

```
mount server:/usr/man  /mnt
```

Файловая система, которую компьютер `server` экспортирует как `/usr/ man`, сейчас доступна локально в `/mnt`. Чтение файлов из удаленной системы может быть медленнее, чем чтение локальных файлов, но должно быть быстрее, чем работа через ftp.

Если команда **mount** выдает сообщение об ошибке

```
mount: failed, reason given by  server: Permission denied,
```

это, скорее всего, означает, что в файле `exports` отсутствует разрешение на доступ с этого компьютера или файл ошибочен, или после его модификации не запустили **exportfs**.

Сообщение

```
RPC: Program not registered
```

говорит о том, что **nfsd** или **mountd** (уже) не запущены на NFS-сервере. Возможно, в файлах hosts.{allow,deny} вам запрещен доступ к **portmapper** или **mountd** с этой машины. Для размонтирования файловой системы используйте команду **umount**:

```
umount /mnt
```

Можно автоматически монтировать удаленную файловую систему. Для этого надо добавить в файл /etc/fstab строку, соответствующую нужной файловой системе:

```
# Device      Mountpoint      FStype  Options       Dump Pass
<server>:</shared/dir> </local/mount/point> nfs <options> 0 0
```

Например:

```
server:/usr/man  /mnt     nfs     0 0
```

Параметры, которые можно включить в поле options, определяют поведение NFS. Остановимся на них подробнее.

Клиент NFS может по-разному реагировать на ошибки соединения с NFS-сервером. Если соединение прервалось (завис или неожиданно завершился **nfsd** на сервере, оборвалась связь и т. п.), NFS-клиент может вести себя согласно двум типам поведения: soft и hard.

Типы задаются одноименным параметром в строке монтирования файловой системы:

- soft: NFS-клиент попытается сообщить об ошибке программе, которая через него пытается получить доступ к файлу, расположенному в удаленной файловой системе. Некоторые программы адекватно реагируют на такие ошибки и сами их обрабатывают. Этот тип поведения лучше не задавать, потому что большинство программ реагируют неадекватно, начинают паниковать, и все кончается потерянными данными. Этот параметр ни в коем случае не следует применять для дисков, на которых расположена почта пользователей! Потери почты часто ведут к нервным срывам и другим крупным неприятностям.
- hard NFS-клиент просто приостановит операцию ввода/вывода до восстановления нормальной работы NFS-сервера. Обратившаяся к нему за удаленным файлом программа просто остановится и будет ждать продолжения ввода/вывода. При необходимости вы сможете уничтожить «подвисший» процесс командой **kill**. Рекомендуется устанавливать именно такой параметр при монтировании удаленных файловых систем, потому что такое поведение более удобно при краткосрочных сбоях в сети. Когда сервер NFS будет снова запущен, программа продолжит свою работу с прерванного места.

Запись в /etc/fstab с учетом параметра hard будет выглядеть так:

```
server:/usr/man  /mnt     nfs     hard     0 0
```

Оптимизация работы NFS

По умолчанию NFS-клиент читает и пишет блоками по 4096 или 8192 байта. Некоторые ядра операционных систем и драйверы сетевых карт (или их комбинации) не обрабатывают такие большие блоки как целое, то есть разбивают их на несколько пакетов в зависимости от значения MTU. Стандартное MTU для Ethernet-сетей в Linux и FreeBSD – 1500 байт, такой пакет как раз идеально входит в максимальный по размеру фрейм Ethernet – 1536 байт. Имеет смысл поэкспериментировать и выяснить оптимальные значения параметров `rsize` и `wsize`.

На оптимальный размер влияют два фактора: размер пакета в сети и размер страницы оперативной памяти. Например, в последних ядрах Linux при размере `rsize`, большем или равном размеру страницы памяти, выполняется опережающее кэширование, улучшающее производительность.

Linux NFS-HOWTO предлагает такой метод тестирования производительности записи на удаленный диск на NFS-сервере:

```
time dd if=/dev/zero of=/mnt/testfile bs=16k count=4096
```

Предполагается, что удаленный диск смонтирован в каталог `/mnt`.

По этой команде будет создан файл размером 64 Мбайт, который будет заполнен нулями. Используйте большие значения `count` (количество блоков размера `bs`), если на вашем компьютере достаточно свободной оперативной памяти для кэширования. Размер файла должен превышать размер кэша по крайней мере на NFS-сервере. Данные должны быть записаны на удаленный диск, а не переданы в удаленную оперативную память!

Эту операцию надо проделать несколько раз и усреднить значения, выдаваемые программой **`time`**.

Получив усредненное значение времени записи, скопируйте этот файл на свою машину.

```
time dd if=/mnt/testfile of=/dev/null bs=16k
```

Linux NFS-HOWTO рекомендует и эту операцию повторить несколько раз, но помните о кэшировании файла как кэшем NFS-сервера, так и локальным кэшем своей машины. Файл должен быть обязательно больше размера кэша!

После многократного копирования файла снова усредните полученное значение времени.

Теперь размонтируйте файловую систему и смонтируйте ее с новыми параметрами. Повторите опыт. Изменяйте `rsize` и `wsize`, но в разрешенном диапазоне (обычно от 1024 до 16 384 байт, с кратностью 1024). Имеет смысл посмотреть документацию по **`nfsd`** и **`mount`** в вашей системе, чтобы определить точно допустимые значения. Делайте так до

тех пор, пока не поймете, каково приходится экспертам PC Magazine, когда они тестируют десяток серверов на производительность одним и тем же тестом.

Выберите те значения `rsize` и `wsize`, которые дали лучшую производительность. Возможно, `rsize` будет отличаться от `wsize`.

Есть сведения, что при слишком больших значениях `rsize` и `wsize` вывод некоторых программ (например, **`ls`**) будет частично теряться. Также возможен неполный вывод файлов без сообщений об ошибке чтения.

Несмотря на всю проделанную работу, вы можете убедиться, что заданный по умолчанию размер в 4096 байт – самый лучший. Не обессудьте: вы сами решили все проверить самостоятельно.

Чтобы задать найденные значения `rsize` и `wsize` явно для последующего автоматического монтирования при старте системы, в `/etc/fstab` нужно указать эти параметры:

```
server:/usr/man  /mnt  nfs  hard,rsize=1024,wsize=1024 0 0
```

Параметры в `/etc/fstab` разделяются запятыми.

Для увеличения скорости записи на удаленный диск в NFS можно запретить синхронную запись на сервер. Согласно спецификации NFS необходимо, чтобы запросы NFS на запись не считались законченными до записи данных на диск сервера. Это ограничивает скорость записи. Асинхронная запись значительно увеличивает скорость записи в NFS. Некоторые системы (например, Linux) всегда выполняют асинхронную запись, но в других системах параметр `dir-async` или нечто похожее поможет увеличить скорость записи. Правда, асинхронная запись увеличивает риск потери данных при сбое.

Безопасность клиента NFS

Можно не заботиться о безопасности (почти!), если ваш компьютер не подключен ни к какой внешней сети и никто даже через обычный модем не соединяется с вашей сетью извне. К сожалению, на практике таких систем почти не бывает.

Система NFS несколько наивна: по умолчанию клиент доверяет серверу и наоборот. Необходимо проверить, есть ли у суперпользователя клиента возможность поиграть роль суперпользователя сервера и наоборот. Примите меры (они описаны в предыдущем разделе), если это возможно.

Помните, что если взломали клиентскую машину, то захотят добраться и до сервера. Ставьте заслон не против соседа-администратора, а против того, кто будет взламывать его учетную запись.

Имеет смысл запретить выполнение программ с установленным битом suid в файловой системе NFS, задав при монтировании параметр

nosuid. Сделав так, мы не позволим работающему под именем суперпользователя на сервере установить бит suid на программы в файловой системе сервера, а затем войти на машину клиента под именем обычного пользователя и запустить программу с suid. Потому что если бы он смог это сделать, то приобрел бы права пользователя root на клиентской машине.

Можно запретить выполнение файлов на смонтированной файловой системе с помощью параметра noexec. Его применяют реже, так как файловая система сервера может содержать некоторые скрипты или программы, которые необходимы всем. Помните: команду **mount** может выполнять только администратор.

Кэширование удаленных файловых систем с помощью biod

Демон **biod** (иногда **rpciod**) кэширует удаленную файловую систему, чтобы повысить скорость работы с файлами. Не забудьте разрешить его запуск в стартовых скриптах. Можно также проверить, стала ли работа эффективнее после его запуска. Новые версии клиентов NFS осуществляют такое кэширование автоматически, и специальная программа типа **biod** не требуется.

17

Печать в UNIX

За печать в UNIX отвечают демон **lpd** и сервисные программы **lpr**, **lpq** и **lprm**.

Печатать с UNIX-машин приходится нечасто, поэтому у многих системных администраторов нет опыта работы с системой печати. В европейских и американских университетах встречаются такие сети компьютеров под UNIX, в которых на общий сетевой принтер печатают с многих рабочих станций. В российской практике UNIX-машина скорее будет использоваться как сервер печати для сети Windows-машин. Такую конфигурацию удобнее всего создавать с помощью пакета Samba (см. главу 22).

Для печати файла с UNIX-машины дайте команду **lpr** *имя_файла*.

Для просмотра очереди печати предназначена команда **lpq**, для удаления заданий на печать из очереди – **lprm**.

Если надо печатать на принтер, присоединенный к другому компьютеру, то в файле `/etc/printcap` должны быть указаны имя компьютера и принтера, к которым следует обращаться. Однако локальному пользователю в такой ситуации все равно будет казаться, что печать выполняется на локальный принтер с определенным именем. Можно даже сделать удаленный принтер принтером по умолчанию, если назвать его именем `lp`. Имя принтера по умолчанию можно переопределить, указав новое имя в качестве значения переменной `PRINTER`.

Все доступные принтеры описываются в файле `/etc/printcap`. Здесь для каждого принтера указываются различные параметры. В UNIX нет драйверов принтеров, программы сами должны уметь работать с этими устройствами. В `/etc/printcap` описываются принтеры, установленные в системе. Одному и тому же физическому устройству может соот-

ветствовать несколько виртуальных принтеров из /etc/printcap. Печать по умолчанию производится на принтер с именем lp.

В зависимости от описания фильтра из /etc/printcap программа-спулер **lpd** может запустить для обработки задания один из фильтров (программ предварительной допечатной обработки задания на печать).

Программа **lpd** обращается к printcap каждый раз при необходимости напечатать что-либо, поэтому можно динамически изменять базу данных принтеров printcap.

Одна строка в printcap описывает один принтер.

Пример простого файла printcap:

```
lp:lp=/dev/lp1:sd=/usr/spool/lp1:sf:sh:mx#0
ulp:lp=/dev/lp1:sd=/var/spool/lpd/dj:if=/var/spool/lpd/dj/
filter:sh:pw=80
```

Поля файла printcap отделяются друг от друга двоеточиями. Первое поле – это имя принтера. При печати на принтер с именем, отличным от lp, имя принтера придется явно указывать в строке вызова **lpr**.

В нашем примере описаны два принтера – lp и ulp. Оба описывают одно и то же устройство (файл устройства, указанный в параметре lp, одинаков). Параметр sd определяет каталог, в котором спулер lpd будет сохранять временные файлы.

Параметр sh запрещает печатать служебную информацию вверху каждой страницы. В больших компьютерных залах, где на один принтер печатают десятки пользователей, имеет смысл на каждой странице печатать сведения о том, кто отправил ее на печать. Также в таких случаях удобно печатать страницы-разделители, которые отделяют одно задание на печать от другого. В большинстве случаев в российской практике ни то, ни другое не имеет смысла: редко где много пользователей печатают на один принтер и путают документы. Параметр mx обозначает максимальный размер файла в блоках размера BUFSIZ. Ноль означает «без ограничений». По умолчанию параметр BUFSIZ имеет значение 1000. Параметр pw указывает размер страницы в символах.

Более подробно список параметров можно изучить в **man printcap**.

При печати задание на печать попадает в очередь, которую обрабатывает демон печати **lpd**. Он вызывает специальные программы для обработки файла перед тем, как послать файл на принтер. Эти программы принято называть фильтрами. Фактически **lpd** организует конвейер, через который прогоняется печатаемый файл до того, как попасть в принтер. Какой именно фильтр (или фильтры) будет запущен, зависит от того, какой ключ был передан **lpr** (см. табл. 17.1).

Если устройство печати (параметр lp в описании принтера в файле printcap) указано как port@machine и параметр rm (remote machine)

не указан, то все данные будут напрямую переданы в указанный порт port компьютера machine.

Таблица 17.1. Значения ключей lpr

Ключ	Фильтр	Объяснение
p	pr | if	обычный текст + pr(1)
none	if	обычный текст
c	cf	cifplot
d	df	DVI (tex)
g	gf	plot(3)
n	nf	ditroff
f	rf	Fortran
t	tf	troff
v	vf	растровое изображение

Удаленная машина может быть другим UNIX-компьютером или сетевым принтером (то есть принтером, у которого есть собственный сетевой интерфейс). Параметры ss и sr указывают файл, в который надо сохранять сведения о передаваемых через сеть данных, включая объем задания на печать.

Демон **lpd** записывает протокол печати (включая сообщения об ошибках), используя syslog. Для записи он идентифицирует себя как источник **lpr** и указывает стандартные уровни протоколирования (подробнее об источниках и уровнях в syslog рассказано в главе 29).

При печати текстовых файлов на UNIX-принтере помните, что принтерам часто необходимо в явном виде объяснить, что в конце строки надо выполнить перевод строки и возврат каретки. Если на листе печатается только одна первая строка текстового файла или первые строки «лесенкой», ищите причину в отсутствии возврата каретки. Возможно, надо в описании принтера указать другой фильтр.

18

Время в UNIX

Который час

Чтобы узнать, который час, наберите **date**. Эта команда в UNIX выдает и дату, и системное время. Иногда пользователи или администраторы для получения точного системного времени вводят команду **time**. Это неправильно, так как в UNIX команда **time** не имеет отношения к текущей дате и времени. Она принимает один аргумент, расценивает его как команду (имя выполняемого файла), запускает эту команду и измеряет время ее выполнения. Таким образом, **time** сообщает, как долго выполнялась программа, имя которой ей передали.

Время в UNIX вычисляется в секундах, за точку отсчета принято 1 января 1970 года. Но человек плохо воспринимает время в секундах, поэтому UNIX его переводит в приемлемый формат с помощью специальных функций.

Программы UNIX понимают спецификацию времени в виде смещения от Гринвича в часах, поэтому, например, письма, которые вам присылают из США коллеги, приходят с корректным временем отправления.

Проблема 2000 года не затронула UNIX. Разработчики предвидят уже давно, что проблема, связанная с летоисчислением в UNIX, появится в 2031 году, но к тому времени все перейдут на 64-разрядные версии UNIX, а это отодвинет проблему на многие столетия вперед.

Для установки времени надо дать команду **date** в формате

```
date hhmm # hhmm - часы минуты
```

или

```
date ccyyMMddhhmm.ss
```

где

- *cc* – 19 или 20 (в зависимости от века – двадцатого или двадцать первого)
- *yy* – год (99, если 1999, или 06, если 2006; формат не проживет дольше, чем до 2099 года)
- *MM* – месяц
- *dd* – число месяца
- *hh* – час
- *mm* – минуты
- *ss* – секунды

В различных диалектах **date** может иметь разные ключи и иногда принимать более сложный формат даты (например, 29 Jan 2002).

Синхронизация времени: timed

В компьютерных сетях остро стоит вопрос о синхронизации времени. Дело в том, что на каждом компьютере в сети время идет по-своему. Одни компьютеры имеют точные встроенные часы, другие нет. На одних компьютерах кто-то следит за временем, на других часы предоставлены сами себе, их показания никто не проверяет. Однако время достаточно важно в сети. Например, если вы получаете почту от человека, на компьютере которого часы неверны, то почта может улечься в глубины вашего почтового ящика, ибо многие программы сортируют входящую почту по времени отправки. Представьте себе, куда попадет письмо, помеченное позавчерашним днем! Или позапрошлым годом... Иногда я случайно обнаруживал новое письмо от приятеля, отправленное якобы два года назад.

Время важно и для синхронизации задач в сети. Скажем, будет неприятно, если вы производите удаленное резервное копирование на ленту ночью по расписанию, а компьютер, где стоит стример, считает, что сейчас белый день. Куда хуже, если этот компьютер тоже должен делать резервное копирование на тот же стример, но днем.

Кроме того, не всякое ПО корректно работает с переходом на «летнее» и «зимнее» время, хотя последние версии FreeBSD и Linux справляются с этим без труда. Конечно, разработчикам UNIX трудно уследить за частыми изменениями законодательства о времени в разных странах: в России, например, правила перехода на летнее и зимнее время менялись дважды.[1]

[1] С момента выхода первого издания книги их изменили еще раз. Проконсультируйтесь у юриста, если сомневаетесь, какой именно закон регламентирует переход на летнее и зимнее время. Да, все равно не забывайте следить за временем на ваших серверах, даже если вы давно отвыкли спрашивать «который час?»

Время в UNIX может синхронизироваться с другими компьютерами с помощью демонов времени.

Широко известны два демона – **timed** и **ntpd** (в некоторых диалектах – **xntpd**). Демоны **timed**, запущенные на нескольких компьютерах в сети, общаются между собой и сообщают друг другу текущее время своих систем. Это время усредняется, и **timed** на каждой машине ускоряет или замедляет системные часы так, чтобы приблизить локальное системное время к усредненному. Для общения **timed** использует протокол ICMP.

Работа демона **timed** построена по схеме master-slave (главный-подчиненный), когда одни (или один) timed-серверы в сети считаются главными, а остальные – подчиненными. Например, удобно один – самый быстрый и точный – компьютер в сети синхронизировать каким-нибудь внешним источником точного времени (радиосигналом или показаниями сервера точного времени, полученными демоном **ntpd**). Для всех остальных компьютеров в сети этот главный сервер времени в локальной сети будет служить образцом – они будут сверять по нему свои часы.

При старте **timed** устанавливает часы своей системы по главному timed-серверу в сети, а если таковой не обнаружился, инициирует выборы главного сервера из имеющихся. Периодически **timed** проверяет, доступен ли главный timed-сервер в сети. Если нет, то вновь инициируются его выборы. При помощи ключа **-F** можно указать демону **timed** явно, какой сервер будет считаться главным. Самый главный timed-сервер в сети должен доверять только самому себе.

Поэтому если принята схема с одним главным сервером, на котором запущен **ntpd** для синхронизации с внешним источником точного времени и **timed** для синхронизации времени в локальной сети, то необходимо запретить демону **timed** на таком сервере доверять другим серверам:

```
timed -F localhost
```

При этом **timed** не будет пытаться менять время того компьютера, где он запущен. Это важно, так как два разных демона не должны одновременно пытаться изменить время одной и той же системы. Поэтому на главном сервере времени в вашей сети изменять время будет только **ntpd**, а синхронизировать остальные компьютеры по этому времени будет **timed**.

Для роли главного сервера времени в локальной сети подойдет только машина с минимальной нагрузкой на нее. Имейте в виду, что в большинстве систем сообщения, которые ядро выводит на консоль, вызывают короткий отказ от обработки прерываний. Компьютер, в котором много проблем (например, с оборудованием или настройкой сети), не может корректно вести счет даже своему собственному времени: он слишком озабочен выводом мириад сообщений на консоль. Каждое со-

общение вносит задержку в десяток-другой миллисекунд. Следовательно, в эти миллисекунды не обрабатываются прерывания от таймера.

Демон **timed** способен вносить коррекцию в показания системных часов, вызванных такими задержками и неточностью таймера.

Сообщения в системном файле протокола от **timed** обычно несут следующую информацию:

- **failed to respond – сервер времени, которому доверяет ваш timed**, перестал отвечать (там аварийно завершился timed, оборвалось соединение с сетью, сервер перегружен, сервер перезагружается, питание сервера выключил пьяный монтер, сервер украли);
- **failed to respond to initial time settings – ваш timed**, установленый на машине с несколькими сетевыми интерфейсами, выбрал себе главный сервер в одном сегменте сети, а подчиненным намерен быть в другом.[1]

Управление timed

Уже запущенным демоном **timed** можно управлять с помощью программы **timedc**.

Ключи **timed**:

- **-d** – включает режим отладки, демон не отсоединяется от терминала, как обычно;
- **-i** *сеть* – указанная сеть игнорируется; для того чтобы игнорировать несколько сетей, надо указать этот параметр несколько раз. Речь идет только о сетях, непосредственно присоединенных к данному компьютеру;
- **-F** *host* – список машин, которым **timed** доверяет. Демон **timed** может расценивать как главную только ту машину, которой он доверяет. Параметр **-F** включает в себя параметр **-M**. Если **-F** не указан, то **timed** доверяет всем компьютерам;
- **-M** – разрешает демону **timed** при необходимости стать главным демоном **timed** в сети;
- **-n** *network* – добавить сеть *network* в список допустимых сетей; все неуказанные сети игнорируются; можно опустить этот параметр, тогда все сети будут считаться допустимыми; можно указать

[1] Справедливо считают, что в компьютерных книгах самое интересное пишут в сносках. Эта сноска не исключение. Демоны **timed** используют протокол TSP (Time Service Protocol), который работает через широковещательные пакеты UDP. Эти пакеты распространяются только в пределах одного сегмента сети. Поэтому компьютер с несколькими сетевыми интерфейсами удобно делать главным timed-сервером во всех сегментах, подключенных к нему. Разумеется, бывают исключения, вызванные спецификой конкретной сети.

несколько сетей, тогда надо повторить ключ **-n** для каждой допустимой сети;

- **-t** – включить протоколирование получаемых демоном **timed** сообщений и записывать их в файл /var/log/timed.log. Можно включать и выключать протоколирование посредством управляющей программы **timedc**, когда **timed** уже запущен.

Ключи **-i** и **-n** взаимно исключают друг друга. Оба этих ключа требуют указания вместе с ними сетей, которые на самом деле непосредственно присоединены к компьютеру.

Если **timed** не передан ни один ключ, он будет работать со всеми сетями, к которым присоединен компьютер. Если ни один из ключей **-M** и **-F** не указан, то **timed** всегда будет только подчиненным сервером времени.

Демон **timed**, работающий в качестве главного в сети, называет свой файл протокола /var/log/timed.masterlog.

Синхронизация времени: ntpd

Общие сведения

Кроме **timed** есть более совершенный демон синхронизации времени **ntpd**, его файл конфигурации – /etc/ntp.conf. В Solaris и некоторых других диалектах UNIX этот демон называется **xntpd**. Обратите также внимание на программы **ntptrace** и **ntpq**, которые сопровождают этот демон (их может не быть в вашем дистрибутиве).

Для синхронизации времени в глобальной сети был придуман протокол NTP (Network Time Protocol). Впервые в виде программного обеспечения этот протокол реализовал профессор Делавэрского университета (University of Delaware) Дэвид Миллз (David Mills). Сейчас основной является четвертая версия (NTP v4) этого протокола.

Работа протокола основана на том, что в Интернете существуют «эталонные» серверы времени.

Эталонные серверы делятся на два слоя (stratums): «главные» и «вторичные» серверы. Лучше обращаться к вторичным серверам, так как нагрузка на главные очень велика. В России есть два вторичных сервера:

RU ntp.psn.ru (194.149.67.130)	**RU sign.chg.ru (193.233.46.10)**
Расположение: Пущино, Московская обл., Россия	Расположение: Научный центр в Черноголовке, Московская обл., Россия
Синхронизация: вторичный NTP слоя 2 под управлением Linux на платформе Alpha	Синхронизация: вторичный NTP слоя 2 под управлением FreeBSD 3.1 на платформе PC

RU ntp.psn.ru (194.149.67.130)	RU sign.chg.ru (193.233.46.10)
Обслуживаемая территория: Россия	Обслуживаемая территория: Россия
Политика использования: открытый доступ, перед началом использования предупредите администратора сервера по e-mail о желании синхронизироваться с этим сервером	Политика использования: открытый доступ, перед началом использования предупредите администратора сервера по e-mail о желании синхронизироваться с этим сервером
Адрес для писем: clockmaster@psn.ru	Адрес для писем: Andrew Neporada (andrew@chg.ru) или time@sign.chg.ru

Перед тем как настроить свой сервер для обращения к эталонному, обычно надо написать письмо администратору эталонного сервера о том, что вы намерены воспользоваться последним.

Демон **ntpd** был разработан для синхронизации времени по протоколу NTP. Этот демон синхронизирует системное время с эталонными серверами времени в Интернете. Современная реализация этого демона умеет работать с протоколом NTP версии 4 и обратно совместима с ПО, использующим протоколы NTP версий 3 (RFC 1305), 1 и 2 (RFC1059, RFC1119).

Поскольку нагрузка на серверы времени постоянно растет, так как все больше организаций нуждаются в синхронизации своих сетей с глобальным временем, для оптимизации нагрузки был создан пул серверов времени. Для обращения к случайно выбранному серверу надо в качестве сервера времени в файле `/etc/ntp.conf` указать сервер pool.ntp.org. Когда сервер времени попытается выяснить IP-адрес компьютера pool.ntp.org, ему будет выдан случайно выбранный адрес какого-либо сервера времени. Для более надежного вычисления времени по показаниям вторичных и третичных серверов ntpd обращается к нескольким серверам. Вот пример файла `ntp.conf`, в котором заданы серверы из пула (четыре последних строки действительно должны быть одинаковыми, это не опечатка):

```
driftfile /var/ntp/ntp.drift
server pool.ntp.org
server pool.ntp.org
server pool.ntp.org
server pool.ntp.org
```

Чтобы узнать, какими именно серверами из пула серверов времени фактически пользуется ваш сервер ntp, запускайте **ntpq -p**:

```
ntpq -p
remote          refid        st t when poll reach delay   offset  jitter
xaqua.          morcic.RI.   3  u 489  1024 377   575.771 -216.95 233.014
subnet.at       CARNe
+pluto.         sombrero.    2  u 548  1024 377   50.858  8.973   443.790
fips.at         cs.tu-
```

```
+rainbow.    ns4.univie.  3  u 511  1024 377   136.659 13.141  25.597
bksys.at     ac.at
*hostmaster  hora.cs.     2  u 504  1024 377   73.758  16.704  5.571
.org         tu-berli
```

Значения столбцов:

- `remote` – сервер времени, к которому обращается наш **ntpd**
- `refid` – сервер времени, с которого забирает время сервер, указанный в колонке remote
- `st` (stratum) – слой, к которому относится сервер remote
- `t` – тип сервера (local, unicast, multicast или broadcast)
- `when` – когда время было получено последний раз
- `poll` – интервал опроса в секундах
- `reach` – регистр доступности (восьмеричное значение)

  и далее, в миллисекундах:
- `delay` – текущая ожидаемая задержка передачи пакета
- `offset` и `jitter` – расхождение и дисперсия

Первый символ перед именем сервера, с которого наш **ntpd** заберет показания времени, означает состояние этого сервера. Знак * указывает на сервер, используемый в настоящий момент. Проследить, из какого первичного источника наш **ntpd** получает время, можно с помощью программы **ntptrace**:

```
ntptrace
localhost: stratum 3, offset 0.000013, synch distance 0.31493
hostmaster.org: stratum 2, offset 0.013122, synch distance 0.06889
hora.cs.tu-berlin.de: stratum 1, offset 0.013267, synch distance
0.00034, refid 'PPS'
```

Другие знаки показывают, что:

- `пробел` – сервер недоступен или его данные отвергнуты по иной причине
- `x` – данные сервера в данный момент считаются недостоверными
- `.` – с сервером трудно синхронизироваться
- `-` – сервер не будет использован при следующем опросе
- `+` – сервер жив и будет использован при следующем опросе
- `*` – время этого сервера используется для установки в настоящий момент локального времени

Состояние серверов постоянно изменяется в зависимости от загрузки сети и от нагрузки на сами серверы.

Свежий список NTP-серверов с открытым доступом к ним можно получить по адресу *http://www.eecis.udel.edu/~mills/ntp/servers.htm*. Подробности о службе NTP и новых версиях связанного с NTP программного обеспечения читайте по адресу *www.ntp.org*.

Алгоритм работы ntpd

Алгоритм работы основан на том, что в мире существуют эталонные источники времени: цезиевые часы или радиосигналы точного времени со спутников. Такие эталонные источники в терминах NTP входят в слой 0 (stratum 0) и являются максимально точными из доступных эталонов времени. Те компьютеры, к которым непосредственно присоединены эталоны слоя 0, представляют собой серверы точного времени слоя 1 (stratum 1). Работающие на них демоны **ntpd** распространяют точное время дальше, серверам слоя 2 (stratum 2). Именно к этим серверам обращаются многие обычные серверы точного времени в локальных сетях организаций. Таким образом, серверы времени **ntpd** в наших локальных сетях образуют слой 3. Номер слоя фактически означает дистанцию (количество промежуточных серверов времени) от данного компьютера до эталонного источника точного времени.

Каждый NTP-сервер расценивает один или несколько источников точного времени как заслуживающие доверия. Они определяются в файле `/etc/ntp.conf`. При старте **ntpd** запрашивает их о текущем времени.

Каждый из NTP-серверов опрашивается несколько раз для получения достоверной статистики задержки передачи пакета. Если указано несколько серверов, каждый из них опрашивается многократно. Обычно тратится по 5 минут на каждый сервер для получения показаний времени, которые можно считать точными.

Запрос выглядит следующим образом: клиент вкладывает в пакет свое текущее системное время и посылает этот пакет серверу как запрос о текущем времени. Сервер вкладывает в пакет время передачи пакета, вычисленное как разность между временем отправки и временем получения пакета сервером. Затем он вкладывает туда свое текущее время и отправляет пакет обратно. Клиент запоминает свое время получения пакета. Таким образом, клиент может рассчитать задержку передачи пакета как половину от «времени передачи туда и обратно минус время обработки пакета сервером».

Если разница между эталонным и текущим системным временем не превышает 128 мс (миллисекунд), то **ntpd** подстраивает частоту системных часов так, чтобы они догнали или затормозились до эталонного времени. Подстройка происходит медленно и плавно. Например, для Linux стандартная скорость подстройки составляет 0,5 миллисекунды в секунду.

Если разница превышает 1000 секунд, то **ntpd** полагает, что имеет место фатальный сбой времени на локальной машине, и завершается с предсмертным воплем, который записывается в `/var/log/messages`.

Если разница больше 128 мс, но меньше 1000 секунд, то **ntpd** по умолчанию выставляет время сразу же, просто вызывая функцию `settimeofday()`.

Коррекция времени, как можно видеть, может быть постепенной или резкой. При резкой коррекции системное время изменяется скачком. Это можно запретить, если указать при запуске **ntpd** ключ **-x**. Не рекомендуется изменять время резко, особенно в тех случаях, когда оно изменяется «назад», то есть текущее системное время до изменения оказывается большим, чем после него (часы спешат).

При передаче информации **ntpd** использует UDP-порт с номером 123.

Управление ntpd

Демон **ntpd** при старте читает файл `/etc/ntp.conf` или принимает параметры в командной строке. Главное, что ему надо, – это знать его эталонные серверы и режим работы.

Если файл `/etc/ntp.conf` не удается прочесть и посредством параметра **-c** не указан никакой другой файл конфигурации, то **ntpd**, если он был скомпилирован с поддержкой NetInfo, прислушивается к широковещательным сообщениям в сети и может сам найти себе эталон времени.

Можно получить данные о конфигурации и изменить настройки **ntpd** на лету, используя программы **ntpq** и **ntpdc**.

При старте **ntpd** меняет значение `umask` на 022, если оно было нулевым.

В **ntpd** доступны следующие ключи:

- **-a** – включить аутентификацию (по умолчанию);
- **-A** – выключить аутентификацию;
- **-b** – синхронизироваться по широковещательным сообщениям;
- **-c** *conffile* – использовать указанный файл вместо `/etc/ntp.conf`;
- **-d** – включить режим отладки; служебная информация выводится на экран; может использоваться вместе с **-q**;
- **-k** *keyfile* – указывает файл с ключами и прочей информацией для аутентификации в NTP;
- **-l** *logfile* – указывает файл протокола; по умолчанию протоколирование ведется через syslog;
- **-L** – слушать виртуальные IP-адреса;
- **-m** – синхронизировать часы по широковещательным (multicast) сообщениям NTP на группу multicast-адресов 224.0.1.1; требует поддержки multicast в ядре;
- **-n** – не запускать дочерние процессы (don't fork);
- **-N** *priority* – запускаться с определенным приоритетом;
- **-p** *pidfile* – указывает файл, в который записывается PID процесса **ntpd**;

- **-q** – установить время и завершиться, подобно **ntpdate**. Может использоваться вместе с **-g** и **-x**;
- **-x** – всегда корректировать время плавно независимо от срока отставания или опережения эталонных часов системными.

Дополнительную информацию по **ntpd** (**xntpd**) можно получить в **man**.

Файл ntp.conf

Файл /etc/ntp.conf содержит параметры настройки демона **ntpd**.

В этом файле команды располагаются по одной в строке, за ключевым словом команды могут идти аргументы. Строки, начинающиеся символом «#», представляют собой комментарии. Пустые строки игнорируются. Аргументы отделяются от команды и друг от друга пробелами.

Аргументы могут быть именами компьютеров или IP-адресами, а также спецификацией времени (в формате числа с плавающей точкой) и текстовой строкой.

Детали конфигурации можно изучить по **man ntp.conf**. Мы рассмотрим один из возможных примеров конфигурации:

```
server ntplocal.example.com prefer
server timeserver.example.org
server ntp2a.example.net
driftfile /var/db/ntp.drift
```

Параметр server указывает, какие серверы будут использоваться по одному в каждой строке. Если один из серверов указан с аргументом prefer, как ntplocal.example.com в нашем примере, то этому серверу отдается предпочтение перед остальными. Ответ от предпочитаемого сервера будет отброшен, если он значительно отличается от ответов других серверов. В ином случае он будет использоваться вне зависимости от других ответов. Аргумент prefer обычно указывается для серверов NTP, о которых известно, что они очень точны. Например, серверам из слоя 2 имеет смысл отдавать предпочтение перед сервером вашего провайдера или главного офиса компании. Впрочем, если ваша компания живет по собственному времени, ее главный сервер времени стоит считать самым предпочтительным.

Параметр driftfile задает файл, в котором надо сохранять смещение частоты системных часов. Демон **ntpd** использует его для автоматической компенсации естественного смещения часов. При этом обеспечивается достаточно правильная настройка часов, даже если компьютер на некоторое время отключается от внешнего источника информации о времени.

В файле, указанном в параметре driftfile, хранится информация о предыдущих ответах от серверов NTP. Этот файл содержит внутреннюю информацию для NTP. Он не должен изменяться никакими другими процессами.

В файле ntp.conf также задаются параметры, управляющие доступом к вашему серверу.

По умолчанию сервер NTP будет доступен всем хостам в Интернете. Параметр restrict позволяет ограничить количество машин, которые могут обращаться к вашему серверу.

Для того чтобы запретить всем машинам обращаться к вашему серверу NTP, добавьте строку

```
restrict default          ignore
```

в файл /etc/ntp.conf. Чтобы разрешить синхронизировать свои часы с вашим сервером только компьютерам вашей сети, но запретить им настраивать сервер, вместо указанной добавьте строчку

```
restrict 192.168.1.0 mask 255.255.255.0 nomodify
```

Здесь 192.168.1.0 – это ваша сеть, а 255.255.255.0 – ее сетевая маска.

Файл /etc/ntp.conf может содержать несколько директив restrict. Дополнительную информацию о возможностях управления доступом можно прочесть в разделе Access Control Support в **man ntp.conf**.

Чтобы NTP-сервер загружался при старте системы, занесите его запуск в стартовые скрипты. Например, во FreeBSD для этого достаточно немного модифицировать /etc/rc.conf:

```
xntpd_enable="YES"
```

Для указания дополнительных ключей и параметров для **ntpd** можно использовать переменную xntpd_flags:

```
xntpd_flags="-c /some/other/configfile -d /some/driftfile"
```

Если NTP-сервер только что установлен или настроен, то для его запуска без перезагрузки машины надо выполнить команду **ntpd**, не забыв задать дополнительные параметры из переменной xntpd_flags в файле /etc/rc.conf. К примеру:

```
ntpd -c /some/other/configfile -d /some/driftfile
```

Практические советы

Если надо синхронизировать часы только при загрузке, то можно использовать для этого программу **ntpdate**. Это удобно для машин, на которых запускается **ntpd**. Демон **ntpd** изменяет время постепенно, а **ntpdate** устанавливает время вне зависимости от того, насколько велика разница между текущим временем машины и точным временем. Если разница между эталонным и текущим системным временем меньше 0,5 секунды, то **ntpdate** вызывает функцию **adjtime()** для

постепенной коррекции времени. Если больше, то он вызывает функцию **settimeofday()** для директивной установки времени.

Во FreeBSD для запуска **ntpdate** при старте системы добавьте строчку

```
ntpdate_enable="YES"
```

в файл /etc/rc.conf. При этом обязательно надо указать тот сервер, с которым надо синхронизировать время, и другие параметры в переменной ntpdate_flags. Установите ее в файле /etc/rc.conf:

```
ntpdate_flags="-b -s your.ntp.server.ru"
```

Ключ **-b** требует установить время с помощью функции settimeofday() независимо от разницы между системным и эталонным временем. Ключ **-s** перенаправляет протоколирование со стандартного вывода в syslog.

Программу **ntpdate** можно вызывать не только при загрузке системы. Если в процессе работы системные часы отклоняются не сильно, а высокая точность (30 мс или выше) не нужна, то достаточно раз в два часа запускать **ntpdate** для синхронизации с эталонным сервером из **cron**:

```
1 */2 * * * root /usr/sbin/ntpdate -b -s my.timeserver.IP
```

Дополнительная литература

[9] Документация по NTP *http://www.eecis.udel.edu/~ntp/ntpfaq/NTP-s-algo.htm* или *http://www.ntp.org*

19

Установка и настройка почтового сервера

Что такое почтовый сервер

Почтовый сервер представляет собой совокупность программ, которые выполняют всю работу с почтой: получают ее от программ-клиентов, обрабатывают, отправляют к месту назначения, принимают от других почтовых серверов, отдают программам-клиентам по их запросу.

Программы, обеспечивающие работу почтового сервера, выполняют следующие роли: пользовательский агент (Mail User Agent – MUA), сервер передачи почты (Mail Transfer Agent – MTA) и сервер выдачи почты (Mail Access Agent – MAA). Какие именно программы могут выполнять эти роли, мы рассмотрим ниже.

Современный мир предъявил довольно высокие требования к способностям почтовых серверов. Так, пользовательский агент, программа, позволяющая человеку принимать и отправлять почту, должна быть чрезвычайно дружелюбна, понимать пользователя с полуслова или хотя бы с «полуклика» мышью. Современный пользовательский почтовый агент часто интегрирован в какую-нибудь многофункциональную среду и дает возможность не только использовать письма в этой среде, но и эффективно управлять ими: раскладывать по папкам, сортировать по дате, размеру, имени отправителя и другим параметрам, которые только могут прийти в голову.

А программы передачи почты благодаря массовым рассылкам вирусов и спама пришлось нагрузить задачей, которая и вовсе не свойственна почтальонам: они блокируют рассылку явного спама, анализируют все письма на предмет похожести на спам и взаимодействуют с анти-

вирусными службами на почтовом сервере для удаления вирусов из писем. Если бы обычные почтальоны умели столько же, они стали бы нашими лучшими друзьями: перед тем как положить корреспонденцию нам в ящик, они удаляли бы из нее бесполезные рекламные проспекты, надоедливые газеты бесплатных объявлений, чудовищные купоны со скидками на ненужные товары и, самое главное, выкидывали бы письма, содержащие споры сибирской язвы и прочих несимпатичных возбудителей заразы. Ниже мы более подробно рассмотрим функции программ передачи почты и конкретные примеры настроек этих функций в программах sendmail и postfix.

Серверы выдачи почты за последние годы тоже расширили спектр своих умений. Хороший MAA умеет не только проверять имя и пароль пользователя и выдавать ему почту, но еще и шифровать передаваемые письма, управлять почтовым ящиком, поддерживать в этом ящике древовидную структуру вложенных друг в друга папок и т. д.

В этой главе вплоть до раздела «Установка и настройка sendmail» описывается теория работы почтового сервера – что он должен уметь и как это делается. Если сухому языку теории вы предпочитаете привлекательный язык примеров, переходите сразу к «Установке и настройке sendmail» (или «Установке и настройке postfix», если вы сделали именно такой выбор сервера передачи почты).

На рис. 19.1 изображена схема передачи письма от отправителя к получателю. Заметьте, что программа – пользовательский агент получателя – не должна быть запущена в момент прихода письма. Непосредственно между собой взаимодействуют только MUA и MTA отправителя, а также MTA отправителя и MTA получателя.

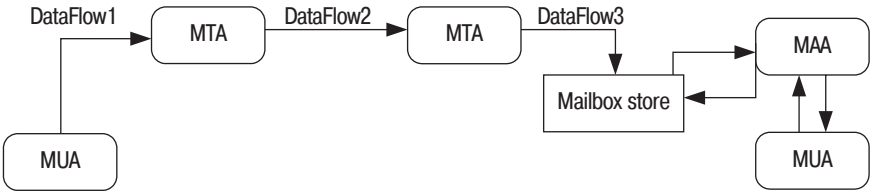


Рис. 19.1. *Схема передачи письма*

Задачи сервера передачи почты (MTA)

Каждый почтовый сервер обслуживает пользователей одного или нескольких доменов. Говорят, что он является релеем (relay) для этих доменов. Термин «relay» можно было бы перевести как «промежуточный почтовый сервер», но в разговорном языке администраторов устойчиво употребляется калька «релей», и мы пока нарушать традицию не будем.

Основная задача сервера передачи почты в отношении писем, направленных в его «родной» домен, т. е. домен, для которого он является ре-

леем, – принимать электронные письма и раскладывать их по почтовым ящикам получателей (пользователей). Если письмо адресовано пользователю домена, для которого данный почтовый сервер релеем не является, то его надлежит отправить тому серверу, который является таким релеем.

Однако кроме собственно передачи почты сервер может выполнять ряд вспомогательных, но очень важных функций: обеспечивать защиту от спама, от вирусов, перенаправлять письма по другим адресам, если это задано конфигурацией, передавать для обработки письма заданного типа определенным программам, блокировать пересылку писем, адресованных с заданных адресов или в такие адреса. Самое интересное, конечно, это как можно бороться со спамом. На сегодняшний день придуманы такие способы:

- «черные списки» IP-адресов тех почтовых серверов, которые уже известны, как распространители спама (blacklists);
- «черные списки» IP-адресов почтовых серверов, позволяющих передавать через себя письма, адресованные не в их домен не из их домена (например, почтовый сервер домена spam.yes.com пересылает письма, адресованные из домена spam.ru в домен yahoo.com, а должен бы пересылать письма, адресованные только в его домен spam.yes.com либо из его домена по любым адресам); такие «добрые» почтовые серверы называются «открытыми релеями» (open relays);
- «черные списки» потенциально опасных IP-адресов, например, адресов модемных пулов крупных интернет-провайдеров; такие адреса назначаются обыкновенным пользовательским машинам, которые, если станут жертвой вируса, могут превратиться в открытые релеи и начать пересылать спам, подчиняясь чужой злой воле;
- «серые списки» – этот механизм реализуют самые современные программы борьбы со спамом;
- отслеживание всей цепочки почтовых серверов, через которые прошло письмо (обычно почтовый сервер проверяет на присутствие в черном списке только сервер, непосредственно открывающий соединение с ним);
- контент-анализ текста письма, включая распознавание слов в графических изображениях;
- контент-анализ заголовков (Subject, From и To) на предмет поиска характерных для спама шаблонов.

Задачи почтового клиента (MUA)

Почтовых клиентов немало. «По науке» почтовый клиент называется MUA (Mail User Agent). Наиболее известные для UNIX – это pine, elm, mail. Для Windows – TheBat!, Mozilla, Thunderbird, Eudora, Microsoft Outlook, Outlook Express, Microsoft Exchange Client. Из всей этой ком-

пании можно рекомендовать либо TheBat!, либо Microsoft Outlook – наиболее надежные и универсальные программы. Разумеется, есть еще встроенные в корпоративные системы почтовые клиенты – для Novell Groupwise или Lotus Notes, но их бессмысленно использовать отдельно от их серверной части. Последние версии Thunderbird говорят хорошо и об этом продукте – вы можете выбрать его, если хорошо относитесь к разработкам Mozilla.

Почтовый клиент выполняет несколько функций:

- редактирование почтовых сообщений;
- организация хранения почтовых сообщений, в том числе поддержка иерархической древовидной системы папок; в случае поддержки протокола IMAP такое папкохранилище для большего удобства можно устроить на сервере; почтовые клиенты, умеющие работать с Microsoft Exchange Server, позволяют это делать с помощью специфического для этого продукта протокола;
- отправка почтовых сообщений по одному из распространенных протоколов передачи почты, в подавляющем большинстве случаев – SMTP;
- прием почтовых сообщений от почтового сервера посредством взаимодействия с агентом выдачи почты; для реализации этой функции почтовые клиенты умеют работать с поддержкой протоколов POP3 или IMAP4

Почтовые клиенты могут иметь как текстовый, так и графический интерфейс; самый простой почтовый клиент под UNIX – программа mail, более сложные, но тоже имеющие текстовый интерфейс – elm и pine. В отличие от mail, они обычно не устанавливаются в момент установки системы.

Задачи сервера выдачи почты (MAA)

Упомянутые MUA (TheBat!, Eudora, Microsoft Outlook и др.) умеют обращаться к серверам выдачи почты по протоколам POP3 (Post Office Protocol, version 3) и IMAP4 (Internet Message Access Protocol, version 4).

Мы будем употреблять термин «клиент» в значении «MUA», говоря о передаче почты. Термин MAA (Mail Access Agent) обозначает сервер, умеющий обрабатывать запросы клиентов, которые желают получить почту, хранящуюся на компьютере, который является сервером почты (принимает и хранит ее).

Сервер выдачи почты должен по запросу от почтового клиента вынимать из почтового ящика на сервере сообщения и передавать их клиенту. Дополнительно он может обслуживать древовидную структуру почтовых папок (folders), обеспечивая раскладку писем из папки Inbox в другие папки, если это поддерживается программой-клиентом, а также удалять сообщения из почтового ящика. Сервер выдачи почты

обычно поддерживает передачу клиенту как сообщений целиком, так и только заголовков сообщений – для того, чтобы пользователь мог выбрать, что он хочет удалить сразу, а что – получить на свой компьютер и спокойно изучить. Это спасает от лишних затрат времени и денег тех, кто вынужденно получает спам, например, сотрудников пресс-служб, отделов кадров, учебных отделов и прочих, чьи адреса широко опубликованы или легко вычисляемы. Спросите тех, кто читает почту по адресу *job@well-known.com*: не приходится ли им ежедневно выгребать завалы спама из своей папки Inbox?

Серверы передачи почты работают по протоколу IMAP4 или POP3. Протокол IMAP4 часто называют протоколом IMAP, а POP3 – просто POP. Поскольку другие версии этих протоколов ныне не в ходу, это упрощение легко принять.

Наиболее известные и распространенные серверы POP3 – это pop3d и qpopper (распространяются в составе практически любого дистрибутива UNIX), cucipop и cyrus. Последний фактически является большим пакетом, хорошо интегрируемым с postfix, и в него входит также IMAP-сервер.

Семейство IMAP-серверов скромнее: наиболее известны всего два варианта. Первый – wu-imapd – IMAP-сервер в версии Вашингтонского университета (University of Washington). Сайт этого продукта – *http://www.washington.edu/imap/*. Второй – Cyrus, адрес этого проекта в сети – *http://asg.web.cmu.edu/cyrus/imapd/*.

Какой сервер выдачи почты предпочесть? Какой протокол выбрать?

Отличия протоколов IMAP4 и POP3

IMAP4 отличается от POP3 тем, что обеспечивает:

- поддержку общих почтовых ящиков, включая уведомление о том, что содержимое ящика изменилось с момента предыдущего обращения;
- уведомление о приходе новой почты;
- поддержку нескольких почтовых ящиков для одного пользователя;
- управление удаленными почтовыми ящиками;
- возможность создания иерархического дерева почтовых ящиков;
- возможность не только скачать письмо из удаленного почтового ящика, но и добавить в него письмо;
- возможность устанавливать специфические флаги для каждого сообщения (типа «important», «answered»);
- просмотр структуры сообщения без скачивания (проверку на наличие приложенного файла и т. п.);
- скачивание сообщения частями (например, если вы хотите получить один прикрепленный файл, а другой не хотите);

- поиск в почтовом ящике на сервере (это позволяет избежать скачивания всех сообщений клиентом для поиска в них).

В 2003–2004 годах появились или развились до уровня индустриального стандарта несколько хороших почтовых клиентов, работающих с IMAP, такие как Mozilla Thunderbird, TheBat! и Microsoft Outlook. Благодаря этому использование IMAP-серверов в качестве хранилищ значительных объемов корпоративной почты стало еще более оправдано, чем раньше. Серверы выдачи почты по POP3 сейчас сохраняются в тех организациях, где хранение почты на сервере не требуется и, более того, нежелательно. Например, сервис POP3-сервера обычно предлагают интернет-провайдеры, которым сам бог велел заботиться не о содержимом почтовых ящиков пользователей, а о его регулярном скачивании последними – за деньги, разумеется.

Таким образом, IMAP следует предпочесть там, где важнее организация структурированного хранилища почты на сервере, POP3 – там, где важнее совместимость со старыми почтовыми программами-клиентами, и там, где хранение почты на сервере не является приоритетной задачей.

Более того, если вы – провайдер, то следует избегать хранения почты клиентов на своем сервере, поэтому надо применять такие программы, которые могут принудительно удалить сообщение, как только клиент его получит. Это, например, умеет делать cucipop.[1] Остальные серверы выдачи почты вежливо ждут, пока программа-клиент сама попросит удалить уже полученное ею письмо, а будет ли она это делать, зависит только от ее настроек. Много ли вы видели провайдеров, которые хотят попасть в такую зависимость от покупателей своих услуг?

Если же вы – администратор большой сети, где пользователи часто пересаживаются с места на место, хранение почты на сервере может быть чрезвычайно удобной альтернативой перемещаемым профилям, в которых хранится база сообщений владельца профиля.[2]

Корпоративные заказчики, желающие заплатить за IMAP-сервер деньги, могут обратиться, например, на сайт *http://www.gordano.com/* – там их осчастливят за несколько сотен долларов – минимум.

---

[1] Это полезная функция, но ее применение нарушает RFC1939. Если это единственное соглашение (закон, инструкция, предписание), которые вы готовы нарушить, задумайтесь, следует ли начинать скользкий путь нарушений именно с него? Может быть, выбрать что-то повыгоднее?

[2] Профиль пользователя (user profile) и перемещаемый профиль (roaming profile) – это принятый в Windows-системах механизм хранения настроек пользователя в домене. Применение этого механизма удобно для пользователя, но нередко приводит к увеличению объема работы для администратора сети и иногда требует полного удаления профиля из-за возникновения непреодолимых сбоев в его внутренней структуре. Разумеется, под этим мнением не станут подписываться маркетологи Microsoft, но практика многих администраторов подтверждает его. – *Примеч. авт.*

Есть распространенное заблуждение: если что-то дорого стоит, то оно хорошее. Эта мысль родилась у честных предков нынешнего человечества, которое постоянно сталкивается с обманом. Иначе зачем бы человечеству содержать штат адвокатов, судей, присяжных, обществ защиты разных прав и т. п.? Покупая любой программный продукт, знайте, что в мире UNIX, как правило, есть аналогичный бесплатный товар не хуже качеством. Бывают исключения, традиционно подтверждающие правило.

Более подробную информацию о протоколе IMAP можно получить на *imap.org*.

Для получения писем клиентом от сервера годится и POP3, и IMAP4. Если вам проще работать с POP3, а дополнительные возможности вас не интересуют, будьте непреклонны. Потом расскажете внукам, как отбивались от просьб настойчивых пользователей установить IMAP-сервер.

Всегда помните о том, что выход есть из самых трудных ситуаций, например, на одном компьютере легко могут уживаться и POP3-, и IMAP4-сервер.

Установка и начальная настройка sendmail

Многие версии UNIX поставляются с предустановленным и во многом настроенным демоном sendmail, либо при установке системы программа-установщик выполняет настройку этого демона, исходя из сведений, которые вы ей сообщите.

Счастливые обладатели таких поставок системы, которых удовлетворяет простая конфигурация их почтовой службы, могут смело пропустить этот раздел. Для остальных разберем подробнее, как устанавливать sendmail и что в нем настраивать.

Программа sendmail компилируется точно так же, как любая другая программа под UNIX. То есть вначале вы внимательно читаете файлы `INSTALL` и `README` из дистрибутива, а затем честно выполняете то, что там написано. Это – если дистрибутив получен в исходных текстах. Если это не так, зайдите за исходными текстами на *www.sendmail.org*: ими там снабжают всех желающих совершенно безвозмездно.

После компиляции программы в ней уже содержатся все мыслимые фунциональные возможности.[1] Теперь остается сообщить ей, какие из них мы желаем задействовать. Это делается посредством создания

[1] Здесь мы рассматриваем функциональные возможности на примере sendmail 8.12.11, предыдущие версии обладали меньшей функциональностью, а последующие гарантированно будут включать в себя больше механизмов, которые можно будет активировать командами в файле конфигурации. – *Примеч. авт.*

файла конфигурации, в котором указывается обязательная настроечная информация, например, имя нашего компьютера, имена файлов конфигурации с нашими указаниями, от кого и для кого можно принимать почту, а для кого – категорически нет (/etc/mail/access), а также необязательная информация: куда следует обращаться за «черным списком» спамеров, какие внешние модули мы собираемся задействовать для вылавливания спама и вирусов и т. п.

Файл конфигурации **sendmail** исторически создавался так, чтобы он был понятен программе и непонятен простому смертному. Поэтому разобраться в нем нелегко. Обычно даже опытные администраторы используют вспомогательный файл настроек-макросов для того, чтобы из него потом сформировать файл sendmail.cf, главный конфигурационный файл **sednmail**.

Файл настроек-макросов с расширением .mc представляет собой заготовку, которую надо обработать с помощью макропроцессора **m4** для превращения в файл sednmail.cf. Пример файла .mc легко найти в дистрибутиве **sendmail**. Вот такой файл myhost.mc требуется для реализации необходимого минимума функций современной почтовой системы:

```
divert(0)dnl
OSTYPE(bsdi2.0)dnl
DOMAIN(generic)dnl
FEATURE(mailertable, hash -o /etc/mail/mailertable)dnl
FEATURE(domaintable, hash -o /etc/mail/domaintable)dnl
FEATURE(access_db)dnl

MAILER(local)dnl
MAILER(smtp)dnl
MAILER(uucp)dnl
```

Первые три строки описывают самые общие свойства почтовой системы, и в данном случае говорят о том, что ей следует себя вести стандартным для систем BSDI способом, соблюдая принятые в таких системах соглашения. Этот вариант прекрасно подходит и для систем FreeBSD.

Строки, начинающиеся с ключевого слова FEATURE, описывают дополнительную функциональность, которая придается почтовой программе. В данном примере использованы макросы, разрешающие работу с таблицами назначения подпрограмм-почтальонов для конкретных доменов (mailertable и domaintable) и таблицей явных запретов и разрешений на прием почты с указанных адресов (access_db).

Строки, начинающиеся с MAILER, включают в почтовой программе соответствующие подпрограммы, являющиеся почтальонами-приемщиками и доставщиками почты по указанным протоколам (local – локальная доставка, smtp – работа с другими почтовыми серверами по протоколу SMTP, uucp – обмен почтой по протоколу UUCP).

Однако для расширения функциональности имеет смысл добавить сюда ряд новых возможностей:

```
FEATURE(virtusertable, hash -o /etc/mail/virtusertable)dnl
FEATURE(`enhdnsbl',`relay.njabl.org',`',`t',
`127.0.0.2.')dnl
```

Таблица виртуальных пользователей позволит вам переадресовывать почту, пришедшую в адрес определенных пользователей вашего домена, на другие адреса. Это может быть удобно для сохранения старого адреса за сотрудником, ушедшим работать в другую организацию и фактически читающим свою почту с другого почтового сервера, а также для создания служебных адресов типа abuse@coolfighter.com, письма на которые на самом деле должны попадать по адресу nobody.answers@da.nu.ee. Формат вышеописанных таблиц прост и интуитивно ясен, например, скажем, таблица виртуальных пользователей, согласно приведенному макросу FEATURE, должна находиться в файле `/etc/mail/virtusertable` и содержать строку:

```
abuse@coolfighter.comnobody.answers@da.nu.ee
```

Самый интересный макрос из перечисленных – enhdnsbl, это включение расширенной проверки письма на спам. Такая проверка предполагает обращение к черному списку IP-адресов компьютеров, известных как распространители спама. Если IP-адрес почтового сервера, обратившегося к нашему, входит в черный список, письмо будет отвергнуто без колебаний. У этой проверки есть два недостатка: во-первых, проверяется только тот сервер, который непосредственно обратился к нашему. Если спамерский сервер переслал письмо «чистому» почтовому серверу, который «не засвечен» в черном списке, а уже этот последний сервер обратился к нам, то наш сервер письмо примет. Второй недостаток связан с технологией внесения в черные списки и состоит в том, что если «чистый» почтовый сервер попал в черный список по ошибке, то пока он не будет из него исключен, письма с него не примет ни один почтовый сервер, читающий этот черный список.

Хранители черных списков, например, широко известный сервис SpamCop (*http://www.spamcop.net*) предлагают во избежание потерь важных писем, распознанных почему-либо как спам, не блокировать прохождение таких писем, а помечать их определенным заголовком (тэгом, от слова tag), чтобы пользователи имели возможность с помощью своих почтовых клиентов удалять их или помещать в дальний угол своего почтового ящика. На практике такая пометка – достаточно редкое явление, но в ближайшее время можно ожидать, что подавляющее большинство почтовых серверов будет ее выполнять.

Встает вопрос, откуда же нам узнать, какие серверы черных списков следует использовать? Адрес `relay.njabl.org` стал известен нам из этой книги, но, наверное, есть и другие источники? В следующем разделе обсуждается и этот вопрос.

Сейчас вернемся к тому, что следует сделать с файлом myhost.mc после того, как в него добавлены все необходимые макросы. Его надо превратить в файл sendmail.cf. Это делается с помощью команды **m4 myhost.mc**. По этой команде макропроцессор **m4** развернет макросы, внесенные в myhost.mc в длинные описания, понятные программе **sendmail**, и в том же каталоге, где лежит myhost.mc, появится файл myhost.cf. Для того чтобы превратить его в действующий файл конфигурации **sendmail**, его надо скопировать в /etc/mail и там переименовать в sendmail.cf. Конфигурация готова. Можно запускать sendmail. Для запуска в режиме демона применяйте команду

```
/usr/sbin/sendmail -bd -q30m
```

Параметр q задает период обработки очереди, его можно задать иным, чем 30 минут, но не следует делать его слишком коротким, чтобы не добавлять излишней нагрузки серверу.

При создании файла .cf необходимо, чтобы макропроцессор **m4** «увидел» файл макроопределений, позволяющий раскрыть макросы из файла .mc в файл конфигурации .cf. Для этого удобно файл .mc создавать в дереве файлов дистрибутива **sendmail** в каталоге sendmail-8.12.11/cf/cf (подставьте номер вашей версии вместо 8.12.11) и **m4** запускать в этом же каталоге.

О настройке **sendmail** здесь рассказано очень коротко – ровно столько, чтобы можно было создать простую работающую конфигурацию. Более сложные вопросы рассмотрены в книге Брайана Косталеса (Bryan Costales) Sendmail 3rd Edition, O'Reilly & Associates, 2002.

Программа **sendmail** имеет один основной файл конфигурации /etc/sendmail.cf. Начиная с версии 8.9 он находится в /etc/mail:

```
-r--r--r--  1 root  wheel  47893 Oct 29  2001 /etc/mail/
sendmail.cf
```

В каталоге /etc/mail сосредоточены остальные, вспомогательные файлы конфигурации:

Название	Как определен в sendmail.cf	Комментарий
access	Kaccess hash -T<TMPF> /etc/mail/access	настройка доступа
aliases	O AliasFile=/etc/mail/aliases	псевдонимы
mailertable	Kmailertable hash /etc/mail/mailertable	маршрутизация (в частности, для мэйлера UUCP)
mynames	по умолчанию – /etc/mail/local-host-names	домены, для которых принимается почта
relay-domains	FR-o /etc/mail/relay-domains	домены, которые могут отправлять почту через этот **sendmail**

Название	Как определен в sendmail.cf	Комментарий
statistics	O StatusFile	файл почтовой статистики
userdb	O UserDatabaseSpec =/etc/mail/userdb	куда переадресовывать почту
virtusertable	по умолчанию – /etc/mail/virtusertable	переадресация почты с одних компьютеров на другие

Файлы с именами, заканчивающимися на .db, – это хэшированные (индексированные) базы записей, построенные по одноименным файлам конфигурации. То есть access.db строится по файлу access и т. п.

Для построения файла .db используется команда **makemap**. Формат индексированного файла может быть разным. Этот формат обязательно указывается в файле sendmail.cf. Самый давний (не всегда самый лучший) формат – это hash. Команда для превращения текстового файла в индексированную базу выглядит так:

```
makemap hash access < access
```

Поскольку текстовых файлов и файлов индексированных баз в /etc/mail несколько, удобно применить команду **make** для компиляции измененных файлов в базу. Вот пример файла /etc/mail/Makefile:

```
all: access.db mailertable.db virtusertable.db

access.db: access
     /usr/sbin/makemap hash access < access

mailertable.db: mailertable
     /usr/sbin/makemap hash mailertable < mailertable

virtusertable.db: virtusertable
     /usr/sbin/makemap hash virtusertable < virtusertable
```

В терминологии, принятой в **sendmail**, *мэйлером* (*mailer*) называется подпрограмма, принимающая и доставляющая письма по одному определенному протоколу – esmtp, uucp и т. п. Мэйлер local доставляет почту локальным пользователям компьютера. В состав **sendmail** входят несколько мэйлеров. Какие именно, определяется при компиляции. Параметры каждого мэйлера должны быть описаны в /etc/sendmail.cf.

Для получения статистики почтового сервера **sendmail** применяется команда **mailstats**:

```
mailstats
Statistics from Mon May  6 21:23:11 2002

M      msgsfr  bytes_frommsgsto bytes_to msgsrej msgsdis Mailer
3      1653    51295K    146    425K     14735   0       local
```

```
5      130916  3097066K   127595 2751952K 1788    0       esmtp
=============================================================
T      132569  3148361K   127741 2752377K 16523   0
C      133367             128319          16523
```

В выводе **mailstats** фигурируют следующие обозначения:

- M – номер мэйлера;
- msgsfr – количество сообщений от мэйлера;
- bytes_from – послано килобайт от мэйлера;
- msgsto – количество сообщений, пришедших к мэйлеру;
- bytes_to – получено килобайт мэйлером;
- msgsrej – количество сообщений, которым отказано в приеме;
- msgsdis – количество сообщений, молча отправленных в помойку. Документация говорит, что над этими сообщениями выполнена операция DISCARD. Она состоит в том, что сервер принимает сообщение и «глотает» его, сообщив об успешной доставке отправителю, но не доставляя получателю. Используется для введения в заблуждение злостных спамеров;
- Mailer – имя мэйлера;
- T – общее число (total);
- C – количество соединений.

При настройке нового домена и почтового сервера для них иногда возникает ошибка, ведущая к сообщениям «mail loops back to myself».

```
553 relay.domain.net config error: mail loops back to myself
554 <user@domain.net>... Local configuration error
```

FAQ по **sendmail** дает в этом случае такую рекомендацию: «Согласно записям MX, почта для домена (скажем, domain.net) была перенаправлена на указанный хост (в нашем случае relay.domain.net), но он не распознается как domain.net. Добавьте строку domain.net в файл /etc/sendmail.cw (если вы используете FEATURE(use_cw_file)[1]) или добавьте строку Cw domain.net в /etc/sendmail.cf».

В современных конфигурациях файл sendmail.cw по умолчанию часто называется relay-domains.

---

[1] Конструкция FEATURE(use_cw_file) используется в файле .mc, из которого при установке **sendmail** при помощи программы m4 компилируется файл конфигурации sendmail.cf. Проще говоря, наличие этой конструкции означает, что существует файл со списком доменов, которые обслуживает данный почтовый сервер. Если вам точно известно, что такой файл есть, можно быть уверенным, что конструкция по умолчанию присутствовала в нужном месте при установке.

Вместе с **sendmail** умеет работать ряд антивирусных программ, например, коммерческая антивирусная программа DrWeb под UNIX (*www.drweb.ru*), позволяющая вылавливать из почтовых сообщений вирусы. Благодаря этому ваши пользователи меньше рискуют получить зараженное вирусом письмо. Более интересными альтернативами вам, возможно, покажется бесплатный пакет ClamAV (см. главу 29).

Sendmail: куда обратиться за черным списком?

Существует немало источников, где хранятся черные списки. Наиболее известны сервисы *www.spamcop.net*, *relays.ordb.org*, *cbl.abuseat.org*, *www.nl.sorbs.net*. По адресу *http://www.declude.com/Articles.asp?ID=97* поддерживается список адресов, по которым можно обратиться за черными списками спамеров. Для того чтобы проверять письма по нескольким черным спискам подряд, для надежности надо в файле .mc вписать несколько строк типа FEATURE('enhdnsbl'....)

Какие именно параметры следует писать, можно уточнить на соответствующих сайтах держателей черных списков, например по адресу *http://www.ordb.org/faq/#usage*.

Разные черные списки закрывают доступ разным категориям почтовых серверов: одни закрывают доступ с компьютеров, известных как open relays (принимающих и отправляющих почту без ограничений и проверок – от кого и кому угодно, что нарушает соглашения Интернет-сообщества), другие запрещают прием почты с компьютеров, которые фактически рассылали спам, третьи блокируют почту с тех компьютеров, чьи IP-адреса выделены из диапазонов динамически выделяемых адресов крупных провайдеров. В зависимости от того, с каких именно серверов вы хотите блокировать прием почты, пользуйтесь соответствующими списками. Например, закрыв доступ с open relays, вы избавитесь лишь от малой части спама.

Некоторые черные списки известны своей строгостью, например sorbs.net – очень строгий список; единожды попав в него, почтовый сервер имеет мало шансов быть оттуда удаленным. Именно поэтому популярность этого списка сейчас уже сильно упала по сравнению с периодом его расцвета.

Для того чтобы sendmail работал с подавляющим большинством черных списков, корректно будет настроить его, просто повторив с указанием правильного адреса черного списка строку

```
FEATURE(`enhdnsbl',`relay.njabl.org',`',`t',
`127.0.0.2.')dnl
```

(не забудьте заменить relay.njabl.org на фактический адрес другого списка!)

Sendmail: как установить внешний фильтр спама и вирусов?

Спамеры весьма изобретательны в подделке обратных адресов и заголовков, поэтому борьба с ними исключительно средствами почтовых серверов оказалась проигранной. На помощь пришли специализированные программы анализа почтовых сообщений. Наиболее удачными и распространенными проектами к 2005 году стали SpamAssasin (*http://spamassassin.apache.org/*) и Milter-Sender (*http://www.milter.info/sendmail/milter-sender/*). Обе эти разработки позволяют реализовать разные технологии борьбы с вирусами. Как правило, дистрибутивы подобных свободно распространяемых пакетов содержат подробную инструкцию по их установке. Здесь мы остановимся на действии только одного из них – пакета Milter-Sender.

Для установки этого пакета потребуется загрузить его дистрибутив, установить библиотеку libsnert, которая требуется для работы программы milter-sender, и затем установить саму программу. После этого в файл конфигурации .mc потребуется внести изменения, указанные в файле-образце `/etc/mail/milter-sender.mc.sample` из дистрибутива. Последующая обработка этого файла макропроцессором m4 добавит требуемые изменения в файл `.cf`, и новый `sendmail.cf` будет готов. После копирования в `/etc/mail` и перезапуска sendmail последний будет готов к совместной работе с программой milter-sender. Не забудьте ее запустить в качестве демона ДО перезапуска sendmail с новым файлом конфигурации!

Похожим образом устанавливается и поддержка работы с антивирусным пакетом clamav (подробнее о нем рассказано в главе 29).

Вообще говоря, в файле `sendmail.cf` есть специальная секция для добавления туда информации о так называемых мильтерах (milter – от mail filter), вспомогательных программах, которым передается почтовое сообщение для анализа, и эту секцию можно изменять не только посредством добавления макросов в .mc, но и напрямую, модифицируя в текстовом редакторе сам файл `sendmail.cf`.

Установка и начальная настройка postfix

После установки программы postfix из дистрибутива в `/etc/mail` или `/etc/postfix` появится файл `main.cf`, который и содержит все основные настройки Postfix. Подробная информация обо всех возможных командах настройки содержится по адресу *http://www.postfix.org/postconf.5.html*. Вообще, программа postfix превосходно документирована, и всю документацию вы легко найдете на странице *http://www.postfix.org/documentation.html*.

Для того чтобы postfix корректно работал, необходимы по крайней мере следующие указания в файле `main.cf`:

```
command_directory = /usr/sbin
daemon_directory = /usr/lib/postfix
myhostname = mail-gw.company.com
mydomain = company.com
myorigin = $mydomain
mynetworks = 127.0.0.0/8, 192.168.1.0/24
alias_maps = hash:/etc/aliases
sendmail_path = /usr/sbin/sendmail
newaliases_path = /usr/bin/newaliases
mailq_path = /usr/bin/mailq
setgid_group = postdrop
manpage_directory = /usr/local/man
sample_directory = /etc/postfix
readme_directory = no
transport_maps = hash:/etc/postfix/transport
```

Последняя строка необязательна, но полезна, в таблице transport можно указать, с помощью какого протокола и на какой хост надо отправлять письма, адресованные в некий домен; это бывает удобно, если требуется указать нестандартный или попросту отсутствующий в DNS маршрут почты, например, локальную маршрутизацию локального же домена с почтового сервера внутрь корпоративной сети.

После модификации любых таблиц следует запускать программу postmap, например, в случае таблицы transport это будет выглядеть так:

```
postmap transport
```

В ответ на это в каталоге появится файл `transport.db`, содержащий хэшированную базу данных адресов и маршрутов.

Перезагрузка `postfix` выполняется командой `postfix reload`, а требование немедленной обработки очереди сообщений – `postfix flush`.

Postfix: куда обратиться за черным списком?

Для обращения к серверам – хранителям черных списков – следует использовать следующие директивы в файле main.cf:

```
maps_rbl_domains = relays.ordb.org
smtpd_client_restrictions = reject_maps_rbl
```

Ограничиваться указанием всего одного сервера необязательно, можно таких директив написать несколько штук подряд. Список подходящих сайтов для запросов о наличии отправителя в черном списке можно получить по адресу, указанному выше, в разделе «Sendmail: куда обратиться за черным списком?».

Установка POP3-сервера

Есть несколько разных POP3-серверов, которые можно взять из дистрибутива вашей системы или скачать из Интернета. До 2000 года были весьма популярны GNU-pop3d и qpopper, прославившиеся многочисленными «дырками», позволявшими их взломать. Наиболее надежным в то время считался cucipop.

Сейчас наиболее известны:

- qpopper от Qualcomm (*http://www.eudora.com/products/#freeservers*) – есть поддержка SSL;
- GNU-pop3d – один из первых pop3-серверов;
- teapop – поддерживает нестандартную аутентификацию, например, через СУБД MySQL;
- ipop3d (от University of Washington, можно скачать в комплекте с IMAP-сервером – см. выше);
- cucipop – может игнорировать требование сохранить почту на сервере, может работать как через inetd, так и сам по себе, очень надежен.

Новые версии любого из перечисленных серверов, предположительно, свободны от «дыр», но до сих пор самым надежным и удобным в настройке слывет cucipop.

Как настраивать pop3-сервер? После компиляции и установки (см. главу 13) в каталоге `/usr/local/bin` или там, куда по умолчанию ставится ваш pop3-сервер, появится его исполняемый файл. В зависимости от того, какой выбран сервер, это будет qpopper, ipop3d, cucipop или что-то иное. POP3-сервер обычно запускается через **inetd**. Значит, надо вписать полный путь к свежеустановленному pop3-серверу в `/etc/ inetd.conf`. Проверьте, чтобы в начале строки не стоял знак комментария:

```
pop3 stream tcp nowait root /usr/local/etc/ipop3d ipop3d
```

По умолчанию при установке многих систем POP3-сервер устанавливается и настраивается автоматически. В этом случае настраивать ничего не придется.

После модификации `/etc/inetd.conf` не забудьте перезапустить **inetd**.

Подробнее об **inetd** рассказано в главе 10.

Обычно pop3-сервер отдает пользователю почту, лежащую в `/var/mail` или `/usr/mail` в файле, имя которого совпадает с именем пользователя. Некоторые pop3-серверы на время сеанса с пользователем создают файл `/vat/mail/`*`имя_пользователя`*`.lock`, который нужен для блокировки параллельного доступа к почтовому ящику во время работы с ним. Иногда при аварийном завершении сеанса передачи почты этот файл остается. Удалять файл надо, только если пользова-

тель не может обратиться к своему почтовому ящику из-за старого, уже потерявшего смысл файла `*.lock`.

Как видите, этот раздел очень короткий, потому что настраивать в любом POP3-сервере почти нечего. Каждый из перечисленных серверов включает какие-нибудь дополнительные функции, о которых лучше прочесть в документации, распространяемой вместе с исходными текстами серверов.

Настройка IMAP-сервера

Важно отметить, что IMAP-сервер Вашингтонского университета не имеет в дистрибутиве программы **configure**. Это значит, что надо внимательно прочитать файлы `BUILD`, `INSTALL` и `README` из дистрибутива, чтобы точно знать, как его устанавливать в условиях конкретной системы. Обратите внимание, что, как рассказано в файле `BUILD`, IMAP-сервер по-разному устанавливается в системах, где поддерживается аутентификация через `/etc/shadows` и через систему PAM (о PAM см. главу 22). Пока PAM реализован только в последних версиях FreeBSD (4.x), Linux и в Solaris 8.

После компиляции и установки IMAP-сервера понадобится проверить, есть ли запись о протоколе imap в `/etc/services`:

```
imap143/tcp
```

и в `/etc/inetd.conf`:

```
imap   stream   tcp   nowait   root   /usr/local/sbin/imapd
imapd
```

Если нет, надо туда добавить эти строки (путь к **imapd** может быть другим).

qmail

Еще одним популярным почтовым сервером (MTA) является **qmail**. Он менее распространен, чем **sendmail**, но несколько проще в настройке. Подробности о нем доступны по адресу *www.qmail.ru*.

20

Основы работы с веб-сервером Apache

Как устроен сервер Apache

То, что вы сейчас прочтете об устройстве Apache, написано с точки зрения системного администратора, а не программиста. Поэтому я надеюсь на ваше снисхождение ко мне. Я действительно почти не уделяю внимания тем замечательным технологиям, которые применяются в Apache для интеграции с Perl, кэшированию данных, Apache API, правилам написания модулей для Apache и некоторым другим вопросам, связанным с архитектурой Apache. Устройство веб-сервера здесь описано в самых общих чертах и обозначает лишь контуры проблем, с которыми может встретиться вебмастер и администратор веб-сайта.

Итак, главной программой пакета Apache является **`httpd`**. Это демон, отвечающий за обслуживание http-запросов. Для обработки каждого запроса запускается своя копия демона. Чтобы не тратить время на запуск копии демона по приходе каждого запроса, несколько копий запускаются заранее.

Программа **`httpd`** имеет модульную структуру. Модули могут быть вкомпилированы статически или загружаться по мере надобности. Загрузка модуля происходит при запуске **`httpd`**, когда он встречает в файле конфигурации `httpd.conf` инструкцию **`LoadModule`**.

При запуске **`httpd`** запускает свои копии от имени пользователя nobody (если вы не меняли эту настройку специально, проследите, чтобы такой пользователь в системе был). Количество копий определяется в файле конфигурации. По умолчанию оно равно 4 или 5, задайте это значение, исходя из своих реальных потребностей.

В файле конфигурации многие настройки устанавливаются по умолчанию, и они довольно разумны. После изменения любых настроек при работающем сервере **httpd** дайте команду **/usr/local/apache/bin/apachectl restart**. Остальные способы заставить **httpd** переконфигурироваться на новый лад применять не рекомендуется. Программа **apachectl** специально написана для управления сервером Apache. Иногда, если PID запущенного сервера **httpd** не совпадает с /var/run/httpd.pid (путь к httpd.pid может быть иным), **apachectl** гневается и сообщает об этом на консоль, откуда его запускают. Это лечится удалением файла httpd.pid и последующим запуском **httpd** вручную, вот так: **/usr/local/apache/bin/httpd**.

Пакет Apache состоит из нескольких программ, из которых наиболее важны **httpd**, **apachectl** и **htpasswd**.

Дополнительные сведения о настройке веб-сервера Apache можно получить из документации по нему на веб-сайте Apache *www.apache.org*.

Установка Apache

Самый надежный способ установки apache – из исходных текстов. Если нас устраивает конфигурация по умолчанию, мы следуем процедуре, рекомендованной в файле INSTALL, включенном в дистрибутив:

```
./configure
make
make install
```

На этапе **./configure** строится Makefile и файл config.status, в котором записывается вызов **configure**. Он записывается в точности так, как его надо будет вызвать в будущем, если мы захотим перекомпилировать Apache, полностью сохранив конфигурацию пакета. «В точности» означает «с указанием всех подразумеваемых и явно указанных ключей». Под конфигурацией понимается расположение дерева каталогов, в которых хранятся файлы конфигурации Apache и выполняемые программы пакета (**httpd**, **htpasswd**, **apachectl**). Кроме этого, конфигурация описывает дополнительные модули, которые следует включить в **httpd**.

Запись конфигурации в config.status сослужит нам добрую службу в дальнейшем, если мы захотим установить новую версию Apache, сохранив функциональность пакета. Например, встроенная поддержка PHP требует добавления в исходные тексты Apache подпрограмм работы с PHP. Последние поставляются не с исходными текстами Apache, а с исходными текстами пакета php. Лучше их местоположение будет записано в config.status единожды, чем вы каждый раз будете вспоминать, где же они находятся.

Перед запуском **configure** надо установить в систему все программы, которые будут использоваться для работы Apache. Например, если пла-

нируется запускать SQL-сервер на том же компьютере, что и Apache, то SQL-сервер надо установить заранее. Если в Apache будет поддержка Perl и PHP, установите пакеты perl и php до установки Apache.

При вызове **configure** следует помнить, что надо:

- указать дополнительные модули, которые должны быть включены в конфигурацию, если их там нет по умолчанию. Если планируется загружать модуль командой **LoadModule** и модуль уже скомпилирован, то указывать его в вызове **configure** не надо;
- указать модули, входящие в конфигурацию apache, но не компилируемые по умолчанию, если это требуется;
- указать путь к интерпретатору Perl;
- включить модуль загрузки динамических объектов (для того чтобы потом загружать новые модуди командой **LoadModule** в `httpd.conf` без перекомпиляции Apache), модуль подсоединяется ключом **--enable-module=so**.

Естественно, перед компиляцией Apache надо прочесть файл `INSTALL` из дистрибутива. Нежелание читать документацию приводит к необходимости компилировать все повторно. Дурная голова ногам покоя не дает...

Как правило, процедуры **make** и **make install** проходят без сучка, без задоринки. Однако могут возникнуть проблемы из-за того, что какая-нибудь библиотека не была найдена. В коде Apache вызывается много разных функций. Надо иметь в виду, что файлы типа `Makefile` составляют люди, а людям свойственно ошибаться. Не так-то легко учесть особенности всех систем, в которых можно установить Apache. Ведь речь идет о нескольких десятках диалектов UNIX, кроме того, есть еще и Win32-системы. Поэтому, если на этапе компиляции произошел сбой, надо проверить:

1. Выполняете ли вы компиляцию от имени root? Попытка установить программу с правами обычного пользователя часто обречена на неудачу (кто попало не имеет права записи в системные каталоги).
2. Все ли программы, которые хочет вызвать **make**, лежат в каталогах, записанных в `PATH`? Надо содержать `PATH` в порядке, чтобы там было все необходимое.
3. Есть ли место на диске, куда ставится Apache? Для распаковки и компиляции всех программ пакета на диске должно быть свободно не менее 12 Мбайт.

Если пока все в порядке, в сбое виноваты, скорее всего, не вы. Будем искать причину дальше.

Скорее всего, **make** не хватило какой-нибудь функции (сообщение типа «undefined reference to ‘name’»). Это значит, что в объектных файлах после компиляции и в подключаемых библиотеках не нашлось нужной функции. Отчего?

1. Библиотеки расположены не там, где их ожидает найти **make**. Проверьте пути в `Makefile`.
2. Слишком старые (или слишком новые) библиотеки. Искомая функция переехала в другую библиотеку (или вообще недоехала). Поправьте `Makefile`: укажите программе **make** ключами, где искать библиотеки.
3. Иногда все это не помогает. Мне однажды пришлось просто найти библиотеку, в которой была нужная функция (искать можно по **man** *имя_функции*). Затем я извлек ее в обычный объектный файл и записал его в дерево дистрибутива Apache (там **make** находит все без проблем):

```
ar x libc.a fstat.o
```

Программа **ar** умеет извлекать объектные файлы из библиотек. В этом примере `libc.a` – стандартная библиотека C, а `fstat.o` – нужный объектный файл потерявшейся функции.

Основные настройки Apache

Все настройки Apache хранятся в файле `/usr/local/apache/conf/httpd.conf`.[1] Путь к файлу `httpd.conf` может быть иным. Этот файл разделен на секции, каждая из которых описывает свой уровень настроек. Одни настройки относятся ко всему веб-серверу, другие – к отдельным сайтам, которые обслуживает веб-сервер.

Вообще существуют следующие уровни:

- Сервер
- Виртуальный сервер (фактически сайт; сервер может обслуживать запросы к нескольким сайтам)
- Каталог (файл)
- `.htaccess` (файл, определяющий доступ к тому каталогу, внутри которого он находится)

Соответственно существуют директивы (ServerName, ServerAdmin, ServerRoot и многие другие, они называются server-wide directives), относящиеся к серверу в целом. Директивы, заданные внутри пар директив

```
<VirtualHost имя> </VirtualHost>
<Directory имя> </Directory>
<Files имя> </Files>
```

[1] В старых версиях Apache настройки также содержались в `srm.conf` и `access.conf`. Кроме того, индивидуальные настройки, относящиеся к каталогам, записываются в файлы `.htaccess`, их синтаксис совпадает с синтаксисом `httpd.conf`.

распространяются только на соответствующий виртуальный хост (то есть отдельный сайт), каталог или файл(ы). Существуют и некоторые другие директивы ограничения области действия, например <DirectoryMatch>, <Location> и др. Самую свежую информацию о директивах можно почерпнуть в документации по Apache на *www.apache.org*. Копия этой документации распространяется вместе с исходными текстами Apache и устанавливается в `$prefix$/htdocs/manual`, где `$prefix$` – это каталог, указанный **configure** при установке как начальный каталог дерева каталогов Apache. По умолчанию это `/usr/ local/apache/`.

Работа с SSI

Механизм, называемый SSI (Server Side Includes), позволяет в html-файлах давать инструкции «включить вот сюда содержимое файла, например `mainmenu.inc`». Сама инструкция в файле выглядит довольно просто:

```
<!--#include virtual="mainmenu.inc" -->
```

Такие инструкции удобны: можно заранее разработать только оформление сайта, а содержимое обновлять по мере необходимости. К тому же это позволяет включать однотипные блоки информации во многие страницы, но при изменении этой информации будет достаточно поменять один файл.

Однако для того чтобы это работало, надо включить в Apache поддержку SSI.

Для этого в файле `/usr/local/apache/conf/httpd.conf` надо:

- указать директиву **Options -Indexes +Includes** в описании виртуального сайта или каталога. Для SSI предназначено только **+Includes**, а **-Indexes** – это требование не выдавать список файлов в каталоге, если там не нашлось `index.html`. Очень рекомендую!
- добавить директивы в раздел AddType:

```
AddType text/html .shtml .shtm .html .htm
AddHandler server-parsed .shtml .shtm .sht .html .htm
```

AddType сообщает веб-серверу, что файлы, имена которых заканчиваются на `.shtml`, тоже являются html-файлами, как `html` и `htm`. AddHandler говорит, что файлы с такими именами должны проверяться сервером перед отправкой пользователю на наличие директив. Если файл не определить как проверяемый[1] на сервере (server-parsed), сервер и не подумает вместо директивы `include` вставить содержимое

[1] «Проверяемый» в данном контексте означает «проверяемый на предмет исполнения специальных директив, если они содержатся в файле».

файла. Просто потому, что обычные файлы он отдает по запросу пользователя, не заглядывая в них.

По общему соглашению файлы, содержащие директивы SSI, должны иметь расширения `.shtml` или `.shtm`. Для того чтобы обычные html-файлы тоже проверялись сервером перед отправкой посетителю сайта, были «server-parsed», надо вписать расширения `.html` и `.htm` в директиву AddHandler, как показано выше. Однако это замедлит работу сервера, поэтому при высокой нагрузке на веб-сервер так лучше не делать.

Если файлы, проверяемые на сервере, могут быть индексными файлами каталогов, то надо не забыть добавить `index.shtml` в директиву `DirectoryIndex`. Она содержит имена файлов, которые сервер расценивает как индексные. Это значит, что при обращении к каталогу по URL *http://www.server.edu/news/* будет выдан файл `index.html` или `index.shtml` из этого каталога:

```
DirectoryIndex index.html index.shtml
```

Как заставить выполняться CGI-скрипты

Прежде всего, CGI-скрипт должен сам соответствовать правилам CGI. То есть в начале своего вывода он должен сообщить тип выдаваемого документа – обычно простой текст (plain text) или html. Затем должна идти обязательная пустая строка, за которой – сам документ. Вот пример CGI-скрипта на Perl:

```
print 'Content-type: text/html',"\n";
print "\n";
print '<HTML><BODY>';
print '…… ';
print '</BODY></HTML>';
```

Скрипты могут быть написаны на любом языке программирования, могут представлять собой текстовые файлы (тогда их должен интерпретировать **sh**, **perl** или иной интерпретатор команд) или двоичные файлы, предназначенные для выполнения.

Для того чтобы скрыть истинное расположение CGI-скриптов от посетителей сайта, полагается их укладывать в какой-нибудь укромный каталог вне основного дерева каталогов веб-сайта.

Например, каталоги сайта лежат в `/usr/local/apache/gio`. В каталоге `gio` есть каталоги `news`, `prices` и т. д. А форум этого сайта написан на языке Perl и представляет собой коллекцию скриптов, вызываемых по URL *http://www.gio.ru/cgi-bin/*.pl*. Однако никакого подкаталога `cgi-bin` в каталоге `/usr/local/apache/gio` нет. Скрипты на самом деле лежат в каталоге `/usr/local/apache/cgi-bins/gio/`. Об этом веб-серверу говорит запись `alias` в файле конфигурации в секции виртуального хоста `gio`:

```
<VirtualHost www.gio.ru>
…
alias cgi-bin /usr/local/apache/cgi-bins/gio/

</VirtualHost>
```

Для того чтобы сервер мог запустить на выполнение файлы `*.pl` из каталога `/usr/local/apache/cgi-bins/gio/`:

1. Эти файлы должны быть доступны для выполнения пользователю nobody, от имени которого запущен **`httpd`**.
2. В файле `httpd.conf` должно быть описание каталога `/usr/local/apache/cgi-bins/gio/`, а в описании должно быть разрешение запускать файлы в этом каталоге:

```
<Directory /usr/local/apache/cgi-bins/gio/>
Options ExecCGI
</Directory>
```

Обработка скриптов на PHP

Язык PHP (Personal Home Page, *www.php.net*) был создан Расмусом Ледорфом (Rasmus Lerdorf). Первая версия PHP была реализована в конце 1994 – начале 1995 года. Она не предназначалась широким массам пользователей. Наибольшее распространение получили более поздние, ставшие индустриальным стандартом версии PHP 3 и 4.

Как правило, интерпретатор PHP встроен в веб-сервер. В сервер Apache модуль поддержки PHP может быть вкомпилирован статически, а может загружаться динамически в виде отдельного модуля.

Программы на языке PHP часто встроены в код HTML-страниц, поэтому для их специальной интерпретации надо сделать меньше шагов, чем для запуска CGI-скриптов.

В `httpd.conf` должны присутствовать инструкции:

```
LoadModule php4_module libexec/libphp4.so  # загрузить
                                           # поддержку php4
AddType application/x-httpd-php .php4  # считать файлы .php4
                                       # содержащими php
```

Можно заставить обрабатывать файлы с другими именами, например `.php`, `.phps`, `.phtml` и т. д., так же, как файлы, содержащие PHP. Если в коде такого файла будет только HTML, веб-сервер не обидится, а если кроме HTML обнаружится PHP, то последний будет интерпретирован правильно:

```
AddType application/x-httpd-php .php
AddType application/x-httpd-php-source .phps
AddType application/x-httpd-php .php3
AddType application/x-httpd-php .php4
AddType application/x-httpd-php .phtml
```

Пример полного описания веб-сайта (virtual host) в файле `httpd.conf` с разрешенным выполнением скриптов на PHP приведен ниже:

```
LoadModule php4_module        libexec/libphp4.so
AddType application/x-httpd-php .php4
AddType application/x-httpd-php .php
AddType application/x-httpd-php-source .phps
AddType application/x-httpd-php .php3
AddType application/x-httpd-php .php4
AddType application/x-httpd-php .phtml
<VirtualHost 192.168.4.54>
    ServerAdmin noc@very.spb.ru
     ServerName www.very.spb.ru
     DocumentRoot /usr/local/apache/very
     DirectoryIndex index.htm index.html
     ErrorLog logs/very-error_log
     CustomLog logs/ very-access_log common
     Options -Indexes ExecCGI +Includes
     ScriptAlias /cgi-bin/ /usr/home/very/cgi-bin/
</VirtualHost>
```

Отметим, что приведенный здесь вариант, когда директива **`Options ExecCGI`** указана не для каталога, а для целого виртуального хоста, нельзя считать особенно удачным. Так можно делать только в случае, если вы полностью доверяете администратору и создателю этого сайта. Намного лучше определять **`Options ExecCGI`** только для конкретного каталога.

Обработка скриптов на Perl

Скрипты, написанные на Perl, могут выполняться встроенным модулем Apache (**`mod_perl`**) или стандартным интерпретатором Perl, установленным в системе. Первый вариант работает быстрее, но **`mod_perl`** занимает довольно много места в памяти. Надо учитывать, что потребуется как минимум 1 Мбайт оперативной памяти на каждый Perl-скрипт, обрабатываемый **`mod_perl`** (плюс около 15 Мбайт на сам модуль). Приведенные здесь цифры условны, они могут меняться в зависимости от конфигурации и конкретной реализации.

Для того чтобы разрешить серверу выполнять скрипты на Perl, необходимы те же действия, что и для разрешения выполнения любых других скриптов.

В описании сервера или сайта (виртуального хоста) надо указать, что файлы `*.pl` – это скрипты. За это отвечает директива AddHandler. Затем указывается местоположение скриптов (псевдоним каталога `cgi-bin`). В нижеследующем примере предполагается, что скрипт, например `show.pl`, будет вызываться по URL *http://www.for.spb.ru/cgi-bin/show.pl*, а фактически скрипт `show.pl` будет расположен в каталоге сервера `/usr/local/apache/spb_forum_cgi-bin/`.

Описание каталога, в котором фактически находятся скрипты (но ни в коем случае не всего сервера!), должно содержать директиву **Options ExecCGI**. Директива **AllowOverride** нужна для того, чтобы файл `.htaccess` в этом каталоге определял специфические параметры доступа к скриптам. Если вы делаете скрипты доступными для всех, то директивы **AllowOverride** в описании каталога быть не должно.

Если вы загружаете модуль встроенной поддержки Perl **mod_perl**, обязательно ознакомьтесь с документацией к нему. Любые динамически подгружаемые модули обычно требуют директив LoadModule и AddType.

```
<VirtualHost 192.168.5.19>
      ServerName www.for.spb.ru
     AddHandler cgi-script .pl
     ScriptAlias /cgi-bin /usr/local/apache/spb_forum_cgi-bin
     DocumentRoot /usr/local/apache/forum
 <Directory /usr/local/apache/spb_forum_cgi-bin>
     Options -Indexes ExecCGI
     AllowOverride AuthConfig
</Directory>
    ErrorLog logs/spb-forum-error_log
    CustomLog logs/spb-forum-access_log common
</VirtualHost>
```

Протоколирование запросов и событий

Последний пример в предыдущем разделе содержал интригующие строки:

```
ErrorLog logs/spb-forum-error_log
CustomLog logs/spb-forum-access_log common
```

Эти строки определяют, что в каталоге `logs` будут лежать файлы протокола виртуального сервера. Каждый виртуальный сервер должен иметь свои собственные файлы протокола. Это важно и для статистики, и для поиска ошибок в скриптах и страницах. Относительный путь к файлу в `httpd.conf` начинается от корня сервера, то есть от каталога, указанного в директиве ServerRoot, обычно `/usr/local/apache/`.

Безопасность httpd

Веб-сервисы очень популярны в Интернете, ими пользуются многие, и это ведет к тому, что веб-сервер становится более уязвим, чем любая другая программа UNIX. Тому есть несколько причин:

- о деталях работы этих сервисов осведомлено больше людей, чем о работе, скажем, gopher-серверов;

- количество людей, которые увидят веб-страницу уязвимого веб-сервера, может быть очень велико, это становится соблазном в конкурентной борьбе и в распространении вирусов через веб-страницы;
- большинство уязвимостей демонов под UNIX связано с ошибками в их программировании, следовательно, наличие уязвимости связано только с квалификацией разработчика программы; выбирая проверенное программное обеспечение от авторитетного разработчика, вы с большой вероятностью защищаете себя от крупных ошибок ПО; в случае с веб-сервером вам приходится также полагаться на грамотность пользователей, которые пишут для своих веб-страниц разнообразные скрипты на PHP, Perl и т. д.; вероятность их ошибки добавляется к вероятности ошибки разработчика. Таким образом, эта вероятность приближается к 100%.

Наверное, можно найти и другие причины повышенной уязвимости веб-серверов. Из-за того что веб-сервер в системе является объектом повышенной опасности, существуют правила безопасности, которые непременно надо соблюдать. Пренебрежение ими обычно ведет к большим финансовым потерям. Последние обычно связаны с хищением информации (например, номеров кредитных карт из интернет-магазинов), взломом серверов (часто взлом веб-сервера открывает дорогу к взлому остальных служб на том же компьютере), резким увеличением входящего трафика (запросов к троянским коням или для сохранения информации на взломанном сервере).

Для того чтобы свести к минимуму вероятность взлома веб-сервера Apache, надо:

- указать в файле `httpd.conf`, что запускать сервер следует от имени nobody. Первая копия **`httpd`**, запускаемая из стартовых скриптов, запустится от имени root, но все остальные – от nobody;
- все каталоги `cgi-bin` хранить вне дерева каталогов содержимого веб-сервера (корень последнего указывается директивой DocumentRoot);
- хранить все файлы `htpasswd`, используемые для аутентификации пользователей, вне дерева каталогов содержимого веб-сервера (именно `htpasswd`, где хранятся пароли, а не .htaccess, которые как раз должны лежать в каталогах, доступ к которым они ограничивают);
- сделать все файлы `.htaccess` доступными для чтения, но запретить в них запись всем, кроме root. Например, иметь такие права доступа:

  -rw- r- - r- - rootwebmaster….htaccess

- разрешить использование файлов `.htaccess` индивидуально для каждого каталога (или веб-сайта, виртуального хоста). Но только для тех каталогов, где это в действительности нужно. О применении директивы AllowOverride для этого рассказано в следующем

разделе. По умолчанию **httpd** не обращает внимание на файлы `.htaccess`. Файл `.htaccess` не должен быть доступен пользователям для изменений. И пользователю незачем иметь возможность записи своего `.htaccess`!

- запретить запуск скриптов из каталогов, которые вы не контролируете. Следите за правами по умолчанию, которые присваиваются файлам при обновлении содержимого сайтов через ftp (иногда по умолчанию устанавливается бит запускаемости для всех файлов – это совершенно излишне);
- ограничить доступ пользователей (особенно незнакомых; владельцам хостингов следует держать ухо востро!) к файловой системе сервера, к таблицам баз данных и самому серверу баз данных, особенно если пользователи сами пишут интерфейсные программы для связи веб-сервера и сервера баз данных. Необходимо установить максимально возможные ограничения, такие, чтобы пользователи могли работать, но не более необходимого минимума;
- читайте файлы протоколов! ЧИТАЙТЕ их! Там можно обнаружить много интересного. Намного лучше обнаружить что-нибудь новенькое там, чем потом обнаружить что-нибудь на первой странице веб-сайта вашей компании;
- ни в коем случае не указывайте владельцем всех процессов httpd пользователя root вместо nobody! Это просто подарок хакерам! Если вы предоставляете бесплатный хостинг и пренебрегаете основными правилами безопасности, считайте, что вас уже взломали (проверьте прямо сейчас, возможно, так оно и есть).

Аутентификация в Apache

Способ аутентификации является глобальной установкой для Apache.

Для того чтобы настройки прав доступа для конкретного каталога (из файла `.htaccess`) имели приоритет перед стандартными, надо в описании этого каталога (или целого сайта, в дереве которого лежит каталог) указать настройку `AllowOverride AuthConfig`.

Типичный файл `.htaccess` выглядит так:

```
AuthType Basic
AuthUserFile /usr/local/apache/conf/htpasswd.users
AuthName "special directory"
Require valid-user
```

Файл `htpasswd` с именами и шифрованными паролями пользователей создается программой **htpasswd** (см. **man htpasswd**). Этот файл естественно следует хранить в стороне от дерева каталогов, доступного для чтения пользователям.

В главе 24 рассказывается о том, как выполнять аутентификацию пользователей веб-сервера через PAM. Этот способ применим в частности для аутентификации пользователей веб-сайта через контроллер домена сети Microsoft.

Для получения дополнительной информации об аутентификации пользователей в Apache стоит также посмотреть раздел «user authentication» в FAQ по Apache и описание модуля `mod_access` в документации.

Ограничение количества одновременных соединений

В некоторых случаях хорошая посещаемость сайта «выходит боком». Если сайт обслуживается не слишком мощным компьютером, последний может не вынести нагрузки. Наверное, самым показательным примером может служить крах веб-серверов основных новостных служб 11 сентября 2001 года, в день ужасного террористического акта в США. В тот день миллионы людей хотели узнать последние новости через Интернет, и администрациям многих новостных служб пришлось приостановить работу своих сайтов. CNN (*www.cnn.com*) вскоре после начала страшных событий выставила единственную страницу с фотографией пожара в WTC, поскольку обеспечить работу в обычном режиме, с лентой новостей и видеоматериалами, их веб-сервер был не в состоянии.

Серверы российских новостных служб тоже полегли под резко возросшей нагрузкой. Единственным источником русскоязычных новостей остался Росбизнесконсалтинг (*www.rbc.ru*), где администрация загодя создала многократный резерв устойчивости к нагрузке. Из известных мне европейских новостных сайтов весь день 11 сентября продержался только Guardian Unlimited (*www.guardian.co.uk*), где новостная лента была доступна все время.

В обычной обстановке нагрузка на веб-сервер тоже может неожиданно возрасти.

Если все веб-сайты, размещенные на вашем сервере, представляют собой исключительно совокупности html-текстов и изображений в форматах gif и jpg, то проблем с нагрузкой у вас не будет еще долго.

А вот выдача или помещение информации в базу данных через веб-интерфейс может серьезно нагрузить сервер. Сам `httpd` не будет сильно нагружен, но запуск нескольких десятков скриптов, общающихся с базой данных, может усложнить ситуацию. Притом, что запустившие их копии `httpd` занимают память в ожидании завершения скриптов, объем свободной памяти станет минимальным. Приход каждого следующего http-запроса может вызвать интенсивный свопинг. Как только перестанет хватать памяти, работа многократно замедлится. Это фактически приведет к отказу от обслуживания.

Атаки на серверы, имеющие целью довести их до отказа от обслуживания запросов, как раз и происходят по аналогичной схеме.

Для того чтобы ограничить количество одновременных запросов к серверу, можно, во-первых, ограничить максимальное число запускаемых копий **httpd**.

Это делается с помощью директивы

```
MaxClients n
```

где *n* – количество копий **httpd**. По умолчанию полагается равным 256.

Однако иногда проблема в том, что надо ограничить доступ лишь к одному-единственному каталогу или даже скрипту. Например, разрешить лишь одному пользователю в каждый момент времени запускать такой скрипт.

Благодаря модулю **mod_limitipconn**, который написал Дэвид Джао (David Jao), стало возможным ограничить количество одновременных запросов с одного IP-адреса к любому объекту, обслуживаемому **httpd**.

Например, следующим образом можно ограничить число запросов к любым URL, содержащим `cgi-bin`:

```
# загрузка модуля limitipconn
LoadModule limitipconn_module libexec/mod_limitipconn.so
# ограничение числа соединений
<VirtualHost 192.168.2.14>
    ServerName very.busy.com
    <IfModule mod_limitipconn.c>
         <LocationMatch ".*cgi-bin.*">
          MaxConnPerIP 1
         </LocationMatch>
</IfModule>
</VirtualHost>
```

В этом примере количество одновременных обращений к URL, содержащим подстроку `cgi-bin`, с одного IP-адреса ограничивается одним соединением.

Чтобы в файл протокола не сыпались сообщения об отвергнутых «лишних» соединениях, можно указать в описании виртуального хоста следующую директиву (она доступна при загруженном модуле **mod_limitipconn**):

```
CustomLog logs/very-busy-access_log common env=!LIMITIP
```

Без нее файл протокола `very-busy-access_log` будет завален сообщениями об отказе в доступе, особенно если на сервер пытаются провести атаку.

21

Настройка http-cache Squid

Установка и основные настройки

Установка squid

Как и прочие пакеты, **squid** устанавливается достаточно просто: дистрибутив раскрывается в отдельный каталог, а затем в этом каталоге выполняются команды

```
./configure
make
make install
```

После этого остается отредактировать файл конфигурации `squid.conf` и проверить, что на диске, выделенном для кэша, достаточно места.

Главные настройки

Надо указать, где будет храниться кэш. Там должно быть достаточно места. Практика показывает, что в сети с двадцатью компьютерами для кэша достаточно 100–200 Мбайт. Сотне компьютеров маловато даже 500 Мбайт, а желательно иметь не меньше гигабайта. Можно считать, что оптимальный размер кэша при среднем «потреблении» файлов из Интернета пользователями сети составляет около 20 Мбайт на каждый компьютер, с которого пользователи имеют доступ в Интернет.

```
/var/cache ufs 256 16
```

По умолчанию **squid** выставляет 16 каталогов первого уровня. Обычно используются первые 3–4. Если в вашем случае это так, смело изменяйте строку в файле конфигурации. Незачем занимать место на дис-

ке просто так и вводить **squid** в смущение. Пусть каталогов будет поменьше, эффективность только повысится:

```
/var/cache ufs 4 128
```

Перед изменением этого параметра стоит поставить эксперимент и посмотреть, насколько **squid** на самом деле использует каталоги кэша.

Надо указать максимальный размер кэша в памяти. Имейте в виду, **squid** в действительности займет несколько больше места, чем вы укажете, оставьте запас. Если установлено всего 64 Мбайт памяти, не надо отдавать пакету **squid** больше 48. Если сервер выполняет еще какие-то обязанности (почтовый сервер, DNS и т. д.), **squid** обойдется и кэшем в 32 Мбайт.

Если места немного, обратите внимание на установку максимального размера файла, хранимого в кэше. Возможно, диск маловат для хранения объектов размером по 50 Мбайт. С другой стороны, вы здорово сократите свой входящий трафик, если у вас в организации любят перекачивать одни и те же видеофильмы из Интернета втайне друг от друга. Тогда, увеличив максимальный размер кэшируемого объекта, вы сэкономите время сотрудников и деньги компании: видеофильм будет сохраняться в кэше и выдаваться по запросу мгновенно.

Пакет **squid** представляет собой хорошее средство анализа трафика компании. Это удивительно, но компания может сэкономить несколько сотен долларов ежемесячно, если создаст локальные копии наиболее популярных порносерверов у себя в сети. Именно так и поступили в одном университете (его название я не открываю по соображениям безопасности).

Чужим здесь не место

Излишняя открытость в сети часто вредит. Не следует открывать свой **squid** для всех желающих тянуть через него файлы. Это должно быть разрешено только сотрудникам компании, имеющим доступ в локальную сеть. По умолчанию через **squid** может получать файлы кто угодно.

Это грозит по крайней мере двумя типами неприятностей:

- За счет вашей компании кто-то будет вытягивать информацию из Интернета. Условия подключения к сети у разных провайдеров различны. Если кто-то платит за зарубежный трафик больше, чем за локальный, он может обращаться внутри города к вашему squid с требованием скачать файл из-за рубежа.
- Неизвестные люди будут вытаскивать информацию через ваш сервер. Возможно, что они взламывают чей-то банк или магазин или пользуются хранилищами детской порнографии, или добывают через сеть военные секреты. Во-первых, незачем даже бездействием потакать преступникам, во-вторых, надо ли вам, чтобы чужие грязные следы вели через сеть вашей организации? Общение со следова-

телями, знаете ли, отвлекает системного администратора тихой компании от выполнения непосредственных обязанностей.

В файле `squid.conf` определение тех, кому дозволено пользоваться этим squid, выполняется в два этапа, как и любые другие разрешения или запрещения в `squid.conf`.

Вначале определяется так называемый список управления доступом (ACL – Access Control List). Учитывая краткость термина ACL, он и будет упоминаться в дальнейшем в этом разделе. В данном контексте[1] ACL – это список или шаблон, идентифицирующий запрос. Идентификацию **squid** умеет производить по широкому спектру признаков, в частности по запрошенному URL, по IP-адресу источника запроса или сервера, которому предназначен запрос, по части URL, по части доменного имени и по доменному имени. Особенно интересна идентификация по части URL, потому что она позволяет выделить такие URL, как, например,

```
http://www.sexytetki.com/index_2.html
```

или

```
http://counter3.sextracker.com/c2/id/0/182450
```

После того как задан ACL, наступает вторая фаза: определяется, что делать с такими запросами – выполнять их (allow) или не выполнять (deny). Поэтому типичный ACL для запрета доступа к кэшу всем, кроме своих пользователей, выглядит так:

```
acl localdomain srcdomain co.spb.ru
acl localipnet src 192.168.5.1-192.168.5.254
```

Затем в части определений доступа пишем

```
http_access allow localdomain
http_access allow localipnet
http_access deny all
```

Последняя строка на самом деле должна быть последней в определении доступа, так как правила доступа проверяются сверху вниз до первого совпадения.

Инициализация каталога-хранилища

Перед тем как запускать **squid** в работу, надо инициализировать его хранилище. Хранилище представляет собой дерево каталогов. Путь к нему указывается директивой **cache_dir** в файле `/etc/squid/`

[1] Термин ACL используется также в некоторых фильтрах пакетов (firewalls), в частности в операционной системе IOS маршрутизаторов Cisco для определения списков хостов, которым разрешен или запрещен определенный тип доступа.

squid.conf (путь к файлу squid.conf может быть иным). Инициализация каталогов производится командой

```
squid -z
```

По этой команде создается необходимое для **squid** дерево каталогов хранилища. Файл swap.state в корневом каталоге дерева удалять нельзя: это индексный файл хранилища. Его удаление равносильно полному стиранию кэша.

Борьба с баннерами

Одним из видов паразитного трафика являются баннеры. Собственно говоря, сетевое сообщество давно придумывает методы борьбы с непрошеной рекламой. В ближайшие годы индустрия отбора информации будет развиваться. Человек разумный все меньше и меньше желает платить деньги за то, что ему совсем не нужно. Даже если эти деньги у него есть.

Мы прекрасно понимаем, что эту книгу читают интеллигентные и образованные люди. Как правило, сотрудники российских компьютерных компаний гордятся тем, что среди специалистов, общающихся в Интернете на профессиональные темы, много людей хороших и отзывчивых и мало нехороших и злобных. Поэтому я позволю себе, опираясь на этот феномен, призвать коллег, то есть вас, уважаемые читатели, к разумному потреблению товаров и услуг. Если сегодня вы не скачаете ненужный вам баннер, завтра озаботитесь блокировкой порнопродукции на кэширующем сервере своей компании, а послезавтра в лесу откажетесь разбить пустую бутылку не из лени, а заботясь о чистоте леса, то, может быть, через неделю этот мир станет чуть-чуть чище и приятнее для проживания, а излишки денег будут направлены на стимулирование производителей полезного, а не его прожигателей.

Теперь ближе к делу. Как блокировать паразитный трафик, если он передается через **squid**?

Во-первых, полезны разумные организационные меры. Можно изучить файл протокола **squid** (обычно это /var/log/squid/access.log, но путь может быть иным; он задается в squid.conf). Он изобилует ссылками на музыку (*.mp3), файлы с URL типа *www.sexy.com/girls/marina.jpg* и т. п.? Возможно, следует подготовить доклад начальству. Например, указать, что 300 из 700 уплаченных за прошлый месяц единиц денег за трафик заплачены за интерес сотрудников к картинкам сексуальной направленности. Разумеется, никто не предлагает поощрять доносительство и сообщать, кто именно тратит казенные деньги таким образом. Но ввести официальный запрет на выкачивание объемных файлов музыки и картинок полезно.

Во-вторых, если начальство считает, что порнографию следует поощрять, вы можете потребовать покупки более мощного канала связи. В конце концов, вам не хватает ночи для выкачивания последнего дистрибутива FreeBSD!

В-третьих, изучение файла протокола может показать, что до 10% трафика могут занимать баннеры. К счастью, в Интернете можно легко найти списки баннерных систем и характерных элементов URL, которые позволяют идентифицировать ссылку на баннер. Искать следует по ключевым словам «banner», «bad_url», «блокировка баннеров».

Поскольку число баннерных систем и шаблонов, по которым надо опознавать ведущие к баннерам URL, множится день ото дня, эти шаблоны организуются в файлы (например, `bad_url`) и складываются в отдельный каталог (например, `/etc/squid/acls`). Маленький пример:

```
head bad_url
www.gameport.ru
www.wwwcatalog.net
www.burstnet.com/cgi-bin/ads/
www.netcity.ru/cgi-bin/images/images.pl
www.burstnet.com/gifs/
www.russianstory.com/cgi-rs/showme.cgi
1000.stars.ru/cgi-bin/1000.cgi
count.ru/cgi-bin
x.counter.ru/counter.gif
```

Пример взят наугад, если имена в нем совпали с реальными именами поставщиков баннеров, то это случайность. Не следует использовать его в рабочих целях.

После создания подобного файла нужно указать в `squid.conf`, как этот файл использовать.

Вот соответствующие две части `squid.conf`:

```
acl good_url    url_regex     "/etc/squid/acls/good_url"
acl bad_url     url_regex     "/etc/squid/acls/bad_url"

acl bad_dom dstdomain  "/etc/squid/acls/porn/domains"
acl bad_url_porn urlpath_regex "/etc/squid/acls/porn/urls"
…
http_access deny bad_url !good_url
http_access deny bad_dom !good_url
http_access deny bad_url_porn !good_url
http_access allow localdomain
http_access allow localipnet
http_access deny all
```

В первой части указывается, где лежат шаблоны. Так, есть файл *good_url* и файл *bad_url*. Шаблоны оттуда представляют собой регулярные выражения для сравнения с запрошенным URL. На это указывает ключевое слово `url_regex`. Кроме того, список «плохих» доме-

нов содержится в файле *porn/domains*, а еще один список подлежащих блокировке URL находится в *porn/urls*.

Во второй части указывается, что следует запретить доступ к URL, если он попадает хотя бы в один из шаблонов bad_url, bad_dom или bad_url_porn, но при этом отсутствует в good_url (последнее задается как !good_url – НЕ из good_url).

Такая конструкция с good_url нужна для того, чтобы случайно попавший в список сайт можно было включить в good_url. Например, на порнографическом сайте размещены материалы, которые сотрудники вашего музея антропологии исследуют по работе. Или баннерная система расположена на том же сайте, где и нужная документация. Кроме того, иногда шаблоны составляют не совсем корректно и хорошо иметь возможность быстро дать доступ к нужным материалам раздосадованному сотруднику. После того как он получит доступ, можно будет спокойно разобраться, почему же его любимый сайт рецептов молочных коктейлей соответствует шаблону из bad_url_porn.

Кроме баннеров, естественно, вы можете блокировать доступ к порносерверам, серверам-хранилищам музыкальных файлов и т. п. Политика ограничений в сети всецело в ваших руках.

Западный опыт показывает, что в коммерческих компаниях, где у сотрудников может возникнуть желание судиться с компанией ради денег, об ограничении доступа к информации через корпоративную сеть следует оповещать заранее и лучше письменно.

Запрещение передачи баннеров через **squid** приводит к неприятному побочному эффекту: вместо баннера **squid** выдает сообщение об ошибке доступа к ресурсу. Часто это выглядит как сообщение ERROR в позиции, где мог бы быть баннер. Это не всегда красиво. Как правило – омерзительно. В squid.conf можно задать параметр, определяющий то, какой файл будет выдаваться по ошибке доступа. Например, это может быть текст «блокированная информация» или даже пустое поле. Размещать там графику не следует, так как размер потенциального баннера может быть много меньше придуманной вами графической вставки. В этом случае форматирование страницы «покосится». Пустое поле идеально с точки зрения форматирования, но неинформативно. Возможно, взвесив все «за» и «против», вы решите ничего не менять в этой настройке **squid**.

Ограничение доступа в Сеть

В некоторых организациях считают полезным ограничивать доступ в Сеть некоторым категориям работников. Например, если сотрудники торгового зала будут посвящать рабочий день плаванию в волнах Интернета, продажи могут перестать расти, а торговый зал опустеет: сначала в нем не станет продавцов, а потом и покупателей. И установ-

ка терминала с бесплатным доступом в Сеть при входе в магазин к тому времени может уже не спасти положение. Во-первых, **squid** позволяет требовать аутентификации ото всех желающих пройти в Интернет и, во-вторых, ограничить посещение столь привлекательного виртуального пространства по времени. В одной из компаний, которые я консультирую в области работы с сетями, сотрудникам разрешен неограниченный доступ в Сеть, но только после 19:00. А в рабочее время – только если им известны имя и пароль главного бухгалтера. Который, разумеется, держит их в секрете.

Как ввести обязательную аутентификацию и как ограничить время доступа в Сеть?

Прежде всего, укажем программу, которая будет выполнять аутентификацию и файл, в котором лежит база имен и паролей:

```
authenticate_program /usr/local/squid/bin/ncsa_auth   /etc/
squidpasswd
```

Можно использовать и иные программы, эта поставляется вместе со **squid**. Файл `/etc/squidpasswd` подготавливается заранее, его можно создать с помощью **htpasswd** из пакета Apache или сделать, используя веб-интерфейс к **htpasswd**, например, по адресу

http://www.earthlink.net/cgi-bin/pwgenerator.pl.

Теперь определим соответствующие списки (ACL):

```
acl validusers proxy_auth REQUIRED    # у каждого спросят имя
                                      # и пароль
acl vitaluser proxy_auth vital
acl buhuser proxy_auth buh
acl vitaltime1 time 00:00-09:00
acl vitaltime2 time 18:00-23:59
```

А теперь применим их, указав, у кого и когда будет доступ в Интернет. Если указать строку

```
http_access allow validusers
http_access deny all
```

то все, кто аутентифицировался успешно (ввел имя и пароль, которые есть в источнике аутентификации), получат доступ без ограничений. А если вместо этого указать конкретные списки, в которых упомянуты определенные пользователи, то доступ получат только они:

```
http_access allow vitaluser vitaltime1 vitaltime2
http_access allow buhuser
http_access deny all
```

В обоих случаях те, кто не ввел верной пары имя/пароль, до Интернета допущены не будут.

Ведение и анализ протоколов

Протокол, который записывает **squid**, достаточно информативен. Он показывает, когда, откуда и что было запрошено, а также какого размера оно было и из какого источника **squid** пытался его взять.

Был ли запрошенный объект успешно передан клиенту, который его запросил, выяснить невозможно.

Пример протокола **squid**:

```
1041079441.642    540 192.168.5.20 TCP_MISS/200 405 GET
http:// hit1.hotlog.ru/cg
i-bin/hotlog/count? - FIRST_UP_PARENT/proxy.soros.spb.ru
text/ plain
1041079441.962      4 192.168.5.20 TCP_DENIED/403 1047 GET
http://images.edgenetwk.com/ads/125x125/survey125x125.gif -
NONE/- -
1041079442.072    109 192.168.5.20 TCP_IMS_HIT/304 217 GET
http://www.edgefiles.com/images/left-menubg-ns.gif - NONE/-
image/gif
1041079442.293    219 192.168.5.20 TCP_IMS_HIT/304 217 GET
http://www.edgefiles.com/images/left-menutopright.gif -
NONE/- image/gif
1041079442.294    110 192.168.5.20 TCP_IMS_HIT/304 217 GET
http://www.edgefiles.com/images/leftmenu-header.gif - NONE/-
image/gif
```

Для анализа протокола `access.log` удобно использовать небольшую утилиту на языке Perl. Мне она понадобилась для того, чтобы преобразовывать время в записях squid из стандартного машинного UNIX-формата в читаемый человеком формат даты. Естественно, подобный скрипт легко приспособить и для сбора статистики. Этот пример я предлагаю вашему вниманию. Данный скрипт, во-первых, выдает преобразованные строки с указанием времени в удобном формате на экран (в стандартный вывод) и, во-вторых, записывает в файл `report.txt` сведения о том, с каких машин сколько данных было скачано и сколько всего данных прошло через **squid**.

```
#!/usr/bin/perl
# открываем стандартный ввод, туда надо направить
# файл access.log при запуске скрипта
open(LOG, "<-") || die "Does stdin not exist?
# обнуляем счетчик переданных байтов
$traffic=0;
# обнуляем счетчик переданных байтов для каждой машины в сети
for ($i = 0; $i < 255; $i++)
          {
                        $TrafIP[$i]=0;
          }

while(<LOG>)
{
```

```
# читаем входной поток по строчкам и делим строки на поля
  ($time, $length, $ipaddr, $code, $smth, $action, $url) =
split(' ');
# преобразуем время к удобочитаемому формату
($sec,$min,$hour,$mday,$mon,$year,$wday,$yday,$isdst)=localt
ime ($time);
  $mon=$mon+1;
  $year=$year-100;
# выдаем в выходной поток преобразованную строку
# с информацией
  print "$mday.$mon.$year. $hour:$min: $action $length by
$ipaddr from $url\n";
# добавляем указанное в строке количество байт к счетчику
  $traffic=$traffic+$length;
# выделяем IP-адрес из строки
  $ipaddr =~ s/.*\.//g;
# добавляем к счетчику байтов, скачанных машиной с этим
адресом
  $TrafIP[$ipaddr]=$TrafIP[$ipaddr]+$length;
}
# закрываем входной поток
close(LOG);
# пишем поадресный отчет по трафику
open(REP,">/var/log/squid/report.txt")||die "Could not open
file";
for ($i = 0; $i < 255; $i++)
 {
 print REP $i, "\t", $TrafIP[$i],"\n";
 };
# завершаем отчет
print REP "TOTAL: $traffic\n";
close(REP);
```

Применяя этот скрипт (или подобный ему) для анализа трафика, имейте в виду:

Squid на самом деле пишет в протокол не то, что он скачал, а то, что у него запросили. Поэтому наличие в файле протокола запрошенного файла размером 16 Гбайт не означает, что такой файл действительно скачали. Возможно, его передача прервалась.

Скрипт предполагает, что все компьютеры, которые обращаются к **squid**, находятся в одной IP-сети класса C. Если это не так, модифицируйте блок подсчета индивидуальных рекордов по скачиванию файлов.

В сети, где этот скрипт применялся для анализа трафика, он дал достоверные результаты, что не гарантируется в других сетях.

Если вас волнует слишком высокий трафик вашей компании, обратите внимание также и на e-mail. Более 10% интернет-трафика может генерироваться письмами. Проверьте работу **sendmail**: не исключено, что вам следует поставить ограничения на максимальный размер письма или закрыть открытый релей.

22

UNIX в роли Windows NT (пакет Samba)

Сеть, в которую объединены компьютеры, управляемые различными операционными системами, называется гетерогенной. В таких сетях приходится работать довольно часто, поскольку на рабочих местах пользователей сети обычно установлены ОС фирмы Microsoft (Windows 9x, Windows NT, Windows 2000 и т. д.). В то же время многие серверные приложения (такие как системы управления базами данных, веб-серверы, почтовые серверы) часто работают под управлением UNIX.

Для совместной работы Windows- и UNIX-машин с одними и теми же файлами используют файловую систему CIFS (Common Internet File System). Фактически это виртуальная файловая система, так как она описывает правила посылки запроса файла или принтера через сеть и правила ответа на запрос, а не правила хранения данных на диске. В Windows и DOS эта файловая система была реализована с помощью протокола SMB (Session Message Block). SMB – это протокол для общего пользования файлами, принтерами, последовательными портами и коммуникационными средствами типа именованных каналов (named pipes) или мэйлслотов (mail slots).

Протокол SMB использует для передачи данных протокол нижнего уровня NetBT (NetBIOS поверх TCP/IP).

При работе NetBT используются TCP-порты 137, 138 и 139.

Файловая система CIFS реализована в UNIX с помощью пакета Samba, в который входят следующие программы: **`smbd`** – демон, представляющий собой аналог Server Service в Windows NT; **`nmbd`** – демон, являющийся аналогом Computer Browser Service в Windows NT; **`smbclient`** – ftp-подобная программа под UNIX для соединения и передачи файлов

между UNIX и сервером CIFS (например, **smbd** на любой машине в сети или «настоящим» сервером Windows NT); **smbstatus** – программа, сообщающая о состоянии демонов **smbd** и **nmbd** и соединениях с ними в данный момент. В пакет входят и некоторые другие программы. Файл конфигурации пакета `smb.conf` обычно устанавливается в каталог `/etc`.

Пакет Samba постоянно совершенствуется, текущую версию и подробную документацию можно получить по адресу *www.samba.org*. Как и подавляющее большинство полезных программ под UNIX, этот пакет распространяется бесплатно. Если ваша UNIX-машина установлена в уже работающей Windows-сети, то после установки Samba настраивать надо только UNIX-машину. Никаких дополнительных настроек Windows-компьютеров не потребуется. Они обнаружат, что в их сети появился еще один файловый сервер (или сервер печати – в зависимости от того, как вы собираетесь использовать Samba).

UNIX с установленным пакетом Samba может играть роль контроллера домена сети Microsoft. Помните, что домен с точки зрения сети Microsoft и домены Интернета (типа *yandex.ru*) – совершенно разные вещи. Старые версии пакета Samba были способны «прикидываться» только резервным контроллером домена, а новые умеют делать вид, что UNIX-машина является главным контроллером домена сети Microsoft.

При настройке пакета Samba надо четко представлять себе среду, в которой ему придется работать. Фирма Microsoft сделала удивительное число странных расширений (или сужений?) протокола SMB. В разных продуктах Microsoft используется пять (!) версий протоколов, с помощью которых данные передаются по сети. Имейте в виду, что разные версии Windows-систем иногда имеют противоречивые установки этих протоколов. Так, в Windows 3.11 for Workgroups все буквы пароля при передаче по сети всегда переводились в верхний регистр при использовании протокола LANMAN1 и не модифицировались в протоколе COREPLUS. Samba поддерживает все эти протоколы, в том числе COREPLUS. Все системы до Windows 98 (не включая ее саму) передавали пароль по сети открытым текстом, а начиная с Windows 98 стали его шифровать.

Если в сети совместно работают системы, шифрующие пароль и не шифрующие его, то придется либо включить в Samba шифрование паролей, либо научить Windows 98 сотоварищи не шифровать пароль. Последнее делается так:

- запускаете **regedit**;
- находите в реестре раздел `HKEY_LOCAL_MACHINE\System\CurrentControlSet\Services\VxD\Vnetsup`;
- вставляете туда новый параметр типа `DWORD` с именем `EnablePlainTextPassword`;
- присваиваете ему значение «1» (один).

Готово! Теперь Windows будет посылать пароли по сети незашифрованными. В старых версиях Samba был только такой способ, но он не самый лучший. Гонять по сети пароли открытым текстом не очень безопасно.

В одном из петербургских университетов системный администратор взял себе в помощь студента и дал ему пароль root на одной из UNIX-машин. Студент установил программу-сниффер (sniffer – слухач), которую можно бесплатно взять в Интернете. Через сутки у него в руках были все пароли всех пользователей сети, работавших как в UNIX, так и в Windows. После того как это обнаружилось, студенту было сделано строгое внушение, а системному администратору пришлось пересмотреть политику шифрования паролей в сети. С telnet все пользователи UNIX были переведены на ssh, а Windows 95 заменили на Windows 98.

Поэтому рассмотрите возможность установки на машины сети только тех систем Microsoft, которые поддерживают шифрование паролей, а вместо telnet лучше используйте ssh.[1]

Установка Samba выполняется весьма просто. Дистрибутив содержит подробные инструкции по установке и файлы с комментариями по разным аспектам работы с пакетом Samba. Читайте файл `INSTALL` перед установкой!

Единственное, что сравнительно сложно настроить при установке Samba, – это файл `smb.conf` – конфигурационный файл Samba. Им пользуются все программы этого пакета. Его надо модифицировать *перед* тем, как запускать smbd или nmbd. Команды запуска smbd и nmbd надо прописать в стартовые скрипты. Впрочем, можно запускать smbd и по запросу через `/etc/inetd.conf`. В SuSe Linux по неизвестным нам причинам такой способ запрещен.[2]

Файл `/etc/smbd.conf` всегда разбит на секции. Названия секций заключаются в квадратные скобки. Есть предопределенные секции – `[global]`, `[homes]` и `[printers]`. В случае если Samba будет работать как контроллер домена, может понадобиться секция `[netlogon]`.

Все остальные секции – это просто имена разделяемых ресурсов. Например, если Samba «прикидывается» сервером с именем swet, то секция `[usrman]`, описанная в `smb.conf`, будет описывать разделяемый ресурс `\\swet\usrman`.

Изучим файл `smb.conf` на живом примере. Файл обычно начинается с секции `[global]`, влияющей на работу Samba в целом.

---

[1] Одним из лучших эмуляторов терминала под Windows с поддержкой протокола SSH является программа putty. Ее можно скачать по адресу *http://www.chiark.greenend.org.uk/~sgtatham/putty/*.

[2] По крайней мере, в тех версиях SuSe Linux, которыми мы располагали.

```
[global]
# имя рабочей группы (домена) - LAB12
   workgroup = LAB12
# не использовать DNS для поиска имен, не известных WINS
# если сделать yes, как по умолчанию,
# остановка DNS подвесит smbd - сеть встанет
   dns proxy = no
# уровень отладки. Определяется вашим любопытством и
# серьезностью возникшей проблемы
   debug level = 1
# "сила" samb'ы при выборах master browser'a
# 255 - максимум
   os level = 255
# безопасность - на уровне пользователя
   security = user
# настройки сокетов. см. man setsockopt
   socket options = TCP_NODELAY
# с каких интерфейсов можно принимать запросы
   interfaces = \
     193.114.38.65/27 \
     193.114.38.33/27
# каким машинам дозволено обращаться
   hosts allow = 193.114.38.64/255.255.255.224
# быть броузером local master
   local master = yes
# быть броузером domain master
   domain master = yes
# стараться им быть во что бы то ни стало
   preferred master = yes
# как себя описывать в сети
   server string = Main File Server
# имя сервера (например, swet)
   netbios name = SWET
# сколько символов в пароле может быть в верхнем регистре[1]
   password level = 2
# быть контроллером домена - обслуживать
# входы пользователей в домен
   domain logons = yes
# запускать им logon script при входе
# скрипт помещен в файл имя_пользователя.bat
# и должен лежать в каталоге [netlogon]
   logon script = %U.bat
# кому поручить подсчет свободного места, если спросят
   dfree command = /usr/local/bin/smbdfree
# какой язык установить для имен в файловой системе
   character set = koi8-r
```

[1] Предположим, password level=2 и полученный от клиента пароль – FRED. Если он не подошел, Samba попробует еще «Fred», «fred», «fRed», «frEd», «freD». Пример взят из **man smb.conf**.

```
# какой язык использует клиент в его файловой системе
   client code page = 866
# сохранять ли регистр букв в именах файлов
   preserve case = yes
   short preserve case = yes

; Хотите, чтобы Samba была WINS-сервером
   wins support = yes
# у нее неплохо получается,

; если WINS-сервер уже есть, укажите его тут,
# только строку раскомментируйте
;   wins server = 192.168.1.1

# зарезервированный ресурс, появляется сам при входе
# пользователя
# путь к нему - домашний каталог пользователя UNIX

[homes]
   comment = Home directory
# должно быть видно при просмотре сети?
   browseable = no
# давать ли пользователю право на запись сюда?
   writeable = yes
# если guest ok, то ресурс подключается без пароля с правами
# гостевой учетной записи
   guest ok = no

[common]
# путь к ресурсу
     path = /usr/home/common
     guest ok = no
     comment = Common use directory
# read-only - антоним writeable
     read only = no
# с какими правами доступа создавать файлы и каталоги
     create mode = 0770
     directory mask = 0770
     browseable = yes
# ресурс для входа в сеть, если Samba - контроллер домена
[netlogon]
# где лежат logon-скрипты и политики
     path = /usr/data/netlogon
     guest ok = no
;     read only = yes

# зарезервированный ресурс, описывает принтеры
# принтеры должны существовать в UNIX
# и быть описаны в /etc/printcap
[printers]
   comment = All Printers
   browseable = no
# можно ли печатать на принтерах…
   printable = yes
```

```
# …кому попало?
   public = no
   read only = yes
   create mode = 0700
# куда скидывать spool
   directory = /tmp
# как удалять задания из очереди
   lprm command = /usr/bin/lprm -P%p %j
```

Оказавшись перед необходимостью решения любых нестандартных проблем, внимательно читайте **man smb.conf**. Пакет Samba отлично документирован.

Для выяснения состояния **smbd** дайте команду **smbstatus**:

```
/usr/bin/smbstatus
INFO: Debug class all level = 1   (pid 30903 from pid 30903)

Samba version 2.2.2
Service      uid      gid      pid     machine
----------------------------------------------

No locked files
```

Для снятия блокировок файлов изучите блокировки и PID процессов **smbd**, ответственных за них. Каждое новое соединение с сервером CIFS (Samba) обслуживается своим собственным **smbd**. Блокировка снимается посылкой сигнала KILL или STOP этому **smbd**. Будьте осторожны: такое принудительное снятие блокировок может привести к потере данных пользователем.

Для доступа к другим файловым серверам с той машины, где установлена Samba, а также для проверки доступа к локальным ресурсам используйте программу **smbclient**.

Самая свежая версия пакета Samba – 3.0.14a (на май 2005 года). В ней появилось много полезных возможностей, по сравнению с прежними версиями samba. Из тех, что не упомянуты выше, надо отметить отличную поддержку национальных языков в именовании файлов, использование расширенных прав доступа в файловой системе для эмуляции прав доступа, характерных в Windows-системах, возможность работы с LDAP в качестве источника аутентификации и учетной информации о пользователе (впрочем, это было и в версии 2.2.x), усовершенствована программа **smbclient** и значительно увеличена производительность **smbd**.

23

Установка и настройка СУБД MySQL

Установка MySQL

Для установки СУБД MySQL надо прежде всего обзавестить свежим дистрибутивом. Работать с залежалыми версиями не советую, поскольку старые экземпляры содержали весьма странные ошибки и не поддерживали некоторые базовые конструкции языка SQL.

Дистрибутив распаковывается командой

```
tar xzvf имя_файла_дистрибутива.tgz
```

В дистрибутиве есть файл `INSTALL` (или `INSTALL-SOURCE`), в котором написано все об установке **mysql**. В файле много информации, которая вам не понадобится. Надеюсь, что вы разделяете принципы, изложенные в главе 13. И поэтому скомпилируете **mysql** из исходных текстов. В этом случае ищите в файле `INSTALL-SOURCE` раздел `Installing a MySQL Source Distribution`.

Затем, как и для установки многих других пакетов, надо выполнить команды

```
./configure --prefix=/usr/local/mysql
make
make install
```

Эти команды надо выполнять, находясь в каталоге, куда раскрыт дистрибутив. Полный перечень того, что надо сделать для установки **mysql**, приведен ниже (включая распаковку архива; параметр `VERSION` замените на версию вашего дистрибутива).

```
groupadd mysql
useradd -g mysql mysql
```

```
gunzip < mysql-VERSION.tar.gz | tar -xvf -
cd mysql-VERSION
./configure --prefix=/usr/local/mysql
make
make install
scripts/mysql_install_db
chown -R root  /usr/local/mysql
chown -R mysql /usr/local/mysql/var
chgrp -R mysql /usr/local/mysql
/usr/local/mysql/bin/mysql_install_db
/usr/local/mysql/bin/safe_mysqld --user=mysql &
```

Последней командой вы запускаете **mysqld**.

Имеет смысл внимательно прочесть замечания в файле INSTALL-SOURCE, ибо они могут быть очень важны. Например, при компиляции **mysql** определенными версиями компилятора необходимо использовать указанные там ключи. Если этого не сделать, то получившийся демон **mysqld** будет неожиданно рушиться без предупреждения.

Для работы важны две программы: **mysqld** (собственно сервер баз данных) и **mysql** – программа, обеспечивающая простой текстовый интерфейс к нему. При настройке прав доступа вам может понадобиться еще и команда **mysqladmin**. Сервер БД обычно запускается при старте системы. Не забудьте вписать соответствующую строку в /etc/rc.local или /etc/inittab. Для назначения прав доступа к серверу БД, для корректировки таблиц, управления БД и т. п. удобна программа **mysql**. Для этой работы полезно знать язык SQL. Лучшая книга на русском языке по MySQL и языку SQL – это [9].

Сервер баз данных MySQL работает с базами данных, состоящими из таблиц. Каждая таблица состоит из записей. Каждая запись – из полей.

После установки MySQL существует только одна база данных – **mysql**. Она создается командой **/usr/local/mysql/bin/mysql_install_db**.

Настройка mysql: права доступа, создание БД

Сразу после установки MySQL нет иных пользователей, кроме root. В этот момент обращайтесь к **mysql** так: **mysql -u root**.

Сразу после установки вызовите **mysql** и установите пароль пользователя root, а также добавьте необходимых пользователей.

Создание баз данных выполняется командой **mysqladmin**. Обычно утилиты MySQL лежат в каталоге /usr/local/mysql/bin/.

В базу данных **mysql** входят таблицы db, user и host (и не только, но эти самые важные для нас). Эти таблицы определяют права доступа пользователей к базам данных. В таблице user перечисляются все пользователи, которые должны иметь доступ к серверу баз данных. Прежде чем начинать работать с базой, надо определить как минимум

одного пользователя, например root. Обязательно дайте ему пароль. Пароль задается открытым текстом при работе с **mysql**, а в базе данных он должен храниться в зашифрованном виде. Например, можно указывать пароль так:

```
SET PASSWORD FOR root@localhost = PASSWORD('q');
```

Можно указывать пароль при установке прав доступа к базе:

```
GRANT ALL PRIVILEGES ON *.* TO user@localhost IDENTIFIED BY 'q';
```

Первая звездочка означает «для всех баз данных», вторая – «для всех таблиц». Если до выполнения команды **GRANT** пользователь не был занесен в таблицу user, запись о нем будет автоматически добавлена.

Есть два способа добавлять пользователя и модифицировать информацию о нем: командой **GRANT** (или **SET PASSWORD**) или непосредственным редактированием таблицы user командами **INSERT** или **UPDATE**. В последнем случае не забудьте при указании пароля вызвать функцию **password** (она встроена в **mysql**):

```
INSERT INTO user (Host,User,Password)
VALUES('localhost','user',PASSWORD('q'));
FLUSH PRIVILEGES;
```

Вместо **FLUSH PRIVILEGES;** можно выйти из **mysql** и дать команду **/usr/local/mysql/bin/mysqladmin -p reload**.

Это нужно для обновления таблиц доступа к базам данных.

Ключ **-p** указывается для того, чтобы **mysql** и **mysqladmin** спрашивали пароль. Иначе не спросят и выдадут ошибку аутентификации. По умолчанию имя пользователя считается таким же, как и имя, от которого вы запускаете **mysql** или **mysqladmin**. Если вы работаете под именем ivan, а к **mysqladmin** хотите обратиться от имени root, укажите ключ **-u**:

```
/usr/local/mysql/bin/mysqladmin -p -u root reload
```

Если требуется, чтобы пользователь мог соединяться с данным компьютером из любого места, то наряду с указанием разрешенного для соединения компьютера localhost надо добавить разрешение соединяться отовсюду:

```
GRANT ALL PRIVILEGES ON *.* TO user@"%" IDENTIFIED BY 'q';
```

«Отовсюду» обозначается в таблицах доступа к **mysql** в виде знака процента. Мы привыкли в шаблонах ставить знак «*», чтобы показать, что допускается любое значение. Здесь роль звездочки играет процент. Его надо экранировать кавычками от специальной интерпретации.

Команда **GRANT** впервые реализована в MySQL версии 3.22.11. В более ранних версиях она не выполнит никаких действий.

Распространенные ошибки

При работе в **mysql** в конце каждой команды надо ставить точку с запятой. И чаще всего ошибка состоит в том, что это забывают делать.

Еще одна распространенная проблема – отсутствие прав доступа к таблицам базы данных, даже при том, что доступ был разрешен. Отсутствие доступа к таблицам означает, что надо дать права доступа пользователям, компьютерам и отдельно определить доступ к таблицам. В **mysql** защита реализована созданием трех основных таблиц: db (определяет права доступа к базе данных), user (дает права пользователю) и host (определяет права доступа с конкретных компьютеров). Обратите внимание, что **mysql** требует явно указывать разрешение доступа с localhost, не ограничиваясь разрешением доступа «отовсюду». Для **mysql** «отовсюду» значит «из любого места, кроме localhost». Некоторые версии требовали указывать полностью определенное доменное имя, а не просто имя компьютера, потому что mysql получал от функции **gethostbyaddr()** именно полностью определенное имя.

Иногда требуется добавить в таблицу базы данных информацию, которая уже содержится в каком-то файле. Если файл имеет регулярный формат, то есть похож на таблицу, можно импортировать информацию в таблицу командой **LOAD DATA INFILE**. Будьте внимательны с разделителями полей! Присутствие отличающихся друг от друга разделителей (например, и табуляций, и пробелов, и запятых) в разных строках файла-источника может вызвать путаницу. Возможно, для успешного добавления информации из файла в базу данных вам пригодится более подробная информация об этой процедуре, доступная по адресу *http://www.mysql.com/doc/en/LOAD_DATA.html*.

Иногда надо остановить длинный вывод на экран, если по команде **SELECT** выводится очень много строк. Делайте это, как везде в UNIX: <Ctrl>+<S>. Возобновление вывода: <Ctrl>+<Q>.

Основные операторы SQL

Основные операторы SQL (их синтаксис легко забыть, если вы работаете с SQL редко):

```
SELECT * FROM имя_таблицы WHERE имя_поля=значение;
```

Вместо «*» можно указать в скобках список полей для вывода. Значение, если это текст, должно быть заключено в кавычки.

```
INSERT INTO имя_таблицы [(список_полей)]
VALUES(список_значений);
```

Список – это значения, разделенные запятыми. Текстовые значения должны быть заключены в кавычки или апострофы:

```
INSERT INTO user (Host,User,Password)
VALUES('localhost','dummy','');
```

Эти операторы следует писать в одну строку, здесь просто не хватило ширины страницы.

```
UPDATE имя_таблицы SET поле=значение WHERE поле=значение
[AND поле=значение];
```

Пример **UPDATE:**

```
UPDATE user SET password=PASSWORD("newpass")WHERE user="
alina" AND host="%.mail.ru";
```

Если вы забыли структуру базы данных и само ее имя, то можете попросить **mysql** освежить вашу память. Для того чтобы просмотреть структуру базы данных, следует вызвать **mysql** и дать команды:

- **show databases** – показать список баз данных, хранимых на этом сервере;
- **use** *имя_базы_данных* – соединиться с базой *имя_базы_данных*;
- **show tables** – показать список таблиц этой базы данных;
- **show columns** – показать список полей базы данных.

Если вы создаете какую-то базу данных, старайтесь не придумывать ее структуру на ходу. Всегда храните на диске в надежном месте файл с командами SQL, которые могут при необходимости воссоздать структуру базы данных (имя самой базы, имена таблиц и полей, определения полей, то есть их типы и размеры).

Дополнительная литература

[10] Яргер Р. Дж., Риз Дж., Кинг Т. «MySQL и mSQL. Базы данных для небольших предприятий и Интернета». – СПб: Символ-Плюс, 2000.

По договору между издательством «Символ-Плюс» и Интернет-магазином «Books.Ru-Книги России» единственный легальный способ получения данного файла с книгой ISBN 5-93286-084-7 «UNIX. Практическое пособие администратора. 2-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» (www.symbol.ru), где именно Вы получили данный файл.

24

Аутентификация. PAM-модули

Аутентификация в любой программе

Аутентичность – это подлинность, соответствие подлинному, истинному. Например, если юная барышня спешит на свидание, надев откровенный топик[1], она, скорее всего, аутентична. Потому что ее одежда соответствует ситуации, она выглядит нефальшиво. Но если эта барышня в душе монахиня, то она сейчас неаутентична, потому что ее одежда не соответствует ценностям, которые она исповедует.

Аутентификация – это процесс, в ходе которого пользователь подтверждает системе свою аутентичность. Он делает это с помощью пароля, ключа или чего-то подобного. Он сообщает системе свое имя и пароль (ключ, код отпечатка пальца и т. п.). Если пароль соответствует имени, аутентичность подтверждена. Если нет, налицо попытка обмануть систему, выдать себя за другого.

Обычно аутентификация производится при попытке соединения с ресурсом, доступ к которому ограничен. После аутентификации может идти авторизация – предоставление пользователю определенных прав доступа к ресурсу. При авторизации личность пользователя влияет на то, какие именно права доступа он получит.

Каждый раз, входя в UNIX, вы подвергаетесь аутентификации. Программа login сопоставляет ваши имя и пароль с теми, что сохранены в `/etc/passwd`.

---

[1] Топик (уменьшительное от англ. top) – майка предельно простой формы на бретельках. Не следует путать этот топик с другим, так как на жаргоне увлеченных конференциями в сети «топиком» называют тему сообщения (англ. topic).

Обычно в UNIX выполняется стандартная аутентификация с использованием файла паролей (`/etc/passwd` или его производных типа `/etc/shadow`). Однако иногда требуется обеспечить единую аутентификацию на нескольких компьютерах и других устройствах. Таким устройством, например, может быть сервер удаленного доступа Cisco. Тогда может понадобиться создание единой базы аутентификации. Обращение к ней может происходить по протоколу TACACS (широко распространенному в серверах удаленного доступа Cisco) или RADIUS (в серверах удаленного доступа Nortel). Бывают и другие возможности аутентификации.

В прошлом не было единого механизма, который давал бы общий интерфейс аутентификации для любой сетевой службы и любого способа аутентификации.

Аутентификация с помощью PAM

В октябре 1995 года сотрудники SunSoft Самар (V. Samar) и Шемерс (R. Schemers) предложили стандарт PAM (Pluggable Authentication Modules).

В основу PAM были положены следующие принципы:[1]

- системный администратор должен иметь возможность выбрать, какой механизм аутентификации будет использоваться в системе по умолчанию: от простого ввода пароля до биометрической проверки или использования смарт-карт;
- для каждого приложения может быть выбран свой механизм аутентификации;
- система аутентификации должна работать с любым пользовательским интерфейсом – текстовым и графическим, удаленным или локальным;
- с каждым приложением может использоваться несколько механизмов аутентификации, например можно потребовать от пользователей аутентифицироваться и через Kerberos, и через RSA;
- системный администратор должен иметь возможность сделать аутентификацию прозрачной через несколько механизмов так, чтобы пользователь мог всего один раз набрать пароль;
- система аутентификации должна разрешать иметь несколько паролей;
- системные подпрограммы, полагающиеся на систему аутентификации, не должны зависеть от изменений этой системы;
- архитектура системы аутентификации должна быть модульной, позволяющей в случае необходимости вставить любой модуль с новым алгоритмом аутентификации;

[1] Принципы изложены по OSF RFC 86.0 October 1995.

- для обратной совместимости PAM API должен поддерживать существующие вызовы подпрограмм аутентификации.

Однако при разработке PAM специально не принимались во внимание следующие аспекты:

- что будет происходить после аутентификации (например, будут ли сторонние программы иметь доступ к результатам аутентификации);
- будет ли реализована однократная регистрация пользователя в сети (Single SignOn – когда пароль вводится один раз, и этого достаточно для аутентификации при доступе к любому сервису на любом компьютере в сети);
- как пересылать пароль через сеть, будет ли он послан открытым текстом.

Как устроен PAM

PAM– это набор библиотек, позволяющих гибко настраивать аутентификацию в UNIX. В настоящее время, насколько мне известно, PAM реализован в Linux, FreeBSD, Solaris 8, HP-UX и, по некоторым данным, в ряде других систем.

Основа PAM – это API (Application Program Interface), предоставляемый программам, требующим аутентификации (login, su, ftp и др.). Программа вызывает функцию PAM API. С другой стороны, PAM имеет SPI (Service Provider Interface), через который происходит трансляция вызова API к источнику аутентификации (файлу **/etc/passwd**, серверу TACACS и т. п.). API является общим для всех программ, а SPI содержит по одному модулю на каждый вариант аутентификации (один – через **/etc/passwd**, другой – через Kerberos и т. п.). Архитектура PAM изображена на рис. 24.1.

Посредством PAM системный администратор может установить свой способ аутентификации для любой сетевой службы. Более того, спектр способов весьма широк. Наиболее впечатляющей, на мой взгляд, является возможность аутентификации на основе базы данных о пользователях любого домена Windows NT.

Самой значительной возможностью PAM является гибкость настройки аутентификации. Системный администратор может указать любой способ аутентификации для любой программы. Правда важно, чтобы программа поддерживала аутентификацию через PAM. Например, веб-сервер может не уметь работать с PAM. Apache умеет это делать, хотя для этого требуется загружать отдельный модуль Apache.

Настройки PAM хранятся в файле `/etc/pam.conf` или в каталоге `/etc/pam.d/`. Если такой каталог существует, то подпрограммы PAM игнорируют содержимое `/etc/pam.conf`.

Следовательно, вы должны выбрать, как именно вы будете настраивать PAM. Многие современные системы устанавливаются с предопре-

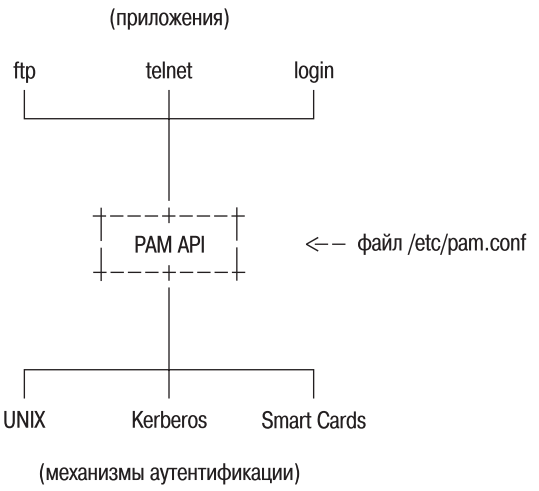


Рис. 24.1. *Архитектура PAM*

деленными настройками PAM. Так, FreeBSD 4.x устанавливается без каталога `/etc/pam.d`, но с уже готовым файлом `/etc/pam.conf`. В RedHat Linux 6.x ситуация обратная: при установке создаются файлы в каталоге `/etc/pam.d/`.

В файле `/etc/pam.conf` или файлах каталога `/etc/pam.d/` (по одному файлу на каждое приложение, использующее PAM) определяется соответствие между приложением (программой) и PAM-модулями, которые выполняют аутентификацию.

Модуль PAM представляет собой динамически присоединяемый модуль (файл с расширением `.so`).

PAM делит аутентификацию на четыре независимых задачи:

- управление учетными записями
- управление аутентификацией
- управление паролями
- управление сеансами

Эти задачи в конфигурационном файле идентифицируются ключевыми словами account, auth, password, session и соотносятся с типичными запросами пользователя к ресурсу, доступ к которому ограничен.

Ниже эти задачи описываются в соответствии с документацией по PAM:

- account – описывает службу проверки учетной записи: «не истек ли срок действия пароля? имеет ли пользователь право доступа к запрошенному ресурсу?»;
- authentication – описывает службу проверки идентичности пользователя. Эта служба обычно реализована посредством диалога меж-

ду пользователем и программой аутентификации: «представился таким-то? сообщи пароль!». Именно здесь от пользователя требуется сообщить верные и соответствующие друг другу имя и пароль (или иное подтверждение того, что пользователь аутентичен). Некоторые методы аутентификации (например, смарт-карты) не предполагают диалога, идентификацию выполняет специальная аппаратура, которая общается с написанным специально для нее модулем PAM. В этом проявляется гибкость PAM: старый добрый login может не задавать вопросов «login:», «password:». Не меняя программ, требующих аутентификацию, PAM позволяет аутентифицировать пользователя любым способом. В будущем появятся новые способы аутентификации, и их тоже можно будет реализовать через PAM-модули;

- password – описывает службу замены пароля; обычно такая служба тесно связана со службой проверки учетной записи. Например, по истечении срока действия пароля система требует у пользователя новый пароль;
- session – описывает работы, которые должны быть выполнены до того, как ресурс будет предоставлен, или после того, как он будет освобожден (например, после разрыва соединения между программой-клиентом и вызванной им службой-сервером). Например, это может быть протоколирование начала и конца соединения, монтирование домашнего каталога пользователя и т. п.

Файл(ы) конфигурации

Когда приложение, которое осуществляет аутентификацию с помощью PAM, запускается, оно инициализирует свое соединение с PAM API. Самой главной частью инициализации для нас является чтение файла конфигурации `/etc/pam.conf`. Или нескольких файлов конфигурации из каталога `/etc/pam.d/`, если такой каталог существует.

Файл(ы) конфигурации содержат список модулей PAM, которые будут использоваться для аутентификации.

Для начала рассмотрим вариант с единственным файлом конфигурации `/etc/pam.conf`.

Синтаксис этого файла прост и похож на синтаксис большинства файлов конфигурации в UNIX.

Файл `/etc/pam.conf` состоит из правил. Каждое правило записывается в одной строке, но может быть продолжено на другой. Для этого в конец продолжающейся строки надо поместить символ «\», который будет «экранировать» следующий за ним перевод строки.

Комментарии начинаются знаком «#» и продолжаются до конца строки.

Записи, соответствующие правилам, состоят из пяти полей, поля отделяются друг от друга пробелами, первые три поля безразличны к регистру букв.

```
service type control module-path module-arguments
приложение тип управление путь/к/модулю аргументы_модуля
```

Синтаксис файлов в /etc/pam.d/ идентичен синтаксису /etc/pam.conf, за исключением того, что первое поле «приложение» в них отсутствует. Вместо этого имя приложения дает имя названию файла в /etc/pam.d/. Так, аутентификацию веб-сервера описывает файл /etc/pam.d/httpd. Имя файла должно состоять только из букв нижнего регистра.

Имя приложения other зарезервировано для указания модулей аутентификации по умолчанию, то есть для тех приложений, что не перечислены явно в файлах конфигурации PAM.

Несколько правил могут быть объединены в стек для последовательной аутентификации несколькими PAM-модулями.

Поле «тип» – это название задачи, решаемой модулем, а именно account, auth, password или session.

Третье поле – «управление» – определяет поведение PAM в случае, если модуль примет решение об отказе в аутентификации (сочтет пароль неверным, например). Возможны варианты:[1]

- requisite – ошибка приводит к немедленному прекращению аутентификации;
- required – ошибка приводит к тому, что PAM API возвращает ошибку, но только после того, как будут вызваны все оставшиеся в стеке модули (для этого приложения);
- sufficient – успешная аутентификация достаточна для удовлетворения требований по аутентификации стека модулей (если предыдущий модуль в стеке выдал ошибку аутентификации, то успех аутентификации текущего модуля игнорируется);
- optional – успех или ошибка аутентификации с использованием данного модуля имеет значение, только если это единственный модуль в стеке, ассоциированном с данным приложением и типом аутентификации (то есть нет других модулей «optional»).

Путь к модулю – полное имя файла PAM-модуля для вызова приложением.

[1] Термин «ошибка» мы употребляем в описании вариантов действия PAM в смысле «ошибка пользователя при аутентификации, например указание неверного пароля».

Аргументы модуля – разделенные пробелами аргументы, которые могут использоваться для изменения поведения модуля. Могут быть у любого модуля и определяются в документации на конкретный модуль.

Динамическая библиотека PAM обычно находится в `/usr/lib/libpam.so.X`, а модули расположены в `/usr/lib/pam` или `/lib/pam` и называются `pam_*.so`.

Иногда и даже часто PAM-модуль поддерживает две (auth и account) задачи аутентификации. Однако некоторые модули ограничиваются поддержкой только auth. В то же время приложение иногда требует и поддержки задачи account. В такой ситуации приходится подставлять для выполнения этой задачи модуль **pam_permit.so**, который все разрешает без проверки. Если указать только одну задачу, аутентификация может не заработать вообще.

Аутентификация на UNIX через домен Windows-сети

Модуль PAM, обеспечивающий аутентификацию через домен сети Microsoft (Windows NT), называется **pam_smb_auth.so**. Кроме указания этого модуля в `/etc/pam.conf` или в `/etc/pam.d/` надо создать файл его настроек `/etc/pam_smb.conf`. Этот файл содержит три поля, по одному на каждой строке:

```
domain
PDC
BDC
```

Здесь `domain` – имя домена сети Microsoft, в котором надо аутентифицироваться, PDC – имя главного контроллера домена (Primary Domain Controller), BDC – имя резервного контроллера домена (Backup Domain Controller).

Модуль **pam_smb_auth** работает так: предполагается, что у аутентифицируемого пользователя есть учетная запись и в домене сети Microsoft, и в файле `/etc/passwd`. Если в `/etc/passwd` вместо пароля стоит звездочка (фактически запрет входа), то запрос на аутентификацию будет перенаправлен контроллеру домена сети Microsoft. Если в `/etc/passwd` – обычный зашифрованный пароль, то аутентификация пройдет с использованием `/etc/passwd`. Если вы хотите, чтобы аутентификация всегда шла через домен сети Microsoft, укажите модулю **pam_smb_auth** аргумент `nolocal`.

Рассмотрим пример аутентификации посетителей веб-сайта через домен Windows NT. Цель – ограничить доступ к некоторым страницам корпоративного сайта. В данном случае доступ должны получить все, кто зарегистрирован в корпоративном домене OLYMP, а те, кто не входит в него, не должны иметь право на доступ.

После установки модуля **pam_smb_auth** указываем в /etc/pam_smb.conf, в каком домене мы должны аутентифицировать пользователей httpd. ZEUS – это PDC, а AFINA – BDC. Если у вас нет BDC в домене, то: (1) это плохо, Microsoft рекомендует иметь резервный контроллер домена; (2) напишите то же имя, что и для PDC, PAM-модуль не обидится.

```
cat /etc/pam_smb.conf
OLYMP
ZEUS
AFINA
```

Посмотрим, что есть в каталоге /etc/pam.d/:

```
cd /etc/pam.d/
ls -l
total 11
-rw-r--r--   1 root     root       336 Oct 14  1998 chfn
-rw-r--r--   1 root     root       336 Oct 14  1998 chsh
-rw-r--r--   1 root     root       315 Nov  1  2001 ftp
-rw-r--r--   1 root     root       380 Mar 13 23:45 httpd
-rw-r--r--   1 root     root       384 Oct 14  1998 login
-rw-r--r--   1 root     root       210 May 23  1998 other
-rw-r--r--   1 root     root       239 Apr 27  1998 passwd
-rw-r--r--   1 root     root       284 Aug  5  1998 su
-rw-r--r--   1 root     root       333 Oct 10  1998 xdm
```

Среди файлов имеется httpd, он-то нам и нужен. Редактируем его до нижеприведенного состояния:

```
cd /etc/pam.d/
cat httpd
#%PAM-1.0
#[For version 1.0 syntax, the above header is optional]
#
# The PAM configuration file for the `httpd' service
#
auth     required   /lib/security/pam_smb_auth.so nolocal \
debug
account  required     /lib/security/pam_permit.so
```

Обратите внимание: **pam_smb_auth** (по крайней мере версия 1.1.6, которую я использовал) не имеет поддержки функции account, а **httpd** ее требует. Если не указать

```
account    required     /lib/security/pam_permit.so
```

то аутентификация вообще не будет выполнена.

Не забудьте указать в файле конфигурации Apache

```
AllowOverride AuthConfig
```

для каталога, в котором нужна аутентификация через PAM.

Также создайте в этом каталоге файл `.htaccess` с такими установками:

```
AuthType Basic
AuthName "secure area"
Require valid-user
AuthPAM_Enabled on
```

Самое главное – в последней строке.

Для работы через PAM потребуется указать веб-серверу, как обеспечить взаимодействие между ним и PAM API. В случае Apache надо просто загрузить модуль Apache, который умеет работать с PAM API. В файле конфигурации Apache следует указать

```
LoadModule pam_auth_module      libexec/mod_auth_pam.so
```

Разумеется, модуль **mod_auth_pam.so** должен быть уже установлен в системе к моменту загрузки Apache с такой директивой в файле конфигурации.

Аутентификация в других приложениях через PAM

Теперь поинтересуемся, как происходит аутентификация для ftp и login (обычный вход в системы через telnet или с консоли):

```
cat ftp
#%PAM-1.0
#auth       required    /lib/security/pam_listfile.so
     \ item=user sense=deny file=/etc/ftpusers onerr=succeed
auth       required     /lib/security/pam_pwdb.so
                                    shadow \ nullok
auth       required     /lib/security/pam_shells.so
account    required     /lib/security/pam_pwdb.so
session    required     /lib/security/pam_pwdb.so

cat login
#%PAM-1.0
auth       required     /lib/security/pam_securetty.so
auth       required     /lib/security/pam_pwdb.so
                                    shadow \ nullok
auth       required     /lib/security/pam_nologin.so
account    required     /lib/security/pam_pwdb.so
password   required     /lib/security/pam_cracklib.so
password   required     /lib/security/pam_pwdb.so
                            shadow \ nullok use_authtok
session    required     /lib/security/pam_pwdb.so
```

Настройки для ftp и login приведены по умолчанию.

Подробная информация о PAM доступна по адресу *http://www.kernel.org/pub/linux/libs/pam/*, как и свежие версии библиотек PAM.

25

Основы настройки X Window

Графическая среда системы UNIX совершенно отделена от самой системы. UNIX может легко работать без графической среды, чему и были посвящены все предыдущие главы. В этой главе мы очень кратко познакомимся с принципами работы и настройки X Window. Графическая среда весьма схоже работает во всех диалектах UNIX, поэтому дополнительную информацию можно получить в руководстве по любой системе либо в многочисленных FAQ и HOWTO, посвященных X Window.

X Window – это графическая среда, в которой есть как минимум следующие компоненты:

- X-сервер – программа, отображающая на экране пользователя информацию в графическом режиме. Обычно информация представляется в виде окон, очень похожих на окна Windows. Несмотря на некоторое внешнее сходство, между Windows и X Window нет практически ничего общего.
- X-клиент – программа, которая знает X Window API и умеет требовать от X-сервера вывести некий графический объект на экран.
- X Window API – набор системных вызовов и соответствующих библиотек для вывода графики и взаимодействия между X-серверами и X-клиентами.

Существует довольно много графических сред, основанных на X Window. Наиболее известны GNOME и KDE под Linux. В системах Solaris и HP-UX – среда CDE (Common Desktop Environment).

Главное, что следует уяснить администратору, работающему с X Window, состоит в том, что X-сервер всегда работает на том компьютере, где сидит пользователь. Даже если это обычный персональный компьютер, на нем может работать X-сервер. Лучший X-сервер под все системы Microsoft Windows – это Exceed.

X-клиент – это программа, как правило, работающая под UNIX. Она может выполняться на том же компьютере, где запущен X-сервер, показывающий графическую картинку пользователю, а может работать сколь угодно далеко от него. X-сервер и X-клиент легко взаимодействуют через любую IP-сеть, например через Интернет.

X-клиент направляет команды для X-сервера, руководствуясь значением переменной среды окружения DISLPAY. В ней указывается адрес компьютера, на котором запущен X-сервер, и номер (обычно 0), идентифицирующий сеанс этого X-сервера:

```
export DISPLAY="my.computer.home.com:0"
startx
```

Команда **startx** или **xstart** часто используется во многих системах для запуска X-сервера из командной строки. Можно указать, что X-сервер запущен локально, дав команду **DISPLAY="0:0"**.

При запуске X-сервера на персональном компьютере под управлением Linux или FreeBSD переключение между графическим терминалом X Window и виртуальными терминалами осуществляется комбинацией клавиш <Ctrl>+<Alt>+<Fn>, X-сервер по умолчанию запускается на седьмом виртуальном терминале <Alt>+<F7>.

Если что-то случилось и вам надо аварийно завершить работу X-сервера, это можно сделать, нажав клавиши <Ctrl>+<Alt>+<Backspace>, если вы находитесь в среде X-сервера.

Первым X-клиентом, с которым сталкивается пользователь Solaris, является **dtlogin**. Вообще, многие программы, работающие в Solaris с X-Window, называются «dtчто-то», где dt – это производное от DeskTop (рабочий стол).

X-клиенты могут поддерживать специфичные ключи, связанные с функциональностью этих программ. Однако есть и общие для всех X-клиентов ключи, перечисленные ниже:

-display display

этот параметр задает активный дисплей, т. е. адрес и экземпляр X-сервера, например:

```
-display host.my.domain.com:0
```

-geometry

этот параметр задает начальный размер и положение окна на экране, например, можно запустить программу **xterm** (X-терминал) следующим образом:

```
xterm -geometry 120x25+5+5
```

то есть запустить ее в окне размером в 120 столбцов по горизонтали, 25 строк по вертикали, с отступом в 5 строк и 5 столбцов от верхне-

го левого угла экрана (формат W×H+X+Y, где W – ширина, H – высота, X – отступ по горизонтали, Y – отступ по вертикали).

-bg color

обозначает цвет фона окна, например:

```
-bg yellow
```

-bd color
-bordercolor color

являются синонимами и задают цвет рамки окна приложения.

-bw number
-borderwidth number

задают ширину рамки в пикселах.

-fg color
-foreground color

задают цвет текста или выводимой в окне графики (по умолчанию).

-fn font
-font font

указывает шрифт, который будет использован для отображения текста в окне. Подробнее о шрифтах рассказано в **man xfn**; программа **xfn** применяется для управления шрифтами в графической подсистеме X-Window.

-rv, -reverse

требует от приложения выводить текст или графику в негативном отображении (меняя местами цвета фона и текста/графики); иногда имеет смысл на монохромных дисплеях и может не поддерживаться конкретной программой.

+rv

требует отказаться от вывода текста или графики в негативном отображении, даже если это задано по умолчанию, чтобы приложение могло работать на дисплее, который не поддерживает этот режим.

Есть еще несколько общих ключей, которые употребляются реже.

Настройка с помощью xf86setup

Каждый конкретный X-сервер требует настройки. Если X-сервер запускается под Windows, то можно почти ничего не настраивать. Самые важные настройки – частота кадров, количество цветов, частота горизонтальной развертки, шрифты и т. п. X-сервер возьмет их из уже готовых настроек Windows. Если же X-сервер запускается под UNIX, то для его настройки придется потрудиться. Установить систему так, чтобы настраивать X-сервер вовсе не пришлось, практически невозможно.

Настраивать придется как минимум:

- разрешение экрана;
- частоту строчной (горизонтальной) и кадровой (вертикальной) развертки;
- шрифты;
- параметры мыши.

Можно попытаться настроить все это с помощью программы **xf86setup**. Желательно перед тем, как это делать, прочесть man по ней.

Некоторые системы содержат более удобный вариант этой же программы **XF86setup**. Постарайтесь выяснить с помощью **man**, какой из этих настройщиков X-сервера наиболее хорош для вашей системы.

Самое полное и толковое руководство по настройке X Window из всех, что мне известны, написано Игорем Николаевым из СПбГУ. Руководство можно найти по многим адресам, например *http://knot.pu.ru/faq/xfaq.html*.

Русификация X Window

Русификация X Window состоит в русификации шрифтов и запуске переключателя клавиатуры. Русские шрифты поставляются со всеми современными системами, их также нетрудно найти в Интернете. Что до переключателя клавиатуры, то он должен вводиться в действие так.

В файле `/etc/X11/XF86Config-4` (возможно, он у вас называется иначе, например просто `/etc/X11/XF86Config`) найдите строки с нижеуказанными параметрами и измените (добавьте) их:

```
Section "InputDevice"
 ...
 Driver "keyboard"
# Переключаться по <Ctrl+Shift>
 Option "XkbOptions" "grp:ctrl_shift_toggle"
 ...
EndSection
```

Где узнать больше об X?

Книг по X Window, обобщающих весь необходимый опыт работы с этой системой в России, мы, к сожалению, не знаем. Довольно много информации содержится в вышеупомянутом руководстве от Игоря Николаева. В форумах на *www.opennet.ru* вы можете пообщаться с людьми, которые все время работают с X Window и часто в состоянии помочь советом.

Не забывайте о документации на X Window. Корневой каталог дерева всех программ, библиотек и документации X Window, как правило, – `/usr/X11R6`. В нем довольно много файлов, рассказывающих о том, как настраивать X Window. Обратите внимание на описание программы **dtlogin**, если вы работаете с CDE.

26

Русификация UNIX: обзор

Проблемы русификации UNIX

Русификация UNIX – очень непростое дело. Во-первых, идея локализации UNIX родилась много позже самой UNIX. Во-вторых, за это дело одновременно (или последовательно, что ситуацию только ухудшило) рьяно взялись несколько групп разработчиков. В результате коммерческие версии UNIX часто имеют свою собственную схему русификации. Существующий стандарт русификации в значительной степени поддерживается во FreeBSD и последних версиях Linux.

Однако русификация предполагает, что все программы UNIX должны нормально работать с русским языком. Это означает, что:

- пользователь может читать на экране русскоязычные тексты;
- пользователь может вводить русскоязычные тексты в текстовом редакторе или иным способом (например, с помощью команды `cat > textfile`);
- программы сортировки строк или слов в правильном порядке сортируют строки (слова), которые содержат буквы русского (кириллического) алфавита.

В UNIX нет целостной схемы русификации, которая после указания единственной настройки вела бы к русификации системы и всех программ в ней. Напротив, от того как написана каждая конкретная программа, зависит, возможна ли ее русификация.

Строго говоря, большинству пользователей достаточно, чтобы русифицированы были только командный процессор и текстовый редактор. По крайней мере, системному администратору этого вполне достаточно. Совсем хорошо, если текстовый броузер **`lynx`** тоже понимает русский язык.

Из соображений совместимости с другими UNIX-системами я не советую называть файлы по-русски, так как далеко не все системы поддерживают эту возможность. Однако возможность читать и писать по-русски под UNIX вполне можно себе обеспечить.

В этой главе мы рассматриваем только русификацию текстового (консольного) режима. Русификация графического режима (подсистемы X Window) кратко рассмотрена в главе 25.

Для русификации каждой отдельно взятой программы могут понадобиться свои собственные, специфические для этой программы способы.

Локализация и русификация

Русификация системы является частным случаем ее локализации. Все распространенные в мире операционные системы и очень многие программы так или иначе используются в разных странах. Это значит, что рано или поздно их разработчики сталкиваются с проблемой «понимания» операционной системой языка и других норм стран, где системы эксплуатируются.

Это серьезная проблема. Кроме языка в разных странах различаются форматы указания дат и времени, символ национальной валюты, форматы представления чисел (например, отделение тысяч от сотен запятой, точкой или пробелом).

Кроме этого, разные страны расположены в разных часовых поясах и отличаются друг от друга правилами перехода с зимнего времени на летнее и обратно. Все аспекты, связанные с временем в UNIX, рассмотрены в главе 18.

Под локализацией системы понимается адаптация операционной системы (или программы) к принятым в стране стандартам, а именно: языковым соглашениям, форматам дат и времени, формату представления национальной валюты и формату представления чисел. Под языковыми соглашениями прежде всего понимается соглашение о том, какие символы в наборе символов считаются буквами национального языка и в каком порядке они должны располагаться в таблице символов.

Ключевым понятием локализации является locale (произносится «локэйл», иногда употребляют русскую кальку «локаль»). Это набор вышеупомянутых соглашений о символах национального алфавита и их порядке, форматах даты, времени, валюты и чисел.

Данное понятие не определяется только и исключительно страной применения системы. Например, в Швейцарии могут быть определены три языка (Швейцария/немецкий, Швейцария/французский и Швейцария/итальянский). С другой стороны, один и тот же язык может входить в разные наборы соглашений (США/английский и Великобритания/английский).

Символы и кодировки

Кодировка – это совокупность уникальных символов, которые программа (или операционная система) отличает друг от друга, причем каждый из них имеет свое уникальное представление в этой программе.

Существует несколько кодировок кириллицы.

Первая кодировка – **ISO 8859-5**. Эта кодировка определяет вторую половину кодовой таблицы ASCII. Используется для кириллических текстов в Македонии и Сербии. В России распространения практически не получила.

Вторая кодировка – **CP866** (codepage 866). Также известна под названием «альтернативная кодировка». Она основана на кодировке Latin-1. Оригинальная Latin-1 во второй половине кодовой таблицы ASCII содержит диакритические символы, умляуты и другие символы европейских алфавитов, а также псевдографику. В CP866 все специфические европейские символы заменены на кириллицу, а все псевдографические символы остались нетронутыми. Это оставляет прежним вид программ, использующих для работы текстовые окна с псевдографикой, и позволяет отображать в них кириллицу. Имена файлов в FAT и VFAT кодируются в CP866.

Третья кодировка – **CP1251** (codepage 1251). Это стандарт кодировки второй половины таблицы ASCII, разработанный компанией Microsoft для кириллицы в Windows. Эта кодировка ни с чем не совместима. Старайтесь избегать ее применения в UNIX – это дурной тон. Единственный оправданный случай ее применения – веб-страницы. Поскольку они готовятся, как правило, в программах под Windows, применение CP1251 в них оправданно.

Четвертая кодировка – **KOI8-R**. Разработана в СССР, отличается тем, что символы русского алфавита в верхней части таблицы ASCII расположены так, чтобы их позиции соответствовали их фонетическим аналогам в английском алфавите в нижней части таблицы. Это означает, что если в тексте, написанном в KOI8-R, убирать восьмой бит каждого символа, то текст будет доступен для прочтения, хотя и будет написан «транслитом», то есть русскими словами, но латинскими буквами. Это очень удобно при использовании старых программ. Например, если почтовая программа обрезает при передаче письма 8-й бит, письмо все равно придет адресату доступным для прочтения. Существует также украинская версия кодировки KOI8 – KOI8-U. Кодировка KOI8-R чаще всего применяется в UNIX.

Пятая кодировка – **UNICODE**. Предполагает использование двух или четырех байт для кодирования одного символа, поэтому один шрифт UNICODE может содержать символы всех алфавитов (греческого, японского, латинского, русского и т. д.). Первые 256 символов UNICODE идентичны 256 символам кодировки Latin-1 (она же ISO8859-1).

В UNIX UNICODE еще не нашла широкого распространения, но во многих системах UNIX ее поддержка уже внедряется. Можно попробовать получить дополнительную информацию о UNICODE с помощью команды

```
man 7 utf-8
```

или по адресу *http://www.unicode.org*.

Русификация bash и vi

Поскольку мы договорились русифицировать командный процессор **shell** и текстовый редактор, разберем, как можно русифицировать **bash** и **vi**. Если их версии достаточно свежие, то для этого надо лишь активизировать набор параметров локализации (locale) для России.

Нижеследующая инструкция по активизации этого набора параметров годится только для систем, основанных на библиотеке glibc (в частности, для Linux и FreeBSD).

Если у вас версия библиотеки glibc меньше, чем 2.1.2, создайте в каталоге /usr/share/locale/ символическую ссылку с именем ru_RU.KOI8-R на файл ru_SU.

Затем вставьте в стартовые скрипты или в /etc/profile присвоение переменной LANG значения нужной кодировки, например:

```
export LANG=ru_RU.KOI8-R
```

В диалектах UNIX, основанных на RedHat, надо отредактировать файл **/etc/sysconfig/i18n**, где должна быть строчка

```
LANG=ru_RU.KOI8-R
```

Проверить, успешной ли была установка, можно, выполнив команду **locale**.

Такая установка дает побочный эффект: некоторые программы, например **date**, начинают говорить по-русски.

Более длинный, но более правильный путь состоит в создании отдельного каталога /usr/share/locale/ru_RU.KOI8-R/ (в некоторых диалектах он создается при установке системы).

Некоторые диалекты UNIX устанавливают по умолчанию значения

```
LANG=ru
LC_ALL=ru_RU.KOI8-R
```

Это не совсем верно. Установку LC_ALL лучше исключить, так как LC_ALL – макрос для установки всех параметров локализации. Не все программы корректно работают в таких условиях, более того, не всегда следует именно так устанавливать все параметры. Список параметров и их разумных установок приведен ниже:

- LC_CTYPE – язык одиночных символов
- LC_NUMERIC – формат чисел
- LC_TIME – формат времени
- LC_COLLATE – используется для сравнения строк
- LC_MONETARY – валюта
- LC_MESSAGES – системные сообщения
- LC_PAPER – формат бумаги
- LC_NAME – формат имен
- LC_ADDRESS – формат адресов
- LC_TELEPHONE – формат телефонов

```
LANG="C"
LC_CTYPE="ru_RU.KOI8-R"
LC_COLLATE="ru_RU.KOI8-R"
LC_TIME="C"
```

Для русификации **bash** и **vi** в свежих версиях FreeBSD и Linux этого достаточно.

Имейте в виду, что не все программы поддерживают работу с восьмибитовыми символами, попытка заставить такую программу общаться по-русски обречена на неудачу. Она не сможет работать ни с одной из кодировок национального алфавита.

Большинство современных программ корректно работают с восьмибитовыми символами.

Переключение клавиатуры по умолчанию выполняется во FreeBSD клавишей <CapsLock>, в Linux – <Ctrl>.

Перекодировка русскоязычных текстов

Конвертирование текстов из одной кодировки русского языка в другую выполняется, например, программой **iconv**. Она является частью общесистемной библиотеки libc (glibc) и потому гарантированно присутствует в любом диалекте Linux или BSD.

Синтаксис **iconv**:

```
iconv -f [исходная_кодировка] -t [целевая кодировка]
имя_файла
```

Эта программа выводит результат работы в стандартный поток вывода, поэтому для сохранения его в файле нужно выполнить перенаправление вывода:

```
iconv -f [исходная_кодировка] -t [целевая кодировка] \
файл-источник > файл-результат
```

Список доступных перекодировок можно посмотреть, дав команду

```
iconv -list
```

Довольно широко распространены программы **fromdos** и **todos** из пакета d1489 (автор Андрей Чернов). Пакет входит в дистрибутив FreeBSD.

Русскоязычные имена файлов

Файлы во многих системах UNIX можно называть по-русски, но обеспечение совместимости с другими диалектами UNIX, например, при скачивании файлов с вашего ftp-сервера, а также с SMB-клиентами, если ваш сервер выполняет роль файлового сервера и на него установлен пакет Samba, потребует дополнительных усилий. Придется как минимум внимательно изучить документацию по ftp-серверу и файл `smb.conf`. Несмотря на большие сдвиги в успешной локализации многих диалектов UNIX, я бы пока не ожидал появления универсального стандарта локализации, доступного для всех установленных систем. Впоследствии, когда все работающие системы будут оснащены поддержкой UNICODE, а все системы, не поддерживающие этот стандарт, будут выведены из обращения, именование файлов по-русски станет совершенно беспроблемным. Но на практике русскоязычные имена файлов нередко превращаются в наборы закорючек при переписывании их между серверами, не понимающими кодировок друг друга.

Русификация Solaris

В версии Solaris 9 русификация сделана весьма неординарным способом, поэтому здесь этой теме уделено специальное внимание. К сожалению, к моменту выхода этого издания книги испробовать русификацию Solaris 10 нам не удалось, но весьма вероятно, что нижеследующий материал поможет вам и в более свежих версиях Solaris.

При выборе русской локализации, отличной от RU.UTF-8 (поддержка кириллической страницы unicode), будет невозможно переключаться между языками при вводе текста в системе с помощью нажатия комбинации клавиш. Будет доступно переключение только с помощью мыши в специальном окне, ассоциированном с приложением. Такое «подокно» появится только в системных приложениях. Например, его не будет для приложения OpenOffice 1.1, которое вы, возможно, захотите установить дополнительно после установки системы.

В Solaris 9 реализована встроенная поддержка русского языка во всех кодировках, включая кодировку Unicode (UTF-8). Поэтому все про-

граммы, поставляемые с системой, корректно работают с русским языком. Однако для того, чтобы с русским языком мог работать клавиатурный переключатель, а также программа Netscape Navigator, следует при установке системы выбирать вариант локализации RU.UTF-8.

Как и во всех остальных системах UNIX, поддержка русского языка в любых программах, поставляющихся отдельно от дистрибутива системы, в исходных текстах или двоичном виде, зависит только от разработчика этих программ. Гарантировать корректность такой поддержки нельзя.

При установленной локализации RU.UTF-8 (UNICODE) переключение с английского алфавита на русский при вводе текста осуществляется нажатием клавиш <Compose>+<c>, <c>. Клавиша <Compose> на стандартной 104-клавишной клавиатуре находится между клавишей <Win> и правой клавишей <Ctrl>. Запись <Compose>+<c>, <c> означает, что сначала следует одновременно нажать клавиши <Compose> и <c>, а затем, отпустив их, нажать клавишу <c>. Для обратного переключения языка ввода на английский, следует нажать клавиши <Ctrl>+<пробел>.

Дополнительная литература

[11] RFC 1489 («Registration of a Cyrillic Character Set»).

[12] The Linux Cyrillic HOWTO (russian) *http://www.inp.nsk.su/~baldin*.

[13] Документация по locale *http://www.sensi.org/~alec/locale*.

[14] Документация по locale в X Window *http://www.tsu.ru/~pascal/x_locale/*.

27

Наблюдение за системными ресурсами

Для наблюдения за системными ресурсами в BSD и системах System V принято использовать разные программы: в BSD – **vmstat**, в System V – **sar**. На самом деле вам может встретиться или **vmstat**, или **sar**. В Solaris, который, несомненно, относится к ветви System V, **sar** отсутствует, зато есть **vmstat**.

И **sar**, и **vmstat** по умолчанию запускаются так:

sar *интервал число_измерений*

или

vmstat *интервал число_измерений*

Например, если вы хотите получить десять «мгновенных снимков» состояния операционной системы с интервалом в одну секунду в BSD-системе, введите:

```
vmstat 1 4
procs  memory     page                  disks   faults     cpu
r b w  avm   fre  flt re pi po fr sr    ad0 ad2 in  sy  cs us sy id
0 1 0  52488 1548 11  5  3  3  10 315   83  0   559 416 14 1  9  90
0 1 0  52488 1524 70  17 4  17 7  1807  20  0   260 37  38 1  3  96
0 1 0  52488 1504 7   1  0  0  0  0     0   0   233 21  7  0  1  99
0 1 0  52488 1496 22  11 14 3  22 657   17  0   249 29  26 1  1  98
```

Расшифровка полей вывода:

- `procs` – информация о числе процессов в разных состояниях
- `r` – стоят в очереди к процессору

- b – блокированы в ожидании ресурса (завершение операции ввода/вывода и т. п.)
- w – запущены или спят менее 20 с, но выгружены на диск
- memory – использование виртуальной и оперативной памяти, объем виртуальных страниц в килобайтах
- avm – активные виртуальные страницы. Страница считается активной, если она принадлежит процессу, который запущен сейчас или был запущен в течение предыдущих 20 с
- fre – свободная память
- page – сведения о свопинге в страницах в секунду, среднее по последним пяти секундам
- flt – число «непопаданий» в кэш
- re – page reclaims (simulating reference bits)
- pi – с диска прочитано страниц
- po – страниц выгружено на диск
- fr – освобождено страниц
- sr – сканировано страниц в соответствии с алгоритмом часов
- disks – выполнено обращений к диску
- ad0 – первый IDE-диск (диск master на контроллере IDE primary)
- ad2 – третий IDE-диск (диск master на контроллере IDE secondary)
- faults – средняя за последние 5 с интенсивность прерываний (в секундах)
- in – число прерываний, включая прерывания от таймера
- sy – число системных вызовов
- cs – число переключений контекста CPU
- cpu – использование процессорного времени
- us – время, занятое пользовательскими процессами нормального и низкого приоритета
- sy – системное время (время, в течение которого выполнялся код ядра)
- id – время простоя процессора

Во FreeBSD есть еще программа **systat**, которая при запуске с ключом **vmstat** дает наглядную динамически меняющуюся картину занятости системных ресурсов:

```
systat -vmstat
1 users    Load  0.04  0.01  0.00                    Jun  1 23:21

Mem:KB    REAL            VIRTUAL            VN PAGER  SWAP PAGER
Tot   Share      Tot    Share    Free        in  out     in  out
Act  104792    3552   125424     5444   30808 count
All  498872    6004  4175732     9456         pages
                                                      Interrupts
```

```
Proc:r  p d s w  Csw Trp Sys Int Sof Flt         cow   262 total
          2 12    45  3  203  262 17  2   71020  wire   stray irq7
                                          129544  act    vr0 irq12
0.5%Sys 0.2%Intr 0.2%User 0.0%Nice 99.2%Idl 273140  inact  34 mux
                                                            irq11
|   |   |   |   |   |   |   |   |   |     25168  cache  ata0 irq14
                                           5640   free   ata1 irq15
                                                  daefr  fdc0 irq6
Namei         Name-cache    Dir-cache              prcfr  sio0 irq4
    Calls     hits    %     hits    %               react  100 clk
                                                             irq0
                                                   pdwak  128 rtc
                                                             irq8
                                     zfod          pdpgs
Disks   ad0  acd0   fd0   md0       ofod          intrn
KB/t   0.00  0.00  0.00  0.00      %slo-z  61232  buf
tps       0     0     0     0      tfree       8  dirtybuf
MB/s   0.00  0.00  0.00  0.00              35552  desiredvnodes
% busy    0     0     0     0              32243  numvnodes
```

В любой UNIX-системе есть следующие объекты, за состоянием которых надо следить:

- Процессор. Чрезмерная загрузка процессора говорит о неполадках в системе. Если не обратить на это внимания, система может перестать отвечать на внешние запросы.
- Память. Нехватка оперативной памяти ведет к росту объема свопинга. При критической нехватке оперативной памяти система вообще перестает делать что-либо, кроме свопинга, «уходит в своп». Нехватка виртуальной памяти ведет к тому, что операционная система начинает «отстреливать» процессы для того, чтобы иметь возможность продолжать работу. В такой ситуации возможны непредсказуемые последствия, обычно требующие перезагрузки системы. Захват большого объема памяти одной программой должен вызвать вопрос о причинах этого. Если эта программа исправна и должна постоянно работать с большим объемом памяти, может быть, нужно подумать об увеличении общей памяти системы.
- Прерывания от внешних устройств. Если число прерываний намного больше обычного, это может говорить о ненормальной активности, например в сети. На такие события лучше вовремя обратить внимание: возможно, сервер вскрывают; если на обработку прерываний тратится больше времени, чем обычно, это может говорить о неисправностях устройств или неполадках с драйверами.
- Сеть. Активность сети надо отслеживать особенно внимательно. Соединения с неизвестными вам IP-портами, перекачка необычно больших объемов данных, необычно большое количество установленных соединений – все это может говорить о попытках вскрыть вашу систему, а также о возросшей активности в сети. В первом случае надо защищать систему от злоумышленников (или очищать

ее от них – в зависимости от того, на каком этапе вы обратили внимание на проблему). Во втором – подумать о распределении нагрузки в вашей сети и, возможно, о ее перепланировке. О мониторинге сети подробнее рассказано в главе 10.

- Диск. Дисковое пространство не должно быть все время на грани заполнения. При идеальном распределении разделов UNIX на дисках занятое пространство в каждом разделе никогда не превышает 70% объема раздела. Если диск заполнен на 100%, система перестает стабильно работать. В зависимости от того, какой раздел заполнился, возникают более или менее серьезные проблемы.

Любой демон в UNIX может вести файл протокола обращений к нему. Демон **syslogd** записывает в системные файлы протокола информацию о событиях в системе в соответствии с файлом конфигурации `/etc/syslog.conf`. Если заканчивается место в разделе, где расположен файл протокола, дестабилизируется работа демона. Некоторые процедуры не могут быть завершены при невозможности их протоколирования, например вход пользователя в систему. Из этого следует, что нехватка места в разделе, где лежит файл протокола входов в систему, может вызывать невозможность войти в нее.

- Внешние устройства. При неисправности внешнего устройства система начинает сообщать об этом. Иногда довольно настойчиво. И обижается, если на сообщения не реагируют. Обида может выражаться в увеличении частоты предупреждений о сбое. Чтобы отслеживать такие ситуации, читайте файлы протокола и сообщения на консоли. Не создавайте сбойных ситуаций сами. Например, если вы смонтируете дискету в дисководе, а потом вытащите ее, не дав команды **umount**, это может привести к тому, что система до следующей перезагрузки будет сообщать о сбое дисковода. При этом даже вставление дискеты обратно может не помочь, так как система фиксирует факт исчезновения смонтированного устройства и не обращает внимания на такие мелочи, как его появление.

Для наблюдения за большинством этих объектов отлично годится программа **top**. Она показывает процессы с самым большим аппетитом на процессорное время. Кроме сведений о процессах **top** выводит общую информацию по системе. Программа – интерактивная, показывает текущее состояние системы и процессов. Выход из программы – <Ctrl>+<C> или <q>. Некоторые версии **top** в некоторых диалектах UNIX позволяют управлять процессами из этой программы: задавать для них значение базового приоритета, убивать процесс или посылать ему определенный сигнал.

```
top
1:36am up 63days, 5:14, 1 user, load average: 0.18,0.16, 0.10
29 processes: 27 sleeping, 2 running, 0 zombie, 0 stopped
CPU states:  6.3% user,  1.3% system,  0.0% nice, 92.2% idle
```

```
Mem:190616K av, 187176K used, 3440K free, 0K shrd,19856K buff
Swap:  104416K av,   84348K used, 20068K free 139140K cached

PID  USER  PRINI SIZE RSS  SHARE STATLIB %CPU%MEM TIME   COMMAND
7642 squid 16 0  7607 273M 61936 S    0   4.9 39.2 843:50 squid
11936 root 11 0  1656 16361160 S    0   1.3 0.8  0:00   sendmail
5589 root  10 0  5104 31323132 S    0   0.7 1.6  58:15  named
11935 filip 11 0 864  864 692  R    0   0.5 0.4  0:00   top
1    root  0  0  112  76  76   S    0   0.0 0.0  0:26   init
2    root  9  0  0    0   0    SW   0   0.0 0.0  0:00   keventd
3    root  9  0  0    0   0    SW   0   0.0 0.0  15:24  kswapd
4    root  9  0  0    0   0    SW   0   0.0 0.0  0:00   kreclaimd
5    root  9  0  0    0   0    SW   0   0.0 0.0  0:01   bdflush
6    root  9  0  0    0   0    SW   0   0.0 0.0  0:07   kupdated
9    root  9  0  0    0   0    SW   0   0.0 0.0  0:00   scsi_eh_0
10   root  9  0  0    0   0    SW   0   0.0 0.0  0:00   khttpdmanag
303  root  9  0  224  164 164  S    0   0.0 0.0  5:42   syslogd
311  root  9  0  636  4   4    S    0   0.0 0.0  0:00   klogd
324  root  8  0  180  124 108  S    0   0.0 0.0  0:03   crond
337  root  9  0  104  64  60   S    0   0.0 0.0  0:00   inetd
389  root  9  0  60   4   4    S    0   0.0 0.0  0:00   mingetty
```

Для получения информации о дисках используйте команды **mount**, **df** и **du**. Программа **mount** позволяет выяснить, какие внешние устройства (накопители) подсоединены к компьютеру и смонтированы в файловую систему. По команде **df** выдается информация о занятых и свободных блоках на разделах UNIX. Обычно **df** выдает информацию в килобайтах, но там, где **df** считает блоками, можно выполнить команду **df -k** для получения расчета в килобайтах.

```
df
Filesystem  1K-blocks     Used    Avail Capacity  Mounted on
/dev/ad0s1a    508143    39227   428265     8%    /
/dev/ad0s1f   7071948  4083618  2422575    63%    /usr
/dev/ad0s1e   2032623   380433  1489581    20%    /var
procfs              4        4        0   100%    /proc
```

Команда **du** сообщает о том, сколько места на диске занимает заданный файл, каталог или дерево каталогов. Для того чтобы узнать, сколько места занимает каждый подкаталог текущего каталога, надо дать команду **du** с ключами и аргументами:

```
du -s  *      #     System V
du -d 1       #     BSD
```

Пример:

```
cd /var  # посчитаем, сколько места занимают каталоги в /var
du -d 1
```

```
1       ./account
3       ./at
41      ./backups
2       ./crash
5       ./cron
3996    ./db
6023    ./log
235431  ./mail
2       ./msgs
1       ./preserve
54      ./run
1       ./rwho
20615   ./spool
21132   ./tmp
20      ./yp
62      ./games
8835    ./cache
1       ./lock
296226  .
```

О состоянии swap-разделов рапортует программа **swapinfo** (или **pstat -s**).

```
swapinfo
Device       1K-blocks   Used   Avail  Capacity  Type
/dev/ad0s1b      98176  31152   67024     32%    Interleaved
```

Чтобы узнать, как давно работает система и с какой нагрузкой, используйте **uptime**. Эта программа так же, как и вызов **w**, сообщает среднюю нагрузку за последние 1, 5 и 15 минут соответственно:

```
uptime
 1:21PM  up 52 days, 14:52, 1 user, load averages: 0.01,
0.08, 0.06
```

Вызов **w** дает несколько больше информации, сообщая, кто в данный момент работает с системой интерактивно:

```
w
 1:21PM  up 52 days, 14:52, 1 user, load averages: 0.01,
0.08, 0.06
USER        TTY      FROM              LOGIN@  IDLE WHAT
filip            p0       ip-100.co.spb.ru  1:11PM     - w
who
filip            ttyp0   Aug 13 13:11   (212.193.6.100)
```

Программа **statserial** показывает состояние физических линий последовательного интерфейса, позволяя определить, жив ли подключенный к нему модем или другое устройство. Приводимый ниже вывод **statserial** показывает состояние последовательного порта, к которому подключен исправный модем, который в данный момент не активен (нет передачи данных и нет входящего звонка). На примере системы Linux:

```
statserial /dev/ttyS0

Device: /dev/ttyS0

Signal  Pin  Pin  Direction  Status  Full
Name    (25) (9)  (computer)         Name
-----   ---  ---  ---------  ------  -----
FG       1    -      -           -   Frame Ground
TxD      2    3      out         -   Transmit Data
RxD      3    2      in          -   Receive  Data
RTS      4    7      out         1   Request To Send
CTS      5    8      in          1   Clear To Send
DSR      6    6      in          1   Data Set Ready
GND      7    5      -           -   Signal Ground
DCD      8    1      in          0   Data Carrier Detect
DTR     20    4      out         1   Data Terminal Ready
RI      22    9      in          0   Ring Indicator
```

Для получения подробной информации о сетевых соединениях следует использовать программу **netstat**, описанную в главе 10. Также полезна программа **lsof**, выдающая информацию о том, какой процесс открыл те или иные файлы или сетевые соединения.

Помните также о широких возможностях **ifconfig**, которые, однако, проявляются не во всех диалектах UNIX. Так, в Linux эта программа выдает информацию не только о настройках интерфейса, но и статистику ошибок и успешных передач пакетов через интерфейс:

```
ifconfig
eth0      Link encap:Ethernet  HWaddr 00:0E:0C:4F:19:C0
          inet addr:192.168.116.254  Bcast:192.168.116.255
Mask:255.255.255.0
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:766826 errors:0 dropped:0 overruns:0
frame:0
          TX packets:916471 errors:0 dropped:0 overruns:0
carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:159709388 (152.3 MiB)  TX bytes:687588041
(655.7 MiB)
          Base address:0xdc00 Memory:fc5e0000-fc600000
```

Если в вашем ведении находится много UNIX-машин, рано или поздно может встать задача определения, какое именно оборудование в них установлено. И если заветная тетрадка или база данных с описанием аппаратных конфигураций всех компьютеров куда-то затерялась (или ее никогда не было в природе, будем откровенны?), пригодятся программы диагностики и определения оборудования.

Обратите внимание на программы **lspci** (Linux) и **pciconf** (FreeBSD) для получения списка установленных устройств на шине pci и на программы **hdparm** (Linux) и **atacontrol** (FreeBSD) для получения низкоуровневой информации о жестких дисках.

28

Предотвращение сбоев

Гром не грянет – мужик не перекрестится.
Пословица

Лучший способ борьбы со сбоями – их предотвращение. Эта простая истина часто не в почете у системных администраторов. Попробуем нарушить традицию и начнем применять на практике все, что поможет нам избежать сбоев.

Следующий абзац я прошу вас показать самому главному начальнику в вашей компании.

Некоторые системные администраторы страдают от низкой оплаты их труда. Еще сильнее они страдают от того, что труд их незаметен и не оценивается другими сотрудниками так, как должен. Долгие ночные бдения за монитором не вознаграждаются всеобщей любовью, если все работает как часы. Поэтому некоторые системные администраторы (чаще подсознательно, чем сознательно) делают не все возможное, чтобы системы и сети были безотказны. На самом деле это очень трудно: спроектировать, построить и регулярно обслуживать сеть так, чтобы ни сеть в целом, ни один из компьютеров в ней в отдельности никогда не выходили из строя. Поэтому в сбоях сети нельзя всегда видеть чей-то злой умысел. С другой стороны, работу системного администратора должно оценивать по отсутствию проблем. Если вы не знаете вашего системного администратора в лицо, если он вам ни разу не понадобился за последние полгода, если компьютеры всегда работают, знайте: ваш системный администратор – профессионал высочайшей квалификации.

Если же системный администратор – частый гость в каждом уголке сети, куда его нервно зазывают отчаявшиеся пользователи, знайте: либо отдел информационных технологий получает слишком мало денег, либо квалификация системного администратора стремится к нулю.

Частые поломки оборудования и сбои программного обеспечения говорят о плохом планировании сети.

Плохое планирование не всегда вызвано дурными намерениями персонала ИТ-отдела. Более того, как правило, оно связано с плохой организацией работы в целом: отсутствием четких целей, частыми авралами, недостатком финансирования, неправильной кадровой политикой и т. п.

Если системный администратор в состоянии быстро исправить любую поломку в сети, это замечательно. Но намного лучше, если он умеет предвидеть и предотвращать такие поломки.

Поэтому сначала мы поговорим о том, как не стать героем трагикомедии «Крах сервера, или Пять ночей без сна».

Во-первых, всегда создавайте резервные копии всех конфигурационных файлов вашей системы. Храните их вдалеке от источников опасности: электромагнитов, источников тепла, детей, домашних животных, мест потенциальных протечек. Не загромождайте резервными копиями запасные выходы! И остальными предметами тоже. Носители с резервными копиями должны легко читаться где угодно, а храниться в надежном помещении. Желательно одну копию держать рядом с сервером, а еще одну – в соседнем здании (селе, городе). Для надежности. Записывать резервные копии конфигурационных файлов лучше на дискету или компакт-диск. Обычно дискеты вполне хватает. Дискету надо надписывать, чтобы по ней сразу было видно, что за конфигурационные файлы на ней записаны и когда. Конфигурационные файлы удобно хранить в одном каталоге из соображений удобства резервного копирования. На практике обычно есть несколько каталогов с файлами конфигураций – `/etc`, `/usr/local/etc` и каталоги конфигураций отдельных пакетов – `/usr/local/apache/conf`, `/usr/local/squid/etc` и т. п. Лучше всего добиваться сосредоточения всех файлов конфигураций (или символических ссылок на них) в одном каталоге и его подкаталогах. Традиционно – в `/etc`.

Во-вторых, следите за работой системы. Для этого есть программы **`top`** (описана в главе 27) и **`trafshow`** (глава 10). Последняя помогает познакомиться с частыми видами трафика через ваш UNIX-сервер. Если запустить **`trafshow`** на маршрутизаторе, то можно легко исследовать работу сети. Наличие подозрительных соединений всегда должно вызывать у вас тревогу. Если вы знаете, что ни одна программа в вашем компьютере не принимает запросов на такой-то порт, а **`trafshow`** показывает интенсивную передачу данных в этот порт или из этого порта, не думайте, что это ошибка. Вас взломали. Пора закрывать дыры в системе. Главное в такой момент – понять, что взломали, какую именно программу. И действовать соответственно. Верный признак сбоя – вы не можете войти в систему. Если вы просто забыли пароль, то это не считается. Правда пароль мог поменять доброжелатель, уже взломавший систему.

Наличие в системе давно и прочно повисших процессов-зомби (см. главу 6) говорит об ошибке ядра или процесса `init`. Это может быть сигналом к тому, что пора обновить систему или, по крайней мере, ядро.

Не пренебрегайте командами `df`, `swapinfo` и `top`. Они нужны для того, чтобы проверить, сколько свободного места осталось на диске, в swap-разделе и в оперативной памяти. Помните о том, что уменьшение свободного места на диске до значений, меньших 30% объема диска, может замедлить работу системы. И чем меньше места, тем хуже будет дело. Полностью забитый swap говорит о том, что:

- одновременно загружается слишком много программ (может быть, вы что-то ненужное грузите по умолчанию?);
- какая-то программа активно поглощает память (нужна ли вам такая программа? Если да, то, может быть, стоит подумать об увеличении оперативной памяти?);
- у вас слишком маленький swap-раздел;
- происходит что-то неучтенное нами, но вряд ли хорошее.

Заполненная оперативная память – это нормально, так как свободное место в памяти в UNIX обычно отдается под файловый кэш. Однако если суммарный размер программ, которые должны одновременно работать на вашем компьютере, явно превосходит объем оперативной памяти, не раздумывая делайте одно из двух: либо перепланируйте нагрузку на компьютеры в вашей сети так, чтобы на компьютере с недостаточным объемом памяти запускалось меньше программ, либо добавьте в него память.

Если сбои питания не редкость для вашего компьютера, то для большей надежности пользуйтесь журналируемой файловой системой, например ext3. Тогда после сбоя файловая система автоматически вернется к состоянию, в котором она была до сбоя (кое-что пропадет, но не будет попорченных, недозаписанных на диск файлов).

Если вы сомневаетесь в целостности файловой системы, запускайте `fsck`. Поскольку обычно она запускается при старте системы, ручной запуск имеет смысл, только если автоматический сорвался. Например, `fsck` не справилась с исправлением ошибок и ей требуется интерактивное общение с оператором. Кроме того, `fsck` может пригодиться для ремонта заведомо поврежденных файловых систем.

Обращайте внимание на программы, запущенные от имени тех, кто обычно ничего не запускает. Вообще обращайте внимание на необычное: будьте любопытны! Верьте: воздастся сторицей. Например, системный администратор одного академического учреждения в Петербурге в течение трех суток не интересовался тем, что происходит в его системе. Он был неприятно удивлен, обнаружив во вторник, что в предшествующие три дня неизвестные нашли ошибку в настройке его сервера. Он случайно разрешил доступ к службе ftp анонимным пользователям. В результате на диск его сервера было записано не-

сколько гигабайт музыкальных файлов. Многие неизвестные успели скачать их оттуда. За входящий трафик, превысивший гигабайт в сутки, эта организация заплатила столько, сколько собиралась заплатить за целый год.

Человеческий фактор

Все мы – люди. Ничто человеческое нам не чуждо. Поэтому мы должны помнить о рамках, в которых должны держаться. По крайней мере, на работе. Все, что перечислено ниже, делать НЕЛЬЗЯ:

- в нетрезвом виде менять пароль суперпользователя: забудете, причем сразу;
- редактировать конфигурацию фильтра пакетов (firewall), находясь далеко от него;
- менять настройки систем, находясь в отпуске (придется ехать на работу, если попортите что-нибудь ненароком);
- удаленно перезагружать компьютер, если что-то серьезное было изменено в настройках (например, IP-адрес интерфейса, через который вы с ним работаете);
- оставлять пароль root пустым;
- разрешать пользователям придумывать простые пароли;
- идти на поводу у пользователей, особенно малограмотных;
- убивать полезных демонов. Особенно – `init`, при работе удаленно – `inetd`, локально – `klogd`;
- давать пароль суперпользователя нескольким людям. Этот пункт появился с легкой руки моего друга, который сам считает его сомнительным. Впрочем, советы, которые здесь написаны, все равно никто не выполняет. Посудите сами. Разве вы откажетесь от соблазна после пяти бокалов коньяка удаленно изменить настройки фильтра пакетов, сменить пароль пользователя root и перезагрузить компьютер, причем все это – находясь в отпуске? В любом случае, делясь паролем root с коллегами, помните: у семи нянек дитя без глазу! Размазывание ответственности хорошо для того, чтобы ее избегать. Для эффективной работы следует поручать администрирование системы одному человеку. Делегирование полномочий возможно только по строго определенной и всем известной (и, желательно, напечатанной на бумаге!) процедуре.

Одним из самых неприятных типов сбоев является так называемый отказ в обслуживании (Denial of Service – DoS).

Нередко он представляет собой следствие атаки, проведенной неизвестными (или известными) через Интернет. Предотвратить такую атаку можно, лишь своевременно обновляя программное обеспечение своего компьютера и внимательно следя за информацией от CERT. CERT – это Computer Emergency Report Team, *www.cert.org*. Настоя-

тельно рекомендуется регулярно посещать их сайт и подписаться на рассылку от CERT по свежим найденным уязвимостям систем.

Бывают случаи, когда DoS стал следствием недостаточной продуманности служб, запущенных на сервере. Так, если памяти мало, канал связи маломощный, а вы предоставляете хостинг популярному ресурсу Интернета (порнографическому сайту, например), то вы обрекаете сервер на перегрузку и на DoS в конце концов. Если ваш веб-сервер работает с базой данных через php- или perl-интерфейс, проверьте, как много одновременных запросов он может обслужить. Если на сервере установлено всего 32 Мбайта оперативной памяти, то не исключено, что число одновременных веб-запросов к базе данных не может превышать десятка. Помните, что каждый запрос к веб-серверу требует запуска отдельной копии веб-сервера. Стало быть, для обработки 10 запросов одновременно потребуется столько памяти, сколько занимают десять демонов **`httpd`**.

Следите за использованием ресурсов с помощью **`ps`** и **`top`**.

Еще одним хорошим средством предотвращения неприятностей является ежедневная проверка состояния вашей системы с помощью скриптов. Например, полезно запускать программу **`find`** для поиска «залежавшихся» файлов в каталоге `/var/spool` и его подкаталогах (так вам удастся обнаружить неисправности в подсистемах печати и, возможно, почтовой системе), выполнять проверку сделанного резервного копирования (все ли сделано?) и т.п. Важной деталью такой проверки должна быть отправка письма системному администратору из того скрипта, который выполняет проверку. Получение утром отчета от системы, которая за ночь проверила себя, будет хорошим подарком (или подспорьем – в зависимости от результата), куда приятнее и полезнее, чем получение прочей почты, которой заботливые спамеры наполнили ваш почтовый ящик, пока вы спали.

Да, и не забывайте о резервном копировании, пожалуйста. Перечитайте эпиграф этой главы, и если резервное копирование еще не стало правилом в вашей сети, перечитайте еще и главу 9 – ведь его так просто организовать!

29

Полезные демоны и утилиты

Можно подумать, что заголовок этой главы предполагает бесполезность всех описанных ранее утилит и демонов. Это не так. Здесь собраны сведения о тех программах, которые могут здорово помочь в работе, если о них помнить. Сами эти программы довольно просты, поэтому посвящать им по отдельной главе было бы несправедливо по отношению к более сложным программам.

Архиваторы и упаковщики

Для сжатия больших файлов в UNIX изначально использовалась программа **compress**. Она сжимала файл, а после сжатия добавляла к имени файла `.Z`. Программа **compress** до сих пор поставляется практически со всеми диалектами UNIX. В IRIX и некоторых других системах она является стандартной утилитой сжатия. Чтобы вернуть файл, сжатый **compress**, к прежнему несжатому состоянию, применяется **uncompress**.

В Linux и FreeBSD, а также во многих других диалектах для сжатия используют **gzip**. В нем реализован тот же алгоритм сжатия LZW, что и в pkzip или WinZip, известным по системам Microsoft. Однако он не совместим по формату файлов ни с WinZip, ни с pkzip. Программа WinZip for Windows понимает форматы **gzip**, а **gzip** ее форматы не понимает. Файлы, сжатые **gzip**, распаковывают с помощью **gunzip**. Большинство версий **gzip** не умеют сжимать каталоги с их содержимым – только файлы.

Есть еще утилиты **zip** и **unzip**, которые работают с теми же форматами файлов, что и pkzip, WinZip и другие архиваторы в Microsoft-систе-

мах. Они редко применяются в UNIX – только если надо обеспечить совместимость с Windows-системами.

Программа **tar** может применяться как для архивирования файлов на ленту, так и для создания архивов. Она собирает все указанные ей файлы в один большой файл. Имя файла **tar** никак не модифицирует. В UNIX принято, чтобы имя архива, созданного **tar**, заканчивалось на `.tar`. Об этом должен позаботиться тот, кто запускает **tar**, указав ей верное имя архива.

Получившийся файл сжимают с помощью **gzip**, чтобы он занимал меньше места. Многие дистрибутивы программных пакетов в UNIX упакованы **tar** и **gzip**. У команды **tar** даже есть ключ **Z**. Он говорит **tar**, что после упаковки файлов в архив нужно вызвать **gzip** для сжатия. Такой же ключ используется и для распаковки, когда **tar**, перед тем как начать свою работу, вызывает **gunzip**.

Например, мне надо упаковать дистрибутив Apache, который лежит в `/home/apache`:

```
cd /home
tar cvzf  apache.tar.gz  apache/*
```

Всего три команды в UNIX принимают (и в некоторых версиях даже требуют) ключи без знака «минус» перед ними. Это **ps**, **dump** и **tar**. В данном случае при создании архива я указал следующие ключи:

- **c** (create) – создать архив;
- **v** (verbose mode) – выводить имена всех пакуемых файлов на экран;
- **z** (zip) – после упаковки вызвать **gzip** для сжатия;
- **f** (file) – записать результат в файл, указанный после ключа **f**, а не на первый ленточный накопитель, как по умолчанию.

`apache.tar.gz` – это имя архива, а `apache/*` – то, что надо упаковать. Лучше всего запаковывать целый каталог, а не входить в него и паковать содержимое:

```
# так делать не надо
cd /home/apache
tar cvzf  apache.tar.gz  *
```

Архив так тоже можно создать, но при распаковке он упадет всеми своими сотнями файлов и подкаталогов в тот каталог, где он будет распаковываться. Это неудобно, и так не принято. Дурной тон. Намного вежливее и красивее упаковать сразу целый каталог, как в предыдущем примере. Тогда по команде

```
tar xzvf apache.tar.gz
```

в том каталоге, где лежит архив, будет создан подкаталог `apache` (помните, там было сказано «`apache/*`»?). И уже в него будут распа-

кованы все файлы и подкаталоги архива. В этой команде **tar** мы использовали один незнакомый ключ **-x**. Он требует распаковать архив.

Кроме упаковщика **gzip** есть еще более современный упаковщик **bzip2** и его распаковщик **bunzip2**, которые все чаще употребляются для сжатия и распаковки дистрибутивов. Принцип его работы и синтаксис вызова такие же, как у **gzip**/**gunzip**, а ключи лучше уточнить в **man** или по команде

```
bzip2 --help
```

Антивирусы

В UNIX вирусы не прижились из-за того, что вирус – это программа, выполняющая некий вредоносный код. В UNIX не так просто заставить систему выполнить код, который портит что-то в самой системе: не хватит прав доступа.

Однако UNIX тоже уязвима, как и Windows, несмотря на большую устойчивость к вирусам. Многие программы UNIX пишутся с мелкими, но порой играющими большую роль ошибками. То, что исходные тексты этих программ часто открыты всем, в отличие от текстов коммерческих программ, приводит к тому, что эти мелкие ошибки быстро находят. Но их находят не только пользователи и разработчики, но и взломщики систем. Нетрудно представить себе, что среди них есть талантливые и ловкие люди, которые найдут способ взломать систему, обнаружив ошибку в какой-нибудь распространенной программе.

Поэтому в мире UNIX наиболее распространены не вирусы, а банальное использование «дыр» в программах.

Однако статьи специалистов по ИТ в американских и европейских журналах (см., например, *http://www.claymania.com/unix-viruses.html*) говорят о том, что опасность UNIX-вирусов постоянно растет.

В одной из них делается весьма разумное предположение, что поскольку значительная часть UNIX-систем написана в виде скриптов (взять хотя бы такие важные скрипты, как стартовые), то и вирусы будут облекаться в форму скриптов.

Малая распространенность вирусов под UNIX во многом объясняется меньшей распространенностью UNIX-систем по сравнению с Windows. Поэтому будьте внимательны, анализируя файлы протоколов и текущую работу системы. Замечайте любые странные процессы или новые файлы, которые появились без вашего участия там, где ничего появляться не должно.

Помните, что если обычный пользователь запустит вредоносный скрипт, то пострадать могут только его файлы. К чужим он не имеет доступа. Если имеет, то его сисадмину стоит подумать о поступлении на курсы для новичков в UNIX.

Но если вредоносный скрипт запустит root, то скрипт сможет выполнить любые действия в системе, и последствия будут ужасны! Если вы заполучили скрипт из ненадежных источников, проверьте его перед запуском. Это проще, чем проверить двоичный файл, ведь скрипт – это текст.

Что касается антивирусного программного обеспечения под UNIX, то оно существует и без труда может быть найдено с помощью любой поисковой системы в Интернете. Количество известных вирусов под UNIX пока мало, но оно будет неуклонно расти.

Весьма специфический сорт антивирусного ПО под UNIX представляют собой сканеры почты. Поскольку в письмах сейчас часто передаются вирусы, предназначенные для Windows-систем, хорошей идеей кажется «выкусывание» таких вирусов из писем еще на этапе передачи письма.

Отечественные разработчики антивирусных программ уже подготовили соответствующие модули. Положительные отзывы уже получены от тех, кто установил Dr. Web daemon (*www.drweb.ru*). Также получает распространение и антивирусный модуль лаборатории Касперского (*www.avp.ru*) для **`sendmail`** и **`qmail`**.

Цитируя рекламу производителей Dr. Web, можно сказать, что «Dr. Web daemon предназначен для совместной работы с различными программами в качестве подключаемого фильтра. В комплект входит описание протокола взаимодействия с демоном и исходные тексты клиентского ПО, позволяющие встраивать проверку на наличие вирусов практически в любые схемы обмена данными.

Текущая версия включает в себя исходные тексты и скрипты для интеграции демона с Sendmail, Exim, QMail Postfix, Communigate Pro, Samba, Courier и ZMailer.

Сканер Dr. Web. Хорошо известные сканеры семейства Doctor Web, работающие в режиме командной строки, теперь существуют для Linux, FreeBSD версий 3.xx и 4.xx и Solaris x86.

При использовании ознакомительного ключа, входящего в состав дистрибутивов, на работу программ накладываются следующие ограничения: сканер не производит лечение инфицированных объектов, не производит проверку файлов, находящихся внутри архивов и внутри почтовых баз. На работу демона накладываются те же ограничения, за исключением работы с почтовыми базами.»

Кроме антивирусов этих известных в России марок есть еще два, причем бесплатно распространяемых пакета, которые предназначены для борьбы с вирусами в почтовых сообщениях. Ниже эти пакеты – Anomy Sanitizer и ClamAV – будут рассмотрены подробнее.

Борьба с вирусами: Anomy Sanitizer

Пакет Anomy (*http://mailtools.anomy.net/*) представляет собой набор скриптов, который предназначен для анализа писем и удаления из них подозрительных вложений. Какие вложения считать подозрительными, а какие – нет, вы можете определить сами, редактируя файл `/usr/local/anomy/anomy.conf`. Самая важная часть этого файла – правила фильтрации. Правила задаются посредством определения списка файлов и политики обработки каждого списка. Список файлов задается простым шаблоном, например:

```
file_list_1 = (?i)(your_details.zip)|
file_list_1 +=
(\.(exe|com|vb[se]|dll|ocx|cmd|bat|pif|lnk|hlp|ms[ip]|reg|sc
t|inf
file_list_1 +=
|asd|cab|sh[sb]|scr|cpl|chm|ws[fhc]|hta|vcd|nws|emf))$
```

А политика указывается на следующей строке после определения шаблона и представляет собой строку

*имя_списка*_**policy =**

```
file_list_1_policy = save
file_list_1_scanner = 0
```

В более сложных случаях следует запускать внешний сканер (соответствующие скрипты могут входит в пакет Anomy Sanitizer, дописываться системным администраторам под конкретные задачи фильтрации или могут быть получены из третьих рук). При описании внешнего сканера указывается, какие коды завершения он возвращает для каждого из возможных случаев: письмо не содержит ничего подозрительного; письмо содержало какую-то дрянь, но скрипту ее удалось удалить; письмо по-прежнему содержит что-то подозрительное, и случилось что-то непредвиденное. Возможны и другие случаи, это лишь самые общие.

Для каждого из случаев определяется одна из шести политик:

- `accept` – пропустить письмо с вложением
- `defang` – пропустить письмо, но модифицировать имя файла, чтобы система получателя не смогла сделать вывод о содержимом по имени и запустить вредоносный код без спроса, имя остается узнаваемым
- `mangle` – пропустить письмо, но изменить имя файла до неузнаваемости, чтобы система получателя точно не смогла сделать вывод о содержимом по имени и запустить вредоносный код без спроса
- `save` – записать файл, не меняя его имя, в карантинный каталог
- `drop` – уничтожить вложение

- unknown – продолжить анализ письма с помощью следующих правил, определенных в anomy.conf

Для каждой из четырех групп кодов, возвращаемых внешним скриптом указывается действие, которое следует произвести с вложением, например, согласно нижеследующему описанию, надо:

- если код завершения 0, пропустить письмо с вложением (accept)
- если код завершения 6, записать его в карантинный каталог, изменив имя согласно заданному в anomy.conf шаблону, параметр file_name_tpl (save)
- если код завершения 3 или 8, сделать то же самое (save)
- если код завершения отличается от уже описанных, сделать то же самое (save)

```
file_list_4 = (?i)\.(zip)
file_list_4_policy =  accept:save:save:save
file_list_4_scanner = 0:6:3,8:/usr/local/sbin/
zip_policy.pl %FILENAME
```

Борьба с вирусами: ClamAV

Пакет ClamAV состоит из нескольких частей. Во-первых, это демон **clamd**, который собственно и занимается поимкой вредоносных вирусов. Сигнатуры вирусов содержатся в базе, которую регулярно обновляет демон **freshclam**. Не забудьте его запустить, иначе база быстро устареет и грош ей будет цена! Третий компонент – программа **clamav-milter**, которая служит для вылавливания вирусов в почтовых сообщениях и умеет работать со стандартным интерфейсом мильтеров – почтовых фильтров программы **sendmail**. Установка ClamAV не вызывает особых сложностей и состоит из следующих этапов:

- получение дистрибутива пакета
- компиляция необходимых программ
- настройка пакета (файл конфигурации /etc/clamav.conf или /usr/local/etc/clamav.conf)
- запуск демонов **freshclam** и **clamd**
- настройка и перезапуск **sendmail** (отредактируйте sendmail.cf так, как сказано в документации к ClamAV)

По умолчанию письмо с вирусом будет отброшено, отправитель письма получит уведомление о том, что его письмо не доставлено из-за того, что заражено вирусом, а потенциальный получатель вообще не будет уведомлен о послании. Изменить это поведение можно, модифицируя файл clamav.conf.

Атаки

Кроме уничтожения или лечения вирусов при работе под UNIX системному администратору приходится многое делать для того, чтобы его сервер не стал объектом атаки. Атака на сервер предполагает либо его взлом (получение несанкционированного доступа к данным на сервере), либо доведение сервера до состояния отказа от обслуживания запросов. В последнем случае работа сервера (чаще – одной из его служб) блокируется до перезагрузки службы или всего сервера.

Атаки часто имеют политическую или коммерческую направленность (борьба с конкурентами).

Защита от атак состоит в грамотной настройке служб, своевременном чтении бюллетеней CERT (Computer Emergency Response Team, *www.cert.org*) и обновлении уязвимых версий программ.

Описанию атак на UNIX-системы посвящены мириады статей и книг. В России наиболее известна книга [15], первое издание которой вышло в 1997 году.

Протоколирование событий: syslogd

Демон **syslogd** (файл конфигурации `/etc/syslog.conf`) протоколирует события, информацию о которых ему поставляют другие процессы, вызывая функцию **syslog()**. В файле `syslog.conf` определяется, какие события надо протоколировать и как.

Синтаксис записи в `/etc/syslog.conf`:

источник.уровень действие

Источник – это условное название процесса или группы процессов, которые могут быть источниками информации о событиях. Уровень – это уровень серьезности сообщения (ошибка, фатальный сбой, предупреждение, информационное сообщение и т. п.). Источник в оригинальной документации называется facility, в переводной литературе фигурирует понятие «средство». Однако мы предпочтем термин «источник», так как речь идет именно об источнике сообщений.

Полный перечень источников и уровней есть в **man syslog.conf**.

Действие может представлять собой:

- запись информации о событии в файл;
- пересылку сообщения другому компьютеру (демону **syslogd** на нем);
- отправку сообщения по почте;
- вывод сообщения на экран для вошедших в систему пользователей.

Обычно информация записывается в файл. Принято держать файлы протоколов в /var/log или /usr/log. Имя файла, записанное в поле «действие», приводит к записи информации в этот файл.

После изменения файла конфигурации /etc/syslog.conf необходимо послать сигнал HUP демону **syslogd**. За размером файлов протокола надо следить, иначе они займут все свободное место на разделе. Уровень протоколирования следует ставить из соображений разумной достаточности. Незачем протоколировать все события с уровнем info или notice. Если вы уже настроили сервер, вполне достаточно протоколировать события большинства источников с уровнем error.

Указанный в строке /etc/syslog.conf уровень – это минимальный уровень, который должно иметь событие, чтобы быть занесенным в протокол. Стало быть, если вы указали уровень событий info, то все более серьезные уровни (warning, error и т. д.) тоже будут отвечать этой же строке syslog.conf и события будут записаны туда же, куда и сообщения о событиях уровня info.

Пример минимального файла syslog.conf:

```
# Log all kernel messages to the console.
# Logging much else clutters up the screen.
#kern.*                                 /dev/console

# Log anything (except mail) of level info or higher.
# Don't log private authentication messages!
*.info;mail.none;authpriv.none          /var/log/messages

# Cisco logging
192.*                                   /var/log/cisco

# The authpriv file has restricted access.
authpriv.*                              /var/log/secure

# Log all the mail messages in one place.
mail.*                                  /var/log/maillog

# Everybody gets emergency messages, plus log them on another
# machine.
*.emerg                                 *

# Save mail and news errors of level err and higher in a
# special file.
uucp,news.crit                         /var/log/spooler

#All Other
#*.*                                   /var/log/all
```

Дополнительная литература

[15] Медведовский И., Семьянов П., Леонов Д., Лукацкий А. «Аспекты защиты. Атака из Internet». – М.: Солон-Р, 2002.

A

Краткий справочник по командам и файлам конфигурации UNIX

Справочник основан на открытом источнике «Solaris Admininstror's Quick Reference», Jialong He, *http://www.bigfoot.com/~jialong_he*.

Инициализация среды работы пользователя

`/etc/profile` (`$HOME/.profile`)[1]

Системный (пользовательский) стартовый файл для **sh**, **bash**, **ksh**; системный файл игнорируется при входе для работы в графическом интерфейсе CDE в Solaris

`/etc/.login` (`$HOME/.login`)

Системный (пользовательский) стартовый файл для **csh**, **tcsh**; системный файл игнорируется при входе для работы в графическом интерфейсе CDE в Solaris

`$HOME/.cshrc`

Стартовый пользовательский файл для **csh**, **tcsh** (порядок выполнения: `/etc/.login` → `$HOME/.cshrc` → `$HOME/.login`)

---

[1] Здесь и далее нечто вроде «`/etc/q` (`$HOME/p`) системный (пользовательский) файл настроек» означает, что `/etc/q` – общесистемный файл настроек, а файл `p` из домашнего каталога пользователя (имя каталога хранится в переменной среды окружения `HOME`) – файл пользовательских настроек.– *Примеч. авт.*

`/usr/dt/config/Xsession.d/2000.login`

Общий стартовый файл для среды CDE (HP-UX, Solaris), скрипт в `Xsession.d`

`$HOME/.dtprofile`

Персональный пользовательский стартовый файл для среды CDE (HP-UX, Solaris)

`/etc/skel/*`

Каталог со стандартными файлами конфигурации, которые копируются при создании нового пользователя в его домашний каталог

Настройка сетевых параметров

`/etc/defaultdomain` (только Solaris)

Имя домена NIS/NIS+

`/etc/defaultrouter` (только Solaris)

IP-адрес шлюза (маршрут по умолчанию)

`/etc/resolv.conf`

Имя домена в Интернете, используемые сервер имен (до 3-х штук) и порядок поиска не полностью определенного доменного имени в доменах.

Пример содержимого `resolv.conf`:

```
domain eu.spb.ru
nameserver 129.219.17.200
nameserver 129.219.17.5
search eu.spb.ru spb.ru
```

`/etc/nsswitch.conf` (Solaris, Debian Linux)

Указатель источников информации для разных служб (файл `hosts`, NIS, DNS)

`/etc/hostname.{hme0|le0}` (только Solaris)

Имя хоста для определенного сетевого интерфейса IPv4

`/etc/nodename` (только Solaris)

Имя хоста для определенного сетевого интерфейса IPv6

`/etc/inet/hosts` (только Solaris), `/etc/hosts` (другие UNIX)

Файл соответствий имен и IP-адресов компьютеров

`/etc/inet/netmasks` (только Solaris)

Маски подсетей IP

`/etc/inet/protocols` (только Solaris), `/etc/protocols`

Список протоколов

`/etc/inet/services` (только Solaris), `/etc/services`

Список соответствий номеров портов и имен сетевых сервисов

Инициализация маршрутизации

Solaris	FreeBSD	Linux

`/etc/notrouter`
наличие этого файла требует отключения маршрутизации между интерфейсами
	`/etc/rc.conf`
	`gateway=yes` или `/sbin/sysctl -w net.inet.ip.forwarding=1`
	(«no» и 0 соответственно, если надо запретить маршрутизацию)
		`/proc/sys/net/ipv4/ip_forward`
		если в этом файле содержится 1, то сервер является маршрутизатором, т. е. будет передавать пакеты между интерфейсами, если 0, то нет. Запись в файл 1 или 0 осуществляется так:
		`echo 1 > /proc/sys/net/ipv4/ip_forward`
		или
		`echo 0 > /proc/sys/net/ipv4/ip_forward`
		Для постоянной настройки `/etc/network/options` должен содержать:
		`ip_forward=yes`
		`spoofprotect=yes`
		`syncookies=no`

`/etc/inet/inetd.conf` (`/etc/inet/xinetd.conf`)

Файл конфигурации демона **`inetd`** (**`xinetd`**)

Изменение IP-адреса затрагивает файлы:

Solaris	Linux	FreeBSD

`/etc/hostname.{hme0|le0}`
`/etc/nodename;`
`/etc/inet/hosts;`
`/etc/net/*/hosts;`
`/etc/defautrouter;`
`/etc/resolv.conf`
	`/etc/network/interfaces`
	`/etc/hosts`
	`/etc/resolv.conf`
	`/etc/postfix/main.cf`
		`/etc/rc.conf`
		`/etc/hosts`
		`/etc/resolv.conf`
		`/etc/postfix/main.cf`

Настройка принтера

Локальный принтер (только Solaris)

`lpadmin -p` *`prt_name`* **`-v`** `/dev/ecpp0` **`-D`** “*`description`*” **`-T PS -I postscript`**

параллельный порт: `{/dev/ecpp0 | /dev/bpp0}`

последовательный порт: `{/dev/term/a | /dev/term/b}`

Сетевой принтер (только Solaris)

```
lpadmin -p prt_name -m netstandard -v /dev/null
-D "description" -o protocol={bsd | tcp}
-o dest=prtIP:9100
-o banner={never | optional | always}
-o timeout=300
```

Разные команды печати

lpadmin -d *prt_name*

Определить принтер *prt_name* в качестве принтера по умолчанию

lpstat -d

Показать текущий принтер по умолчанию

lpstat -p *prt_name* **-l**

Показать конфигурацию принтера *prt_name*

Переменные среды окружения, связанные с печатью

LPDEST или PRINTER

Принтер по умолчанию

_default в файле /etc/printers.conf

Системный принтер по умолчанию (Solaris)

nobanner=yes в файле /etc/lp/interfaces/*prt_name*

Убрать баннер при печати (Solaris)

Файловая система (NFS)

/etc/fstab (Solaris – /etc/vfstab)

Файл с таблицей начального монтирования

/etc/exports (Solaris – /etc/dfs/dfstab)

Файл с таблицей экспортируемых каталогов

Конфигурация Sendmail

/etc/mail/sendmail.cf

Файл конфигурации **sendmail**

/etc/mail/aliases

Файл почтовых псевдонимов, запускайте **newaliases** после модификации (для систем с postfix – **postaliases**)

Настройка sendmail в Solaris:

Создание нового файла настроек:

```
cd /usr/lib/mail/cf
cp main -v7sun.mc foo.mc
/usr/ccs/bin/make foo.cf
```

Тестирование нового файла настроек

```
/usr/lib/sendmail -C foo.cf -v email_address </de/null
```

Установка нового файла настроек

```
cd /etc/mail
cp sendmail.cf sendmail.sav
cp /usr/lib/foo.cf sendmail.cf
pkill -HUP sendmail
```

Простые настройки для борьбы со спамом:

Добавьте `FEATURE(access_db)` в `sendmail.mc` (если там еще нет этой строки), выполните установку нового файла конфигурации;

Создайте файл `/etc/mail/access`, содержащий примерно следующее:

```
cyberpromo.com REJECT
sendmail.org RELAY
spam@buyme.com 550 Spammers shan't see sunlight here
```

Выполните компиляцию хэшированной базы `access.db` из файла `access`:

```
makemap hash /etc/mail/access < /etc/mail/access
```

Настройки CDE (Solaris, HP-UX)

`$HOME/.dtprofile`

Пользовательский файл настроек, генерируется из

```
/{usr|etc}/dt/config/sys.dtprofile
```

`$HOME/.dt/dtwmrc`

Файл конфигурации менеджера рабочего стола (выбирается из контекстного меню, вызываемого щелчком правой кнопкой мыши). Генерируется из

```
/{usr|etc}/dt/config/C/sys.dtwmrc
```

`/usr/dt/appconfig/types/C/dtwm.fp`

Скопируйте в `$HOME/.dt/types/myname.fp` и отредактируйте его

`$HOME/.dt/sessionetc`

Пользовательский начальный скрипт запуска CDE

`$HOME/.dt/sessionexit`

Пользовательский завершающий скрипт выхода из CDE

`/{usr | etc}/dt/config/C/Xresources`

Управляет окном входа (программа **dtlogin**)

`/{usr | etc}/dt/config/C/sys.resources`

Параметры сессии; укажите, например, для отмены автоматической блокировки окна по времени:

```
dtsession*lockTimeout: 0
```

`$HOME/.dt/sessions/{current | home}`

Файлы активной сессии

`/{usr | etc}/dt/config/Xconfig`

Файл настройки окна входа в систему (**dtconfig**), после изменения запускайте

```
dtconfig -reset
```

`/usr/dt/config/Xstartup`

Скрипт запускается после входа в систему перед запуском пользовательской сессии

`/usr/dt/config/Xaccess`

Управляет доступом к X-серверу

`/usr/dt/bin/Xsession`

Запуск сессии CDE

`/usr/dt/config/Xsession.d/*`

Автоматически запускается Xsession

`usr/dt/config/Xsession.ow`

Запуск сессии OpenWindow

`/usr/dt/config/Xreset`

Запускается после завершения сессии

`/usr/dt/bin/dtconfig -e|-d|-kill`

`-e|-d`: разрешить (запретить) автозапуск программы **dtlogin**

`-kill`: принудительное завершение **dtlogin** и ее дочерних процессов

`$HOME/.dt/icons`
`$/{usr|etc}/dt/appconfig/icons/C`

Путь поиска пиктограмм

Конфигурация системы

`/etc/passwd`, `/etc/shadow` (FreeBSD – `/etc/master.passwd`), `/etc/group`

Пользователи и группы

`/etc/ftpusers`

Список тех, кому ЗАПРЕЩЕН доступ по ftp к данной системе

`/etc/system`

Настройки ядра Solaris

Параметры ядра (Solaris)

Например:

```
set sys:coredumpsize = 0 # prevent core dump
```

`/etc/motd`, (`/etc/issue`)

Сообщение после (до) входа в систему

`/etc/shells`

Список допустимых командных процессоров

`/etc/initab`

Список запускаемых при старте программ (System V)

`/etc/rc{S, | 1 | 2 | 3 | 5 | 6}.d`

Скрипты для различных режимов работы – runlevels (System V)

`/etc/bootparams`

Файл настроек `rpc.bootparamd` (Solaris)

`/etc/pam.conf`

Файл настроек PAM

`/etc/printers.conf`

Настройки принтеров

`/etc/security/*`

Файлы, связанные с подсистемой аудита (Solaris)

`/etc/syslog.conf`

Файл конфигурации **syslogd**

`/etc/notrouter`

Выключает маршрутизацию между интерфейсами

Параметры по умолчанию для разных программ (Solaris)

`/etc/default/power`

Управление питанием

`/etc/default/su`

Настройки для команды **su**

`/etc/default/sys-suspend`

Список тех, кому можно переводить систему в режим `suspend`

`/etc/default/passwd`

Минимальная длина пароля

`/etc/default/login`

Настройки, связанные с входом в сеть

`/etc/default/init`

Установки часового пояса, маски прав доступа к новым файлам по умолчанию **umask**

Разное

showmount -e *host_name*

Посмотреть, какие файловые системы экспортируются на *host_name*

rpcinfo -p *host_name*

Проверить сервисы RPC на *host_name*

ldd *program* (Solaris)

Посмотреть, какие разделяемые библиотеки нужны программе *program*

`LD_LIBRARY_PATH` – путь поиска разделяемых библиотек

useradd

Добавить нового пользователя

man *foo* **| col -b >** *foo.txt*

Записать страницу man в виде обычного текста (Solaris)

who -r

Показать текущий режим работы (runlevel) – Solaris

fuser -u [-k] */cdrom*

Показать, какой процесс использует */cdrom*

find / -type f -perm -4000 -user root -print

Найти программы, принадлежащие root с установленным битом SUID

fsck -F ufs -y /dev/rdsk/c0t0d0s0

Проверка файловой системы на устройстве `/dev/rdsk/c0t0d0s0` с ответом «yes» на все вопросы

xv -root -quit *foo.gif*

Установить фоновый рисунок в CDE

/etc/init.d/*script* {*start* | *stop*}

Запуск стартовых скриптов (скриптов останова) в каталоге `/etc/init.d`

newfs -v /dev/rdsk/c0t0d0s6

Выполнить форматирование и создание новой файловой системы

`/usr/sbin/sys -unconfig`

Отменить все сделанные во время установки системы Solaris установки (ПРИМЕНЯТЬ С ОСТОРОЖНОСТЬЮ ПОСЛЕ ЧТЕНИЯ ДОКУМЕНТАЦИИ!)

`ndd` (Solaris)

Установка параметров TCP/IP

`ldconfig` (Linux)

Управление разделяемыми библиотеками

В

Указатель веб-адресов

cbl.abuseat.org
filip@unixbook.spb.ru
http://asg.web.cmu.edu/cyrus/imapd/
http://docsrv.caldera.com
http://kirch.net/unix-nt/hotmail.html
http://knot.pu.ru/faq/xfaq.html
http://mailtools.anomy.net/
http://openvpn.net/man.html
http://spamassassin.apache.org/
http://unixbook.spb.ru
http://www.acnet.ge/networking/net_l/routing/internet/cidr.htm
http://www.bigfoot.com/~jialong_he
http://www.claymania.com/unix-viruses.html
http://www.declude.com/Articles.asp?ID=97
http://www.earthlink.net/cgi-bin/pwgenerator.pl
http://www.eecis.udel.edu/~mills/ntp/servers.htm
http://www.eecis.udel.edu/~ntp/ntpfaq/NTP-s-algo.htm
http://www.ehlis.com/adam/solaris/history.html
http://www.eu.spb.ru
http://www.eudora.com/products/#freeservers
http://www.faqs.org/faqs/unix-faq/shell/csh-whynot/
http://www.for.spb.ru/cgi-bin/show.pl
http://www.gio.ru/cgi-bin/.pl*

http://www.gordano.com/

http://www.icann.org/dndr/udrp/approved-providers.htm

http://www.inp.nsk.su/~baldin

http://www.kernel.org/pub/linux/libs/pam/

http://www.levenez.com/unix/

http://www.milter.info/sendmail/milter-sender/

http://www.mysql.com/doc/en/LOAD_DATA.html

http://www.netfilter.org

http://www.nic.aero

http://www.nic.biz/

http://www.nic.coop/

http://www.nic.info/

http://www.nic.museum

http://www.nic.name/

http://www.ntp.org

http://www.opengroup.org

http://www.openssl.org/

http://www.ordb.org/faq/#usage

http://www.postfix.org/documentation.html

http://www.postfix.org/postconf.5.html

http://www.qmw.ac.uk/~tl6345/ca.htm

http://www.redhat.com/mirrors/LDP/FAQ/Linux-FAQ/intro.html#PLATFORM-SUPPORT

http://www.sensi.org/~alec/locale

http://www.server.edu/news/

http://www.spamcop.net

http://www.tldp.org/HOWTO/IP-Masquerade-HOWTO/firewall-examples.html

http://www.tldp.org/HOWTO/IP-Masquerade-HOWTO/kernel-2.4.x-requirements.html

http://www.tsu.ru/~pascal/x_locale/

http://www.unicode.org

http://www.washington.edu/imap/

imap.org

relays.ordb.org

unix book.spb.ru

www.apache.org

www.avp.ru

www.cert.org

www.drweb.ru

www.eu.spb.ru

www.guardian.co.uk

www.nl.sorbs.net

www.opennet.ru

www.qmail.ru

www.rbc.ru

www.ripn.net

www.sarafan.ru

www.spamcop.net

Алфавитный указатель

B

BIND (Berkley Internet Name Domain), 212, 214

D

DNS (Domain Name System), служба имен доменов, 163, 171, 172, 305, 333, 382
  named.conf, 214, 215
  домен, 210, 211, 214, 216, 218, 220, 333, 347
  зона, 172, 212
DNS-сервер, 46, 214–217, 220, 230, 330, 365
  вторичный, 212, 362
  главный, 30, 212, 218, 362

F

firewall, фильтр пакетов, 146, 169, 184, 186, 187, 189–192, 230, 271, 371

I

IMAP-сервер, 295, 297

M

make-файл, 236, 238, 301, 311

P

PAM-модули, 307, 342–349, 387

S

SSL (Secure Socket Layer), протокол защищенных сокетов, 306
swap, 74, 84, 262, 324, 370

T

tcp-wrapper, программа, 180
umask, команда, 71

V

VPN (Virtual Private Network), виртуальная частная сеть, 194, 195

А

агент
  почтовый, 80, 291–294
  транспортный, 291, 292, 307
адрес
  IP, 36, 154, 159, 161, 164, 175, 205, 209, 222, 253
  интерфейса, 159, 169, 198, 230, 270, 367
аутентификация, 179, 342, 343, 345, 348, 349, 381, 382, 388
  NIS, 382
  PAM, 307, 342–349, 387
  классическая, 207, 342, 343, 347, 348, 387, 388

Б

база данных
  aliases, 164, 300, 305, 384
  принтеров, 276, 277, 334
  СУБД MySQL, 306, 336–338, 340
брандмауэры, 185–187, 190–193

В

веб-сервер, 27, 29, 30, 39, 51, 57, 83, 152, 154, 178, 229, 308–312, 314–320, 346, 348
  Apache, 46, 57, 58, 60, 62, 126, 130, 152, 178, 207, 304, 308–319, 327, 343, 348, 349, 369, 374

виртуальные частные сети *см.* VPN
вирусы, борьба с, 304, 378
время жизни, 203, 219

Д

демон
asppd, 250
cron, 103–105, 290, 366
crond, 98, 103, 365
ftpd, 177–179, 206
gated, 168, 231
httpd, 57, 229, 308–320, 346, 348, 372
inetd, 29, 38, 44, 81, 98–100, 178–181, 230, 232, 306, 307, 332, 365, 371, 383
init, 41, 85, 98, 99, 172, 189, 228, 229, 365, 370, 371, 388, 389
kswapd, 84, 365
lockd, 231, 266
lpd, 232, 276–278
mountd, 231, 257, 259, 260, 263–265, 270–272
named, 46, 214–217, 220, 230, 330, 365
nfsd, 231, 257, 259, 260, 263–266, 268, 270–273
nmbd, 330, 332
ntpd, 281, 283–289
portmap, 231, 258, 265, 269, 270
pppd, 133, 190, 194, 250–255
qmail, 307, 376
routed, 168, 231
rpcbind, 258
rpc.mountd, 263, 266
sendmail, 24, 80, 99, 177, 232, 236, 292, 297, 298, 300–302, 304, 305, 307, 329, 365, 378, 384, 385
smbd, 330, 332, 333, 335
statd, 231, 266
syslogd, 98, 230, 252, 364, 365, 379, 380, 387
tcpd, 180, 181
telnetd, 177, 178, 181
timed, 231, 280–283
xntpd, 231, 281, 283, 288, 289
добавление устройства, 55, 247
документация
howto, 45
info, 45, 57, 211, 239, 304, 380
man, 13–15, 39, 41, 43–45, 53, 68, 78, 81, 82, 100, 107, 110, 121, 134, 136, 161, 169, 192, 195, 227, 232, 242, 247, 249, 259–262, 266, 268, 271, 272, 274, 277, 288, 289, 305, 311, 318, 333, 335, 352, 353, 357, 375, 379, 388
доменные имена
поиск, 163, 172, 216, 382, 383
полностью определенные, 214
суффикс по умолчанию, 382
домены, 212, 214, 216, 218, 220, 292, 305, 333, 347
arpa, 216, 220
com, 47, 72, 74, 145, 211, 222, 225, 288, 293, 295, 296, 299, 303, 305, 306, 319, 320, 323–325, 328, 339, 351, 375, 377, 381, 385
mil, 211
net, 147, 151, 158, 160, 168, 170, 173, 188, 195, 203, 211, 218, 221, 224, 225, 243, 259, 288, 299, 302, 303, 314, 325, 327, 377, 383
org, 29, 121, 148, 173, 185, 186, 189, 196, 206, 211, 225, 241, 271, 284, 285, 288, 290, 297, 299, 303–305, 309, 312, 331, 332, 349, 357, 360, 371, 379, 385
uk, 196, 210, 319, 332
второго уровня, 77
корневой, 210, 214
организационные, 210
драйвер, 63

Ж

жесткий диск, восстановление, 40, 55, 58, 229, 234, 370, 389
разделы, 31, 64
разметка, 37
форматирование, 55, 389

З

загрузка, 31, 34, 41, 52, 55, 65, 214, 227, 230, 246, 247
MBR, главная загрузочная запись, 34, 227
компьютера, 31, 34, 41, 52, 55, 65, 214, 227, 230, 246, 247
зона, 172, 212
обратная, 216, 220

И

идентификатор
группы, 49, 55, 85, 261, 262, 267, 268, 335
пользователя, 49, 51, 55, 85, 87, 261, 262, 267, 268, 335
процесса, 85, 97–101, 287, 289, 309, 335, 365
родительского процесса, 85
эффективный, 85, 86
интерфейс
графический, 350, 351
сетевой, 169, 198, 230, 367

К

каталоги
копирование, 44, 58, 59, 61, 80, 81, 106, 140, 141, 144, 145, 385
переименование, 59, 61, 80, 81, 115, 134
права доступа, 68, 70, 87, 88, 217
создание, 60, 217
удаление, 60
команда
arp, 175, 207, 208
awk, 108
cat, 60, 107, 128, 163, 164, 169–172, 179, 187, 220, 252, 348, 349, 354
chgrp, 70, 337
chmod, 68, 70, 87, 88, 217
chown, 70, 71, 217, 337
chpass, 53
config, 245–248, 309, 382, 385
cp, 44, 58, 59, 61, 80, 81, 106, 141, 385
crontab, 103, 104
cu, 251
date, 50, 102, 279, 280, 357
df, 278, 365, 370
du, 365
dump, 40, 139–143, 374, 387
echo, 42, 128, 129, 133, 134, 179, 187, 188, 198, 383
find, 62, 81, 82, 88, 89, 101, 125, 187, 372, 389
fsck, 40, 55, 58, 229, 234, 370, 389
ftp, 30, 38, 50, 107, 123, 140, 178, 179, 181, 182, 186, 188, 200, 206, 222, 242, 257, 271, 318, 343, 348, 349, 370, 387
getfacl, 68
grep, 80, 99, 100, 107, 108, 113–119, 171, 180, 263, 265, 269
halt, 39, 233
hdparm, 367
hostname, 169, 171, 230, 382, 383
ifconfig, 169, 198, 230, 270, 367
kill, 41, 80, 92, 99, 100, 180, 229, 255, 263, 272
killall, 100
ln, 62, 80
lpq, 276
lpr, 276–278
lprm, 276, 335
ls, 58, 61–63, 65, 88, 90, 91, 100, 121, 124, 125, 128, 170, 274, 348
lsof, 367
lspci, 367
m4, 298, 300, 302, 304
mail, 47, 50, 80, 81, 88, 90–92, 140, 221–224, 291, 293, 294, 298–302, 304, 306, 330, 340, 366, 380, 384, 385
mailstats, 301
make, 29, 236–241, 246–249, 301, 309–311, 321, 336, 337, 385
mkdir, 60, 217
mknod, 64
mount, 52, 68, 75, 78, 79, 234, 271–273, 275, 365
mv, 59, 61, 80, 81, 115, 134
netstat, 200–202, 367
newfs, 55, 389
nice, 100, 364
openssl, 196, 197
openvpn, 195, 197, 198
passwd, 48–50, 52, 53, 87, 88, 104, 127, 207, 252, 253, 342, 343, 347, 348, 387, 388
ping, 161–163, 197–200
pkg_add, 240–242
ps, 80, 85, 88, 97–101, 180, 263, 265, 372, 374
reboot, 39, 65
renice, 101
restore, 139, 143
rm, 60, 62, 80, 104, 134, 277
rmdir, 60
rmuser, 53
route, 161, 167–169, 197, 270
rpm, 242

sar, 361
setfacl, 68, 69
shutdown, 39, 41
ssh, 144, 145, 332
stty, 43, 250
su, 32, 51, 52, 67, 87, 171, 218, 221, 267, 343, 348, 360, 388
swapinfo, 366
telnet, 29, 38, 124, 150, 178, 179, 181, 182, 192, 200, 204, 332, 349
time, 61, 102, 162, 199, 203, 219, 231, 232, 273, 279, 282, 284, 327, 329
traceroute, 202, 203
trafshow, 204–207
umask, 71, 287, 388
umount, 78, 79, 257, 272, 364
uptime, 366
команды языка sh
case, 132, 135, 334
for, 96, 133, 135, 148, 164, 176, 179, 187, 188, 224, 232, 315, 316, 328, 329, 331, 348, 373
if, 132, 133, 135, 231–233, 273, 277, 278
while, 132, 133, 135, 255, 328
командный процессор
bash, 45, 49, 120–124, 126–129, 131, 132, 134, 136, 137, 241, 357, 358, 381
csh, 82, 120–124, 127–132, 134–137, 381
ksh, 120–124, 127–129, 131, 132, 136, 137, 381
rsh, с ограничением возможностей, 127
sh, 45, 101, 103, 104, 121–124, 126–129, 131, 132, 134–137, 172, 187, 229, 255, 277, 313, 377, 381
tcsh, 82, 120–124, 126, 128, 129, 131, 136, 137, 381
комментарии, символ #, 230, 234

Л

лирическое отступление, 75, 146, 149, 151, 156, 178, 268, 324, 332
локальный интерфейс, 154, 159, 160, 164, 165, 169, 201, 202, 230

М

макропроцессор m4, 298, 300, 302, 304
маска
подсети, 270
сети, 156, 159, 172, 175
модуль, 249, 347

Н

нагрузка
на компьютер, 98, 101, 207, 341, 361, 362, 364, 365, 369, 370, 372
на сеть, 200–202, 204–207, 367

П

пакет
BIND, 212, 214, 222
Samba, 256, 276, 330, 332–335
память оперативная, 370
печать
подсистема, 232, 276–278
файла, 276–278
поиск
доменных имен, 144, 163, 168, 171, 192, 223, 224, 282, 315, 337, 339, 340, 351, 388
файлов, 62, 81, 82, 88, 89, 101, 125, 187, 372, 389
почта
imap4, 179
pop3, 179
настройка, 80, 99, 177, 232, 236, 292, 297, 298, 300–302, 304, 305, 307, 329, 365, 378, 384, 385
cucipop, 296, 306
imapd, 80, 81, 179, 295, 307
pop3d, 80, 81, 295
права доступа, 66, 68, 70, 86, 88, 259, 317, 337
расширенные, 68
приоритет процесса, 85, 98, 100, 101, 364, 365
повышение и понижение, 101
протокол
ARP, 232
BGP, 158
DHCP, 175, 176, 188
FTP, 187, 188, 224
HTTP, 242, 257
ICMP, 151, 161, 200, 281

IMAP, 80, 294–297
OSPF, 158
POP3, 294–297
SMB, 330, 331
SMTP, 294, 298
TCP/IP, 146, 150–152, 156, 158, 166, 175, 198, 208, 209, 212, 248, 330, 389
UDP, 151, 152, 258, 282
протоколирование, демон, 98, 230, 252, 364, 365, 379, 380, 387
процессор, 50, 97, 120, 126, 127, 130, 207, 362, 364, 365
процессы-зомби, 88, 95, 98, 379

С

сборка программы, 29, 236–241, 246–249, 301, 309–311, 321, 336, 337, 385
сервер
NFS, 231, 257, 259, 260, 263–266, 268, 270–273
времени, 281, 283–289
имен, авторитетный, 176
имен, файл конфигурации, 214, 215
сети
Ethernet, 149, 198, 201, 270, 273, 367
виртуальные частные *см.* VPN
глобальные, 149
класса A, 153
класса B, 154
класса C, 172, 173
спам, 291, 293, 295, 299, 303
борьба с, 304

Т

типы записей DNS
CNAME, 218, 222
MX, 217–219, 221, 222, 302
NS, 212, 218, 220, 221, 223
PTR, 217, 220, 221
SOA, 218, 220, 221
трафик, 175, 200, 205, 214, 324
входящий, 178, 322, 371

У

установка
из исходных текстов, 29, 236–241, 246–249, 301, 309–311, 321, 336, 337, 385
пакета, 240–242
программы, 246, 309, 321, 336, 337

Ф

файл
конфигурации
host.conf, 163, 171
hosts, 145, 163, 164, 171, 180, 181, 209, 210, 259, 265, 269, 270, 272, 333, 382, 383
hosts.allow, 181, 269, 270
nsswitch.conf, 163, 171, 382
printcap, 276, 277, 334
smb.conf, 331–333, 335, 347, 348, 359
сервера имен, named.conf, 214, 215
конфигурации при загрузке, 64, 81, 169, 170, 180, 190, 229, 230, 232, 233, 265, 289, 290, 337, 383
подкачки, 44, 60, 74, 84, 232, 262, 324, 334, 361, 362, 370
протокола, 326
устройства, 63
файлы
копирование, 44, 58, 59, 61, 80, 81, 106, 140, 141, 144, 145, 385
переименование, 59, 61, 80, 81, 115, 134
права доступа, 68, 70, 87, 88, 217
редактирование, 53, 89, 103, 107–112, 115, 120, 126, 127, 357, 358
удаление, 60, 62, 80, 104, 134, 277
фильтры пакетов (брандмауэры), 185–187, 190–193
ipchains, 186
ipfw, 190–193
iptables, 185–187
межсетевой экран, 186

Я

ядро
компиляция, 245–248, 309, 382, 385
сборка, 245–248, 309, 382, 385

По договору между издательством «Символ-Плюс» и Интернет-магазином «Books.Ru-Книги России» единственный легальный способ получения данного файла с книгой ISBN 5-93286-084-7 «UNIX. Практическое пособие администратора. 2-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» (www.symbol.ru), где именно Вы получили данный файл.