Скотт Граннеман

НЕОБХОДИМЫЙ КОД И КОМАНДЫ

Второе издание

Linux®

КАРМАННЫЙ СПРАВОЧНИК

Linux®

КАРМАННЫЙ СПРАВОЧНИК

ВТОРОЕ ИЗДАНИЕ

Linux®

PHRASEBOOK

SECOND EDITION

Scott Granneman

Addison-Wesley

800 East 96th Street, Indianapolis, Indiana 46240 USA

Linux®

КАРМАННЫЙ СПРАВОЧНИК

ВТОРОЕ ИЗДАНИЕ

Скотт Граннеман

Москва · Санкт-Петербург · Киев
2016

ББК 32.973.26-018.2.75
Г77

УДК 681.3.07

Издательский дом "Вильямс"

Зав. редакцией *С.Н. Тригуб*

Перевод с английского и редакция докт. физ.-мат. наук *Д.А. Клюшина*

По общим вопросам обращайтесь в Издательский дом "Вильямс" по адресу:
info@williamspublishing.com, http://www.williamspublishing.com

Граннеман, Скотт.

Г77 Linux. Карманный справочник, 2-е изд. : Пер. с англ. — М. : ООО "И.Д. Вильямс", 2016. — 464 с. : ил. — Парал. тит. англ.

ISBN 978-5-8459-2101-7 (рус.)

ББК 32.973.26-018.2.75

Научно-популярное издание

Скотт Граннеман

Linux. Карманный справочник

2-е издание

Литературный редактор *И.А. Попова*
Верстка *Л.В. Чернокозинская*
Художественный редактор *В.Г. Павлютин*
Корректор *Л.А. Гордиенко*

Подписано в печать 08.06.2016. Формат 84x108/32.
Гарнитура Times.
Усл. печ. л. 14,5. Уч.-изд. л. 16,5.
Тираж 1000 экз. Заказ № 3614.

Отпечатано в АО «Первая Образцовая типография»
Филиал «Чеховский Печатный Двор»
142300, Московская область, г. Чехов, ул. Полиграфистов, д.1

ООО "И. Д. Вильямс", 127055, г. Москва, ул. Лесная, д. 43, стр. 1

ISBN 978-5-8459-2101-7 (рус.)
ISBN 978-0-321-83388-4 (англ.)

Содержание

Об авторе

Скотт Граннеман — писатель, преподаватель и владелец небольшой компании. Им написано семь книг (о Firefox, Linux, Google Apps и Mac OS X) и еще две в соавторстве. Кроме того, он ведет ежемесячную рубрику на *SecurityFocus* — одном из крупнейших и наиболее влиятельных веб-сайтов, посвященных вопросам безопасности, а также публиковал статьи в *Linux Magazine*, когда он выходил в печатном виде.

Будучи преподавателем, Скотт учил тысячи студентов всех возрастов — от детей до стариков — множеству вещей, включая литературу и технологии. В настоящее время он занимает должность адъюнкт-профессора в Вашингтонском и Вебстерском университетах (Сент-Луис), где читает разнообразные курсы по информационным технологиям, социальным сетям, а также по разработке интернет- и веб-приложений. В последние несколько десятилетий фокус его лекций был заметно смещен в сторону Linux и других технологий с открытым исходным кодом. В связи с этим Скотт старается донести до слушателей с разными уровнями подготовки сведения о новых важных направлениях в разработке программного обеспечения.

Как руководитель компании WebSanity, занимающейся проектированием, разработкой и хостингом веб-сайтов, он работает с клиентами в двенадцати штатах, управляя Linux-серверами и инфраструктурой разных фирм, осваивая новые технологии и тесно сотрудничая с другими партнерами и разработчиками системы управления содержимым (Content Management System — CMS), используемой в компании WebSanity.

Посвящение

Эта книга посвящается пользователям Linux всех поколений, как старым, так и новым. Добро пожаловать!

Благодарности к первому изданию (2005 г.)

Никто не способен изучить оболочку Linux самостоятельно. Моими учителями были сотни специалистов с многолетним опытом. В своих книгах они рассказали мне и другим читателям о невероятных возможностях, предоставляемых командной строкой Linux. Книги, веб-сайты, блоги и информационные материалы, раздаваемые на встречах пользователей системы Linux, помогли мне изучить оболочку `bash` и помогают в этом по сей день. Если я смогу донести до своих читателей хоть малую часть того, что смог получить сам, то буду считать свою задачу выполненной.

Помимо моих учителей, я хочу поблагодарить тех, кто непосредственно помогал мне в работе над этой книгой.

Хочу поблагодарить моего агента Лауру Левин (Laura Lewin) за большую и разнообразную помощь, которую она мне постоянно оказывала.

Мои редакторы издательства Pearson не только предоставили мне возможность опубликовать эту книгу, но и поощряли меня по мере необходимости.

Неоценимую помощь в описании программы RPM мне оказал отличный специалист по Linux Роберт Сайтек (Robert Citek), который всегда отвечал на любые мои вопросы.

Мой давний приятель и деловой партнер Дженс Картон (Jans Carton) помогал мне сосредоточиться на конкретных вопросах и был охотным (в большинстве случаев) испытателем многочисленных новых команд и опций. Думал ли я, когда мы познакомились в пятом классе школы, что он станет таким выдающимся специалистом?

Джерри Брайан (Jerry Bryan) внимательно читал написанный мною текст и исправлял все мелкие грамматические ошибки и опечатки, сделанные мною. Обещаю, Джерри, что когда-нибудь смогу отличить "возможно" от "вероятно".[1]

Моя жена, Дениза Либерман (Denise Lieberman), терпеливо выслушивала мои бессвязные восклицания, когда мне приходила в голову новая идея, даже если она ничего не понимала из того, что я говорю. Это настоящая любовь. Спасибо, Дениза!

И наконец, хочу упомянуть мою маленькую Либби, собаку породы ши-тцу, которая всегда находила подходящий момент, чтобы положить мне лапы на колени и потребовать, чтобы я погладил и почесал ее.

Благодарности ко второму изданию (2015 г.)

К моменту выхода в свет второго издания многое изменилось (к сожалению, Либби покинула нас и отправилась на небеса), но одна вещь осталась неизменной: как и прежде, писать эту книгу мне помогало множество людей, которые заслуживают признания и благодарности.

Марк Тэбер (Mark Taber), ответственный редактор издательства Pearson, поддерживал меня в течение всего *очень* длинного периода работы над книгой. Я не могу в полной мере отблагодарить Вас, Марк. Именно благодаря Вам эта книга увидела свет.

Моя жена, Робин Уолтмен (Robin Woltman), прочитала каждую главу и сделала множество неоценимых редакторских исправлений и предложений. Ты сделала большую работу, Робин!

Роберт Сайтек (Robert Citek) просмотрел многие главы, сделав предложения и технические правки, которые всегда были весьма кстати. Именно к Роберту я обращаюсь каждый раз, когда мне требуется помощь!

Крэйг Бачек (Craig Buchek) просмотрел одну или две главы и сделал несколько полезных замечаний. Этого и следовало

[1] В оригинале: *may* и *might* — *Примеч. ред.*

ожидать от бывшего добросердечного диктатора группы пользователей Linux из Сент-Луиса.

Том Кирк (Tom Kirk) одолжил мне несколько ноутбуков из своей огромной коллекции, чтобы я мог проверить некоторые факты для главы о сетях. Том, ты — мой спаситель и лучший специалист по аппаратным средствам, которых я знаю.

Кроме моих друзей, перечисленных выше, несколько человек сообщили мне по электронной почте о проблемах, которые они обнаружили в первом издании.

- Джо Хэнчарик (Joe Hancharik) указал на проблему в разделе "Wildcards and What They Mean" главы 1. Это очень глупая ошибка с моей стороны.
- Уильям Х. Фергюсон (William H. Fergusson) нашел другую глупую ошибку в раздее "Copy Files" главы 3.
- Брайан Грир (Brian Greer) указал на проблему, которую я должен был заметить в разделе "Troubleshooting Network Problems" главы 15.

Спасибо, господа! Благодаря таким читателям, как Джо, Уильям и и Брайан, писать книги становится приятно. Обратная связь — важная вещь, даже если это — отчет об ошибках. Если вы увидите ошибку, сообщите мне, чтобы я мог ее исправить в третьем издании!

От издательства

Вы, читатель этой книги, и есть главный ее критик и комментатор. Мы ценим ваше мнение и хотим знать, что было сделано нами правильно, что можно было сделать лучше и что еще вы хотели бы увидеть изданным нами. Нам интересно услышать и любые другие замечания, которые вам хотелось бы высказать в наш адрес.

Мы ждем ваших комментариев и надеемся на них. Вы можете прислать нам бумажное или электронное письмо, либо просто посетить наш веб-сайт и оставить свои замечания там.

Одним словом, любым удобным для вас способом дайте нам знать, нравится или нет вам эта книга, а также выскажите свое мнение о том, как сделать наши книги более интересными для вас.

Посылая письмо или сообщение, не забудьте указать название книги и ее авторов, а также ваш обратный адрес. Мы внимательно ознакомимся с вашим мнением и обязательно учтем его при отборе и подготовке к изданию последующих книг.

Наши электронные адреса:

E-mail: info@williamspublishing.com
WWW: http://www.williamspublishing.com

Наши почтовые адреса:

в России: 127055, г. Москва, ул. Лесная, д.43, стр. 1
в Украине: 03150, Киев, а/я 152

Введение

Среди всех элементов системы Linux самым важным, пожалуй, является командная строка. Если вы организуете работу сервера под управлением Linux, то интерфейсные средства, вероятнее всего, будут исчерпываться оболочкой. Если система Linux установлена на вашей рабочей станции, то терминал, скорее всего, будет включен постоянно. Начинающим пользователям кажется, что они никогда не прибегнут к помощи командной строки. Однако чем больший опыт они приобретают, тем чаще обращаются к оболочке.

Оболочка во многом определяет богатые возможности и гибкость системы Linux. С помощью командной строки можно выполнять действия, которые были бы немыслимы при работе с графическим пользовательским интерфейсом. Независимо от того, насколько мощными являются такие инструменты, как KDE или GNOME (а также IceWM, или XFCE, или одна из множества других оконных сред), оказывается, что многие действия гораздо быстрее и эффективнее выполнить, пользуясь только командной строкой. Если вы хотите освоить Linux, то начинать изучение надо с командной строки.

Традиционный метод получения информации о командах — вызов страниц справочного руководства `man`. Хотя информация, представленная на них, очень полезна, зачастую ее не хватает. Причина заключается в отсутствии примеров. Конечно, кое-где приводятся несколько примеров, но больших и содержательных примеров в справочном руководстве нет. В связи с этим у пользователей любого уровня квалификации возникает проблема: одно дело — видеть полный список всех опций с их описаниями, а другое — видеть, как эти опции используются в реальных задачах.

В этой книге приведены примеры, которых так недостает справочному руководству. Я использую Linux около двадцати лет и считаю, что знаю почти все об этой прекрасной и мощной операционной системе. Я настолько сильно люблю

использовать командную строку, что у меня всегда есть открытый терминал. Более того, Linux-серверы моей компании не имеют графического пользовательского интерфейса (как я люблю!), так что я *должен* использовать терминал, чтобы работать с ними. Но я всегда жалуюсь — как и мои друзья-пользователи системы Linux из группы LUG (Linux User Group) — на недостаток примеров, найденных на страницах справочника `man`. Когда меня попросили написать *Карманный справочник по Linux* (*Linux Phrasebook*), включить в него сотни примеров, иллюстрирующих большинство важных команд Linux, я ответил: "Я не могу ждать! Такую книгу я купил бы, не задумываясь!"

Результат моих усилий вы держите в руках. Эта книга о командах Linux, которые необходимо знать. Использование каждой из них поясняется на примерах. Данная книга — всего лишь справочник, который пригодится вам сейчас и в ближайшие годы, и все же я надеюсь, что вы получите некоторое удовольствие, читая ее.

ЗАМЕЧАНИЕ

Посетите наш веб-сайт, зарегистрируйте эту книгу по адресу `informit.com/register`, и вы получите удобный доступ к дополнительным материалам и информации о возможных ошибках.

На кого рассчитана эта книга

Я старался написать книгу так, чтобы она была полезна как для новичков, только приступающих к изучению Linux, так и для опытных пользователей, применяющих оболочку для решения разных задач: от администрирования до программирования. Если вы только начали изучать Linux, эта книга расскажет вам о возможностях оболочки; если же вы работаете с этой системой многие годы, она напомнит вам о средствах, о которых вы давно забыли, или научит некоторым специальным приемам работы.

В настоящее время существуют различные оболочки: `csh`, `tcsh`, `zsh` и многие другие. Я использую оболочку `bash` (Bourne Again Shell), по умолчанию включаемую практически в каждый дистрибутивный пакет Linux. Эта оболочка хороша не только тем, что используется повсеместно: она предоставляет богатые возможности и обеспечивает гибкость в работе. Научившись работать с `bash`, вы без труда перейдете к любой другой оболочке, но саму ее в любом случае необходимо знать.

В ходе работы над книгой я использовал систему Debian — одну из наиболее распространенных версий системы Linux, дистрибутивный пакет которой легко получить. Несмотря на это, команды, которые я описываю, должны без каких-либо проблем выполняться и в вашей системе. Единственное, но важное различие связано с работой от имени пользователя `root`. Вместо регистрации под именем `root` некоторые дистрибутивные пакеты (например, Ubuntu) предполагают использование команды `sudo`. Другими словами, вместо того, чтобы вызывать `lsof firefox` от имени `root`, пользователь системы Ubuntu задает команду `sudo lsof firefox`.

ЗАМЕЧАНИЕ

Внимательные читатели первого издания, вероятно, заметили, что я использовал систему Ubuntu (или, как я ее называл, K/Ubuntu, подчеркивая, что я использовал среду KDE в системе Ubuntu). Теперь я постарался выбрать более общий вариант и предпочел систему Debian, на дистрибутивном пакете которой основана система Ubuntu.

Стремясь удовлетворить интересы как можно более широкого круга читателей, я привожу команды так, как будто их вызывает пользователь `root`, не указывая имя программы `sudo`. Если перед командой стоит символ `#`, это означает, что в системе зарегистрирован пользователь `root`; именно под этим именем вам придется войти в систему, чтобы выполнить данную команду. При работе с системой Ubuntu и другими

аналогичными дистрибутивными пакетами в этом случае указывается имя программы `sudo`.

В заключение, для того чтобы не увеличивать объем книги, я сократил строки вывода команд. Например, на вашем компьютере, работающем под управлением операционной системы Linux, после ввода команды `ls -l` вы обычно видите строку вывода

```
-rwxr-xr-x 1 scott admins 1261 Jun 1 2012 script.sh
```

Вы увидите эту строку в разных местах книги, там где это будет уместно, но чаще вы обнаружите что-то вроде

```
-rw-r--r 1 scott admins script.sh
```

В данном случае я сократил информацию, которую не считал важной, чтобы вывод занимал одну строку, а не две. Дополнительные строки могли бы сбить читателей с толку, поэтому я внес те изменения там, где это казалось целесообразным (или когда мой редактор был шокирован возможным количеством страниц в книге!!).

ПОДСКАЗКА

Большинство информации, которую я привожу о системе Linux, относится и к вариантам системы UNIX, например BSD и OS X. Обратите внимание на то, что я говорю *большинство*, но не *вся* и *не подавляющее большинство*. Если вы будете помнить об этом, то данная книга поможет вам работать и с этими операционными системами.

О втором издании

Когда Марк Табер, мой редактор в издательстве Pearson, начал разговор о втором издании *Linux Phrasebook*, я воспользовался этой возможностью. Я сам использовал свою собственную книгу в качестве справочника по нескольку раз в месяц и благодаря этому за эти годы заметил ошибки (каждый раз

вздрагивая при этом), и обнаружил много вещей, которые хотел бы изменить, удалить или добавить.

Моя цель состояла в том, чтобы сделать второе издание бестселлером для новых читателей, а также для владельцев первого издания *Linux Phrasebook*. Это нельзя назвать так называемым новым изданием с немногочисленными и небольшими изменениями. Совсем нет. Ниже перечислены изменения, которые я внес в новое издание.

- Я разделил прежнюю главу 2 "Основы" на две главы: главу 2 "Навигация по файловой системе", и главу 3 "Создание и разрушение". Старая глава 2 была непропорционально длинной и переполненной разнообразными командами. Новое разделение делает ситуацию намного более управляемой и разумной (хотя это означало, что каждую последующую главу пришлось перенумеровать).
- Я удалил главу 6 "Печать и управление заданиями печати" и главу 16 "Организация сети в системе Windows", просто потому, что теперь они уже не кажутся столь важными, как это было десять лет назад. Кроме того, большинство людей, которым нужно распечатать какой-нибудь текст или связаться друг с другом в сети на основе системы Windows, будут использовать инструменты графического пользовательского интерфейса, которые больше подходят для такой работы. Впрочем, не приходите в отчаяние: оригинальные главы из первого издания все еще можно найти на моем веб-сайте (`www.granneman.com/linux-redaction`)[2].
- Я добавил новую главу 7 "Управление текстовыми файлами с помощью фильтров". Она содержит *большой* объем новой информация, и я знаю, что вы сочтете ее очень полезной!
- Я удалил разделы из старых глав 2 (теперь главы 2 и 3), 3 (теперь глава 4), 7 (8), 8 (9), 9 (10), 10 (11), и 14 (15). Вы

[2] Актуальность ссылок не гарантируется. — *Примеч. ред.*

найдете оригинальные разделы из первого выпуска на моем веб-сайте (`www.granneman.com/linux-redaction`)[3].

- Я добавил новые разделы в главах 1–6 и 8–13. Кроме того, я почти вдвое увеличил главу 15, оставив устаревшие команды (поскольку они все еще есть в большинстве дистрибутивов) и добавив новые.
- Я переместил разделы в новые главы, где они будут более уместными. Особенно это касается главы 8.
- Я пересмотрел содержание каждой главы, исправил ошибки, переписал части, которые были неясными, добавил дополнительные примечания и подсказки и улучшил примеры, добавляя или пересматривая текст.
- Мимоходом я упомянул еще много команд, таких как `ssh-agent`, `wput`, `htop`, `dnf`, `pandoc`, `rename`, `whoami` и `iconv`.
- Я включил небольшие фрагменты информации о многих вещах, которые полезно знать, например, о переменных, циклах `for`, заданиях для программы `cron`, аргументах и файле `sources`. "Есть многое на свете ..."

И наконец, подсказка: если какие-либо ссылки не работают, попытайтесь найти их в архиве Интернета Wayback Machine, который можно найти по адресу `https://archive.org`. После этого сообщите мне, чтобы я мог исправить ссылку в будущих изданиях.

Спасибо за то, что дочитали книгу до этого места. Я действительно надеюсь, что вы получите удовольствие от нового издания *Карманного справочник по Linux*!

Основные соглашения

В данной книге использовались следующие соглашения.

- Для того чтобы выделить программный код на фоне обычного текста, используется моноширинный шрифт. Таким шрифтом в книге представляются символы, которые

[3] Актуальность ссылки не гарантируется. — *Примеч. ред.*

отображаются на экране компьютера. Например, в двух следующих абзацах используется моноширинный шрифт.

- По умолчанию команда `df` выводит результаты в килобайтах, однако они будут проще для восприятия, если использовать опцию `-h` (или `--human-readable`).
- `Этот текст выводится на экран терминала.`
- Стрелка ↳ в начале строки кода означает, что она является слишком длинной и не помещается на странице книги. Это подразумевает продолжение кода на той же строке без разрыва.
- Кроме того, в книгу включены фрагменты дополнительной информации, имеющей отношение к основному материалу.

ЗАМЕЧАНИЕ

Приводятся интересные сведения, связанные с контекстом.

ПОДСКАЗКА

Описывается способ, позволяющий проще или быстрее решить конкретную задачу.

ПРЕДУПРЕЖДЕНИЕ

Предупреждение о возможных проблемах. Зная о них, вы сможете быстрее устранить ошибки.

1

Общие сведения о работе с командной строкой

Перед тем как вплотную приступить к изучению оболочки `bash`, необходимо обсудить некоторые особенности операционной системы. Зная их, вы сможете более эффективно работать с материалом этой книги. Некоторые из этих особенностей вам известны, другие покажутся не столь очевидными.

Файлы, и ничего, кроме файлов

Все, с чем вы встретитесь в системе Linux, — это файлы. Абсолютно все! Очевидно, что текстовый документ — это файл. Изображения, аудиоданные в формате MP3 и видеофрагменты — это, несомненно, файлы.

Но как насчет каталогов? Оказывается, это тоже файлы — файлы специальных типов, содержащие информацию о других файлах. Дисковые устройства — это большие файлы. Сетевые соединения — тоже файлы. Даже исполняемый процесс — это файл.

С точки зрения системы Linux файл представляет собой поток битов или байтов. Система не интересуется тем, что означает каждый байт, — это забота конкретных программ, выполняющихся в операционной системе. Для Linux и документ, и сетевое соединение — всего лишь файлы. Как обрабатывать текстовый документ, знает редактор, а сетевое приложение умеет работать с сетевым соединением.

На протяжении всей книги будут упоминаться файлы. Подобные упоминания можно воспринимать так: "файлы, каталоги, подкаталоги и любые другие объекты". Многие из

команд, которые мы вскоре рассмотрим, одинаково хорошо работают и с документами, и с каталогами, поэтому смело экспериментируйте с ними.

ЗАМЕЧАНИЕ

Строго говоря, выражение "файлы, и ничего, кроме файлов" верно не совсем. Точнее, как заметил Linux Torvalds, говорить "потоки байтов, и ничего, кроме потоков байтов". Более подробно эта концепция описана в статье Википедии "Everything is a file", расположенной по адресу `http://en.wikipedia.org/wiki/Everything_is_a_file`, а также в статье "What 'Everything Is a File' Means on Linux" на сайте How-To Geek по адресу `www.howtogeek.com/117939/htg-explains-whateverything-is-a-file-means-on-linux/`.

Максимальная длина имени файла

Те, кто имеет за плечами опыт работы с системой MS-DOS, помнят, что в ней длина имени файла была ограничена восемью символами, за которыми могли следовать три символа расширения. В результате получались "чрезвычайно информативные" имена, например `MSRSUME1.DOC`. В системах Mac, предшествующих OS X, имя файла могло содержать до 31 символа. Несмотря на то что имена файлов были достаточно длинными, иногда приходилось прибегать к сокращениям, расшифровку которых вскоре уже никто не помнил.

В системе Linux (и, конечно же, Unix) имена файлов могут насчитывать до 255 символов. Этого с лихвой хватает, чтобы сформировать имя, полностью описывающее назначение файла. Если вы используете хотя бы половину допустимой длины, у вас возникнет другая проблема — как разместить имя на экране дисплея. Если же этот вопрос вас не волнует, смело

формируйте любое имя и следите лишь за тем, чтобы вам было удобно с ним работать.

На практике желательно, чтобы имена файлов не превышали 80 символов — именно столько их помещается в строке терминала. Если вы не уложитесь в 80 символов, часть имени файла будет перенесена на другую строку. Ограничивать длину имени — не требование, а всего лишь совет. Все 255 символов в вашем полном распоряжении.

Регистр символов в именах файлов

В отличие от Windows и Mac OS, в системе Linux имена файлов чувствительны к регистру символов. В частности, вы можете встретить в одном каталоге все три приведенных ниже файла.

- `bookstobuy.txt`
- `BooksToBuy.txt`
- `BoOkStObUy.txt`

С точки зрения файловой системы Linux это различные имена. Если вы попытаетесь создать файлы с этими же именами в Windows или Mac OS, то для файла `BooksToBuy.txt` система предложит задать другое имя или отказаться от попыток создать его. Причина в том, что в каталоге на этот момент находится файл `bookstobuy.txt`.

Чувствительность к регистру символов также означает, что при вводе команд они должны в точности совпадать с именами файлов, поддерживающих их. Так, например, удаляя файл с помощью команды `rm`, нельзя вводить `RM`, `Rm` или `rM`. Надо также следить за написанием имен, задаваемых в качестве параметров. Если вы захотите удалить файл `bookstobuy.txt`, а укажете имя `BooksToBuy.txt`, то лишитесь совсем не того файла, с которым предполагали расстаться.

Из сказанного можно сделать вывод, что Linux требует точности. Кстати, точность — это совсем не плохо. В то же время

Linux обеспечивает такую степень гибкости, которую нельзя получить при работе с другими системами. Подобное сочетание необходимой точности и предоставляемой гибкости делает Linux очень удобным в применении, однако некоторые пользователи поначалу испытывают неудобства при работе с этой системой.

Специальные символы в именах файлов

В каждой операционной системе определены символы, которые нежелательно или вовсе нельзя использовать в именах файлов и каталогов. Так, в Mac OS к числу подобных запрещенных символов относится двоеточие (:), а пользователям Windows нельзя включать в имена файлов обратную косую черту (\). Есть такие символы и в системе Linux. Перед тем как переходить к их рассмотрению, уточним сначала, какие знаки наверняка допустимы в именах файлов:

- числа;
- буквы (как верхнего, так и нижнего регистра);
- точка (.);
- знак подчеркивания (_).

Что же касается остальных символов, то в них нельзя быть полностью уверенным. Возможно, некоторые из них не станут источником проблем, с другими возникнут сложности из-за того, что та или иная оболочка будет интерпретировать их специальным образом, а некоторые символы недопустимо применять ни при каких обстоятельствах.

Безусловно, запрещена косая черта, так как этот символ используется для разделения каталогов и файлов. Предположим, вы создали список книг, которые хотите приобрести, и собираетесь сохранить этот список в виде файла с `books/to_buy.txt`. Такое имя вы выбрали потому, что собираетесь впоследствии создать также файлы `books/on_loan.txt` и `books/lost.txt`. Однако попытка создать файл `/home/scott/documents/`

books/to_buy.txt окончится неудачей. Оболочка посчитает, что books — это каталог в составе каталога documents, и не сможет найти его.

Вместо косой черты следует ввести знак подчеркивания (этот символ без проблем был интерпретирован в части имени файла to_buy). Можно также и вовсе обойтись без разделителей и назвать файл booksToBuy.txt или BooksToBuy.txt.

Для формирования имени файла можно использовать дефисы, например books-to-buy.txt, но знаки подчеркивания предпочтительнее. Если вы все же хотите использовать именно символ -, то не помещайте его в начале имени файла (-books_to_buy.txt) или после пробела (books - to buy). Как вы увидите в дальнейшем, знак - служит признаком опции при вводе команды. Как будет показано в главе 3 "Создание и уничтожение", команда rm удаляет файл, и если вы зададите команду rm -books_to_buy.txt, то оболочка отобразит следующее сообщение об ошибке:

```
rm: invalid option -- b
```

Если необходимо, можете включать в имена файлов пробелы, например, допустимо создать файл books to buy.txt, однако при этом на вас возлагается дополнительная обязанность — сообщать оболочке, что пробелы являются частью имени. Дело в том, что оболочка обычно интерпретирует пробелы как разделители между параметрами. При попытке удалить файл books to buy.txt оболочка интерпретирует соответствующую команду следующим образом: удалить файл books, затем файл to и, наконец, файл buy.txt. Мало того, что файл books to buy.txt останется на диске, вы можете случайно удалить нужные вам файлы.

Как же поступать с пробелами в именах файлов? Этот же вопрос возникает по отношению к символам * и ?, применение которых будет рассмотрено в следующем разделе. Одинарная и двойная кавычки также имеют специальное назначение. Поступать со всеми указанными выше символами можно поразному, но лучше всего не использовать их в именах файлов.

Если по каким-то причинам обойтись без них нельзя, надо указывать перед каждым символом специального назначения обратную косую черту. Этим вы сообщите оболочке о том, что специальное значение символа надо игнорировать и интерпретировать его как обычный знак. Вводить команды подобным образом довольно утомительно, так как надо убедиться, что обратная косая черта указана везде, где необходимо.

```
$ rm Why\ shouldn\'t\ I\ name\ files\ with\ \*\?.txt
```

Есть и более простой способ — поместить имя файла в кавычки. Результат будет таким же, как если бы вы предварили каждый символ специального назначения обратной косой чертой.

```
$ rm "Why shouldn't I name files with *?.txt"
```

Такой подход даст нужный результат, нельзя лишь при вводе команды забывать о кавычках. Несмотря на обилие решений, самое лучшее из них — вовсе не включать символы специального назначения в имя файла. В табл 1.1 перечислены некоторые из таких символов и приведены рекомендации по работе с ними.

Таблица 1.1. Использование специальных символов в именах файлов

Символ	Рекомендации по использованию
/	Нельзя использовать ни при каких обстоятельствах
\	Должен быть предварен таким же символом. Применять не рекомендуется
-	Нельзя использовать в начале имени файла или каталога
[ ]	Каждый из этих символов должен быть предварен обратной косой чертой. Применять не рекомендуется
{ }	Каждый из этих символов должен быть предварен обратной косой чертой. Применять не рекомендуется
*	Должен быть предварен обратной косой чертой. Применять не рекомендуется
?	Должен быть предварен обратной косой чертой. Применять не рекомендуется
'	Должен быть предварен обратной косой чертой. Применять не рекомендуется
"	Должен быть предварен обратной косой чертой. Применять не рекомендуется

Символы групповых операций

Предположим, в одном из каталогов на вашем компьютере содержатся двенадцать файлов с изображениями и один текстовый файл.

```
libby1.jpg          libby8.jpg
libby2.jpg          libby9.jpg
libby3.jpg          libby10.jpg
libby4.jpg          libby11.jpg
libby5.jpg          libby12.jpg
libby6.jpg          libby1.txt
libby7.jpg
```

Представьте себе, что вам надо удалить эти файлы, используя команду `rm` (она будет рассмотрена в главе 3). Конечно, файлы можно удалять по одному, но это утомительно, да и компьютер предназначен для автоматизации рутинных операций. В данном случае целесообразно применить символы групповых операций, которые позволяют задать с помощью одного параметра команды одновременно несколько файлов.

Групповые операции задаются посредством

- звездочки (`*`);
- знака вопроса (`?`);
- квадратных скобок (`[]`);
- фигурных скобок (`{}`).

ЗАМЕЧАНИЕ

Фигурные скобки отличаются от остальных. Первые три групповые операции интерпретируются оболочкой `bash` как расширение имен файлов (filename expansion), а фигурные скобки означают раскрытие скобок (brace expansion). Первые три групповые операции соответствуют существующим файлам, а фигурные скобки можно применять как для использования существующих файлов, так и для создания новых.

Звездочка отмечает любое (в том числе нулевое) количество любых символов. В табл. 1.2 приведено несколько примеров использования звездочки и показано, на какие файлы воздействует соответствующая команда.

Таблица 1.2. Использование символа * в составе команды

Команда	Файлы
`rm libby1*.*`	`libby10.jpg` — `libby12.jpg`, а также `libby1.txt`
`rm libby*.jpg`	`libby1.jpg` — `libby12.jpg`, но не `libby1.txt`
`rm *txt`	`libby1.txt`, но не `libby1.jpg` — `libby12.jpg`
`rm libby*`	`libby1.jpg` — `libby12.jpg`, а также `libby1.txt`
`rm *`	Все файлы в каталоге

Символ `?` соответствует одному произвольному символу. Примеры его применения приведены в табл. 1.3.

Таблица 1.3. Использование символа * в составе команды

Команда	Файлы
`rm libby1?.jpg`	`libby10.jpg` — `libby12.jpg`, но не `libby1.txt`
`rm libby?.jpg`	`libby1.jpg` — `libby9.jpg`, но не `libby10.jpg`
`rm libby?.*`	`libby1.jpg` — `libby9.jpg`, а также `libby1.txt`

Квадратные скобки позволяют задавать один символ из набора (например, выражение `[12]` означает символы `1` и `2`, но не `12?`, а выражение `[abc]` означает символы `a`, `b` или `c`, но не `ab`, `bc` или `abc`) или символ, принадлежащий определенному диапазону (например, `[1-3]`, что соответствует символам `1`, `2` или `3`). В последнем случае границы диапазона разделяются дефисом. В табл. 1.4 приведено несколько примеров использования квадратных скобок.

Таблица 1.4. Использование квадратных скобок в составе команды

Команда	Файлы
`rm libby1[12].jpg`	`libby11.jpg` и `libby12.jpg`, но не `libby10.jpg`

Окончание табл. 1.4

Команда	Файлы
`rm libby1[0-2].jpg`	`libby10.jpg` — `libby12.jpg`, но не `libby1.jpg`
`rm libby[6-8].jpg`	`libby6.jpg` — `libby8.jpg`

Эти символы будут встречаться вам на протяжении всей книги, поэтому постарайтесь запомнить их назначение, так как они существенно упрощают работу с файлами.

Скобки `{}` также задают два вида соответствий: строки и диапазоны. Если вы хотите раскрыть скобки для строк, то должны разделить свой список запятыми, а затем оболочка `bash` перечислит список всевозможных комбинаций, такой, например, как приведен ниже.

```
$ ls
huan.jpg libby.gif libby.jpg libby.png
.libby.tiff
$ ls libby.{jpg,png}
libby.jpg libby.png
$ ls {libby,huan}.jpg
huan.jpg libby.jpg
```

Ваш список строк на самом деле должен быть списком, хотя в этом случае вы не сможете просто использовать выражение `{jpg}`, или оно просто не будет распознаваться.

В замечании, приведенном ранее в этом разделе, упоминалось, что раскрытие фигурных скобой может использоваться и для создания файлов, и для установления соответствия с уже существующими. В главе 3 будет описана команда `mkdir`, которая создает каталоги. Рассмотрим пример, в котором используется раскрытие фигурных скобок для быстрого создания нескольких каталогов.

```
$ mkdir {dogs,cats,wombats}
$ ls
cats dots wombats
```

В этом случае не использовалось соответствие существующим файлам или каталогам (для которых используются символы *, ? и []); вместо этого использовались фигурные скобки, чтобы передать команде mkdir имена новых каталогов, которые необходимо создать.

Важно иметь в виду, что, в то время как групповая операция [] соответствует отдельным символам, операция {} соответствует строкам, но строка может состоять из единственного символа! Это может создать некоторую путаницу, которую можно увидеть в результатах четвертой команды.

```
$ ls
testa testab testb testbc testc
$ ls test[a,b]
testa testb
$ ls test{a,b}
testa testb
$ ls test[ab,bc]
testa testb testc
$ ls test{ab,bc}
testab testbc
```

Вторая и третья команды, test[a, b] и test{a, b}, приводят к одинаковым результатам, потому что они обе ищут отдельные символы. Но как только вы забудете, что операция [] работает только с отдельными символами, получите неожиданные результаты. Команда test[ab, bc] не выводит на экран строки testab и testbc, а вместо этого возвращает строки testa, testb и testc, потому что, с учетом особенностей этой команды, вы приказали ей искать строку testa, затем testb, затем test, (т.е. строку test, сопровождаемую запятой!), затем снова testb, а потом testc. Команда нашла три соответствия (игнорируя дубликат запроса) и выдала их вам. С другой стороны, поскольку операция {} работает со строками, она знает, что вы хотите видеть строки testab и testbc, и выводит их на экран. И это, надо надеяться, делает различия между квадратными и фигурными скобками немного более четкими.

Что касается диапазонов, то эта функция позволяет определить начальную и конечную точку внутри скобок { и }. Они могут быть либо числами, либо буквами, но не тем и другим одновременно. Кроме того, они должны быть разделены двумя точками. Рассмотрим примеры, используя команду touch (описываемую в главе 3).

```
$ touch finny{1..3}
$ ls
finny1 finny2 finny3
$ touch finny{a..c}
$ ls
finny1 finny2 finny3 finnya finnyb finnyc
```

Интересно, что можно объединить два варианта раскрытия фигурной скобки, и все возможные комбинации будут либо обнаружены среди существующих, либо созданы.

```
$ touch finny{1..3}{a..c}
finny1a finny1b finny1c finny2a finny2b
↳finny2c finny3a finny3b finny3c
```

Итак, чтобы подытожить сведения о символах подстановки, полезно знать, что вы можете объединить их в любые комбинации. Хотите найти соответствия libby1.jpg, libby2.jpg и libby1.txt, но не libby3.jpg или libby.jpg? Используйте команду libby[1-2].*, и дело в шляпе! Как насчет Libby1.jpg, libby1.jpg (напомним, что в системе Linux верхний регистр L и нижний регистр l относятся к разным файлам), libby2.jpg и Libby3.txt, но не libby4.jpg? Используйте команду ?ibby[1-3].{jpg, txt}. И так далее, и тому подобное.

Мы будем использовать символы подстановки по всей книге, поэтому целесообразно рассмотреть их уже сейчас. Они позволяют работать с файлами с помощью командной строки, что намного проще, и вы увидите, насколько они полезные.

Специальные файлы, на которые влияет командная строка

Всюду в этой книге мы будем ссылаться на некоторые скрытые сценарии запуска в вашем корневом каталоге (скрытые, потому что они начинаются с точки, и поэтому часто в разговорной речи упоминаются как *файл с точкой* (*dotfiles*)). Эти сценарии могут заметно влиять на работу вашей оболочки. Мы никогда не будем говорить о них слишком подробно, потому что для этого просто нет места, но они заслуживают внимательного изучения. Оболочка может удовлетворять разные потребности, и вы должны настроить эти файлы, чтобы удовлетворить именно свои.

ЗАМЕЧАНИЕ

Если вас интересуют мои файлы с точкой, посетите мой блог, расположенный по адресам `http://ChainsawOnATireSwing.com` и `https://github.com/rsgranne/syno-dotfiles`. Веб-служба GitHub позволяет очень легко просматривать файлы с точкой, созданные другими людьми, — достаточно зайти на сайт `http://dotfiles.github.io`. Приложив немного старания, вы тоже сможете сохранять свой файл с точкой на веб-службе GitHub!

Прежде чем рассказать о сценариях запуска, сначала рассмотрим два способа классификации оболочек: *с регистрацией* (login) и *без регистрации* (nonlogin), а также *интерактивный* (interactive) и *неинтерактивный* (noninteractive) режимы.

Оболочка входа в систему с регистрацией — сюрприз! — запускается вашей операционной системой, когда вы входите в систему локально или по протоколу SSH (его мы рассмотрим в главе 16). Когда вы запускаете оболочку регистрации, оболочка `bash` настраивает ее, читая файл `/etc/profile`, который

описывает окружение. Затем оболочка bash ищет следующие файлы по порядку, останавливаясь на первом найденном.

- ~/.bash_profile. Напоминает файл .profile, но относится только к оболочке bash.
- ~/.bash_login
- ~/.profile. Совместим с оболочкой Борна sh, поэтому оболочки, совместимые с ней, могут использовать его.

Эти файлы, помимо прочего, устанавливают переменные окружения, которые передаются любым процессам, запускаемым оболочкой входа в систему, включая подоболочки (подоболочки создаются, когда оболочка запускает другую оболочку в соответствии со своим сценарием). Кроме переменных окружения, эти файлы могут также обращаться к программам, зависящим от оболочки, которые вы хотите автоматически настроить или запустить в момент входа в систему, такие как ssh-agent (см. главу 16) и bash-completion (см. http://chnsa.ws/7d).

Оболочка входа в систему без регистрации — это оболочка, в которой вы не регистрируетесь (снова сюрприз!). Например, подоболочка — это оболочка входа в систему без регистрации. Эта оболочка открывается в средах GNOME, KDE, Unity или в любой другой настольной среде Linux или многооконном файловом менеджере.

ЗАМЕЧАНИЕ

Интересно, что система Mac OS X интерпретирует все оболочки как оболочки входа в систему с регистрацией, поэтому вместо любых файлов можно использовать файл .profile.

Итак, оболочки могут быть с регистрацией и без регистрации. Но кроме этого, они могут быть интерактивными и неинтерактивными.

Интерактивная оболочка — это оболочка, которая отвечает на команды, введенные вами, иначе говоря, реагирует на

ваши действия и отправляет вывод в потоки STDOUT и STDERR (см. главу 5). Неинтерактивная оболочка обычно используется, когда выполняется сценарий оболочки, во время которого вы не вводите команды и получаете вывод непосредственно.

ЗАМЕЧАНИЕ

Интерактивная оболочка вполне может выполнять сценарии, а неинтерактивная может реагировать на команды, но это редкость, поэтому я постараюсь сохранить общность изложения.

Когда вы запускаете интерактивную оболочку входа в систему без регистрации, оболочка bash считывает и выполняет команды из файла /etc/bash.bashrc, если он есть (одни дистрибутивы включают его, а другие не делают этого). Этот файл относится ко всем пользователям, использующим оболочку bash. После этого оболочка bash использует в качестве источника данных файл .bashrc каждого пользователя, расположенный в его корневом каталоге. Вы должны использовать этот файл для своих параметров настройки, специфичных для оболочки bash, например опции (см. http://chnsa.ws/7c) и prompts (см. http://chnsa.ws/7b).

В файл .bashrc можно также помещать псевдонимы и функции (см. главу 12), но как только вы приобретете побольше опыта, то узнаете, как лучше их загружать в файл .bash_aliases (и возможно, в файл .bash_functions). Для того чтобы вызвать команды из файла .bash_aliases, поместите следующие команды в файл .bashrc.

```
if [ -f ~/.bash_aliases ]; then
source ~/.bash_aliases
fi
```

Если файл .bash_aliases существует, то он станет источником данных; если нет — то будет проигнорирован. Как вы увидите, этот файл — прекрасный инструмент.

После выхода из оболочки входа в систему с регистрацией оболочка bash считывает и выполняет команды из файла ~/.bash_logout, если он существует. Он используется нечасто, но среди тех, кто его использует, одной из наиболее популярных является команда clear, которая обсуждается в следующем разделе.

ПОДСКАЗКА

Если хотите настроить свою среду оболочки bash еще точнее, то изучите разные варианты использования файла ~/.inputrc. С формальной точки зрения, когда вы редактируете файл ~/.inputrc, то не настраиваете оболочку bash; на самом деле вы настраиваете библиотеку Readline, которую оболочка bash использует для редактирования командных строк. Так или иначе, с помощью этого файла можно делать удивительные вещи. Некоторые из них описаны в моем сообщении в блоге на сайте www.chainsawonatireswing.com/2012/05/13/fun-with-inputrc-part-1. Ознакомьтесь также с превосходными идеями Бретта Терпстры (Brett Terpstra) на сайте http://brettterpstra .com/2015/07/09/shell-tricks-inputrc-binding-fun/.

Если на экране слишком много информации, сделайте перезагрузку

```
clear
```

Ваша первая команда, та, которую стоит использовать, читая эту книгу — clear. Допустим, вы проверили несколько команд: некоторые успешно, другие менее успешно. Ваш терминал теперь заполнен командами, строками вывода, ошибками и мусором. Для того чтобы избавиться от них, введите команду

clear, и все это исчезнет и будет заменено приглашением в левом верхнем углу, так, будто вы начинаете работу с начала.

Вот и все. Команда clear — это одна из нескольких команд Linux, у которой нет параметров или опций. Это быстрый и простой способ сразу вернуться в начало.

ЗАМЕЧАНИЕ

В главе 12 вы узнаете о команде history, которая позволяет увидеть и выполнить заново все предыдущие команды. Учтите, что использование команды clear не влияет на вашу историю; она просто очищает экран.

Выводы

В данной главе мы рассмотрели вопросы, которые имеют отношение к работе системы Linux и которые, возможно, неочевидны для некоторых начинающих пользователей. Приведенные здесь сведения, возможно, несколько прояснят материал последующих глав. Они помогут ответить на вопрос: почему система отказывается скопировать каталог, в имени которого содержатся пробелы, или как удалить одновременно 1000 файлов, или почему не работает команда RM bookstobuy.txt? Вооружившись минимальными знаниями, вы сможете избежать многих ошибок, которые часто допускают новички.

Теперь самое время перейти к рассмотрению конкретных команд.

2

Основные команды

Эта глава посвящена основным командам — тем, которые каждый специалист применяет по нескольку раз в день. Эти команды можно сравнить с молотком, отверткой и плоскогубцами, которые рабочий использует очень часто и старается держать под рукой. Изучив команды, рассмотренные в данной главе, вы сможете управлять оболочкой и получите массу полезных сведений о своих файлах, каталогах, а также о системном окружении. В частности, вы узнаете о метаданных — данных, описывающих ваши данные, — которые система Linux вынуждена отслеживать (и вы, возможно, будете удивлены, узнав, насколько их много).

ЗАМЕЧАНИЕ

Редактируя книгу для второго издания, я удалил раздел, посвященный команде `mkdir -v` (она показывает вам, что именно выполняет команда `mkdir`) и команде `rm -v` (делает тоже самое, но для команды `rm`). Исходный текст вы можете найти на моем сайте `www.granneman.com/linux-reductions`. Кроме того, я перенес разделы, посвященные командам `touch`, `mkdir`, `cp`, `mv`, `rm` и `rmdir`, в главу 3 (для этого пришлось перенумеровать все последующие главы!). Кроме того, раздел, посвященный команде `su`, был перемещен в главу 8, где он более уместен.

Вывод списка файлов и каталогов

`ls`

Команда `ls` — вероятно, одна из самых распространенных. Ведь перед тем как выполнять какие-либо действия с файлами,

необходимо выяснить, есть ли они в каталоге (напомним, что термины *файл* и *каталог* являются взаимозаменяемыми). Сделать это позволяет команда `ls`, которая отображает список файлов и подкаталогов, находящихся в конкретном каталоге.

ЗАМЕЧАНИЕ

Команда `ls` может показаться очень простой, — просто покажи мне файлы! — однако это впечатление обманчиво. Существует бесчисленное число вариантов этой команды, и с некоторыми из них вы вскоре познакомитесь.

После ввода команды `ls` вы получите сведения о содержимом текущего каталога. Сразу после регистрации в системе текущим становится ваш рабочий каталог. Задайте команду `ls`, и вы увидите информацию, подобную приведенной ниже.

```
$ ls
alias Desktop   iso   pictures program_files   todo
bin   documents music podcasts src             videos
```

Вывод содержимого произвольного каталога

```
ls [папка]
```

Для того чтобы узнать содержимое каталога, не обязательно, чтобы этот каталог был текущим. Предположим, вы находитесь в своем рабочем каталоге, а хотите узнать, что содержится в каталоге `music`. Для этого достаточно ввести команду `ls` и указать после нее имя каталога, содержимое которого вы хотите определить.

```
$ ls music
Buddy_Holly   Clash   Donald_Fagen  new
```

В этом примере мы использовали относительный путь, но с тем же успехом можно указать полный, или абсолютный, путь.

```
$ ls /home/scott/music
Buddy_Holly   Clash   Donald_Fagen  new
```

Возможность указания относительного или абсолютного пути очень удобна в тех случаях, когда необходимо узнать, какие файлы содержатся в каталоге, но вы не хотите перемещаться по файловой системе. Вы не уверены, есть ли у вас видеофайл Bengal Tiger, который ваш брат записал в зоопарке? Выполните следующую команду (~ — это псевдоним, обозначающий рабочий каталог текущего пользователя):

```
$ ls ~/videos
airhorn_surprise.wmv
apple_knowledge_navigator.mov
b-ball-e-mail.mov
carwreck.mpg
nerdtv_1_andy_hertzfeld
nerdtv_2_max_levchin_paypal
nerdtv_3_bill_joy
tiger.wmv
Ubuntu_Talk-Mark_Shuttleworth.mpeg
```

Да, файл `tiger.wmv` существует.

Использование символов групповых операций при определении содержимого каталога

```
ls *
```

Вы только что научились находить файл в каталоге; теперь рассмотрим более быстрый способ решения той же задачи. Если вы знаете, что нужный вам файл Bengal Tiger содержит данные в формате Windows Media (я уже слышу звуки неодобрения!) и, следовательно, имеет расширение `.wmv`, то вы

можете использовать символы групповых операций, чтобы вывести только файлы, оканчивающиеся на `.wmv`.

```
$ ls ~/videos
airhorn_surprise.wmv
apple_knowledge_navigator.mov
b-ball-e-mail.mov
carwreck.mpg
nerdtv_1_andy_hertzfeld
nerdtv_2_max_levchin_paypal
nerdtv_3_bill_joy
tiger.wmv
Ubuntu_Talk-Mark_Shuttleworth.mpeg
$ ls ~/videos/*.wmv
airhorn_surprise.wmv   tiger.wmv
```

Существует еще один метод, который также предполагает применение символов групповых операций: в данном случае можно искать файл, в имени которого содержится слово `tiger`.

```
$ ls ~/videos/*tiger*
tiger.wmv
```

ЗАМЕЧАНИЕ

Если символам групповых операций соответствует реальный каталог, то содержимое этого каталога будет выведено на экран. Если вы хотите игнорировать это соответствие и избежать вывода на экран содержимого подкаталогов, добавьте опцию `-d`.

Просмотр содержимого подкаталогов

```
ls -R
```

При необходимости можно просмотреть содержимое нескольких подкаталогов с помощью одной команды. Предположим, вы пришли на встречу членов группы пользователей Lunux и сразу окунулись в энергичный водоворот событий.

"Эй! — кричит кто-то из ваших знакомых, — нет ли у кого-нибудь ISO-файл новой версии Kubuntu?" Вы припоминаете, что загрузили ее несколько дней назад, но, чтобы быть уверенным, выполняете следующую команду (вместо `ls -R` можно задать `ls --recursive`):

```
$ ls -R ~/iso
/home/scott/iso:
debian-6.0.4-i386-CD-1.iso knoppix ubuntu

/home/scott/iso/knoppix:
KNOPPIX_V7.2.0CD-2013-06-16-EN.iso
↳KNOPPIX_V7.4.2DVD-2014-09-28-EN.iso

/home/scott/iso/ubuntu:
kubuntu-15.04-desktop-amd64.iso
↳ubuntu-15.04-desktop-amd64.iso
ubuntu-14.04.3-server-amd64.iso
```

И действительно, в каталоге `~/iso/ubuntu` есть файл `ubuntu-15.04-desktopamd64.iso`. Опция `-R` задает рекурсивный просмотр каталога `iso`, и вы получаете сведения о содержимом этого каталога и всех его подкаталогов. Для каждого каталога отображается его путь относительно того каталога, с которого вы начали работу, затем выводится двоеточие, а после него — содержимое каталога. Учтите, что при большом количестве подкаталогов опция рекурсивного выполнения становится бесполезной, так как на экран выводится большой объем информации и найти сведения о нужном файле проблематично. Конечно, если вам надо всего лишь убедиться, что в подкаталогах данного каталога содержится большое количество файлов, эта опция подходит как нельзя лучше, однако необходимость в подобных сведениях возникает нечасто.

Вывод содержимого каталога в один столбец

```
ls -1
```

До сих пор мы имели дело с установками команды `ls` по умолчанию, согласно которым содержимое каталога выводится в алфавитном порядке в несколько столбцов, причем между соседними столбцами помещается как минимум два пробела. Но что делать, если вам надо вывести данные в другом формате?

Если вывод в несколько столбцов вас не устраивает, вы можете отобразить результаты выполнения команды `ls` в один столбец. Для этого используется команда `ls -1` (или `ls --format=single-column`).

```
$ ls -1 ~/bin
Desktop
documents
iso
music
pictures
src
videos
```

Если в каталоге содержится очень много файлов, да к тому же вы задали опцию рекурсивного выполнения, то выводимый список может стать неуправляемым. Поэтому, убедившись, что процесс вывода данных, полученных по команде `ls -lR ~/`, грозит затянуться надолго, нажмите комбинацию клавиш <Ctrl+C>, чтобы прервать работу.

Вывод содержимого каталога с запятыми в качестве разделителей

```
ls -m
```

Еще один вариант рассматриваемой нами команды предназначен для тех, кто по каким-то причинам не хочет, чтобы выходные данные были оформлены в виде одного или нескольких столбцов. Отказаться от такого форматирования позволяет опция `-m` (или `--format=commas`).

```
$ ls -m ~/
alias, bin, Desktop, documents, iso, music, pictures,
↳podcasts, program_files, src, todo, videos
```

Для того чтобы лучше запомнить данную опцию, надо принять во внимание, что буква `m` дважды встречается в слове *comma* (запятая). Эта опция пригодится в тех случаях, когда вы пишете сценарий и вам необходимо иметь имена файлов, разделенных запятыми.

Отображение скрытых файлов и каталогов

```
ls -a
```

До сих пор мы получали информацию о видимых файлах и каталогах, но необходимо помнить, что помимо них могут существовать и скрытые файлы. Подобных файлов достаточно много и в вашем рабочем каталоге. Невидимым файл становится в том случае, если его имя начинается с точки. Если вы хотите, чтобы информация о невидимых файлах также отображалась на экране, вам следует использовать опцию `-a` (или `--all`).

```
$ ls -a ~/
.              .gimp-2.2        .openoffice.org1.9.95
..             .gksu.lock       .openoffice.org1.9
.3ddesktop     .glade2          .openoffice.org2
```

```
.abbrev_defs .gnome           .opera
.acrorc      .gnome2          .padminrc
.adobe       .gnome2_private  pictures
alias        .gnome_private   podcasts
[Листинг сокращен для экономии места]
```

Просматривая результаты, полученные с помощью команды `ls -a`, легко заметить следующее. Во-первых, эта команда отображает как скрытые, так и обычные файлы; например, в данном примере вы видите имя `.gnome` и `pictures`. Во-вторых, в списке есть элементы `.` и `..`; одна точка соответствует текущему каталогу, а две точки — каталогу, расположенному выше по иерархии, т.е. родительскому по отношению к текущему. Эти два скрытых каталога присутствуют в любом каталоге вашей системы, и удалить их невозможно. Отобразить их позволяет опция `-a`. В некоторых каталогах число скрытых файлов очень велико, а о наличии некоторых из них вы даже не догадываетесь.

ПОДСКАЗКА

Я только что написал, что команда `ls -a` демонстрирует все файлы с точкой, включая ссылки на файлы `.` и `..`; однако, если вы не хотите видеть эти две ссылки, используйте команду `ls -A`.

Отображение информации о типах файлов

`ls -F`

Обычная команда `ls` не сообщает о файлах, находящихся в каталоге, ничего, кроме их имен. Отображаемая с ее помощью информация не дает даже возможности отличить обычный файл от каталога. Для того чтобы получить дополнительные сведения, надо использовать опцию `-F` (или `--classify`).

```
$ ls -F ~/bin
adblock_filters.txt fixm3u*       pix2tn.pl*
addext*             flash.xml*    pop_login*
address_book.csv    getip*        procmail/
address_book.sxc    homesize*     programs_usual*
address_book.xls    html2text.py* quickrename*
backup_to_chaucer*  list-urls.py*
```

Объем этих сведений невелик. Звездочка после имени файла говорит о том, что файл является исполняемым, а косая черта — это признак каталога. Если после имени файла не указан никакой символ, значит, это обычный файл. Символы, которые могут присутствовать в списке после имен файлов, перечислены в табл. 2.1.

Таблица 2.1. Символы, обозначающие типы файлов

Символ	Тип файла
*	Исполняемый файл
/	Каталог
@	Символьная ссылка
|	FIFO (синоним — *именованный канал*)
=	Сокет

Отображение информации в цвете

```
ls --color
```

Опция `-F`, рассмотренная ранее, позволяла получить информацию о файле с помощью предопределенных символов. Кроме того, вы можете указать оболочке выводить информацию в цвете. Это дает дополнительные возможности для классификации содержимого каталога. В некоторых версиях Linux вывод в цвете предусмотрен по умолчанию, но, если такая установка в вашей системе отсутствует, воспользуйтесь опцией `--color` (я знаю, что текст черно-белый, так что просто представьте это себе).

```
$ ls --color
adblock_filters.txt   fixm3u       pix2tn.pl
addext_flash.xml      pop_login
address_book.csv      getip        procmail
address_book.sxc      homesize     programs_kill
address_book.xls      html2text.py programs_usual
backup_ssh_to_chaucer list-urls.py quickrename
```

В нашем случае исполняемые файлы отображаются зеленым цветом, каталоги — синим, а обычные файлы — черным цветом (который в большинстве оболочек установлен по умолчанию). В табл. 2.2 приведены соглашения по использованию цвета для отображения типов файлов.

Таблица 2.2. Соответствие цвета типу файла

Цвет	Тип файла
Цвет, используемый оболочкой по умолчанию	Обычный файл
Green	Исполняемый файл
Blue	Каталог
Magenta	Символьная ссылка
Yellow	FIFO
Magenta	Сокет
Red	Архив (`.tar`, `.zip`, `.deb`, `.rpm`)
Magenta	Изображение (`.jpg`, `.gif`, `.png`, `.tiff`)
Magenta	Аудиофайл (`.mp3`, `.ogg`, `.wav`)

ПОДСКАЗКА

Если вы хотите выяснить, какому типу файла соответствует тот или иной цвет, то введите команду `dircolors --print-database` и ознакомьтесь с результатами. Команду `dircolors` можно также использовать для того, чтобы изменить соответствие цветов типу файлов.

Используя опции `--color` и `-F`, вы можете быстро составить представление о том, какие типы файлов находятся в каталоге.

```
$ ls -F --color
adblock_filters.txt    fixm3u*       pix2tn.pl*
addext* flash.xml*     pop_login*
address_book.csv       getip*        procmail/
address_book.sxc       homesize*     programs_kill*
address_book.xls       html2text.py* programs_usual*
backup_ssh_to_chaucer* list-urls.py* quickrename*
```

Информация о правах доступа и владельцах файлов

`ls -l`

Вы уже знаете, как получить дополнительные сведения о содержимом каталога, но в некоторых случаях необходима более подробная информация. Сейчас мы рассмотрим, как определять размер, владельца и права доступа к файлу для различных категорий пользователей. Получить эти сведения позволяет опция `-l` (или `--format=long`).

```
$ ls -l ~/bin
total 2951
-rw-r--r-- 1 scott scott 15058 2015-10-03 18:49
↳adblock_filters.txt
-rwxr-xr-- 1 scott root     33 2015-04-19 09:45
↳addext
-rw-r--r-- 1 scott scott 84480 2015-04-19 09:45
↳addressbook.xls
-rwxr--r-- 1 scott scott    55 2015-04-19 09:45
↳batchprint_home
drwxr-xr-x 9 scott scott 1080  2015-09-22 14:42
↳bin_on_bacon
-rwxr-xr-- 1 scott scott  173  2015-04-19 09:45
↳changeext
-rwxr-xr-- 1 scott root   190  2015-04-19 09:45
```

```
↳convertsize
drwxr-xr-x 2 scott scott   48  2015-04-19 09:45
↳credentials
[Листинг сокращен и отредактирован для экономии места]
```

Опция `-l` означает "long". Как видите, она задает отображение подробных сведений о файлах, содержащихся в каталоге. Рассмотрим типичную строку, начиная с самого правого элемента.

Как нетрудно догадаться, заканчивается строка именем файла. Тип этого файла можно отобразить, задавая опцию `-F`, например `ls -lF`. Также можно указать и отображение в цвете (`ls -lF --color`).

Перемещаясь по строке влево, вы встретите информацию о дате и времени. Они определяют момент последней модификации файла (дата отображается в формате год-месяц-день, а время отсчитывается посредством 24-часового цикла).

Левее даты располагается число, которое обозначает размер файла в байтах. Размер приводится и для каталогов, что может вызвать определенные затруднения у начинающих пользователей. В приведенном примере размер каталога `bin_on_bacon` составляет 1080 байтов, т.е. немногим более одного килобайта, но содержится в нем 887 Кбайт данных. С другой стороны, согласно данным, полученным с помощью команды `ls -l`, размер каталога `credentials` составляет 48 байтов, но этот каталог пуст. В чем же дело?

В главе 1 было сказано, что каталоги — это специальный тип файлов, в которых находится информация о содержимом. В данном случае содержимое каталога `credentials` исчерпывается родительским каталогом с именем `..`, поэтому он составляет 48 байтов, а каталог `bin_on_bacon` содержит более 30 файлов, поэтому его размер равен 1080 байтам.

Двигаясь влево по строке, мы встретим столбцы, в которых отображается информация о владельце файла и группе. Как видно из предыдущего примера, почти все файлы принадлежат пользователю `scott` и группе `scott`. Исключение составляют файлы `addext` и `convertsize`, для которых указаны пользователь `scott` и группа `root`.

ЗАМЕЧАНИЕ

Принадлежность файла пользователю и группе можно изменить. Как это сделать, вы узнаете в главе 8 (для этой цели используются команды `chown` и `chgrp`).

Во втором слева столбце содержится число. Оно сообщает, сколько существует "жестких" ссылок на этот файл. Для каталога данное число обозначает количество файлов, содержащихся в нем (подробности — в главе 3), включая два скрытых указателя — `.` (текущий каталог) и `..` (родительский каталог). Это означает, что даже если подкаталогов не существует, то команда все равно вернет число 2.

Итак, мы достигли самого левого столбца. В нем отображаются права доступа к каждому файлу и каталогу. Непосвященному данная последовательность символов не скажет ничего, но она чрезвычайно информативна для тех, кто обладает необходимым минимумом знаний. В данном столбце содержится десять элементов, которые можно условно разделить на четыре группы. Первой группе принадлежит первый символ; знаки в позициях 2–4 составляют вторую группу; символы, занимающие позиции 5–7, принадлежат третьей группе; и четвертой группе принадлежат символы в позициях 8–10. Так, например, последовательность символов для каталога `credentials` можно разделить следующим образом: `d|rwx|r-x|r-x`.

Первая группа сообщает тип файла. Как вы уже знаете, опции `-F` и `--color` отображают типы файлов различными способами. Еще один способ реализует опция `-l`. Символ `d` в первой позиции означает, что в данной строке представлен каталог. Если же в этой позиции находится символ `-`, значит, речь идет об обычном файле. (Даже если файл является исполняемым, команда `ls -l` отобразит в первой позиции символ `-`; в этом случае опции `-F` и `--color` предоставят более подробную информацию.) В табл. 2.3 перечислены символы, которые могут присутствовать в первой позиции строки, отображаемой по команде `ls -l`.

Таблица 2.3. Символы, представляющие права доступа, и типы файлов

Символ	Тип файла
-	Обычный файл
-	Исполняемый файл
d	Каталог
l	Символьная ссылка
s	Сокет
b	Блочное устройство
c	Символьное устройство
p	Именованный канал (FIFO)

ПОДСКАЗКА

Для того чтобы увидеть все или почти все буквы, приведенные в табл. 2.3, выполните команду `ls -l /dev`.

Остальные девять символов, составляющих вторую, третью и четвертую группы, задают права для владельца файла, группы и все остальных пользователей системы. Для файла `addext` из рассмотренного выше примера права заданы так: `rwxr-xr--`. Это означает, что владелец `scott` имеет права `rwx`, группа (в данном случае она тоже называется `scott`) — права `r-x`, а все остальные пользователи имеют права `r--`. Как же расшифровать эту запись?

В каждом случае буква `r` обозначает "чтение разрешено"; буква `w` — "запись разрешена"; буква `x` — "выполнение разрешено". Символ `-` в соответствующей позиции означает "данное действие запрещено". Если дефис указан вместо буквы `r`, это значит "чтение запрещено". Также запрещается запись или выполнение, если символ `-` находится на месте `w` или `x`.

Вернемся еще раз к файлу `addext` и правам доступа `rwxr-xr--`. Теперь нам понятно, что владелец (`scott`) может читать, записывать или выполнять файл, члены группы (`root`) могут читать файл и запускать его на выполнение, но не записывать информацию в него. Любой другой пользователь может читать файл, а запись и запуск на выполнение ему запрещены.

Теперь, когда вы знаете, как расшифровывается информация о правах, обратите внимание на то, что некоторые сочетания символов встречаются очень часто. Например, для многих файлов установлены права `rw-r--r--`; это означает, что владелец может выполнять операции чтения и записи, а остальные пользователи, в том числе члены группы, могут только читать файл. Для программ вы часто встретите права `rwxr-xr-x`, т.е. каждый пользователь может читать и выполнять программу, но вносить изменения в файл может только его владелец.

С каталогами дело обстоит несколько по-другому. Права `r`, `w` и `x`, относящиеся к файлу, интуитивно понятны: они соответствуют чтению, записи и выполнению. Но как выполнить каталог?

Начнем с самого простого — права `r`. В случае каталога `r` означает, что пользователь может просматривать содержимое каталога с помощью команды `ls`. Символ `w` говорит о том, что пользователь может включать в каталог новые файлы, а также переименовывать и удалять существующие. Символ `x` означает доступ к каталогу, т.е. право выполнять команды, выполняющие определенные действия с файлами из этого каталога, или обращаться к подкаталогам данного каталога.

Как видите, опция `-l` предоставляет обширные возможности, но она оказывается еще более полезной в сочетании с другими опциями. Вы уже знаете опцию `-a`, которая отображает все файлы, содержащиеся в каталоге, поэтому для вас очевиден будет результат выполнения команды `ls -la` (вместо `-la` можно также задать `--format=long --all`).

```
$ la -la ~/
drwxr-xr-x   2 scott scott  200 2015-07-28 01:31
↳.alias
drwx------   2 root root     72 2015-09-16 19:14
↳.aptitude
-rw-------   1 scott scott 8800 2015-10-18 19:55
↳.bash_history
-rw-r--r--   1 scott scott   69 2015-04-20 11:00
↳.bash_logout
-rw-r--r--   1 scott scott  428 2015-04-20 11:00
```

```
↳.bash_profile
-rw-r--r--   1 scott scott 4954 2015-09-13 19:46
↳.bashrc
[Листинг сокращен для экономии места]
```

ЗАМЕЧАНИЕ

Если владельцем файла является `scott` или если так называется группа, которой принадлежит файл, то эти данные для экономии места будут удаляться из листингов, приведенных ниже.

Вывод информации в обратном порядке

`ls -r`

Если вам не подходит информация о файлах, выводимая в алфавитном порядке, вы можете изменить порядок на обратный с помощью опции `-r` (или `--reverse`).

```
$ ls -lar ~/
-rw-r--r--    1 scott scott  4954 2015-09-13 19:46
↳.bashrc
-rw-r--r--    1 scott scott   428 2015-04-20 11:00
↳.bash_profile
-rw-r--r--    1 scott scott    69 2015-04-20 11:00
↳.bash_logout
-rw-------    1 scott scott  8800 2015-10-18 19:55
↳.bash_history
drwx------    2 root root      72 2015-09-16 19:14
↳.aptitude
drwxr-xr-x    2 scott scott   200 2015-07-28 01:31
↳alias
[Листинг сокращен и отредактирован для экономии места]
```

ЗАМЕЧАНИЕ

Обратите внимание: данная опция задается буквой нижнего регистра (`-r`, а не `-R`). Опция `-r` задает обратный порядок вывода, а `-R` означает рекурсивное выполнение.

Когда вы используете опцию `-l`, имена файлов и каталогов располагаются по алфавиту. Опция `-r` изменяет этот порядок на обратный, но сортировка по-прежнему производится по имени файла.

Сортировка по дате и времени

```
ls -t
```

Возможность сортировки по алфавиту удобна, но в некоторых случаях желательно расположить содержимое каталога по возрастанию или по убыванию даты и времени. Для того чтобы сделать это, надо наряду с опцией `-l` указать опцию `-t` (или `--sort=time`). Для того чтобы расположить отсортированные данные в обратном порядке, надо кроме `-l` задать `-tr` (или `--sort=time --reverse`). Использовать обратную сортировку очень удобно: более новые данные появляются внизу, и их легче обнаружить, потому что именно в этом месте находится знак приглашения после завершения выполнения команды!

```
$ ls -latr ~/
-rw------- 1 scott scott 8800 2015-10-18 19:55
↳.bash_history
drwx------ 15 scott scott 1280 2015-10-18 20:07
↳.opera
drwx------ 2 scott scott 80 2015-10-18 20:07
↳.gconfd
drwxr-xr-x 2 scott scott 432 2015-10-18 23:11
↳.qt
drwxr-xr-x 116 scott scott 5680 2015-10-18 23:11
```

```
drwx------ 3 scott scott 368 2015-10-18 23:12
↳.gnupg
drwxr-xr-x 12 scott scott 2760 2015-10-18 23:14
↳bin
drwx------ 4 scott scott 168 2015-10-19 00:13
↳.Skype
[Листинг сокращен и отредактирован для экономии места]
```

Все файлы, за исключением последнего, были модифицированы в один и тот же день. Если бы опция `-r` не была указана, последний файл отобразился бы первым.

ЗАМЕЧАНИЕ

Обратите внимание на то, что в последнем примере было указано четыре опции `-latr`. Их можно задать и по отдельности, `-l -a -t -r`, но при этом пришлось бы вводить лишние дефисы и пробелы. Гораздо быстрее и проще объединить их в одну последовательность символов. Существуют также "длинные" варианты опций. Такая опция начинается с двух дефисов и состоит из одного или двух слов. Их нельзя объединять и надо задавать раздельно. Другими словами, команду, использованную в предыдущем примере, можно задать и так: `ls -la --sort=time --reverse`. Имейте в виду, что этим свойством обладают многие команды и программы Linux, но не все.

Сортировка содержимого каталога по размеру файлов

```
ls -S
```

Содержимое каталога можно отсортировать также и по размеру файлов. Сделать это позволяет опция `-S` (или `--sort=size`).

```
$ ls -laS ~/
-rw-r--r--   1 scott scott 109587 2015-10-19 11:53
↳.xsession-errors
-rw-------   1 scott scott  40122 2015-04-20 11:00
↳.nessusrc
-rw-r--r--   1 scott scott  24988 2015-04-20 11:00
↳.abbrev_defs
-rwxr--r--   1 scott scott  15465 2015-10-12 15:45
↳.vimrc
-rw-------   1 scott scott  11794 2015-10-19 10:59
↳.viminfo
-rw-------   1 scott scott   8757 2015-10-19 08:43
↳.bash_history
[Листинг сокращен и отредактирован для экономии места]
```

При сортировке по размеру самый большой файл указывается первым. Если задан обратный порядок следования, т.е. если указана опция `-r`, вверху списка окажется файл самого меньшего размера.

Представление размеров файлов в кило-, мега- и гигабайтах

```
ls -h
```

В списке из предыдущего раздела содержался файл `.vimrc` размером 15465 байтов, т.е. порядка 15 Кбайт. Преобразовывать в уме байты в килобайты, мегабайты и гигабайты не всегда удобно. Часто бывает предпочтительнее использовать опцию `-h` (или `--human-readable`), которая представляет информацию в виде, удобном для восприятия пользователем.

```
$ ls -laSh ~/
-rw-r--r--    1 scott scott 100K 2015-10-19 11:44
..xsession-errors
-rw-------    1 scott scott 40K 2015-04-20 11:00
..nessusrc
-rw-r--r--    1 scott scott 25K 2015-04-20 11:00
..abbrev_defs
```

```
-rwxr--r--    1 scott scott 16K 2015-10-12 15:45
..vimrc
-rw-------    1 scott scott 12K 2015-10-19 10:59
..viminfo
-rw-------    1 scott scott 8.6K 2015-10-19 08:43
..bash_history
[Листинг сокращен и отредактирован для экономии места]
```

В данном примере размер файлов представлен в килобайтах, на что указывает буква K после соответствующего числа. Если бы файлы были достаточно большими, вы бы увидели букву M (мегабайты) и даже G (гигабайты). Возможно, вы удивитесь, как 40122 байта (размер файла `.nessusrc`) превратились в 40 Кбайт; ведь в килобайте 1024 байта, и, разделив 40122 на 1024, мы получим 39,1816406 байта. Однако команда `ls -h` округляет данное значение до 40 Кбайт. Заметьте, что мегабайт содержит 1048576 байтов, а гигабайт — 1073741824 байта, и при использовании этих единиц измерения также возможны подобные округления. Если необходимо узнать точное значение с точностью до десятичных знаков (кибибайты, мебибайты и т.д.), используйте опцию `--si` (подробнее об этом — в главе 6).

ЗАМЕЧАНИЕ

В своем файле `~/.bashrc` я задал несколько псевдонимов, которые уже много лет упрощают мою работу в системе. Используйте материал из этой главы для экспериментов с приведенными примерами и создавайте псевдонимы, которые точно соответствуют вашим потребностям (более подробно псевдонимы будут описаны в главе 12).

```
alias l='ls -F'
alias l1='ls -1F'
alias la='ls -aF'
alias ll='ls -laFh'
alias ls='ls -F'
```

Определение пути к текущему каталогу

pwd

Может случиться так, что, перемещаясь по каталогам и выводя на экран их содержимое, вы забудете, в каком месте файловой системы находитесь в данный момент. Как узнать, какой из каталогов является текущим? Сделать это позволяет команда `pwd`, имя которой расшифровывается как *print working directory* (вывести текущий каталог).

ЗАМЕЧАНИЕ

Слово "print" в выражении *print working directory* означает вывод на экран, а не на принтер.

Команда `pwd` отображает полный путь к текущему каталогу. Вряд ли вы будете использовать эту команду очень часто, но время от времени она оказывается довольно полезной.

```
$ pwd
/home/scott/music/new
```

Однако вы должны знать один нюанс, который часто вводит людей в заблуждение. В главе 3 я опишу команду `ln`, поэтому вы можете пропустить несколько страниц и прочитать описание команды `ln`, чтобы полностью понять то, что я собираюсь рассказать. Будем считать, что вы это сделали, и приступим к проверке следующих команд:

```
# ls -l
lrwxrwxrwx scott scott websites -> /var/www/
$ cd websites
$ pwd
/websites
$ pwd -P
/var/www
```

Итак, мы видим мягкую ссылку на источник `websites`, которая указывает на каталог `/var/www`. Если выполнить команду `cd` с мягкой ссылкой и ввести команду `pwd`, то вы получите *логический* каталог `/websites`, т.е. *источник* мягкой ссылки, который на самом деле не является целевым каталогом `/var/www`. Именно так работает команда `pwd` по умолчанию. Это эквивалентно вводу команды `pwd -L` (или `--logical`).

С другой стороны, если ввести команду `pwd -P` (или `--physical`), то вы получите цель мягкой ссылки, т.е. каталог `/var/www`.

Не знаю, как вы, но когда я использую команду `pwd`, то обычно хочу узнать реальное физическое (цель), а не логическое местоположение (источник). Поскольку я предпочитаю использовать по умолчанию опцию `-P`, то применяю псевдоним из своего файла `.bash_aliases` (см. главу 12), который выглядит так:

```
alias pwd="pwd -P"
```

Теперь вы знаете не только, как надо использовать команду `pwd`, но и о побочных эффектах, которые с ней связаны.

Переход в другой каталог

cd

Отобразить содержимое любого каталога можно, указав путь к нему в качестве параметра команды `ls`. Однако часто возникает необходимость перейти в другой каталог, т.е. сделать его текущим. В этих случаях надо применять команду `cd` — одну из самых популярных команд любой оболочки.

Команда `cd` очень проста в использовании, в качестве параметра ей передается каталог, который необходимо сделать текущим. При этом можно использовать как относительный (например, `cd src` или `cd ../../`), так и абсолютный путь (например, `cd /tmp` или `cd /home/scott/bin`).

Переход в рабочий каталог

```
cd ~
```

Некоторые полезные возможности команды cd могут несколько упростить работу в системе. Независимо от того, какой каталог является текущим, выполнив команду cd, вы немедленно перейдете в свой рабочий каталог. Эта возможность позволяет сэкономить время. Можно также использовать команду cd ~, так как ~ является сокращением, обозначающим "мой рабочий каталог".

```
$ pwd
/home/scott/music
$ cd ~
$ pwd
/home/scott
```

Переход в предыдущий каталог

```
cd ~
```

Еще один полезный вариант рассматриваемой здесь команды — это cd -. Она осуществляет переход к предыдущему каталогу, после чего автоматически вызывает команду pwd, которая выводит информацию о новом каталоге (правда, в данном случае его уместнее назвать не новым, а старым). Это эквивалентно щелчку на кнопке Back в окне графического менеджера файлов. Действие команды cd - демонстрирует следующий пример:

```
$ pwd
/home/scott
$ cd music/new
$ pwd
/home/scott/music/new
$ cd -
/home/scott
```

Команда `cd` – удобна тогда, когда надо перейти в другой каталог, выполнить в нем некоторые действия, а затем вернуться в тот каталог, с которым вы работали ранее. Информация о каталоге, выводимая на экран, позволяет вам удостовериться в правильности выполненных действий.

Выводы

Если бы вы учились на юридическом факультете, то изучали бы, что такое правонарушение, противоправное деяние и преступление. Если бы вы изучали основы ремонта оборудования, то интересовались бы запоминающими устройствами, жесткими дисками и материнскими платами. Однако, поскольку наша книга посвящена оболочке системы Linux, мы рассмотрели основные команды, которые должен знать пользователь Linux, чтобы эффективно использовать командную строку: `ls`, `pwd` и `cd`. Теперь, зная основы навигации по файловой системе, можно переходить к основам мироздания: созданию и разрушению. С этого момента вы сможете не только наблюдать за своим окружением, но и модифицировать его по своему желанию. Вперед!

3

Создание и уничтожение

У каждого народа есть легенды, в которых рассказывается, как нечто возникло (*космогония*) и в конце концов было уничтожено (*эсхатология*). Например, у викингов есть мифы о странах *Муспельхейм* (*Muspellsheimr*) и *Нифльхейм* (*Niflheimr*), существовавших в начале времен, и конце света — *Рагнарок* (*Ragnarok*). Зороастрийцы сложили миф *Бундахишн* (Bundahishn) о сотворении мира и *Фрашо-керети* (Frashokereti) — о воскресении из мертвых. Библия также начинается с *Книги Бытия*, а заканчивается *Апокалипсисом*. Linux — это не религия, но тоже позволяет создавать и уничтожать сущности. Продолжайте же читать и мудро пользуйтесь обретенной силой!

ЗАМЕЧАНИЕ

Пересматривая книгу для второго издания, я исключил из нее раздел о командах `mkdir -v` и `rm -v` (в котором объяснялось, что такое команды `mkdir` и `rm` и как они работают). Исходные главы можно найти на моем веб-сайте `www.granneman.com/linux-redactions`.

Изменение сведений о времени

`touch`

Команда `touch` не относится к тем, которые используются ежедневно, но мы будем применять ее в этой книге, поэтому удобнее всего ознакомиться с ней именно сейчас. Основное назначение данной команды — установка времени доступа и модификации файла, однако мы будем применять ее с другой

целью. Как ни странно, вспомогательная функция команды важнее для нас, чем основная.

ЗАМЕЧАНИЕ

Применять команду `touch` к файлу можно только в том случае, если вы имеете право записывать информацию в этот файл. В противном случае при попытке выполнить команду `touch` возникнет ошибка.

Для того чтобы одновременно изменить время доступа и время модификации файла (или каталога), надо выполнить команду `touch`, не задавая никаких опций.

```
$ ls -l ~/
drwxr-xr-x  scott scott 2015-10-18 12:07 todo
drwxr-xr-x  scott scott 2015-10-18 12:25 videos
-rw-r--r--  scott scott 2015-09-10 23:12
↳wireless.log
$ touch wireless.log
$ ls -l ~/
drwxr-xr-x  scott scott 2015-10-18 12:07 todo
drwxr-xr-x  scott scott 2015-10-18 12:25 videos
-rw-r--r--  scott scott 2015-10-19 14:00
↳wireless.log
```

В результате выполнения приведенной выше команды изменилось время доступа и время модификации файла `wireless.log`. Это не очевидно, так как по команде `ls -l` отображается только время модификации файла. Файл не использовался около месяца, но команда `touch` изменила информацию о дате и времени.

При необходимости можно изменить время доступа и время модификации по отдельности. Если надо изменить только время доступа, следует указать опцию `-a` (или `--time=access`), а чтобы установить только время модификации, надо использовать опцию `-m` (или `--time=modify`).

Установка произвольного времени для файла

```
touch -t
```

Заметьте, что команда touch не ограничивает вас текущими датой и временем. Вы можете выбрать любую дату, используя опцию -t и задавая значение в следующем формате: [[CC]YY]MMDDhhmm[.ss]. Назначение основных его элементов приведено в табл. 3.1.

Таблица 3.1. Элементы шаблона, используемые для установки времени доступа и времени модификации файла

Символы	Значение
CC	Первые две цифры года, задаваемого четырьмя цифрами
YY	Год, задаваемый двумя цифрами: • значение 00–68 предполагает первые две цифры — 20; • значение 69–99 предполагает первые две цифры — 19; • отсутствующее значение предполагает текущий год
MM	Месяц (01–12)
DD	День (01–31)
hh	Часы (01–23)
mm	Минуты (00–59)
ss	Секунды (00–59)

Важно помнить, что в тех случаях, когда в шаблоне предусмотрены две цифры, а значение представляется лишь одной цифрой, надо указывать ведущий нуль. В противном случае опция будет интерпретирована неправильно. Ниже представлено несколько примеров использования опции -t.

```
$ ls -l
-rw-r--r-- scott scott 2015-10-19 14:00 wireless.log
$ touch -t 197002160701 wireless.log
$ ls -l
-rw-r--r-- scott scott 1970-02-16 07:01 wireless.log
$ touch -t 9212310000 wireless.log
```

```
$ ls -l
-rw-r--r-- scott scott 1992-12-31 00:00 wireless.log
$ touch -t 3405170234 wireless.log
$ ls -l
-rw-r--r-- 1 scott scott 2034-05-17 02:34
↳wireless.log
$ touch -t 10191703 wireless.log
$ ls -l
-rw-r--r-- scott scott 2015-10-19 17:03 wireless.log
```

Сначала мы выясняем текущую дату и время для файла `wireless.log` (`2015-10-19 14:00`). Затем возвращаемся во времени на 35 лет (`1970-02-16 07:01`), а после этого переходим более чем на двадцать лет вперед (`1992-12-31 00:00`). Как видите, мы можем совершить даже скачок в будущее (`2034-05-17 02:34`). Это будет время повсеместного использования компьютеров под управлением Linux, время всеобщего мира и открытого кода. В конце концов, мы возвратимся в настоящее время.

Приведенный пример демонстрирует некоторые особенности работы с командой `touch`. Когда мы возвращались назад более чем на три десятилетия, мы задавали четырьмя цифрами год (1970), кроме того, указали месяц (02), число (16), количество часов (07) и минут (01). Секунды можно не указывать, и мы не делали этого. Данная команда — единственная, в которой год определялся с помощью четырех цифр. Значение 92 в 9212310000 попадает в диапазон 69–99, поэтому в качестве базового выбирается 1900 год. Число 34 в составе значения 3405120234 принадлежит диапазону 0–68, поэтому базовым считается 2000 год. В последней команде год не задается вовсе, указывается только месяц (10), число (19), количество часов (17) и минут (03). В этом случае команда `touch` принимает текущий год. Зная, как пользоваться командой `touch`, мы можем изменять время доступа или модификации файла.

Создание нового пустого файла

touch

Необходимость изменять дату возникает достаточно редко. Однако команда touch имеет еще одно, гораздо более интересное применение. С ее помощью можно воздействовать на файл, который еще не создан. В результате команда создаст новый файл с указанным именем.

```
$ ls -l ~/
drwxr-xr-x scott scott 2015-10-19 11:36 src
drwxr-xr-x scott scott 2015-10-18 12:25 videos
$ touch test.txt
$ ls -l ~/
drwxr-xr-x scott scott 2015-10-19 11:36 src
-rw-r--r-- scott scott 2015-10-19 23:41 test.txt
drwxr-xr-x scott scott 2015-10-18 12:25 videos
```

Зачем же нужно использовать команду touch подобным способом? Причины могут быть разными. Возможно, вы захотите сначала создать файл, а потом записать в него данные. Не исключено, что вам понадобится несколько файлов для того, чтобы проверить, как работает незнакомая вам команда. По мере того как вы будете подробнее изучать оболочку, вы придумаете новые причины для создания пустого файла.

Создание нового каталога

mkdir

Команда touch может создать пустой файл, но как насчет нового каталога? В этом вам поможет команда mkdir.

```
$ ls -l
drwxr-xr-x scott scott 2015-10-19 11:36 src
drwxr-xr-x scott scott 2015-10-18 12:25
↳videos
```

```
$ mkdir test
$ ls -l
drwxr-xr-x scott scott 2015-10-19 11:36 src
drwxr-xr-x scott scott 2015-10-19 23:50
↳test
drwxr-xr-x scott scott 2015-10-18 12:25
↳videos
```

ЗАМЕЧАНИЕ

В большинстве систем новые каталоги, созданные посредством команды `mkdir`, предоставляют владельцу права чтения, записи и выполнения, а членам группы — только права чтения и выполнения. Если вы хотите назначить права по-другому, вам следует воспользоваться командой `chmod`, которая будет рассмотрена в главе 7.

К счастью, в данном случае оболочка следит за правильностью ваших действий: при попытке создать уже существующий каталог, команда завершится с ошибкой и выведет предупреждающее сообщение.

```
$ mkdir test
mkdir: cannot create directory 'test': File exists
```

Создание нового каталога и необходимых подкаталогов

`mkdir -p`

Если вы хотите создать каталог, в нем новый подкаталог, а в нем еще один подкаталог, может показаться, что эту рутинную задачу следует решать так: создать первый каталог с помощью команды `mkdir`, перейти в него, вызвав команду `cd`, создать подкаталог, опять же использовав для этого команду `mkdir`, сделать его текущим и вызвать команду `mkdir` для создания

очередного подкаталога. К счастью, в команде `mkdir` предусмотрена удобная опция `-p` (или `--parents`), существенно упрощающая весь процесс.

```
$ ls -l
drwxr-xr-x scott scott 2015-10-19 11:36 src
drwxr-xr-x scott scott 2015-10-18 12:25
↳videos
$ mkdir -p pictures/personal/family
$ ls -l
drwxr-xr-x scott scott 2015-10-20 00:12
↳pictures
drwxr-xr-x scott scott 2015-10-19 11:36 src
drwxr-xr-x   6 scott scott  632  2015-10-18 12:25
↳videos
$ cd pictures
$ ls -l
drwxr-xr-x scott scott 2015-10-20 00:12
↳personal
$ cd personal
$ ls -l
drwxr-xr-x scott scott 2015-10-20 00:12
↳family
```

Копирование файлов

cp

Каждому пользователю, независимо от того, с какой операционной системой он работает, то и дело приходится копировать файлы. Команда `cp` — одна из главных в оболочке Linux. Она предназначена для копирования файлов и каталогов. Самый простой способ скопировать файл — ввести команду `cp`, указав после нее имя копируемого файла, а затем имя нового файла, который должен получиться в результате копирования. Другими словами, данная команда задается следующим образом: `cp` *файл_для_копирования новый_файл*. Ее также можно описать как `cp` *файл_источник целевой_файл*.

```
$ pwd
/home/scott/libby
$ ls
libby.jpg
$ cp libby.jpg libby_bak.jpg
$ ls
libby_bak.jpg   libby.jpg
```

Данный пример чрезвычайно прост. Изображение копируется в тот же каталог, который содержит исходный файл. Можно также копировать файлы в другой каталог либо копировать их из каталога, который не является текущим.

```
$ ls ~/libby
libby_bak.jpg libby.jpg
$ cp pictures/dogs/libby_arrowrock.jpg
↳~/libby/libby_arrowrock.jpg
$ ls ~/libby
libby_arrowrock.jpg libby_bak.jpg libby.jpg
```

Поскольку в данном случае файл копируется в другой каталог, его имя может оставаться неизменным. В предыдущем примере нам пришлось задать новое имя — `libby_bak.jpg`, так как исходный файл, `libby.jpg`, копировался в тот же каталог.

Если вам надо скопировать файл в текущий каталог из какой-то другой точки файловой системы, используйте в качестве имени целевого каталога точку. (Вы, конечно же, помните, что точка означает текущий каталог? Теперь эта информация пригодилась.) Очевидно, что если вы укажете в качестве второго параметра команды `cp` точку, имя целевого файла будет совпадать с именем исходного.

```
$ ls
libby_bak.jpg libby.jpg
$ cp pictures/dogs/libby_arrowrock.jpg .
$ ls
libby_arrowrock.jpg libby_bak.jpg libby.jpg
```

Если файл, полученный в результате копирования, должен быть помещен в конкретный каталог, можете не указывать имя целевого файла, достаточно задать имя каталога.

```
$ ls -l
drwxr-xr-x scott scott 2015-10-18 12:35 iso
drwxr-xr-x scott scott 2015-10-20 12:34 libby
drwxr-xr-x scott scott 2015-09-29 23:17 music
drwxr-xr-x scott scott 2015-10-16 12:34 pictures
$ ls libby
libby_arrowrock.jpg libby_bak.jpg libby.jpg
$ cp pictures/dogs/libby_on_couch.jpg libby
$ ls libby
libby_arrowrock.jpg libby_bak.jpg libby.jpg
↳libby_on_couch.jpg
```

Выполняя команду, приведенную в данном примере, надо быть уверенным, что каталог `libby` существует, в противном случае файл `libby_on_couch.jpg` будет скопирован в файл `libby` и помещен в текущий каталог.

Копирование файлов с использованием символов групповых операций

cp *

Как уже говорилось выше, система Linux поощряет ленивых пользователей. В частности, она предоставляет возможность скопировать одновременно несколько файлов, используя символы групповых операций. Если вы грамотно именовали файлы, то сэкономите время при их копировании, так как сможете с помощью одной команды обрабатывать несколько файлов.

```
$ pwd
/home/scott/libby
$ ls ~/pictures/dogs
libby_arrowrock.jpg  libby_by_pool_03.jpg
```

```
libby_on_floor_03.jpg
libby_by_pool_01.jpg libby_on_floor_01.jpg
libby_on_floor_04.jpg
libby_by_pool_02.jpg libby_on_floor_02.jpg
$ ls
libby_on_couch.jpg
$ cp ~/pictures/dogs/libby_by_pool*.jpg .
$ ls
libby_arrowrock.jpg  libby_by_pool_02.jpg
libby_on_couch.jpg
libby_bak.jpg        libby_by_pool_03.jpg
libby_by_pool_01.jpg libby.jpg
```

Символ * — не единственный из тех, которые можно применять для групповых операций. Вы можете задавать имена файлов, используя квадратные скобки. Например, если хотите скопировать изображения `libby_on_floor` с номерами 01, 02, 03, но не 04, сделайте это так, как показано в следующем примере:

```
$ pwd
/home/scott/libby
$ ls ~/pictures/dogs
libby_arrowrock.jpg  libby_by_pool_03.jpg
libby_on_floor_03.jpg
libby_by_pool_01.jpg libby_on_floor_01.jpg
libby_on_floor_04.jpg
libby_by_pool_02.jpg libby_on_floor_02.jpg
$ ls
libby_arrowrock.jpg  libby_bak.jpg libby.jpg
libby_on_couch.jpg
$ cp ~/pictures/dogs/libby_on_floor_0[1-3].jpg .
$ ls
libby_arrowrock.jpg  libby_on_couch.jpg
libby_on_floor_03.jpg
libby_bak.jpg        libby_on_floor_01.jpg
libby.jpg            libby_on_floor_02.jpg
```

Вывод подробной информации о копировании файлов

```
cp -v
```

Опция -v (или --verbose) предоставит вам подробную информацию о действиях, выполняемых командой cp.

```
$ pwd
/home/scott/libby
$ ls ~/pictures/dogs
libby_arrowrock.jpg libby_by_pool_03.jpg
↳libby_on_floor_03.jpg
libby_by_pool_01.jpg libby_on_floor_01.jpg
↳libby_on_floor_04.jpg
libby_by_pool_02.jpg libby_on_floor_02.jpg
$ ls
libby_arrowrock.jpg  libby_bak.jpg libby.jpg
↳libby_on_couch.jpg
$ cp -v ~/pictures/dogs/libby_on_floor_0[1-3].jpg .
'/home/scott/pictures/dogs/libby_on_floor_01.jpg' ->
↳'./libby_on_floor_01.jpg'
'/home/ scott /pictures/dogs/libby_on_floor_02.jpg'
↳-> './libby_on_floor_02.jpg'
'/home/ scott /pictures/dogs/libby_on_floor_03.jpg'
↳-> './libby_on_floor_03.jpg'
$ ls
libby_arrowrock.jpg  libby_on_couch.jpg
↳libby_on_floor_03.jpg
libby_bak.jpg        libby_on_floor_01.jpg
libby.jpg            libby_on_floor_02.jpg
```

Благодаря опции -v вы можете не задавать последнюю команду ls, так как вам предоставляется информация, которая позволяет убедиться в том, что файлы скопированы.

ЗАМЕЧАНИЕ

По мере накопления опыта работы с командной строкой Linux вы увидите, что опция `-v` часто интерпретируется программами как *подробно* (*verbose*). Такое соответствие можно обнаружить со многими опциями. Таким образом, запоминать их совсем не трудно!

Как предотвратить копирование поверх важных файлов

```
cp -i
```

Предыдущий пример продемонстрировал важную особенность команды `cp`, которую необходимо учитывать в процессе работы. В одном из более ранних примеров мы скопировали изображения `libby_on_floor` в каталог `libby`, в последнем примере мы снова выполнили копирование трех изображений `libby_on_floor` в тот же каталог `libby`. Несмотря на то что файлы с такими именами уже существовали в каталоге, мы не получили предупреждающего сообщения. Это общий принцип работы Linux: система предполагает, что пользователь знает, что он делает. При перезаписи существующих файлов система не предупредит вас ...если вы специально не попросите ее об этом. Если вы хотите, чтобы команда `cp` оповещала вас, перед тем, как записать один файл поверх другого, используйте опцию `-i` (или `--interactive`). Если вы снова попытаетесь скопировать те же файлы, но укажете опцию `-i`, результаты будут не такими, как раньше.

```
$ pwd
/home/scott/libby
$ ls ~/pictures/dogs
libby_arrowrock.jpg libby_by_pool_03.jpg
↳libby_on_floor_03.jpg
```

```
libby_by_pool_01.jpg libby_on_floor_01.jpg
↳libby_on_floor_04.jpg
libby_by_pool_02.jpg libby_on_floor_02.jpg
$ ls
libby_arrowrock.jpg libby_bak.jpg libby.jpg
↳libby_on_couch.jpg
$ cp -i ~/pictures/dogs/libby_on_floor_0[1-3].jpg .
cp: overwrite './libby_on_floor_01.jpg'?
```

Как видите, команда `cp` приостановила свое выполнение, чтобы спросить, действительно ли вы хотите записать файл `libby_on_floor_01.jpg` поверх другого с таким же именем. Если вы хотите, чтобы файл был скопирован, введите `y`, в противном случае введите `n`. Если вы введете `n`, команда `cp` не прекратит свое выполнение: вам будет предложено подтвердить копирование очередного файла. Есть способ отменить копирование всех последующих файлов: надо нажать комбинацию клавиш <Ctrl+C>, завершив таким образом выполнение команды. В системе Linux нет способа, позволяющего ответить "да" сразу на все вопросы, поступающие последовательно. Таким образом, если вы захотите записать 1000 файлов поверх 1000 существующих, имеющих те же имена, и укажете опцию `-i`, вам придется запастись временем и терпением, так как система ровно 1000 раз спросит, действительно ли вы хотите заменить существующий файл новым.

ПРЕДУПРЕЖДЕНИЕ

У рядовых пользователей редко возникает необходимость в опции `-i`. Для пользователя `root` она очень важна, поскольку его работа постоянно сопряжена с опасностью случайно заменить важный системный файл. По этой причине пользователю `root` желательно создать в файле `.bashrc` псевдоним, наличие которого приведет к вызову `cp -i` вместо обычной команды `cp`.

```
alias cp='cp -i'
```

Копирование каталогов

cp -R

До сих пор мы рассматривали копирование файлов, но нередко приходится копировать целые каталоги. При этом команда cp *исходный_каталог целевой_каталог* работать не будет.

```
$ pwd
/home/scott
$ cp libby libby_bak
cp: omitting directory 'libby'
```

Для копирования каталогов надо использовать опцию -R (или --recursive), уже знакомую вам по команде ls. Опция -R, указанная при вызове команды cp, приводит к копированию каталога и его содержимого.

```
$ pwd
/home/scott
$ ls -l
drwxr-xr-x scott scott 2015-10-17 14:42
↳documents
drwxr-xr-x scott scott 2015-10-18 12:35 iso
drwxr-xr-x scott scott 2015-10-20 17:16
↳libby
$ ls libby
libby_arrowrock.jpg libby_on_couch.jpg
↳libby_on_floor_03.jpg
libby_bak.jpg       libby_on_floor_01.jpg
libby.jpg           libby_on_floor_02.jpg
$ cp -R libby libby_bak
$ ls -l
drwxr-xr-x scott scott 2015-10-17 14:42
↳documents
drwxr-xr-x scott scott 2015-10-18 12:35 iso
drwxr-xr-x scott scott 2015-10-20 17:16
↳libby
drwxr-xr-x scott scott 2015-10-20 17:17
↳libby_bak
```

Использование команды cp для создания резервных копий

```
Cp -a
```

Когда изучите команду cp, вам, возможно, покажется, что она хорошо подходит для создания резервных копий. Действительно, с ее помощью можно решать эту задачу, однако существуют инструменты, гораздо более пригодные для этого (о них вы узнаете в главе 15). С помощью нескольких строк сценария оболочки bash можно обеспечить копирование различных файлов и каталогов. Чаще всего для этой цели используется опция -a (или --archive), которая эквивалента сочетанию трех опций -dpR (или --no-dereference --preserve --recursive). Действие опции -a можно описать так: cp не обрабатывает символьные ссылки (в противном случае объем копируемой информации мог бы неконтролируемо увеличиться), сохраняет основные атрибуты файла, например, сведения о владельце, дату и время, и рекурсивно обрабатывает подкаталоги.

```
$ pwd
/home/scott
$ ls -l
drwxr-xr-x scott scott 2015-10-21 11:31 libby
drwxr-xr-x scott scott 2015-09-29 23:17 music
$ ls -lR libby
libby:
total 312
-rw-r--r-- scott scott 2015-10-20 12:12
↳libby_arrowrock.jpg
-rw-r--r-- scott scott 2015-04-19 00:57 libby.jpg
drwxr-xr-x scott scott 2015-10-21 11:31 on_floor
libby/on_floor:
total 764
-rw-r--r-- scott scott 2015-10-20 16:11
↳libby_on_floor_01.jpg
-rw-r--r-- scott scott 2015-10-20 16:11
```

```
↳libby_on_floor_02.jpg
$ cp -a libby libby_bak
$ ls -l
drwxr-xr-x scott scott 2015-10-21 11:31 libby/
drwxr-xr-x scott scott 2015-10-21 11:31 libby_bak/
drwxr-xr-x scott scott 2015-09-29 23: 17 music/
$ ls -lR libby_bak
libby_bak/:
total 312
-rw-r--r-- scott scott 2015-10-20 12:12
↳libby_arrowrock.jpg
-rw-r--r-- scott scott 2015-04-19 00:57 libby.jpg
drwxr-xr-x scott scott 2015-10-21 11:31 on_floor

libby_bak/on_floor:
total 764
-rw-r--r-- scott scott 218849 2015-10-20 16:11
↳libby_on_floor_01.jpg
-rw-r--r-- scott scott 200024 2015-10-20 16:11
↳libby_on_floor_02.jpg
```

ЗАМЕЧАНИЕ

Если вы еще не поняли, почему я использую именно такие имена файлов, поясню: Libby — это моя собака породы ши-тцу. Во время написания книги она все время крутилась где-то рядом. К сожалению, она умерла в августе 2015 г.

Перемещение и переименование файлов

mv

Как вам уже известно, команда `cp` копирует файлы. А можно ли не копировать, а лишь переместить файл? Для этой цели используется команда `mv`. Подобно команде `cp`, ее имя получается путем удаления гласных из слова “move”.

Прочитав описание mv, нетрудно заметить, что для нее допустимы многие из опций, известных вам по команде cp. Это неудивительно, ведь mv делает то же, что и cp -a, а после этого, если копирование было произведено успешно, удаляет исходный файл.

Самое простое описание mv звучит так: эта команда перемещает файл из одной позиции файловой системы в другую.

```
$ pwd
/home/scott/libby
$ ls
libby_arrowrock.jpg  libby_bak.jpg  libby.jpg
↳libby_on_couch.jpg   on_floor
$ ls ~/pictures/dogs
libby_on_floor_01.jpg  libby_on_floor_03.jpg
↳libby_on_floor_02.jpg  libby_on_floor_04.jpg
$ mv ~/pictures/dogs/libby_on_floor_04.jpg
  libby_on_floor_04.jpg
$ ls
libby_arrowrock.jpg    libby.jpg
↳libby_on_floor_04.jpg  libby_bak.jpg
libby_on_couch.jpg     on_floor
$ ls ~/pictures/dogs
libby_on_floor_01.jpg   libby_on_floor_02.jpg
↳libby_on_floor_03.jpg
```

Как и при использовании команды cp, можно задавать текущий каталог с помощью точки.

```
$ pwd
/home/scott/libby
$ ls
libby_arrowrock.jpg libby_bak.jpg libby.jpg
↳libby_on_couch.jpg on_floor
$ ls ~/pictures/dogs
libby_on_floor_01.jpg libby_on_floor_03.jpg
libby_on_floor_02.jpg libby_on_floor_04.jpg
$ mv ~/pictures/dogs/libby_on_floor_04.jpg .
$ ls
libby_arrowrock.jpg libby.jpg
↳libby_on_floor_04.jpg
libby_bak.jpg libby_on_couch.jpg on_floor
```

```
$ ls ~/pictures/dogs
libby_on_floor_01.jpg libby_on_floor_02.jpg
↳libby_on_floor_03.jpg
```

Если вы перемещаете файл в другой каталог и хотите, чтобы он сохранил свое имя, вам надо задать только имя каталога. При этом имя файла останется прежним.

```
$ pwd
/home/scott/libby
$ ls
libby_arrowrock.jpg libby.jpg
↳libby_on_floor_04.jpg
libby_bak.jpg libby_on_couch.jpg on_floor
$ ls on_floor
libby_on_floor_01.jpg libby_on_floor_02.jpg
↳libby_on_floor_03.jpg
$ mv libby_on_floor_04.jpg on_floor
$ ls
libby_arrowrock.jpg libby_bak.jpg libby.jpg
↳libby_on_couch.jpg on_floor
$ ls on_floor
libby_on_floor_01.jpg libby_on_floor_03.jpg
libby_on_floor_02.jpg libby_on_floor_04.jpg
```

Для того чтобы гарантировать, что `on_floor` — это каталог, желательно указать после него косую черту. Тогда команда примет следующий вид: `mv libby_on_floor_04.jpg on_floor/`. Если окажется, что `on_floor` — не каталог, команда не переместит файл. Так можно предотвратить запись одного файла поверх другого.

ЗАМЕЧАНИЕ

Многие опции команд `cp` и `mv` совпадают и производят одинаковые действия. Например, опция `-i` запрашивает подтверждение на выполнение операции, а если указана опция `-v`, то при копировании и перемещении выводятся подробные сведения о выполненных действиях.

Переименование файлов и каталогов

mv

Как вы вскоре увидите, команда mv может делать нечто большее, чем обычное перемещение файлов. Она позволяет также переименовывать файлы. При перемещении файла указывается целевое имя, которое не обязательно совпадает с именем исходного файла. Именно это свойство пользователи, работающие с оболочкой, издавна используют для переименования файлов и каталогов.

```
$ pwd
/home/scott/libby/by_pool
$ ls -F
libby_by_pool_02.jpg liebermans/
$ mv liebermans/ lieberman_pool/
$ ls -F
libby_by_pool_02.jpg lieberman_pool/
```

Копируя каталог с помощью команды cp, необходимо указывать опцию -R (или --recursive). С командой mv дело обстоит по-другому. Как можно заметить в предыдущем примере, она, будучи вызвана без дополнительных опций, успешно перемещает или переименовывает каталоги. Это объясняется тем, что команда mv намного проще, чем команда cp; она не создает новых файлов и не перемещает их на новое место (если, конечно, вы не переходите с одной файловой системы на другую); она всего лишь изменяет имя отдельного файла или каталога, который вы указали.

ПРЕДУПРЕЖДЕНИЕ

Команда mv имеет одну важную особенность, которую начинающие пользователи часто упускают из виду. Если вы перемещаете ссылку, указывающую на каталог, вам надо внимательно следить за вводимыми данными. Предположим, что в вашем рабочем каталоге

есть ссылка dogs, которая указывает на каталог /home/scott/pictures/dogs, и вы хотите переместить ее в каталог /home/scott/libby. Приведенная ниже команда перемещает лишь саму ссылку.

```
$ mv dogs ~/libby
```

Следующая же команда перемещает каталог, на который указывает эта ссылка:

```
$ mv dogs/ ~/libby
```

Различие между ними лишь в косой черте, которая вводится после ссылки. Если символ / отсутствует, перемещается лишь сама ссылка. Включив этот символ, вы переместите каталог, а не ссылку. Будьте внимательны!

Как Linux хранит файлы

В следующем разделе описывается команда ln (от слова *link*). Однако, прежде чем рассматривать, как работают ссылки, необходимо кратко рассмотреть, как система Linux хранит файлы на компьютере. Вспомним, что в главе 1 мы утверждали, что в системе Linux существуют только файлы, и ничего, кроме файлов. И это правда, но не вся.

Все сущности являются файлами, но что делает файл файлом? Я понимаю, что мы начинаем погружаться в философию, но не пугайтесь. Оказывается, то, что вы считаете файлом, на самом деле представляет собой указатель на индексный дескриптор (*inode*) — структуру данных, в которой хранится информация о том, где именно фактически расположены физические данные на вашем компьютере, и также метаданные об этих данных, такие как тип файла, размер, время изменения, владение, полномочия и т.д. (см. главы 2 и 8).

Когда вы создаете файл, ваша файловая система присваивает ему уникальный номер индексного дескриптора и имя. Оба эти параметра хранятся в таблице индексных дескрипторов на вашем диске. Когда вы обращаетесь к файлу по имени, система

Linux использует это имя, чтобы найти индексный дескриптор, связанный с ним; как только информация сохранена системой Linux в индексном дескрипторе, она уже может влиять на файл. Если вы хотите увидеть номер индексного дескриптора, связанного с файлом, используйте команду `ls -i`.

```
$ ls -i
49889 backup_daily.sh
49796 dotfiles/
49795 dotfiles.sh
49886 extra/
49291 wp_upgrade.sh
```

Видите числа слева от файлов и каталогов? Это номера индексных дескрипторов. Обратите внимание на то, что у каталогов тоже есть номера индексных дескрипторов. Почему? Вспомните, что в главе 1 говорилось, что каталоги это тоже файлы — файлы специальных типов, содержащие информацию о других файлах. Теперь вы знаете то, что на самом деле это значит: каталог или папка — это файл (с его собственным индексным дескриптором), который связывает файлы, содержащиеся внутри, с их индексными дескрипторами.

Большинство файловых систем создает постоянное количество индексных дескрипторов, так что теоретически может возникнуть ситуация, в которой существует много свободного места на диске, но больше невозможно создавать файлы, потому что количество индексных дескрипторов исчерпано (это бывает чрезвычайно редко, но я уже сталкивался с таким явлением). Если хотите знать больше о том, сколько индексных дескрипторов уже имеет ваша файловая система и сколько их есть в запасе, используйте команду `df` из главы 13 с опциями `-h` (для повышения читабельности) и `-i` (или `- inodes`), и вы увидите что-то вроде нижеследующего.

```
# df -h
Filesystem  Size   Used   Avail   Use%  Mounted on
/dev/sda    4.9G   742M   4.1G    16%   /
none        396M   308K   395M     1%   /run
/dev/sdb     42G    33G   8.8G    79%   /var
```

```
$ df -i
Filesystem  Inodes   IUsed  IFree  IUse%  Mounted on
/dev/sda    327680   40627  287053  13%   /
none        505701     826  504875   1%  /run
/dev/sdb    2785280 706754  2078526  26% /var
```

Например, вы можете узнать, что каталог `/var` использует 79% отведенного ему пространства и только 26% доступных структур *inode*. Таким образом, у вас есть возможность создать, например, много маленьких файлов. Разобравшись с индексными дескрипторами, перейдем к команде `ln`.

Создание ссылки на другой файл или каталог

Иногда было бы хорошо иметь один тот же файл в двух или больше местах одновременно или иметь больше одного способа обращаться к файлу или каталогу. В таких случаях используйте команду `ln`, представляющую собой сокращение слова *link* (ссылка).

Существуют два вида ссылок: *жесткая* (по умолчанию) и *мягкая* (вероятно, самая распространенная). Как вы вскоре убедитесь, они довольно сильно отличаются друг от друга.

Если вы по какой-то причине пропустили предыдущий раздел, обязательно прочитайте его, потому что вы должны знать, что такое индексный дескриптор, чтобы понять следующий материал. Я вас подожду … о, вы уже вернулись? Теперь, когда вы знаете об индексных дескрипторах и понимаете, почему они такие важные, рассмотрим несколько характеристик жестких ссылок. Для того чтобы создать жесткую ссылку, используется команда `ln original-file link-name`, но это только начало.

```
$ ls -l
drwxr-xr-x 2 root root 4096 melville/
-rw-r--r-- 1 root root 16 moby_dick.txt
```

```
$ cat moby_dick.txt
Call my Ishmael
$ ln moby_dick.txt white_whale.txt
$ ls -l
drwxr-xr-x 2 root root 4096 melville/
-rw-r--r-- 2 root root 16 moby_dick.txt
-rw-r--r-- 2 root root 16 white_whale.txt
$ ls -i
40966 melville/
25689 moby_dick.txt
25689 white_whale.txt
$ cat white_whale.txt
Call my Ishmael
```

Для начала у нас есть каталог `melville` и текстовый файл, названный `moby_dick.txt`, содержащий слова "Call my Ishmael" (я знаю, знаю ...успокойтесь, читатели *Моби Дика*). Затем я применяю команду `ln`, указывая первый файл, на который мы хотим установить ссылку, и имя жесткой ссылки после него. Заметьте, что, когда я снова выполняю команду `ls -l`, и файл, и жесткая ссылка имеет точно тот же размер, и, кроме того, число `1`, вычисленное для файла `moby_dick.txt` в результате первого применения команды `ls`, изменяется на `2` после второго ее применения. Почему? Потому что это число показывает, сколько жестких ссылок связано с файлом (это не имеет отношения к каталогам и индексным дескрипторам, потому что, как говорилось в главе 2, это число в случае каталога сообщает, сколько в нем подкаталогов). Число `1` означает, что файл существует, поскольку должна существовать по крайней мере *одна* связь между файлом и его индексным дескриптором. Когда вы создаете новую жесткую ссылку, это число увеличивается на единицу, а когда удаляете ссылку, оно уменьшается на единицу. Фактически именно это действие выполняет команда `rm`, описываемая в следующем разделе: она разрывает связь между файлом и его индексным дескриптором; когда вы дойдете до числа `1` и примените команду `rm` к файлу, файл будет окончательно уничтожен.

ЗАМЕЧАНИЕ

Существует еще один способ удаления ссылки — с помощью команды `unlink`: `unlink moby_dick.txt`. Внимательные читатели могли заметить, что я уже использовал эту команду для удаления исходного файла. Она прекрасно работает, потому что файл `white_whale.txt` не поврежден, так как он ссылается на тот же самый индексный дескриптор, а значит, на те же самые данные.

В предыдущем примере, в котором использовалась команда `s -i`, индексные дескрипторы, связанные с файлами `moby_dick.txt` (исходный файл) и `white_whale.txt` (жесткая ссылка), имеют один и тот же номер: `25689`. В заключение, выполнив команду `cat white_whale.txt`, я убедился, что текст в файле `white_whale.txt` идентичен тексту в файле `moby_dick.txt`. Все это свидетельствует о том, что на самом деле `moby_dick.txt` и `white_whale.txt` — это один и тот же файл! Однако погодите — это еще не все!

```
$ echo "Call me Ishmael" > white_whale.txt
$ cat white_whale.txt
Call me Ishmael
$ cat moby_dick.txt
Call me Ishmael
$ mv moby_dick.txt moby-dick.txt
$ cat while_whale.txt
Call me Ishmael
$ mv moby-dick.txt melville/
$ cat white_whale.txt
Call me Ishmael
```

На самом деле первая строка книги Moby-Dick — "Call me Ishmael", а не "Call my Ishmael". Для того чтобы исправить ошибку, я использую команду `echo` (которая удаляет все, что я вводил), а затем перенаправляю вывод (см. главу 5) на жесткую ссылку `white_whale.txt`. Затем применяю команду `cat` к обоим файлам, так что теперь они имеют одинаковое содержание, и, следовательно, мы имеем дело с одним и тем же файлом, так

что неважно, какой из них следует изменить, потому что это одни и те же данные.

И еще один интересный факт: если переименовать исходный файл командой `mv` (см. выше в этой главе), то ссылка останется корректной. Почему? Потому что индексный дескриптор не изменился и ссылается на те же самые данные, хранящиеся на диске. Если теперь применить команду `mv` для физического перемещения файла в другое место, то ссылка и в этом случае останется правильной и будет ссылаться на то же самое содержимое. Почему? Смотри предыдущий ответ! С учетом мощи жестких ссылок на них наложен ряд ограничений. Самое важное — невозможно установить жесткую ссылку на каталог, ну никак (для этого используется мягкая ссылка, которая описывается далее). Второе ограничение состоит в том, что жесткая ссылка и исходный файл должны принадлежать одной и той же файловой системе, поэтому невозможно иметь файл в одном разделе диска, а жесткую ссылку — в другом (помните, что каждый раздел диска имеет свой индексный дескриптор!). Если вы попытаетесь это сделать, то получите сообщение об ошибке "Invalid cross-device link".

Теперь, когда вы знаете, как работают жесткие ссылки, настало время разобраться с мягкими ссылками. Для того чтобы создать мягкую ссылку (известную также как *символическая ссылка* (symbolic link), или просто *symlink*), используется тот же синтаксис, что и раньше, но с опцией `-s`, например `ln -s original-file link-name`. Давайте взглянем на один из моих серверов, и я поясню, что вы видите.

```
$ ls -l /
drwxr-xr-x 98 4096 etc/
drwxr-xr-x 2 4096 home/
lrwxrwxrwx 1 28 mb -> /var/shared/multiblog/
lrwxrwxrwx 1 18 shared -> /var/shared/
drwxr-xr-x 17 4096 var/
lrwxrwxrwx 1 12 webconf -> /etc/apache2/
lrwxrwxrwx 1 8 www -> /var/www/
[Листинг сокращен из-за недостатка места]
```

Это частичный листинг корневого каталога на моем сервере. Этот каталог содержит четыре мягких ссылки: mb, shared, webconf и www. Я и мои партнеры устали подключаться к серверу по протоколу SSH, а затем набирать что-то вроде cd /var/www, поэтому мы создали мягкую ссылку с именем www, которая указывает на этот каталог (да, я знаю, что это выглядит нелепо, но если вы будете повторять эти команды постоянно, то вам тоже надоест!). Это действие можно выполнить несколькими способами. Вот первый:

```
$ sudo ln -s /var/www /www
```

В предыдущем примере я использовал абсолютный путь для создания мягкой ссылки /www, указывающей на каталог /var/www (я использовал программу sudo, потому что обычный пользователь не может создавать ссылки или что либо еще в каталоге /). Преимущество этого метода заключается в том, что я могу создать эту ссылку, находясь в любом месте системы, поскольку я использую абсолютный путь. Но что если я поленюсь еще немного?

```
$ cd /
$ sudo ln -s /var/www .
```

Здесь я применил команду cd к каталогу /, в котором хотел создать мягкую ссылку, а затем создал ее, указав вместо ее имени символ . (точку). Тем самым я сообщил команде ln, что хочу, чтобы имя ссылки совпадало с именем файла или папки, на которые я ссылаюсь. В данном примере имя ссылки — www, потому что это последний компонент исходного пути. Это ленивый способ (я считаю, что это хорошо!). А что если я стану еще ленивее?

```
$ cd /
$ sudo ln -s /var/www
```

Здесь я снова применил команду cd к каталогу /, в котором хотел создать мягкую ссылку, затем создал ее, но вместо имени не указал вообще ничего. Тем самым я сообщил команде ln, что хочу, чтобы имя ссылки совпадало с именем файла или папки, на которую я ссылаюсь. Как и ранее, мягкая ссылка имеет имя

www, потому что это последний компонент исходного пути. Вот это на самом деле *ленивый* способ! Независимо от способа создания мягкие ссылки работают как файл или папка, на которую они ссылаются (команды pwd и pwd -P обсуждались в главе 2).

```
$ pwd
/home/scott
$ ln -s /var/www
$ ls -l
lrwxrwxrwx 1 8 www -> /var/www/
$ cd www
$ pwd -P
/var/www
$ ls -l
lrwxrwxrwx 1 8 notes -> /home/scott/notes.txt
$ cat notes
Дальше идет содержимое файла notes.
```

Первая созданная мною мягкая ссылка указывает на каталог, поэтому, когда я применяю команду cd к этой мягкой ссылке, я на самом деле оказываюсь в каталоге, на который ссылался. Затем я применяю команду cat к мягкой ссылке, указывающей на файл, и вижу его содержимое. Если бы я открыл мягкую ссылку в редакторе, то увидел бы фактические изменения содержимого исходного файла.

Итак, теперь вы знаете, что такое мягкие ссылки и как они работают, но чем они отличаются от жестких? О, между ними много различий! Начнем с результатов команды ls, предполагая, что на просматриваемый нами каталог указывают жесткая и мягкая ссылки.

```
$ ls -l
-rw-r--r-- 2 moby_dick.txt
lrwxrwxrwx 12 webconf -> /etc/apache2/
-rw-r--r-- 2 white_whale.txt
$ ls -i
25689 moby_dick.txt
25372 webconf
25689 white_whale.txt
$ ls -i /etc
59948 apache2/
```

Рассмотрим результат выполнения команды `ls -l`. Первый столбец содержит букву `l` (сокращение от слова *link*), если ссылка мягкая, и знак "минус" (означающий "обычный" файл), если ссылка жесткая. Почему? Потому что с точки зрения вашей файловой системы жесткие ссылки — это обычные файлы, связанные с индексными дескрипторами. Другой факт, который вы обязаны обнаружить, заключается в том, что команда `ls -l` демонстрирует указатель на файл или папку, с которыми связана мягкая ссылка, что очень удобно. Для жесткой ссылки это не имеет смысла, потому что исходный файл и любая жесткая ссылка на него на самом деле одно и то же. С точки зрения операционной системы они идентичны, поэтому между исходными файлами и любыми ссылками на них различий не проводится. Далее, поскольку мягкая ссылка может ссылаться только на один файл или папку, но в то же время можно иметь много жестких ссылок на один и тот же файл, то как понять, что на что ссылается? Мой совет: даже не пытайтесь.

Вторая команда — `ls -i` — также демонстрирует ключевые различия между жесткими и мягкими ссылками. Разумеется, жесткие ссылки имеют одинаковые номера индексных дескрипторов, но мягкие ссылки и файлы, на которые они указывают, имеют разные номера, потому что это совершенно разные файлы. В отличие от жестких ссылок, которые на самом деле представляют собой один и тот же файл, мягкие ссылки используют имя для указания на файлы и каталоги, и файл с указанным именем имеет свой собственный индексный дескриптор. Иначе говоря, мягкая ссылка имеет свой собственный индексный дескриптор, указывающий на индексный дескриптор исходного файла, который указывает на реальные данные, записанные на диске!

Еще немного различий: самое важное — мягкие ссылки могут указывать на файлы и каталоги, а жесткие ссылки — только на файлы. В отличие от жесткой ссылки, мягкая ссылка и файл, на который она указывает, не обязаны принадлежать одной и той же файловой системе. Если исходный файл удален или перемещен, то мягкая ссылка разрывается, но имя мягкой ссылки

всегда можно изменить без каких-либо проблем. Результаты сравнения и различия между жесткими и мягкими ссылками приведены в табл. 3.2.

Таблица 3.2. Жесткие и мягкие ссылки

Тип	Жесткая	Мягкая
На что ссылается	Индексный дескриптор	Имя
Ссылка на каталог	Нет	Да
Ссылка за пределы файловой системы	Нет	Да
При перемещении исходного файла	Ссылка остается корректной	Ссылка разрывается
При изменении имени целевого файла	Ссылка остается корректной	Ссылка разрывается
После удаления ссылки	Цель продолжает существовать*	Цель продолжает существовать
Индексный дескриптор ссылки	Совпадает с целевым	Отличается от целевого
Объем памяти	Ничего	Около 4 Кбайт

* Если не удалена последняя ссылка или выполняемый процесс не использует открытый файл.

Жесткие и мягкие ссылки имеют одно и то же предназначение и могут оказаться очень полезными. Используйте их!

Удаление файлов

```
rm
```

Команда `rm` (в ней нашли место только две буквы из слова "remove") безвозвратно удаляет файлы. В системе Linux нет "мусорной корзины". Одно неосторожное движение — и вернуть данные уже нельзя.

Может быть, не все так мрачно? Действительно, при работе с командной строкой "мусорная корзина" не используется, но кое-что все же сделать можно. Если вы остановите машину

в тот момент, когда обнаружите ошибку, и если операционная система еще не записала данные в те сектора, в которых содержался удаленный файл, и если вы сможете умело воспользоваться сложными инструментами восстановления информации, то есть шанс вернуть файл. Но в любом случае это займет много времени. Поэтому будьте внимательны.

ПОДСКАЗКА

Попытки обезопасить команду `rm` предпринимаются многими. Некоторые заменяют ее командой перезаписи во временное хранилище (`http://blogs.adobe.com/cantrell/archives/2012/03/stopusing-rm-on-the-command-line-before-its-too-late.html`)), другие заменяют `rm` новой командой `trash` (`https://github.com/andreafrancia/trash-cli`)).

С другой стороны, иногда хочется быть уверенным, что никто, даже сотрудники спецслужб, не сможет восстановить удаленные файлы. В этом случае вместо `rm` желательно использовать команду `shred`. Эта команда 25 раз перезаписывает область на диске, где хранился файл, так что прочитать данные оказывается невозможно (этот параметр можно уточнить с помощью опции `-n #`). Однако перед использованием команды `shred` обратитесь к справочным страницам `man`, поскольку успех ее использования в большой степени зависит от типа файловой системы, с которой вы работаете.

Команда `rm` проста в использовании, можно даже сказать, слишком проста.

```
$ pwd
/home/scott/libby/by_pool/lieberman_pool
$ ls
libby_by_pool_01.jpg libby_by_pool_03.jpg
libby_by_pool_01.jpg_bak libby_by_pool_03.jpg_bak
$ rm libby_by_pool_01.jpg_bak
$ ls
```

```
libby_by_pool_01.jpg libby_by_pool_03.jpg
↳libby_by_pool_03.jpg_bak
```

Удаление нескольких файлов с помощью символов групповых операций

```
rm *
```

Символы групповых операций, например *, позволяют одним нажатием клавиши удалить несколько файлов.

```
$ pwd
/home/scott/libby/by_pool/lieberman_pool
$ ls
libby_by_pool_01.jpg      libby_by_pool_03.jpg
libby_by_pool_01.jpg_bak  libby_by_pool_03.jpg_bak
$ rm *_bak
$ ls
libby_by_pool_01.jpg libby_by_pool_03.jpg
```

ПРЕДУПРЕЖДЕНИЕ

Будьте предельно внимательны, удаляя файлы с указанием символов групповых операций, потому что вы можете удалить гораздо больше, чем собирались. Классический пример — ввод вместо `rm *txt` команды `rm * txt`. Вместо текстовых файлов будут удалены все файлы, а затем команда предпримет попытку удалить файл `txt`.

Как предотвратить удаление важных файлов

```
rm -i
```

Опция `-i` (или `--interactive`) делает команду `rm` более безопасной. В этом случае команда запрашивает у пользователя

подтверждение на удаление каждого файла. Эта опция незаменима при работе с правами root!

```
$ pwd
/home/scott/libby/by_pool/lieberman_pool
$ ls
libby_by_pool_01.jpg libby_by_pool_03.jpg
libby_by_pool_01.jpg_bak libby_by_pool_03.jpg_bak
$ rm -i *_bak
rm: remove regular file 'libby_by_pool_01.jpg_bak'?
↳y
rm: remove regular file 'libby_by_pool_03.jpg_bak'?
↳y
$ ls
libby_by_pool_01.jpg libby_by_pool_03.jpg
```

Как и в командах, рассмотренных ранее, ответ y на запрос команды означает согласие на удаление файла, а ответ n — приказ сохранить файл. Получив ответ n, команда не прекращает работу, а переходит к обработке следующего файла.

Удаление пустого каталога

rmdir

Удалить файл несложно, но что делать с каталогами?

```
$ pwd
/home/scott/libby/by_pool
$ ls
libby_by_pool_02.jpg lieberman_pool
↳lieberman_pool_bak
$ ls lieberman_pool_bak
libby_by_pool_01.jpg      libby_by_pool_03.jpg
libby_by_pool_01.jpg_bak  libby_by_pool_03.jpg_bak
$ rm lieberman_pool_bak
rm: cannot remove 'lieberman_pool_bak/':
↳Is a directory
```

Действительно, попытка не удалась. Однако существует команда rmdir, специально предназначенная для удаления каталогов. Попробуем использовать ее.

```
$ rmdir lieberman_pool_bak
rmdir: 'lieberman_pool_bak/': Directory not empty
```

И опять неудача. Команда `rmdir` может удалить только пустой каталог. В нашем случае в каталоге `lieberman_pool_bak` содержатся только четыре файла, так что нетрудно удалить их, а затем использовать `rmdir`. Но что делать, если надо удалить каталог, содержащий 10 подкаталогов. Более того, в каждом из них находится 10 подкаталогов, каждый из которых содержит 25 файлов? Работа по удалению каталогов затянется надолго. Однако есть более простой путь.

Удаление файлов и каталогов, содержащих данные

```
rm -Rf
```

Для решения этой задачи надо использовать сочетание опций `-R` (или `--recursive`) и `-f` (или `--force`). Опция `-R` указывает команде `rm` на то, что надо перейти в каждый подкаталог и удалить его содержимое, а опция `-f` говорит, что пользователя не следует беспокоить напоминанием о том, что очередной каталог не пустой.

```
$ pwd
/home/scott/libby/by_pool
$ ls
libby_by_pool_02.jpg lieberman_pool
.lieberman_pool_bak
$ ls lieberman_pool_bak
libby_by_pool_01.jpg      libby_by_pool_03.jpg
libby_by_pool_01.jpg_bak  libby_by_pool_03.jpg_bak
$ rm -rf lieberman_pool_bak
$ ls
libby_by_pool_02.jpg lieberman_pool
```

Это удобный способ избавиться от каталога и всех его подкаталогов.

ПРЕДУПРЕЖДЕНИЕ

Команда `rm -Rf` может удалить важные файлы и разрушить саму операционную систему.

Классическое предупреждение пользователям Linux: не используйте команду `rm -Rf /*`, если работаете как пользователь `root`. Так можно стереть всю систему. Пользователь, сделавший это, выглядит довольно глупо.

В любом случае, используя символы групповых операций в команде `rm -Rf`, надо соблюдать осторожность. Различие между командами `rm -Rf libby*` и `rm -Rf libby*` чрезвычайно велико. Первая из них удаляет в рабочем каталоге все файлы, начинающиеся на `libby`; вторая команда сначала удаляет файл `libby`, а затем и все остальные файлы в каталоге и его подкаталогах.

Если вместо команды `rm -Rf ~/libby/*` вы зададите `rm -Rf ~/libby /*`, у вас возникнут большие проблемы. Сначала исчезнет каталог `~/libby`, а затем вся файловая система.

И еще одно предупреждение пользователям: никогда не задавайте команду `rm -Rf .*/*`, чтобы удалить файлы, начинающиеся с точки, поскольку указанному критерию удовлетворяет каталог `..`, и вы удалите все данные, расположенные выше вашего рабочего каталога.

Еще раз запомните: используя `rm -Rf`, соблюдайте осторожность! Удвойте внимание, если вы работаете с полномочиями `root`!

Удаление проблемных файлов

Прежде чем закончить разговор о команде `rm`, обсудим некоторые особенности, связанные с применением этой команды к вашей файловой системе. Во-первых, как бы вы ни старались, вы не сможете удалить каталог `.` или `..`, так как он необходим для поддержки самой файловой системы. Во-вторых, зачем их удалять? Оставим их в покое!

Как удалить файл, в имени которого содержится пробел? Обычный способ — вызов команды `rm` и указание имени файла в качестве параметра — не подходит, так как команда интерпретирует заданное имя как два отдельных параметра. На самом деле решить эту задачу несложно. Достаточно поместить имя файла в кавычки.

```
$ ls
cousin harold.jpg -cousin_roy.jpg cousin_beth.jpg
$ rm cousin harold.jpg
rm: cannot remove 'cousin':
  No such file or directory
rm: cannot remove 'harold.jpg':
  No such file or directory
$ rm "cousin harold.jpg"
$ ls
-cousin_roy.jpg  cousin_beth.jpg
```

И еще одна проблема: как удалить файл, имя которого начинается с дефиса?

```
$ ls
-cousin_roy.jpg  cousin_beth.jpg
$ rm -cousin_roy.jpg
rm: invalid option -- c
Try 'rm -help' for more information.
```

Команда `rm` воспринимает символ - как признак опции, но в данном случае не распознает опцию, начинающуюся с буквы `c`, за которой следуют символы `ousin_roy.jpg`. В результате команда не знает, как поступить.

Существуют два решения этой проблемы. Во-первых, можно предварить имя файла двумя дефисами (--). Это означает, что следующие за ними данные должны восприниматься не как опция, как имя файла или каталога.

```
$ ls
-cousin_roy.jpg  cousin_beth.jpg
$ rm -- -cousin_roy.jpg
$ ls
cousin_beth.jpg
```

Во-вторых, можно использовать точку как часть пути к файлу и тем самым устранить тот самый пробел перед дефисом, который ввел в заблуждение команду `rm`.

```
$ ls
-cousin_roy.jpg  cousin_beth.jpg
$ rm ./-cousin_roy.jpg
$ ls
cousin_beth.jpg
```

Этот пример еще раз подтверждает: изобретательность пользователей Linux не знает границ. Однако лучше всего не включать дефис в начало файла!

Выводы

Эта глава напомнила мне о моем шестилетнем сыне: он любит создавать фантастических персонажей из своего конструктора Лего и давать им безумные имена, однако через некоторое время он разрушает их, распевая веселые песни. Подобно моему сыну, вы теперь можете создавать файлы и каталоги (`touch`, `mkdir` и `cp`), изменять их местоположение или имена (`mv`), создавать ссылки на них (`ln`) и уничтожать их (`rm` и `rmdir`). Имея в своем распоряжении такие важные команды, вы теперь можете освоить остальные команды. Звучит замысловато? Да вы еще ничего не видели!

4

Получение информации о командах

В предыдущей главе мы начали изучать основные команды системы Linux. Мы рассмотрели довольно много команд них, но осталось гораздо больше. Команда `ls` — мощный инструмент, поддерживающий огромное количество опций, гораздо больше, чем было рассмотрено в главе 2. Как же получить дополнительную информацию об этой команде и других, интересующих вас? И как разобраться в командах, если неизвестны даже их имена? В этом вам поможет данная глава. Вы узнаете, как глубже изучить те команды, о которых вы успели составить представление, те, о которых кроме имени вы не знаете ничего, и даже полностью неизвестные команды.

Начнем с "гигантов" — команд `man` и `info`, а затем перейдем к менее глобальным командам, которые тем не менее используют ту же информацию, что и команда `man`. Дойдя до конца этой главы, вы будете готовы продолжить изучение самых разнообразных средств, доступных посредством оболочки.

ЗАМЕЧАНИЕ

Пересматривая книгу для второго издания, я удалил информацию о команде `man -u` (которая перестраивает базу данных справочной системы `man`) и команде `man -t` (которая выводит страницы `man` на печать). Кроме того, информация о команде `whatis` была перемещена в раздел о команде `man -f`, а информация о команде `apropos` — в раздел о команде `man -k`. Исходный текст можно найти на моем веб-сайте `www.granneman.com/linux-redactions`.

Получение информации о командах с помощью команды man

```
man
```

Хотите больше узнать о командах Linux? Нет ничего проще. Предположим, вам понадобилась информация о команде `ls`. Введите команду `man ls`, и на экране появится страница справочного руководства (сама команда `man` сокращенно означает *manual*, т.е. руководство), на которой подробно описываются все особенности команды `ls`. В качестве параметра `man` можно указать имя любой команды. Все они (или почти все) отражены в справочной системе.

Несмотря на то что справочное руководство чрезвычайно полезно, при работе с командой `man` подчас возникают проблемы. Необходимо знать имя команды (хотя эту трудность можно обойти), кроме того, нередко информация оказывается устаревшей и не отражает средства, появившиеся относительно недавно. Сведения о некоторых командах вовсе отсутствуют. Однако хуже всего, когда описание интересующей вас команды существует и еще не устарело, но, ознакомившись с ним, вы обнаруживаете, что оно практически бесполезно.

Обычно страницы справочного руководства разрабатывают те же специалисты, которые пишут программы (бывают и исключения, но, как правило, это так). Большинство разработчиков, приложения которых были включены в дистрибутив Linux, являются первоклассными программистами, но они не всегда могут доходчиво описать работу программы. Они прекрасно разбираются в предмете, но слишком часто забывают о том, что то, что очевидно для разработчиков, зачастую неизвестно пользователям.

Несмотря на перечисленные выше проблемы, справочное руководство — незаменимый ресурс для пользователей Linux всех уровней квалификации. Если вы хотите работать с командной строкой, вам надо научиться пользоваться

руководством. Как было сказано ранее, использовать команду man несложно. Надо ввести man, а затем указать имя команды, о которой вы хотите получить дополнительные сведения.

```
$ man ls
LS(1)        User Commands              LS(1)
NAME
      ls - list directory contents
SYNOPSIS
      ls [OPTION]... [FILE]...
DESCRIPTION
      List information about the FILEs (the
↳current directory by default).
      Sort entries alphabetically if none of -
↳cftuSUX nor --sort.
      Mandatory arguments to long options are
↳mandatory for short options too.
      -a, --all
            do not hide entries starting with .
      -A, --almost-all
            do not list implied . and ..
```

Объем данных, предоставляемых man, часто бывает очень большим. Для данного примера он составляет более 200 строк. Описание некоторых команд очень короткое, однако есть и такие команды, описание которых гораздо длиннее, чем приведенное в данном примере. Ваша задача — ознакомиться с описанием, которое в большинстве случаев включает в себя следующие разделы.

- NAME. Имя команды и краткое описание.
- SYNOPSIS. Формат вызова команды.
- DESCRIPTION. Подробное описание возможностей, предоставляемых командой.
- OPTIONS. Чаще всего именно этот раздел оказывается самым полезным для пользователей. В нем приводятся опции, предусмотренные для команды, и их краткое описание.
- FILES. Файлы, используемые командой.

- AUTHOR. Автор, который разработал программу, реализующую эту команду; контактная информация.
- BUGS. Замеченные недостатки и сведения о том, куда сообщить, если вы обнаружите новую ошибку.
- COPYRIGHT. Информация об авторских правах.
- SEE ALSO. Другие команды, имеющие отношение к рассматриваемой.

Просмотр страницы справочной системы происходит следующим образом. Для того чтобы перейти на строку вниз, следует нажать клавишу со стрелкой вниз, чтобы переместиться на одну строку вверх, надо нажать клавишу со стрелкой вверх. Перейти вперед на одну страницу позволяет клавиша пробел или <F> (первая буква в слове *forward*), а на страницу назад — клавиша <B> (первая буква в слове *backward*). Когда вы дойдете до конца страницы, выполнение команды `man` автоматически завершится и вы вернетесь в среду оболочки. В некоторых случаях автоматическое завершение не происходит, и тогда надо нажать клавишу <Q>, чтобы выйти из программы. Завершить команду можно в любой момент с помощью клавиши <Q>; при этом не обязательно находиться в конце страницы.

Нередко бывает нелегко найти требуемые сведения на странице, поэтому пользователю предоставляются минимальные средства поиска. Для того чтобы отыскать данные на открытой странице справочного руководства, надо ввести косую черту, затем термин, который надо найти, и нажать клавишу <Enter>. Если этот термин существует, содержимое страницы сдвинется так, чтобы он отображался на экране. Чтобы продолжить поиск, надо повторно нажимать клавишу <Enter> (или <N>). Чтобы вернуться к предыдущему вхождению термина, надо нажать комбинацию клавиш <Shift+N>.

ЗАМЕЧАНИЕ

По умолчанию для реального вывода страниц справочника на экран команда `man` использует команду `less`

(которая рассматривается в главе 6). Приведенные выше комбинации клавиш для навигации по справочнику относятся к команде `less`; для того чтобы получить больше информации об этих комбинациях клавиш, выполните команду `man less`. Если по каким-то причинам вы не хотите использовать команду `less`, то измените переменные окружения `MANPAGER` или `PAGER`. Правда, для того чтобы понять, как они работают, придется потрудиться!

Получение кратких сведений о команде

```
man -f
whatis
```

Если вам известно имя команды, но вы не знаете, какие действия она выполняет, с ее назначением можно быстро ознакомиться, не просматривая всю страницу справочного руководства. Если вы зададите опцию `-f` (или `--whatis`) или выполните команду `whatis`, на экране отобразится раздел с описанием команды.

```
$ man -f ls
ls (1)              - list directory contents
$ whatis ls
ls (1)              - list directory contents
```

Какой из этих вариантов предпочтительнее? Лично мне нравится использовать команду `whatis`, поскольку ее легче запомнить, а мои друзья предпочитают команду `man -f`, потому что она короче. Другим преимуществом команды `whatis` является то, что она поддерживает обработку регулярных выражений и символов групповых операций. Для поиска по базе данных с помощью групповых операций используйте опцию `-w` (или `--wildcard`).

```
$ whatis -w ls*
ls (1)    - list directory contents
lsb (8)   - Linux Standard Base support for Debian
```

```
lshal (1) - List devices and their pro perties
lshw (1)  - list hardware
lskat (6) - Lieutnant Skat card game for KDE
```

Применение символов групповых операций немного замедляет поиск по сравнению с командой `whatis` без опций, но на современных быстрых компьютерах этой разницей можно пренебречь.

Регулярные выражения можно использовать с опцией `-r` (или `--regex`).

```
$ whatis -r ^rm.*
rm (1)    - remove files or directories
rmail (8) - handle remote mail received via uucp
rmdir (1) - remove empty directories
rmt (8)   - remote magtape protocol module
```

ПОДСКАЗКА

Из-за недостатка места регулярные выражения в этой книге не рассматриваются, но вы можете узнать больше, прочитав книгу Бена Форта (Ben Forta) *Освой самостоятельно регулярные выражения. 10 минут на урок* (пер. с англ., ИД "Вильямс", 2005 г.). Настоятельно советую как можно лучше изучить регулярные выражения — они чрезвычайно полезны. Еще лучше, если вы заглянете на сайт `http://xkcd.com/208/`!

Поиск команды по выполняемым ею действиям

```
man -k
apropos
```

Немного поработав с командой `man`, вы обнаружите, что она предоставит нужную вам информацию... при условии, что вы знаете, на какой странице эта информация находится. Но

что делать, если вам известно, какие действия должна выполнять команда, но не знаете ее реального имени? В этой ситуации вам поможет опция `-k` (или `--apropos`). Она позволяет организовать поиск по слову или фразе, описывающим команду. На экран выводится список команд, для которых либо имя совпадает с ключевым словом, либо это слово содержится в разделе `SYNOPSIS`.

```
$ man list
No manual entry for list
$ man -k list
last (1)    - show listing of last logged
↳in users
ls (1)      - list directory contents
lshw (1)    - list hardware
lsof (8)  - list open files
[Листинг сокращен для экономии места]
```

Опцию `-k` надо применять осторожно, так как объем выходных данных может быть очень большим, и вы не сможете найти в них те сведения, которые вам нужны. Если поиск закончился безрезультатно, попробуйте повторить его, задав другое ключевое слово.

ПОДСКАЗКА

Если страница справочника является слишком длинной и вам нелегко найти то, что вы ищете, попытайтесь организовать конвейер из команд `man -k` и `grep`. (Понятие конвейера обсуждается в следующей главе, а команда `grep` — в главе 10.) Пока просто введите команду `man -k list | grep hardware`. В качестве альтернативы можете выполнить команду `apropos`, которая рассматривается в этом разделе!

Команда `apropos` делает то же самое, что и команда `man -k`, за одним важным исключением: подобно команде `whatis` (рассмотренной в предыдущем разделе), для поиска

можно использовать опцию -w (или --wildcard) или -r (или --regex). Впрочем, стоит отметить, что существует возможность использовать опцию -e (или --exact), которая позволяет сузить поиск на конкретном слове или фразе без каких-либо исключений. Например, в предыдущем листинге поиск слова *list* привел к выдаче информации о команде last, потому что в ее описании есть слово *listing*. Посмотрим, что получится, если выполнить тот же самый поиск без опции -e.

```
$ apropos -e list
ls (1)   - list directory contents
lshw (1) - list hardware
lsof (8) - list open files
```

На этот раз команда last не используется, потому нам нужны только результаты, которые содержат слово *list*, а не *listing*. На самом деле список результатов поиска слова list содержит 80 позиций без опции -e и 55 позиций с использованием опции -e, поэтому опция -e делает поиск точнее.

ЗАМЕЧАНИЕ

Слово *apropos* в обыденной речи встречается редко, но на самом деле это литературное слово, означающее "соответствующий" или "уместный." Синонимом этого слова является *appropriate* (подходящий), но оно имеет другой латинский корень. Если верите, найдите точное толкование этих слов в словаре по адресу www.dictionary.com.

Просмотр страницы справочной системы, посвященной конкретной команде

```
man [1-8]
```

В предыдущем листинге вы, вероятно, заметили, что на странице для команды `ls` отображается последовательность символов `LS(1)`. Ранее, когда мы рассматривали опцию `-k`, имя каждой команды сопровождалось числом в скобках. В большинстве случаев после имени отображалось число 1, но после команды `lsof` было выведено число 8. Что же означают эти цифры?

Страницы справочного руководства распределены по разделам с номерами от 1 до 8. Назначение разделов описано ниже (если какой-то из примеров будет непонятен вам, не беспокойтесь; большинство команд являются узкоспециализированными).

1. Команды общего назначения, например `cd`, `chmod`, `lp`, `mkdir`, `passwd`.
2. Низкоуровневые системные вызовы, поддерживаемые ядром, например `intro`, `chmod`.
3. Функции библиотеки C, например `beep`, `HTML::Parser`, `Mail::Internet`.
4. Специальные файлы, к которым, в частности, относятся устройства в каталоге `/dev`, например `console`, `lp`, `mouse`.
5. Информация о форматах файлов и соглашениях, например `apt.conf`, `dpkg.cfg`, `hosts`, `passwd`.
6. Игры, например `atlantik`, `bouncingcow`, `kmahjongg`, `rubik`.
7. Различная информация, в том числе сведения о макропакетах, например `ascii`, `samba`, `utf-8`.
8. Команды системного администрирования, вызываемые пользователем `root`, например `mount`, `shutdown`.

До сих пор все интересовавшие нас команды располагались в разделе 1. Это неудивительно, так как мы изучаем общие вопросы работы с системой Linux. Заметьте, что некоторые ключевые слова соответствуют нескольким разделам, например, информация о chmod находится в разделах 1 и 2, а сведения о passwd можно получить в разделе 1 или 5. По умолчанию принимается наименьший из возможных номеров. Так, если вы введете в командной строке man passwd, то получите информацию о команде с таким именем из раздела 1. Если вас интересует файл passwd, полученная информация вряд ли окажется полезной для вас. Для того чтобы получить страницу для файла passwd, укажите после man требуемый номер раздела.

```
$ man passwd
PASSWD(1)                                PASSWD(1)
NAME
       passwd - change user password
SYNOPSIS
       passwd [-f|-s] [name]
       passwd [-g] [-r|-R] group
       passwd [-x max] [-n min] [-w warn] [-i inact]
↳login
       passwd {-l|-u|-d|-S|-e} login
DESCRIPTION
      passwd changes passwords for user and group
↳accounts. A normal user...
[Листинг сокращен для экономии места]
$ man 5 passwd
PASSWD(5)                                  PASSWD(5)
NAME
      passwd - The password file
DESCRIPTION
      passwd contains various pieces of information
↳for each user account.
[Информация сокращена для экономии места]
```

Получение информации о командах с помощью info

info

Команда man проста в использовании, а предоставляемые ею страницы справочного руководства удобны, однако их нельзя назвать дружественными по отношению к пользователю. В рамках проекта GNU был разработан альтернативный формат — info-страницы, для доступа к которым используется команда info.

Многие пользователи считают, что страницы info написаны лучше, чем страницы man, на них более полно представлена информация, однако страницы man проще в работе. Для конкретной команды страница man только одна, а содержимое страниц info организовано в виде разделов, называемых узлами, которые могут включать подразделы, называемые подузлами. Для того чтобы эффективно работать с системой info, надо освоить навигацию не только в пределах одной страницы, но также между узлами и подузлами. Поначалу бывает довольно трудно найти информацию на страницах info. Как ни странно, именно система, разработанная специально для начинающих, оказывается более сложной в изучении.

ЗАМЕЧАНИЕ

Лично мне страница info очень не нравится, потому, если я вынужден ее использовать (когда в справочнике man нет нужной страницы), я обычно перехожу в поисковую систему Google.

Система info имеет много аспектов (что делает ее интереснее!). Чтобы получить сведения о системе info, надо использовать следующую команду:

```
$ info info
```

В результате ее выполнения будет открыта страница `info` для команды `info`. Попробуем научиться ориентироваться в пространстве Info.

Навигация в системе Info

В пределах одного раздела действуют следующие правила. Чтобы перейти вниз или вперед на одну строку, надо нажать клавишу со стрелкой вниз. Аналогично клавиша со стрелкой вверх позволяет перейти на одну строку вверх или назад. Когда вы достигнете конца раздела, курсор не будет реагировать на попытки сдвинуться вниз.

Если вы хотите переместиться вниз на один экран, надо использовать клавишу <PageDown>, а если на один экран вверх — клавишу <PageUp>. Описанные здесь клавиши не позволят покинуть текущий раздел.

Если вы достигли конца раздела и хотите перейти в его начало, нажмите клавишу <B> (первая буква в слове beginning).

Если, перемещаясь по разделу, вы заметили, что информация выглядит странно, например, символы или слова искажены, нажмите комбинацию клавиш <Ctrl+L>, чтобы обновить содержимое экрана.

Теперь, когда вы знаете, как перемещаться в пределах раздела или узла, рассмотрим навигацию между узлами. Если вы не хотите перемещаться вперед и назад с помощью клавиш <PageDown> и <PageUp>, используйте для перемещения вниз клавишу пробела, а для перемещения вверх — клавишу <Backspace> или <Delete>. Действие этих клавиш несколько отличается от действия клавиш <PageDown> и <PageUp>: если вы достигнете конца раздела, то перейдете к следующему разделу (или подразделу, если таковой существует). Перемещаясь назад и достигнув начала раздела, вы перейдете к предыдущему разделу или подразделу. Используя пробел или клавиши <Backspace> и <Delete>, вы можете просмотреть весь набор info-страниц для конкретной команды.

Чтобы перейти к следующему разделу несколько быстрее и избежать многократных нажатий клавиш, воспользуйтесь клавишей <N> (next). Если раздел, который вы просматриваете, содержит подразделы, то, нажав клавишу <N>, вы пропустите подразделы и сразу перейдете к следующему разделу того же уровня, что и текущий. Если вы просматриваете подраздел и нажмете клавишу <N>, то перейдете к следующему подразделу. Подобно клавише <N>, которая перемещает к следующему разделу того же уровня, клавиша <P> (previous) переместит вас к предыдущему разделу.

Если надо переместиться вперед к следующему элементу, независимо от того, является ли он узлом или подузлом, надо использовать клавишу <]>. Если вы просматриваете узел, в котором есть подузлы, то после нажатия клавиши <]> перейдете к первому из них, если же подузлов нет, то эта клавиша переместит вас к следующему узлу того же уровня. Чтобы переместиться назад подобным образом, надо использовать клавишу <[>.

Переместиться к вышестоящему или родительскому узлу позволяет клавиша <U> (up). При этом будьте осторожны: нажав данную клавишу лишний раз, вы подниметесь выше корневого узла для текущей команды и очутитесь в узле `Directory` — настоящем корневом узле, из которого можно перейти к любому другому узлу info. (Достичь узла `Directory` можно также, нажав клавишу <D> (directory), причем сделать это можно в любой момент.)

Узел `Directory` представляет один из типов страниц, доступных посредством системы info, а именно, страницу `Menu`, в которой перечисляются подтемы текущей темы. Если вы окажетесь на странице `Menu`, то сможете переместиться к одному из указанных в ней подузлов одним из двух способов. Введите `m` (menu), а затем имя подузла, в который вам необходимо перейти. Например, ниже показана первая страница, которую вы увидите, выполнив команду `info info`.

```
File: dir, Node: Top This is the top of the INFO
↳tree
```

```
This (the Directory node) gives a menu of major
↳topics.
Typing "q" exits, "?" lists all Info commands,
↳"d" returns here,
"h" gives a primer for first-timers,
↳"mEmacs<Return>" visits the Emacs manual, etc.

* Menu:

Basics
* Common options: (coreutils)Common options.
* Coreutils: (coreutils). Core GNU (file, text,
↳shell) utilities.
* Date input formats: (coreutils)Date input
↳formats.
* Ed: (ed). The GNU line editor
```

Для того чтобы перейти в пакет Coreutils, надо ввести букву `m` и за ней слово `Core`. После этого можно либо заканчивать ввод, т.е. ввести слово `utils`, либо нажать клавишу <Tab>, и система Info автоматически сформирует имя пункта, соответствующего уже введенным символам. Если система info отказывается формировать имя пункта, значит, вы допустили ошибку либо введенным вам символам соответствуют несколько пунктов меню. Исправьте ошибку или продолжайте ввод до тех пор, пока info сможет однозначно идентифицировать интересующий вас пункт. Если в этот момент окажется, что вам не надо покидать данную страницу, нажмите комбинацию клавиш <Ctrl+G>, чтобы отменить команду, и продолжайте просмотр текущего узла.

Существует и альтернативный способ. Используя клавиши со стрелкой вверх или вниз, вы можете разместить курсор на требуемом пункте меню и нажать клавишу <Enter>.

Если вам надо осуществить не навигацию, а поиск, у вас также есть два способа сделать это. Вы можете искать по заголовкам узлов либо по всему тексту. Для того чтобы искать в заголовках, введите `i` (index), затем ключевое слово и нажмите клавишу <Enter>. Если слово присутствует в заголовке, вы можете перейти к соответствующему узлу. Можно также

повторить поиск и перейти к следующему результату, для этого надо ввести запятую.

Если вы хотите выполнить поиск во всем тексте, введите s (search), затем ключевое слово или фразу и нажмите клавишу <Enter>. Для того чтобы повторить поиск, непосредственно после нажатия клавиши <Enter> нажмите клавишу <S>. Сделать это сложнее, чем ввести запятую, но такой подход позволяет добиться нужных результатов.

Если вы "заблудились" в системе info и вам нужна помощь, нажмите клавишу <?>. В ответ в нижней части окна отобразятся команды системы info. Перемещаться в этом разделе позволяют клавиши со стрелками. Чтобы отменить подсказку, нажмите комбинацию клавиш <Ctrl+X><0>, т.е. сначала нажмите комбинацию клавиш <Ctrl+X>, затем отпустите эти клавиши и нажмите клавишу <0>.

И наконец, чтобы закончить работу с системой info, нажмите клавишу <Q>, и вы вернетесь в оболочку.

Определение путей к исполняемым, исходным файлам и страницам справочного руководства

whereis

Команда whereis чрезвычайно полезна: она сообщает путь к исполняемому файлу программы, ее исходным файлам (если они существуют) и соответствующим страницам справочного руководства. Например, для kword (текстового процессора из набора Koffice) можно получить следующую информацию (она будет предоставлена при условии, что двоичные, исходные и справочные файлы были установлены):

```
$ whereis kword
kword: /usr/src/koffice-1.4.1/kword /usr/bin/kword
↳/usr/bin/X11/kword usr/share/man/man1/kword.1.gz
```

Сначала команда whereis сообщает расположение исходных файлов, /usr/src/koffice-1.4.1/kword, затем информирует о местонахождении исполняемых файлов: /usr/bin/kword и /usr/bin/X11/kword. Как видите, программа kword обнаружена в двух позициях файловой системы. Это несколько необычно, но, тем не менее, в этом нет ничего странного. И наконец, вы получаете информацию о том, где находятся страницы справочного руководства: /usr/share/man/man1/kword.1.gz. Теперь вы знаете, что программа действительно инсталлирована на вашем компьютере, и можете запустить ее.

Если вас интересуют только исполняемые файлы, при вызове whereis укажите опцию -b.

```
$ whereis -b kword
kword: /usr/bin/kword /usr/bin/X11/kword
```

Опция -m нужна тем, кому необходимы лишь страницы справочного руководства.

```
$ whereis -m kword
kword: /usr/share/man/man1/kword.1.gz
```

И наконец, если вы хотите получать только сведения об исходных файлах, укажите при вызове whereis опцию -s.

```
$ whereis -s kword
kword: /usr/src/koffice-1.4.1/kword
```

Команда whereis позволяет быстро получить чрезвычайно важную информацию о программах. Со временем вы начнете ее использовать гораздо чаще, чем предполагаете сейчас.

Сведения об экземпляре программы для запуска

which

Вспомните команду whereis и результаты ее применения к команде kword с опцией -b.

```
$ whereis -b kword
kword: /usr/bin/kword /usr/bin/X11/kword
```

В файловой системе есть два исполняемых файла `kword`. Но какой из них будет запущен при указании имени `kword` в командной строке? Выяснить это можно с помощью команды `which`.

```
$ which kword
/usr/bin/kword
```

Команда `which` сообщает о том, какой вариант команды будет выполнен, если вы зададите ее имя. Если вы введете слово `kword`, а затем нажмете клавишу <Enter>, ваша оболочка выполнит тот файл, который находится в каталоге `/usr/bin`. Если вы захотите выполнить тот файл, который расположен в каталоге `/usr/bin/X11`, вам надо сделать этот каталог текущим посредством команды `cd`, а затем ввести `./kword`. Можно также указать полный путь к файлу: `/usr/bin/X11/kword`.

Команда `which` также представляет собой быстрый способ выяснить, поддерживается ли та или другая команда в вашей системе. Если команда может быть выполнена и путь к соответствующему каталогу указан в переменной окружения `PATH`, вы получите информацию о том, где найти ее. В противном случае отобразится лишь приглашение для ввода очередной команды.

```
$ which arglebargle
$
```

Если вы хотите определить местоположение всех экземпляров файлов, поддерживающих команду (как в случае, если вы задаете выражение `whereis -b`), используйте опцию `-a` (первая буква слова “all”).

```
$ which -a kword
/usr/bin/kword
/usr/bin/X11/kword
```

Выяснение интерпретации команды

type

Команда type показывает, как оболочка bash будет интерпретировать команды, которые вы вводите. Рассмотрим несколько примеров.

Для начала посмотрим, что произойдет, если мы выполним команду type.

```
$ type ls
ls is aliased to '/bin/ls -F --color'
$ type cd
cd is a shell builtin
$ type whereis
whereis is /usr/bin/whereis
$ type mark
mark is a function
mark ()
{
  mkdir -p "$MARKPATH";
  ln -s "$(pwd)" "$MARKPATH/$1"
}
$ type do
do is a shell keyword
```

Вывод команды type показывает, как команда bash интерпретирует каждую команду. Есть пять возможных ответов.

- alias. Созданная вами мягкая ссылка, включая цель ссылки. К жестким ссылкам этот вариант не относится, потому что они вообще не выводятся на экран. (Детали см. в главе 3.)
- builtin. Команда встраивается в оболочку bash и выполняется ею непосредственно, в отличие от вызова внешней программы. Примерами встроенных команд являются alias, cd, echo, history, source и type.

- `file`. Команда, которая не встроена в оболочку, и поэтому называется внешней. Ее примерами являются команды `whereis` и `whatis` из этой главы.
- `function`. Функция (см. главу 12), созданная системой Linux, программой или вами. Как и команда `alias`, команда `type` выводит реальное содержание этой функции.
- `keyword`. Зарезервированное слово, которое используется только оболочкой `bash`, например `do`, `for`, `if`, `then`, `while` и `!`.

Если вы хотите лишь выяснить, к какой из этих пяти категорий относится ваша команда, используйте опцию `-t`.

```
$ type -t ls
alias
$ type -t cd
builtin
$ type -t mark
function
```

Коротко и точно! Опция `-a` приводит к более длинному листингу, потому что в нем указываются не только разновидность команды, но и все места, где ее можно найти.

```
$ type -a ls
ls is aliased to '/bin/ls -F --color'
ls is /bin/ls
```

Итак, теперь вам известно, что команда `ls` относится к категории `alias`, и вы знаете оригинал, к которому относятся псевдонимы, и путь к исходной команде.

Команда `type` не из тех команд, которые используются каждый день, но если вам интересно, почему та или иная команда не работает так, как вы ожидали, команда `type` поможет вам понять причину.

Выводы

Название данной главы говорит о том, что в ней будет продолжен разговор о командах. Теперь вы знаете, что существуют самые разнообразные способы получить дополнительную информацию о тех средствах, которые доступны из командной строки. Основными из них являются команды `man` и `info`. Они используют огромные объемы данных, описывающих практически все команды, доступные на компьютере под управлением Linux. Нельзя также забывать о командах `whereis`, `whatis`, `apropos` и `which`. Они очень полезны, особенно тогда, когда вам не хочется вникать в детали использования команд `man` и `info`. Но далеко не всегда они могут предоставить необходимые сведения. Время от времени приходится читать справочное руководство. Это так же, как и с овощами. Не все любят их, но они полезны.

Многие команды, описанные в данной главе, частично дублируют друг друга. Например, `man -k` — это то же самое, что `apropos`, `man -f` можно заменить `whatis`, а `whereis -b` функционально эквивалентна `which -a`. За вами выбор, какую из них использовать в каждой конкретной ситуации. Несмотря на то что разные команды выполняют одни и те же действия, желательно знать их все. Это поможет вам читать сценарии, написанные другими. Система Linux отличается тем, что предоставляет пользователю богатый выбор возможностей, даже тогда, когда это касается таких простых вещей, как команды, запускаемые посредством оболочки.

5

Объединение команд

В детстве мы все учили числа, а затем учились выполнять с ними арифметические действия, используя для записи символы +, -, ´ и =. На данный момент мы знаем несколько команд, но выполнять их умеем только по одной. На самом деле команды можно объединять в сложные и интересные конструкции посредством различных операций. Для обозначения этих операций используются символы |, >, >>, < и др. Мощь системы UNIX во многом объясняется возможностью объединять команды с помощью этих символов. В этой главе мы рассмотрим "строительные блоки", которые позволят гораздо эффективнее использовать как знакомые вам команды, так и те команды, которые будут подробно обсуждаться в последующих главах.

Последовательное выполнение нескольких команд

```
;
```

Предположим, вам надо выполнить одну за другой несколько команд, и среди них есть такие, которые выполняются достаточно долго. Значит ли это, что вам придется неотлучно сидеть возле компьютера? Допустим, у вас есть большое число записей Джона Колтрейна в формате mp3, которые представлены в виде zip-архива. Вы хотите распаковать их, поместить в новый подкаталог, а затем удалить архив. Пока что вы умеете лишь выполнять команды по одной, например:

```
$ ls -l /home/scott/music
-rw-r--r--  1437931  2015-11-07 17:19
```

```
↳JohnColtrane.zip
$ unzip /home/scott/music/JohnColtrane.zip
$ mkdir -p /home/scott/music/coltrane
$ mv /home/scott/music/JohnColtrane*.mp3
↳/home/scott/music/coltrane/
$ rm /home/scott/music/JohnColtrane.zip
```

Размер файла John_Coltrane.zip составляет 1,4 Гбайт. Даже на быстродействующей машине для его распаковки потребуется длительное время, в течение которого у вас наверняка найдется более интересное занятие, чем сидеть возле компьютера, ожидая завершения операции. Решение проблемы — в объединении команд!

Для объединения команд достаточно записать их все в одной строке, отделив друг от друга точкой с запятой. При этом команды будут выполняться последовательно одна за другой. Выполнение очередной команды начнется лишь после завершения (неважно, успешного или нет) предыдущей. Это нетрудно и действительно экономит время.

Объединенные команды записываются следующим образом:

```
$ ls -l /home/scott/music
-rw-r--r-- 1437931 2005-11-07 17:19
↳JohnColtrane.zip
$ unzip /home/scott/music/John_Coltrane.zip ;
↳mkdir -p /home/scott/music/coltrane ;
↳mv /home/scott/music/John_Coltrane*.mp3
↳/home/scott/music/coltrane/ ;
↳rm /home/scott/music/John_Coltrane.zip
```

Этот метод особенно полезен, когда каждый этап последовательности выполняется довольно долго. Если вы не хотите просто сидеть у компьютера, ожидая, когда он позволит вам ввести следующую команду, можете упорядочить команды с помощью точек с запятыми, а затем предоставить компьютеру выполнить их все сразу, автоматически переходя от одной команды к другой после завершения очередного этапа. За это время вы можете посмотреть фильм, пообедать или сделать какую-нибудь полезную работу.

Каждая команда выполняется, затем завершается, и управление передается следующей команде. Записать несколько команд подобным образом достаточно просто, и такой подход реально сэкономит вам время.

Перед выполнением команды можно задавать небольшую задержку. Если вы хотите сохранить копию экрана, вам поможет следующая объединенная команда (для ее выполнения надо, чтобы на вашем компьютере был установлен продукт ImageMagick; обычно он входит в состав дистрибутивного пакета Linux):

```
$ sleep 3 ; import -frame window.tif
```

В данном случае команда `sleep` реализует задержку на три секунды, затем посредством команды `import` обеспечивается копирование содержимого экрана. За эти три секунды вы можете переместить требуемое окно на передний план, минимизировать другие окна и выполнить прочие несложные действия. Символ `;` очень удобен для логического разделения команд, поэтому он используется достаточно часто.

ПРЕДУПРЕЖДЕНИЕ

Объединяя команды, будьте внимательны, особенно, если удаляете или перемещаете файлы. Убедитесь в том, что вы правильно ввели команды, в противном случае результаты могут быть непредсказуемыми.

Выполнение команды при условии успешного завершения предыдущих

`&&`

Из предыдущего раздела вы узнали, что символ `;` может служить разделителем при объединении команд, например:

```
$ unzip /home/scott/music/JohnColtrane.zip ;
↳mkdir -p /home/scott/music/coltrane ;
↳mv /home/scott/music/John_Coltrane*.*mp3
↳/home/scott/music/coltrane/ ;
↳rm /home/scott/music/John_Coltrane.zip
```

Предположим, вы допустили ошибку и ввели следующее:

```
$ unzip /home/scott/JohnColtrane.zip ;
↳mkdir -p /home/scott/music/coltrane ;
↳mv /home/scott/music/John_Coltrane*.*mp3
↳/home/scott/music/coltrane/ ;
↳rm /home/scott/music/John_Coltrane.zip
```

Вместо команды `unzip /home/scott/music/John_Coltrane.zip` была случайно задана команда `unzip /home/scott/John_Coltrane.zip`. Вы не заметили ошибки, нажали клавишу <Enter> и ожидаете окончания выполнения последовательности команд. Однако команда `unzip /home/scott/John_Coltrane.zip` не может быть выполнена, поскольку такого файла не существует. Тем не менее последовательность команд продолжается как ни в чем не бывало, и управление передается команде `mkdir`, которая завершается успешно. Третья команда (`mv`) также не может быть выполнена, так как файлы mp3, которые требуется переместить, не существуют; они не были созданы командой `unzip`. И наконец, выполняется четвертая команда, удаляющая zip-файл (заметьте, что на этот раз вы задали правильный путь). Теперь вам остается лишь восстановить файл из резервной копии (если она существует) и начать все сначала.

ЗАМЕЧАНИЕ

Вы не верите, что описанное возможно? Я сам совершил нечто подобное несколько дней назад. Ощущение было довольно скверным.

Источник проблемы — символ `;`, который вызывает последовательное выполнение команд без учета того, успешно

ли они завершаются. Гораздо лучшее решение — разделять команды символами &&. В этом случае они также выполняются последовательно, но следующая команда получает управление лишь в том случае, если предыдущая завершилась без ошибки (т.е. если она возвращает код завершения, равный 0). В случае ошибки выполнение всей цепочки команд прекращается. Если вы укажете вместо ; символы &&, выражение в командной строке примет следующий вид:

```
$ unzip /home/scott/JohnColtrane.zip &&
↳mkdir -p /home/scott/music/coltrane &&
↳mv /home/scott/music/JohnColtrane.*mp3
↳/home/scott/music/coltrane/ &&
↳rm /home/scott/music/JohnColtrane.zip
```

Поскольку первая команда unzip не может успешно завершиться, прекращается весь процесс. Когда вы обнаружите это, файл John_Coltrane.zip будет по-прежнему находиться на диске и вы сможете начать все сначала. Это гораздо лучше, чем остаться без архивного файла, не так ли?

Рассмотрим два примера, демонстрирующих преимущества операции &&. В главе 14 вы ознакомитесь с инструментом apt, существенно упрощающим обновление версии Debian системы Linux. При использовании apt сначала обновляется список доступного программного обеспечения, а затем выясняется, есть ли соответствующие дополнения. Если список программ не обновлен, вам не надо даже искать дополнения. Для того чтобы без необходимости не выполнять второй процесс (поиск дополнений), надо разделять команды посредством символов &&.

```
# apt-get update && apt-get upgrade
```

Второй пример выглядит так. Предположим, вы хотите преобразовать файл PostScript в файл PDF, используя команду ps2pdf, затем вывести PDF-файл на печать и удалить файл PostScript. Здесь также пригодится разделитель &&.

```
$ ps2pdf foobar.ps && lpr foobar.pdf && rm foobar.ps
```

Если вместо `&&` вы укажете `;`, а `ps2pdf` не выполнит свою задачу, то файл PostScript будет удален и вы даже не сможете повторить попытку из-за отсутствия исходных данных.

Вы убедились в том, что в ряде случаев предпочтительнее использовать разделитель `&&`? Если нет опасности удалить файл, вполне может подойти разделитель `;`, но если в наборе команд присутствует `rm` или другая команда, подобная ей, то лучше использовать для разделения символы `&&`.

Выполнение команды при условии, что предыдущая завершилась с ошибкой

||

Разделитель `&&` указывает на то, что очередная команда должна быть выполнена только при условии успешного завершения предыдущей. Символы `||` действуют противоположным образом. Если первая команда завершается с ошибкой (т.е. если она возвращает состояние завершения, отличное от 0), то только в этом случае управление передается следующей команде. Подобную конструкцию можно рассматривать как выражение типа "или, или", т.е. выполняется либо первая команда, либо вторая.

Разделители `||` часто используются для оповещения администратора об остановке процесса. Например, чтобы убедиться, что определенный компьютер работает нормально, его надо постоянно опрашивать посредством команды `ping` (подробнее эта команда будет рассматриваться в главе 15). Если `ping` сообщит об ошибке, администратору будет отправлено почтовое сообщение.

```
ping -c 1 -w 15 -n 72.14.203.104 ||
{
    echo "Server down"  | mail -s 'Server down'
↳admin@google.com
}
```

ЗАМЕЧАНИЕ

Вас заинтересовало, что собой представляет команда `|`? Она будет описана в этой главе чуть позже.

Немного поразмыслив, вы найдете множество применений разделителя `||`. Он давно взят на вооружение многими администраторами.

Использование выходных данных одной команды при вызове другой

`$()`

Механизм подстановки команд позволяет включить выходные данные одной команды в другую команду. При этом вторая команда будет выполняться так, как будто вы ввели все символы вручную. Первую команду — ту, выходные данные которой надо включить во вторую команду, — следует поместить в круглые скобки и поставить перед открывающей скобкой символ `$`. Использование подстановки команд пояснит приведенный ниже пример.

Предположим, вы пришли домой с банкета, подключили цифровую камеру к компьютеру, чтобы извлечь из нее новые фотоснимки и поместить их в каталог, имя которого соответствует текущей дате.

```
$ pwd
/home/scott/photos/family
$ ls -1F
2015-11-01/
2015-11-09/
2015-11-15/
$ date "+%Y-%m-%d"
2015-11-24
$ mkdir $(date "+%Y-%m-%d")
$ ls -1F
```

```
2015-11-01/
2015-11-09/
2015-11-15/
2015-11-24/
```

В данном примере первой выполнится команда `date "+%Y-%m-%d"`, а затем ее выходные данные (2015-11-24) будут использованы командой `mkdir` для формирования имени нового каталога. Подстановка команд — очень мощный механизм, и если вы просмотрите сценарии, написанные другими специалистами (их можно легко найти в веб), то обнаружите, что подстановка используется в них очень часто.

ЗАМЕЧАНИЕ

Раньше для организации подстановки применялся символ ` (“обратная галочка” — клавиша в верхней левой части клавиатуры). Теперь же рекомендуется применять более привычные символы — `$()`. Эти символы допускают вложение, т.е. символы `$()` можно использовать внутри других символов `$()`. В то же время символ `, вложенный внутрь других символов `, было бы очень трудно использовать. В конце концов, символы `$()` просто легче увидеть.

Входной и выходной потоки

Для того чтобы понять материал, изложенный в данной главе, необходимо знать, что оболочка Linux поддерживает три потока: *стандартный входной поток* (или стандартный ввод), *стандартный выходной поток* (или стандартный вывод) и *стандартный поток ошибок*. Каждому из этих потоков поставлен в соответствие дескриптор файла (числовой идентификатор) и общепринятое сокращение, кроме того, тот или иной поток может использоваться по умолчанию.

Например, если вы вводите информацию с клавиатуры, вы передаете данные в стандартный входной поток, которому соответствует дескриптор 0 и сокращение `stdin`. Когда ваш компьютер выводит данные на терминал, он использует стандартный выходной поток, которому соответствует дескриптор 1 и сокращение `stdout`. И наконец, если система должна сообщить об ошибке, используется стандартный поток ошибок, которому соответствует дескриптор 2 и сокращение `stderr`.

Обсудим работу с потоками на примере известной вам команды `ls`. Когда вы вводите информацию с клавиатуры, вы используете `stdin`. После того как вы ввели `ls` и нажали клавишу <Enter>, список файлов и каталогов выводится в `stdout`. Если вы зададите в качестве параметра данной команды несуществующий каталог, сообщение об ошибке будет выведено на ваш монитор посредством `stderr`. В табл. 5.1 поясняется использование потоков.

Таблица 5.1. Потоки ввода-вывода

Дескриптор файла (идентификатор)	Название	Общепринятое сокращение	Использование по умолчанию
0	Стандартный входной поток	`stdin`	Клавиатура
1	Стандартный выходной поток	`stdout`	Терминал
2	Стандартный поток ошибок	`stderr`	Терминал

В этой главе вы научитесь перенаправлять ввод и вывод. Вместо того чтобы выводить выходные данные на терминал, вы сможете, например, передать их другой программе. Точно так же, вместо того, чтобы вводить данные с клавиатуры, можно организовать получение их из файла. После того как вы поймете принципы использования `stdin` и `stdout`, вы сможете проделывать с ними поистине головокружительные трюки.

Передача выходных данных одной команды на вход другой

```
|
```

Всем известно, что система Unix собрана из отдельных частей, слабо зависящих друг от друга. Ничто не иллюстрирует этот принцип так ярко, как механизм каналов. Канал задается символом `|`, который помещается между двумя командами. Канал организует передачу выходных данных первой команды на вход второй. Иными словами, `|` перенаправляет `stdout` так, что он соединяется с потоком `stdin` следующей команды.

Ниже приведен простой пример, позволяющий лучше понять механизм каналов. Вы уже знаете команду `ls`, а в главе 5 ознакомитесь с командой `less`, позволяющей постранично просматривать текст на экране. Если вы примените команду `ls` к каталогу, содержащему множество файлов, например `/usr/bin`, то выходные данные быстро промелькнут на экране и их невозможно будет прочитать. Если же посредством канала вы соедините выход `ls` с входом `less`, то информация будет отображаться постранично.

```
$ pwd
/usr/bin
$ ls -1
zipinfo
zipnote
zipsplit
zsoelim
zxpdf
[Листинг сокращен из-за большой длины - 2318 строк!]
$ ls -1 | less
7z
a2p
acidrip
aconnect
```

Связывание команды `ls -l` с командой `less` посредством канала упростит вашу работу.

Рассмотрим более сложный пример; в нем использованы две команды, `ps` и `grep`, которые будут обсуждаться позже (прочитав главу 10, вы узнаете, что команда `ps` предоставляет информацию о выполняющихся процессах, а в главе 13 будет сказано, что `grep` предназначена для поиска строк в файлах). Предположим, программа Firefox начала вести себя странным образом, и вы подозреваете, что это происходит потому, что в фоновом режиме выполняется большое число ее экземпляров. Команда `ps` сообщает о каждом процессе на вашем компьютере, но в большинстве случаев объем выходных данных очень большой. Связав выход `ps` с входом `grep` и выполнив поиск по слову `firefox`, вы сможете немедленно выяснить, выполняется ли данная программа и сколько ее экземпляров имеется в системе.

```
$ ps ux
1504  0.8  4.4  75164 46124 ? S Nov20 1:19 kontact
19003  0.0  0.1   3376  1812 pts/4 S+ 00:02 0:00 ssh
↳admin@david.hartley.com
21176  0.0  0.0      0     0 ? Z 00:14 0:00
↳[wine-preloader] <defunct>
24953  0.4  3.3  51856 34140 ? S 00:33 0:08
↳kdeinit: kword
↳/home/scott/documents/clientele/current
[Листинг сокращен из-за недостатка места]
$ ps ux | grep firefox
scott  8272  4.7 10.9 184072 112704 ? Sl Nov19 76:45
↳/opt/firefox/firefox-bin
```

Из 58 строк выходных данных осталась одна — именно в ней и содержится интересующая вас информация.

ЗАМЕЧАНИЕ

Учтите, что не все программы могут работать с каналами. Например, текстовый редактор `vim` (или `pico`, или `nano`, или `emacs`) забирает у оболочки управление так, что данные с клавиатуры непосредственно передаются

vim, а выходная информация отображается средствами самой программы. Поскольку vim полностью контролирует оболочку, вы не можете использовать каналы для перенаправления вывода. Со временем, поработав с оболочкой, вы научитесь распознавать программы, к которым не применим механизм каналов.

Перенаправление выходных данных в файл

```
>
```

В обычных условиях данные, сгенерированные в результате выполнения команды, выводятся на экран или в поток stdout. Если вы хотите, чтобы информация выводилась не на экран, а в файл, вы можете сделать это, указав символ >.

```
$ pwd
/home/scott/music
$ ls -1F
Hank_Mobley/
Horace_Silver/
John_Coltrane/
$ ls -1F Hank_Mobley/* > hank_mobley.txt
$ cat hank_mobley.txt
1958_Peckin' _Time/
1960_Roll_Call/
1960_Soul_Station/
1961_Workout/
1963_No_Room_For_Squares/
$ ls -1F
Hank_Mobley/
hank_mobley.txt
Horace_Silver/
John_Coltrane/
```

Заметьте, что когда вы вызвали команду ls -F в первый раз, файл hank_mobley.txt отсутствовал. Он был создан, когда вы

использовали символ > для перенаправления вывода. Поступая подобным образом, необходимо соблюдать осторожность: если бы файл hank_mobley.txt существовал ранее, он был бы полностью заменен.

ПРЕДУПРЕЖДЕНИЕ

И еще раз напоминаю: соблюдайте осторожность, используя перенаправление! Вы можете затереть файл с важными данными.

Как предотвратить перезапись файла при перенаправлении

Существует способ предотвратить запись информации в файл поверх существующей при перенаправлении вывода. Если вы установите опцию noclobber, оболочка bash выполнит перенаправление в существующий файл только с вашего разрешения. Чтобы установить noclobber, выполните следующую команду:

```
$ set -o noclobber
```

Теперь, если вы захотите перезаписать файл путем перенаправления вывода, вместо > укажите >|, как показано в следующем примере:

```
$ pwd
/home/scott/music
$ ls -1F
Hank_Mobley/
hank_mobley.txt
Horace_Silver/
John_Coltrane/
$ ls -1F Hank_Mobley/* > hank_mobley.txt
ERROR
$ ls -1F Hank_Mobley/* >| hank_mobley.txt
$ cat hank_mobley.txt
```

```
1958_Peckin' _Time/
1960_Roll_Call/
1960_Soul_Station/
1961_Workout/
1963_No_Room_For_Squares/
```

Сбросить `noclobber` можно таким образом:

```
$ set +o noclobber
```

(Да, вы не ошиблись: опция `-o` означает "включено", а `+o` — "выключено". Не волнуйтесь — вы не сошли с ума. Просто у оболочки `bash` есть такая странность.)

Чтобы опция `noclobber` была установлена постоянно, включите в файл `.bashrc` выражение `set -o noclobber`.

Перенаправление выходных данных и запись их в конец файла

```
>>
```

Как вы уже знаете, символ > задает перенаправление выходных данных из `stdout` в файл. Например, можно легко перенаправить в файл вывод команды, возвращающей информацию о дате.

```
$ date
Mon Nov 21 21:33:58 CST 2005
$ date > hank_mobley.txt
$ cat hank_mobley.txt
Mon Nov 21 21:33:58 CST 2005
```

Однако нельзя забывать, что если указанный файл отсутствует, то он будет создан при выводе перенаправленной информации; если же файл существует, его содержимое будет полностью заменено. Указав >> вместо >, вы присоедините выходные данные в конец имеющегося файла (если файл отсутствует, он, как и ранее, будет создан).

```
$ cat hank_mobley.txt
Mon Nov 21 21:33:58 CST 2005
$ ls -1F Hank_Mobley/* >> hank_mobley.txt
$ cat hank_mobley.txt
Mon Nov 21 21:33:58 CST 2005
1958_Peckin' _Time/
1960_Roll_Call/
1960_Soul_Station/
1961_Workout/
1963_No_Room_For_Squares/
```

ПРЕДУПРЕЖДЕНИЕ

Перенаправляя ввод, удостоверьтесь, что указали именно символы >>. Если вы случайно зададите >, вместо записи в конец файла будет полностью заменено его содержимое.

Использование содержимого файла в качестве входных данных

<

В обычных условиях выходные данные задаются с клавиатуры, т.е. читаются из `stdin`. Подобно тому как `stdout` можно перенаправить в файл, вы можете перенаправить `stdin` так, что входные данные будут поступать не с клавиатуры, а из файла. Зачем это нужно? Некоторые программы не могут самостоятельно открывать файлы. В этом случае решить проблему позволяет символ <, перенаправляющий ввод. Часто это бывает необходимо для оформления сценариев, но это тема совсем другой книги.

Например, команда `tr` (которая будет рассмотрена в главе 7) не принимает файлы на вход; она считывает данные только из потока `stdin`. А что делать, если у вас есть файл и вы

хотите использовать команду tr? Есть хорошие новости — вы можете просто перенаправить поток stdin.

Допустим, мой редактор прислал мне текстовый файл, содержащий список команд, который ему хотелось бы описать в этой книге. Я открываю его и, к своему ужасу, вижу, что он полностью набран в верхнем регистре! Попробуем исправить это положение, вежливо попросив команду tr & прочитать перенаправленный поток stdin.

```
$ cat commands.txt
CP
LS
MKDIR
RM
TR
$ tr 'A-Z' 'a-z' < commands.txt
cp
ls
mkdir
rm
tr
```

Для ввода используется файл commands.txt, но команда tr прекрасно справилась со своей работой. Однако обратите внимание на то, что вывод команда tr направила в поток stdout, а не в файл. Этого и следовало ожидать, но в данном случае это не слишком полезно. Я действительно хотел ввести данные из файла и вывести их в файл. Как этого добиться, вы узнаете в следующем разделе.

Сочетание перенаправления ввода и вывода

```
[команда] < [файл] > [вывод]
```

В предыдущем разделе мы использовали команду tr для того, чтобы изменить регистр, в котором набрано содержание

файла. Впрочем, по умолчанию команда `tr` выводит результаты в поток `stdout`. Что делать, если необходимо записать результаты в новом файле с правильным регистром? Для этого выполните следующую команду:

```
tr 'A-Z' 'a-z' < commands.txt > commands_lower.txt
```

Посмотрим, как это работает, используя очень короткий файл.

```
$ ls
commands.txt
$ cat commands.txt
CP
LS
MKDIR
RM
TR
$ tr 'A-Z' 'a-z' < commands.txt > commands_lower.txt
$ ls
commands_lower.txt commands.txt
$ cat commands_lower.txt
cp
ls
mkdir
rm
tr
```

Конечно, можно просто применить команду `mv`, чтобы присвоить новому файлу имя старого. Если вы на самом деле умный программист, то используете команду `&&` для сцепления команд `mv` и `tr`. Однако, перед тем, как удалить старый файл, сначала убедитесь, что новый файл содержит правильные данные!

Теперь я должен позвонить редактору. Команды набраны в верхнем регистре! В UNIX?! Вот те на!

ЗАМЕЧАНИЕ

Вам могло показаться, что следовало использовать команду

```
tr 'A-Z' 'a-z' < commands.txt > commands.txt
```

Увы, из-за того, как оболочка bash выполняет перенаправление, такая команда создала совершенно пустой файл (подробное объяснение причин приведено в статье "Illustrated Redirection Tutorial" на веб-сайте http://wiki.bash-hackers.org/howto/redirection_tutorial). Вот почему вывод необходимо направлять в другой файл.

Одновременный вывод данных в файл и поток stdout

tee

В этом разделе мы рассмотрели все способы вывода данных в стандартный поток. В предыдущих главах мы видели много разных способов записи результатов выполнения в файлы (в следующих главах их будет еще больше). Но что делать, если мы хотим записать данные в поток stdout *и* файл одновременно? Неужели то возможно?

Да! Это команда tee!

Команда tee разделяет вывод на два потока: один направляется в поток stdout, а другой — в файл (этим объясняется имя команды (tee — тройник): вывод разделяется на два направления, как в водопроводной Т-образной трубе; см. описание в статье http://en.wikipedia.org/wiki/Piping_and_plumbing_fittings#Tee)

Рассмотрим простой пример. Допустим, мы хотим увидеть содержимое папки *и* записать его в файл.

```
$ ls -1 ~/music/Hank_Mobley/ | tee hank_mobley.txt
1958_Peckin'_Time
1960_Roll_Call
1960_Soul_Station
1961_Workout
1963_No_Room_For_Squares
$ ls
```

```
hank_mobley.txt
$ cat hank_mobley.txt
1958_Peckin'_Time
1960_Roll_Call
1960_Soul_Station
1961_Workout
1963_No_Room_For_Squares
```

Результаты работы команды `ls` выводятся в поток `stdout`, в то же время команда `tee` гарантирует, что эти же результаты будут записаны в файл.

Учтите, что если файл, в который мы хотим записать результаты, уже существует, то он будет перезаписан и будет содержать новые данные, предоставленные командой `tee`. Если вы хотите не перезаписывать, а дополнять данные в файле, используйте опцию `-a`.

```
ls -1 ~/music/Hank_Mobley/ | tee -a hank_mobley.txt
```

Результаты выполнения команды `tee` можно передать другой команде! В главе 7 будет рассмотрена команда `sort`; опция `-r` изменяет порядок следования упорядоченных результатов на противоположный.

```
$ ls -1 ~/music/Hank_Mobley/ | tee hank_mobley.txt |
↳sort -r > hank_mobley_reverse.txt
1958_Peckin'_Time
1960_Roll_Call
1960_Soul_Station
1961_Workout
1963_No_Room_For_Squares
$ ls
hank_mobley_reverse.txt hank_mobley.txt
$ cat hank_mobley_reverse.txt
1963_No_Room_For_Squares
1961_Workout
1960_Soul_Station
1960_Roll_Call
1958_Peckin'_Time
```

Возможности просто бесконечны!

Выводы

В начале этой книги мы рассмотрели несколько простых команд, а сейчас вы ознакомились со "строительными блоками", которые позволяют объединять эти команды. Материал последующих глав будет более сложным; в них вы узнаете об обширных возможностях, которые предоставляет Linux. Однако воспользоваться ими можно, лишь владея средствами, рассмотренными в данной главе. Итак, вперед!

6

Просмотр содержимого файлов (как правило, текстовых)

Одна из замечательных особенностей системы Linux состоит в том, что практически все конфигурационные файлы, файлы протоколов и файлы с информацией о системе представлены в текстовом формате. В настоящее время эти текстовые файлы практически всегда используют всеобъемлющую UNIX-совместимую кодировку UTF-8, вытеснившую древний (по компьютерным меркам 45 лет — это античная старина!) стандарт ASCII. Поскольку текстовые файлы лежали в основе системы UNIX с самого начала ее появления, существует множество программных команд, которые можно использовать для просмотра содержимого текстовых файлов. В этой главе мы рассмотрим команды, наиболее часто применяемые для чтения текстовых и других файлов.

Распознавание типа файла

```
file
```

Обычно тип файла совершенно очевиден. Если он имеет расширение `txt` — это текстовый файл; `jpg` — графический; `html`— веб-страница. Однако ситуации не всегда бывают такими ясными, как ожидалось. В системе UNIX файлы не обязательно должны иметь суффиксы имен (расширения); суффиксы — это всего лишь рудименты других операционных систем. Что будет, если у файла нет расширения (эта ситуация встречается чаще, чем хотелось бы)? Что, если расширение файла ни о чем вам не говорит? Во всех этих ситуациях было

бы желательно знать, с файлом какого вида вы работаете. Для этого используется команда `file`. Она выполняет разнообразные проверки указанных файлов, чтобы выяснить, какой формат они имеют (эти проверки описаны на справочной странице `man file`). Затем команда выводит в поток `stdout` результат, который она считает наиболее правдоподобным (и он почти всегда является правильным). Рассмотрим пример коллекции разнообразных файлов, которую, я надеюсь, вы никогда не запишете в один и тот же каталог!

```
$ ls -lF
838005x.docx
event.h
Lovecraft/
mtr*
Outline.md
Paper.doc
test
test.sh*
tix.jpg
www@
Yeats.txt
$ file *
838005x.docx: Zip archive data, at least v2.0 to
↳extract
event.h:      ASCII C program text
Lovecraft:    directory
mtr:          ELF 64-bit LSB executable, x86-64,
↳version 1 (SYSV), dynamically linked (uses shared
↳libs), for GNU/Linux 2.6.8, stripped
Outline.md:   UTF-8 Unicode English text
Paper.doc:    CDF V2 Document, Little Endian, Os:
↳Windows, Version 5.1, Code page: 1252, Author:
↳JohnDoe, Template: Normal, Last Saved By: John,
↳Revision Number: 14, Name of Creating Application:
↳Microsoft Office Word, Total Editing Time: 30:00,
↳Create Time/Date: Mon Mar 26 11:35:00 2012, Last
↳Saved Time/Date: Tue Mar 27 11:54:00 2012, Number
↳of Pages: 9, Number of Words: 2101, Number of
↳Characters: 11978, Security: 0
test:         HTML document text
test.sh:      Bourne-Again shell script text
```

```
↳executable
tix.jpg:        JPEG image data, JFIF standard 1.01
www:            symbolic link to '/var/www'
Yeats.txt:      ASCII English text
```

Эти результаты имеют несколько интересных особенностей.

Во-первых, рассмотрим несколько текстовых файлов, созданных простым текстовым редактором: `event.h`, `Outline.md`, `test`, `test.sh` и `Yeats.txt`. Обратите внимание на то, что команда `file` различает разные *виды* текстовых файлов и даже может описать природу более специализированных файлов: `event.h` — заголовочный файл на языке C, `test` — HTML-файл (хотя в этом случае расширение файла не указано; в принципе, расширение файла указывать не обязательно, команда `file` и без этого способна распознавать виды файлов), а файл `test.sh` — сценарий оболочки `bash`.

ЗАМЕЧАНИЕ

Я ожидал, что файл `Outline.md` будет идентифицирован как файл на языке Markdown, но, увы, не получилось. Впрочем, файлы на языке Markdown — это обычные текстовые файлы, написанные на языке Markdown, поэтому никакой ошибки в этом нет, просто результат получился не настолько точный, как я хотел. Обзор языка Markdown, достойный самой высокой похвалы, можно найти в Википедии (`http://en.wikipedia.org/wiki/Markdown`); на сайте универсального конвертера текстовых файлов Pandoc, который понимает язык Markdown (`http://johnmacfarlane.net/pandoc/index.html`), а также в моем блоге (`www.chainsawonatireswing.com/?s=markdown`).

Во-вторых, исполняемые файлы системы Linux идентифицированы правильно, независимо от того, были ли они скомпилированы (`mtr`) или представляли собой сценарии оболочки (`test.sh`). Команда `file` способна распознавать сценарии

оболочки bash, а также сценарии, написанные на Perl, Python и многих других языках.

В-третьих, команда file легко распознает каталоги и символические ссылки (а также специальные файлы других видов, которые можно увидеть, выполнив команду ls -F). Графические файлы тоже представлены с указанием дополнительной информации, например, версии графического стандарта (в случае формата GIF, например, команда выдает что-то вроде GIF image data, version 89a, 500 x 500).

И наконец, файлы 838005x.docx и Paper.doc были созданы текстовым процессором Word (я знаю, знаю), но команда file выдала для них разные результаты! О файле Paper.doc, который студент сохранил в старом формате Word и прислал мне, команда file выдает чрезмерно детализированную информацию, которая включает автора, версию Windows, шаблон, количество слов и даты. Файл 838005x.docx был сохранен в новом формате Word, который по существу представляет собой архивированный набор XML-документов. Если вы укажете опцию -z в команде file, то она попытается заглянуть в архив, чтобы выяснить его содержимое.

```
$ file -z 838005x.docx
838005x.docx: XML document text (Zip archive data,
↳at least v2.0 to extract)
```

Вам еще не понятно, *что* за файл перед вами? Тогда примените к нему команду file!

Вывод содержимого файла в stdout

cat

Пользователям системы DOS доступна команда type, отображающая содержимое текстового файла на экране. Пользователи Linux могут применять для той же цели команду cat.

```
$ cat Yeats_-_When_You_Are_Old.txt
WHEN you are old and grey and full of sleep,
And nodding by the fire, take down this book,
And slowly read, and dream of the soft look
Your eyes had once, and of their shadows deep;

How many loved your moments of glad grace,
And loved your beauty with love false or true,
But one man loved the pilgrim soul in you,
And loved the sorrows of your changing face;

And bending down beside the glowing bars,
Murmur, a little sadly, how Love fled
And paced upon the mountains overhead
And hid his face amid a crowd of stars.
$
```

Команда `cat` выводит файл на экран, а затем возвращает управление оболочке. Если содержимое файла не помещается на экране, воспользуйтесь средствами прокрутки.

При использовании команды `cat` может возникать следующая проблема: если объем документа, предназначенного для просмотра, слишком велик, текст быстро промелькнет на экране, и прочитать его будет невозможно (попробуйте для примера выполнить команду `cat Melville_-_Moby_Dick.txt`; конечно, соответствующий файл должен быть в вашей файловой системе). Выход — использовать команду `less`, которая будет обсуждаться далее в этой главе. Она обеспечивает постраничный вывод текста.

Конкатенация файлов и вывод их в stdout

```
cat файл1 файл2
```

Имя команды `cat` представляет собой сокращение от слова "concatenate" (конкатенация), т.е. объединение файлов. Основное назначение данной команды — объединение нескольких файлов в один. Вывод на экран одного файла — лишь частный

случай ее использования. Предположим, у вас есть фрагмент сонета Шекспира и фрагмент текста Бредбери, которые вы хотите просмотреть одновременно.

```
$ cat With_rue.txt Oh_when_I_was_in_love.txt
WITH rue my heart is laden
  For golden friends I had,
For many a rose-lipt maiden
  And many a lightfoot lad.

By brooks too broad for leaping
  The lightfoot boys are laid;
The rose-lipt girls are sleeping
  In fields where roses fade.

OH, when I was in love with you
  Then I was clean and brave,
And miles around the wonder grew
  How well did I behave.

And now the fancy passes by
  And nothing will remain,
And miles around they'll say that I
  Am quite myself again.
```

Заметьте, что команда `cat` не включает между файлами ни строку дефисов, ни другой разделитель. Тексты, содержащиеся в файлах, следуют непосредственно один за другим. Если вы хотите, чтобы они были разделены, надо предусмотреть соответствующие данные в исходных файлах, например, включить в конец первого файла пустую строку.

Конкатенация файлов и запись результатов в другой файл

```
cat файл1 файл2 > файл3
```

В предыдущем примере мы объединяли два файла и отображали их на экране, т.е. выводили в `stdout`. Однако такое

решение не всегда устроит вас. В ряде случаев, обрабатывая командой cat два файла, надо сохранить результаты и в файле. Сделать это можно, перенаправив вывод в файл (см. главу 5).

```
$ ls
Oh_when_I_was_in_love.txt With_rue.txt
$ cat With_rue.txt Oh_when_I_was_in_love.txt >
↳Housman.txt
$ ls
Housman.txt Oh_when_I_was_in_love.txt With_rue.txt
```

Теперь мы можем выполнять с файлом Housman.txt. любые действия. Например, добавить в конец этого файла любой текст.

```
$ cat Loveliest_of_trees.txt >> Housman.txt
```

Заметьте, что на этот раз мы использовали символы >>. Это существенно, потому что, например, следующая команда не дала бы желаемых результатов:

```
$ cat Loveliest_of_trees.txt Housman.txt >
↳Housman.txt
```

Если вы попытаетесь выполнить конкатенацию содержимого файла и записать результаты в этот же файл, то получите сообщение об ошибке.

```
cat: Housman.txt: input file is output file
```

Конкатенация файлов и нумерация строк

cat -n

При работе с текстом, особенно с исходным кодом программы, удобно, чтобы строки были пронумерованы. Для того чтобы сгенерировать номера строк, надо использовать при вызове cat опцию -n (или --number).

```
$ cat -n With_rue.txt Oh_when_I_was_in_love.txt
     1   WITH rue my heart is laden
     2     For golden friends I had,
```

```
 3  For many a rose-lipt maiden
 4    And many a lightfoot lad.
 5
 6  By brooks too broad for leaping
 7    The lightfoot boys are laid;
 8  The rose-lipt girls are sleeping
 9    In fields where roses fade.
10
11  OH, when I was in love with you
12    Then I was clean and brave,
13  And miles around the wonder grew
14    How well did I behave.
15
16  And now the fancy passes by
17    And nothing will remain,
18  And miles around they'll say that I
19    Am quite myself again.
```

Номера строк могут оказаться невероятно полезными, а команда `cat` обеспечивает быстрый способ нумерации строк в файле.

ЗАМЕЧАНИЕ

Если вам нужны альтернативные функциональные возможности, попробуйте вместо `cat` использовать команду `tac`. Как вы, вероятно, заметили, имя `tac` — это `cat` наоборот. Команда `tac` объединяет файлы в обратном порядке. Круто!

Постраничный вывод текста

`less`

В ряде случаев команда `cat` очень полезна, но она совершенно не подходит для просмотра файлов большого объема, так как при этом по экрану пробегает сплошной поток символов, прочитать которые практически невозможно. Если вам надо

просмотреть длинный файл (в данном случае длинным можно считать файл, содержимое которого не помещается на экране), вместо команды cat лучше использовать команду less.

Команда less организует постраничный вывод текста. Подобные возможности обеспечивают команды more и pg. Сама команда less была реализована в 1985 году как расширение команды more.

С помощью less файл открывается предельно просто.

```
$ less Milton_-_Paradise_Lost.txt
```

Текст, выводимый командой less, занимает весь экран; для навигации используется клавиатура. При желании вы можете завершить выполнение less и вернуться к командной строке. Для управления отображением текста используются клавиши, описанные в табл. 6.1.

Таблица 6.1. Основные клавиши команды `less`

Основные клавиши	Выполняемые действия
<PageDn>, E или пробел	Перемещение вперед на одну страницу
<PageUp> или B	Перемещение назад на одну страницу
<Enter>, E, J или стрелка вниз	Перемещение вперед на одну строку
Y, K или стрелка вверх	Перемещение назад на одну строку
G или P	Перемещение вперед к концу файла
1G	Перемещение назад к началу файла
<Esc+)> или стрелка вправо	Прокрутка вправо
<Esc+(> или стрелка влево	Прокрутка влево
Q	Завершение работы команды less

Как видите, одни и те же действия можно выполнить различными способами. Чаще всего при работе с less используются клавиши, сдвигающие текст вперед на одну страницу и завершающие работу программы.

Для отображения информации о файле надо нажать клавишу <=>. В результате в нижней части экрана будет выведена информация, подобная представленной ниже.

```
Milton_-_Paradise_Lost.txt lines 7521-7560/10762 byte
166743
↳/237306 70% (press RETURN)
```

Для того чтобы вернуться в режим просмотра текста, надо нажать клавишу <Enter>.

Нумерацию строк можно реализовать не только средствами `cat`, но и с помощью команды `less`. Очевидно, что номера строк будут отображаться лишь во время работы с командой `less`. После нажатия клавиши <Q> номера исчезнут. Для того чтобы в начале каждой строки отображался ее номер, надо при вызове `less` указать опцию `-N` (или `--LINE-NUMBERS`).

```
$ less -N Milton-_-Paradise_Lost.txt
```

ПОДСКАЗКА

Если вы используете команду `ls --color` (см. главу 2) и объединяете ее в конвейер с командой `less`, то могут возникнуть проблемы. Вот что вы получите:

```
$ ls
Burroughs Howard Lovecraft
$ ls --color | less
ESC[0mESC[01;34mBurroughsESC[0m
ESC[01;34mHowardESC[0m
ESC[01;34mLovecraftESC[0m
```

Что это за мешанина символов? Дело в том, что для работы с цветами оболочка `bash` использует невидимые управляющие символы и, когда вы передаете результаты работы команды `ls` команде `less`, эти символы вносят большую путаницу. Для того чтобы исправить ситуацию, используйте опцию `-R` (или `--RAW-CONTROL-CHARS`) команды `less`:

```
$ ls --color | less -R
Burroughs
Howard
Lovecraft
```

Кстати, некоторые из вас, вероятно, подумали, что к вам это не относится, поскольку вы не используете опцию `ls --color`.

Я тоже ...честно. Но мои псевдонимы (см. главы 5 и 15) для команды `ls` содержат опцию `--color`, поэтому легко забыть, что на самом деле я *всегда* использую эту опцию! Если вы используете команду `ls --color` непосредственно или через псевдонимы, не забывайте добавлять опцию `-R` в команду `less` там, где это необходимо.

Поиск с помощью программы постраничного просмотра

Если вы просматриваете с помощью `less` большой файл, бывает трудно обнаружить нужный фрагмент текста. Предположим, например, что вас интересует, называл ли Джон Мильтон в "Потерянном рае" яблоком плод, ставший причиной изгнания Адама и Евы. Работая с программой `less`, введите символ `/` и укажите шаблон для поиска. При необходимости можете даже использовать регулярные выражения. Окончив ввод шаблона, нажмите клавишу <Enter>, и программа `less` представит вам первый фрагмент текста, соответствующий шаблону (если такой фрагмент существует). Если поиск закончится неудачей, вы получите следующее сообщение.

```
Pattern not found (press RETURN)
```

При необходимости вы можете повторить поиск как вперед, так и назад по тексту. В табл. 6.2 описаны средства поиска, поддерживаемые командой `less`.

Таблица 6.2. Команды поиска, предусмотренные в программе `less`

Команда	Действие
/шаблон	Поиск в прямом направлении (возможно использование регулярных выражений)
`n`	Повторный поиск в прямом направлении
`N`	Повторный поиск в обратном направлении

ЗАМЕЧАНИЕ

Кстати, Мильтон никогда явно не использовал слово "яблоко"; вместо этого он использовал исключительно слово "плод". Я это хорошо знаю, потому что несколько лет писал диссертацию об английской литературе XVII века. Я ее так и не закончил. Свидетельство этому — данная книга и то, что я никогда не посвящал ни одну книгу Джону Мильтону.

Редактирование файлов, отображаемых средствами постраничного просмотра

Программа `less` — это не редактор, а лишь средство просмотра, однако вы можете передать файл, отображаемый с помощью `less`, текстовому редактору, например `vim` или `nano`. Для редактирования надо нажать клавишу <V>. Попробуйте сделать это. Откройте файл с помощью программы `less` и нажмите клавишу <V>. Через одну-две секунды на экране отобразятся интерфейсные элементы текстового редактора. Внесите необходимые изменения, завершите работу редактора, и вы снова вернетесь в программу `less`, но в ней уже будет отображаться измененный файл.

Если вам не подходит редактор, вызываемый после нажатия клавиши <V>, замените его любым другим. Например, если вы хотите использовать `vim`, то перед вызовом `less` выполните следующую команду:

```
$ export EDITOR=vim
```

В течение сеанса работы данную команду достаточно вызвать один раз. После этого всякий раз, когда вы обратитесь к `less`, с ней будет связан редактор `vim`. Если вы начнете новый сеанс, вам снова придется устанавливать редактор. Для того чтобы не делать этого каждый раз вручную, включите в файл `.bashrc` следующую строку:

```
export EDITOR=vim
```

ЗАМЕЧАНИЕ

Я люблю редактор `vim`; его и его предшественника `vi` можно встретить в любой части системы UNIX everywhere. Так почему же я ничего о нем не написал? Потому что он заслуживает отдельной книги! Начинающим пользователям рекомендую прочитать книгу *Learning the vi and Vim Editors*, а затем заглянуть на сайт `http://zzapper.co.uk/vimtips.html` для дальнейшего совершенствования.

Просмотр первых десяти строк файла

head

Если вы хотите просмотреть лишь первые десять строк файла, вам не обязательно использовать команду `cat` или `less`. Достаточно вызвать команду `head`, которая выведет первые десять строк файла и завершит свою работу. Проверим это на примере Чосера.

```
$ head Canterbury_Tales.txt
Here bygynneth the Book of the Tales of Caunterbury

General Prologue

Whan that Aprill, with his shoures soote
The droghte of March hath perced to the roote
And bathed every veyne in swich licour,
Of which vertu engendred is the flour;
Whan Zephirus eek with his sweete breeth
Inspired hath in every holt and heeth
$
```

Команда `head` очень удобна в том случае, если вам надо быстро оценить начало файла, независимо от его размера. Практически мгновенно вы узнаете, тот ли текст содержится в файле, который вы ищете.

Просмотр первых десяти строк нескольких файлов

head файл1 файл2

Команду head можно использовать для просмотра начальных строк нескольких файлов. Полученные при этом данные похожи на те, которые предоставляются командой cat, за исключением того, что отображаемое содержимое файла ограничивается несколькими строками, а между файлами включаются разделители.

```
$ head Canterbury_Tales.txt Paradise_Lost.txt
==> Canterbury_Tales.txt <==
Here bygynneth the Book of the Tales of Caunterbury

General Prologue

Whan that Aprill, with his shoures soote
The droghte of March hath perced to the roote
And bathed every veyne in swich licour,
Of which vertu engendred is the flour;
Whan Zephirus eek with his sweete breeth
Inspired hath in every holt and heeth

==> Paradise_Lost.txt <==
Book I

Of Man's first disobedience, and the fruit
Of that forbidden tree whose mortal taste
Brought death into the World, and all our woe,
With loss of Eden, till one greater Man
Restore us, and regain the blissful seat,
Sing, Heavenly Muse, that, on the secret top
Of Oreb, or of Sinai, didst inspire
That shepherd who first taught the chosen seed,
```

В качестве разделителей используются пустые строки и специальные заголовки. Они позволяют легко отличить один файл от другого.

Просмотр произвольного числа строк из файлов

Если вас не устраивает тот факт, что из каждого файла извлекается ровно десять строк, укажите команде head отображать другой объем текста. Для этой цели используется опция -n, за которой следует число, например 5 (можно также задать опцию --lines=5). Если вы зададите два или более файла, из начала каждого файла будет выбрано указанное число строк.

```
$ head -n 5 Canterbury_Tales.txt Paradise_Lost.txt
==> Canterbury_Tales.txt <==
Here bygynneth the Book of the Tales of Caunterbury

General Prologue

Whan that Aprill, with his shoures soote
==> Paradise_Lost.txt <==
Book I

Of Man's first disobedience, and the fruit
Of that forbidden tree whose mortal taste
Brought death into the World, and all our woe,
```

Заметьте, что в число "пяти строк" входят пустые строки, имеющиеся в файле.

Можно задать опцию --lines=100, например, чтобы убедиться, что содержимое файла превышает 10 строк, количество которых задано по умолчанию.

Просмотр указанного количества байтов из начала файла

```
head -c
```

Опция -n позволяет задать число строк, просматриваемых из начала файла. Но что делать, если вам надо задать не

количество строк, а число байтов, или килобайтов, или даже мегабайтов? (Последнее в большинстве случаев *совершенно* бессмысленно, так как объем информации окажется слишком большим.) Для этой цели используется опция `-c` (или `--bytes=`).

Например, для того, чтобы просмотреть первые 100 байтов файла, содержащего Кеттерберийские рассказы Чосера, надо воспользоваться следующей командой:

```
$ head -c 100 Canterbury_Tales.txt
Here bygynneth the Book of the Tales of Caunterbury

General Prologue

Whan that Aprill, with his sh
```

Указанное число байтов совсем не обязательно должно приходиться на конец строки или пробел между словами. Так, в данном примере текст обрывается на середине слова.

Для того чтобы отобразить первые 100 Кбайт этого же файла, надо переписать команду таким образом:

```
$ head -c 100KB Canterbury_Tales.txt
Here bygynneth the Book of the Tales of Caunterbury

General Prologue

Whan that Aprill, with his shoures soote
The droghte of March hath perced to the roote
And bathed every veyne in swich licour,
Of which vertu engendred is the flour;
Whan Zephirus eek with his sweete breeth
Inspired hath in every holt and heeth
```

Аналогично можно задать отображение первых 100 мегабайтов, но объем всей книги значительно меньше. Формально соответствующая команда имеет следующий вид:

```
$ head -c 100MB Chaucer_-_Canterbury_Tales.txt
```

Заметьте, что мегабайт — это 1000000 байтов (1000 × 1000). Я так и слышу, как вы возмущаетесь: "Что?! Мегабайт не равен 1000 килобайтам! Он состоит из 1024 килобайтов!"

Я не хочу становиться в позу непререкаемого авторитета, но не считаю, что слово мегабайт означает то, что вы имеете в виду. Начиная с 2000 года используется новый набор префиксов (в Википедии есть хорошая статья на эту тему: `http://en.wikipedia.org/wiki/Mebibyte`). Старые и новые префиксы, а также их смысл и значения приведены в табл. 6.3.[1]

Таблица 6.3. Команды поиска, предусмотренные в программе `less`

Старый префикс	Значение	Смысл	Смысл	Значение	Новый префикс
Килобайт (Кбайт)	10^3	1000 байтов	1024 байтов	2^{10}	Кибибайт (КиБ)
Мегабайт (Мбайт)	10^6	1000 Кб	1024 МиБ	2^{20}	Мебибайт (МиБ)
Гигабайт (Гбайт)	10^9	1000 Мб	1024 ГиБ	2^{30}	Гебибайт (ГиБ)
Терабайт (Тбайт)	10^{12}	1000 Гб	1024 ТиБ	2^{40}	Тебибайт (ТиБ)
Петабайт (Пбайт)	10^{15}	1000 Тб	1024 ПиБ	2^{50}	Пебибайт (ПиБ)
Эксабайт (Эбайт)	10^{18}	1000 Пб	1024 ЭиБ		Эксбибайт (ЭиБ)

Очень многие из вас (и я в том числе!) думали, что мегабайт — это 1024 килобайтов. Оказалось, что эту величину следовало называть *мебибайтом*; он действительно состоит из 1024 *кибибайтов*. Теперь представьте себе, что я купил крутой жесткий диск емкостью три терабайта. Предполагается, что я должен сказать: "Полюбуйтесь на мой жесткий диск емкостью три *тебибайта*!" Люди подумают, что я просто заикаюсь от счастья. Кроме путаницы с новыми терминами, стоит упомянуть, что слово *терабайт* у меня всегда ассоциировалось с огромным свирепым динозавром. А слово *тебибайт* ассоциируется с чем-то миниатюрным и незаметным.

Теперь вы должны понимать разницу между значениями, которые используются в качестве опции команды `head -c`. Если вы хотите увидеть десятичные значения, используйте опции `kB`, `MB`, `GB` и т.д. Если же вы хотите увидеть двоичные значения 1024, используйте опции `K`, `M`, `G` и т.д. Не используйте

[1] См. также статью в русскоязычной Википедии `https://ru.wikipedia.org/wiki/Двоичные_приставки`. — *Примеч. ред.*

опции KiB, MiB, GiB, потому что они не работают. Возможно, это не совсем то, что вы хотели, но это лучше, чем ничего.

Просмотр последних десяти строк файла

tail

Команда head отображает начало файла. Нетрудно догадаться, что команда tail позволяет просмотреть конец файла.

```
$ tail Paradise_Lost.txt
To the subjected plain—then disappeared
They, looking back, all the eastern side beheld
Of Paradise, so late their happy seat,
Waved over by that flaming brand; the gate
With dreadful faces thronged and fiery arms.
Some natural tears they dropped, but wiped them
↳soon;
The world was all before them, where to choose
Their place of rest, and Providence their guide.
They, hand in hand, with wandering steps and slow,
Through Eden took their solitary way.
```

Зачем нужна эта команда? Чаще всего она применяется для просмотра окончания файла протокола. Именно там располагаются последние записи.

Просмотр последних десяти строк нескольких файлов

tail *файл1 файл2*

Если команда head позволяет просматривать первые строки нескольких файлов, логично ожидать, что команда tail также может обрабатывать больше одного файла.

```
$ tail Paradise_Lost.txt Miller's_Tale.txt
==> Paradise_Lost.txt <==
```

```
To the subjected plain—then disappeared
They, looking back, all the eastern side beheld
Of Paradise, so late their happy seat,
Waved over by that flaming brand; the gate
With dreadful faces thronged and fiery arms.
Some natural tears they dropped, but wiped them
↪soon;
The world was all before them, where to choose
Their place of rest, and Providence their guide.
They, hand in hand, with wandering steps and slow,
Through Eden took their solitary way.
==> Miller's_Tale.txt <==
With othes grete he was so sworn adoun
That he was holde wood in al the toun;
For every clerk anonright heeld with oother.
They seyde, "The man is wood, my leeve brother";
And every wight gan laughen at this stryf.
Thus swyved was this carpenteris wyf,
For al his kepyng and his jalousye;
And Absolon hath kist hir nether ye;
And Nicholas is scalded in the towte.
This tale is doon, and God save al the rowte!
```

Подобно `head`, команда `tail` включает между файлами разделители, упрощающие работу с информацией.

ПОДСКАЗКА

Очень хорошо, что утилита `tail` позволяет просматривать несколько файлов, но утилита `multitail` справляется с этой задачей еще лучше. Почему? Потому что, вместо того, чтобы конкатенировать результаты и спрессовывать их в один большой ком, как это делает утилита `tail`, утилита `multitail` показывает последние несколько строк каждого заданного файла одновременно в отдельных окнах на терминале. Эту утилиту можно загрузить из репозитория дистрибутивного пакета (см. главу 14) или с сайта `www.vanheusden.com/multitail/`. Более подробную информацию об этой замечательной

маленькой программе можно найти в статье Уильяма фон Хагена (William von Hagen) "Monitoring logs and command output" (`www.ibm.com/developerworks/aix/library/au-monitorlogs/`).

Просмотр произвольного числа последних строк из файлов

`tail -n`

Продолжая аналогии между `head` и `tail`, можно предположить наличие опции, посредством которой задается число строк, предназначенных для отображения. Действительно, такая опция существует. Это опция `-n` (или `--lines=`). По умолчанию отображается десять строк. Хотите увидеть окончания сразу нескольких файлов? Укажите их при вызове команды.

```
$ tail -n 4 Paradise_Lost.txt Miller's_Tale.txt
==> Paradise_Lost.txt <==
The world was all before them, where to choose
Their place of rest, and Providence their guide.
They, hand in hand, with wandering steps and slow,
Through Eden took their solitary way.
==> Miller's_Tale.txt <==
For al his kepyng and his jalousye;
And Absolon hath kist hir nether ye;
And Nicholas is scalded in the towte.
This tale is doon, and God save al the rowte!
```

Данный вариант команды `tail` удобен в тех случаях, когда надо просмотреть несколько файлов протоколов. Однако есть и лучшее решение. Вы узнаете о нем из следующего раздела.

Просмотр обновляемых строк в конце файла

```
tail -f
```

Файлы протоколов претерпевают постоянные изменения. Команда `tail` дает "мгновенный снимок" файла, а затем возвращает управление командной строке. А если вам надо еще раз обратиться к файлу протокола? Необходимо снова вызвать команду `tail`, затем еще и еще раз.

Если при вызове команды `tail` указать опцию `-f` (или `--follow`), программа не завершит работу. Она будет отображать последние десять строк (или другое количество, заданное посредством опции `-n`), причем изменения будут сразу же представлены на экране. Такой режим чрезвычайно удобен, если вам надо выявить причину некорректной работы программы или всей системы.

Например, файл протокола веб-сервера может выглядеть следующим образом:

```
$ tail -f /var/log/httpd/d20srd_org_log_20151201
"GET /srd/skills/bluff.htm HTTP/1.1"...
"GET /srd/skills/senseMotive.htm HTTP/1.1"...
"GET /srd/skills/concentration.htm HTTP/1.1"...
"GET /srd/classes/monk.htm HTTP/1.1"...
"GET /srd/skills/escapeArtist.htm HTTP/1.1"...
```

В книге трудно представить тот факт, что файл не закрывается. Программа `tail` продолжает работу с файлом и отражает изменения его содержимого. Очередные данные будут выводиться до тех пор, пока вы не нажмете комбинацию клавиш <Ctrl+C>, завершив тем самым работу программы.

Попробуйте применить данную команду к одному из ваших файлов протоколов, например `/var/log/syslog`. Задайте отображение требуемого количества строк, а затем попробуйте поработать с двумя файлами, например `/var/log/syslog` и

/var/log/daemon.log, и посмотрите, что произойдет. Результаты могут оказаться совсем неожиданными.

Есть еще одна интересная опция, которая называется --retry. Она очень полезна в ситуациях, в которых файл может исчезнуть или временно стать недоступным. Представьте себе, например, файл в каталоге /tmp или файл, который вы записали, а потом уничтожили в ходе выполнения сценария оболочки. Комбинация опций -f и --retry позволяет обнаружить практически любой файл, даже если он исчез, а потом появился!

ЗАМЕЧАНИЕ

Если вы знаете идентификатор процесса (PID — process identifier), создавшего файл, который вы отслеживали с помощью утилиты tail, то можно воспользоваться командой tail -f --pid=PID#. Это очень удобно, потому что, когда процесс с указанным идентификатором будет закончен, утилита tail прекратит слежение за ним. Например, допустим, что процесс apache имеет идентификатор 2112. Применим эту команду для наблюдения за файлом, который генерируется этим процессом.

```
$ tail -f --pid=2112 /var/log/apache2/error.log
```

Когда процесс apache будет завершен, утилита tail прекратит слежение за файлом error.log. Учтите, что для этого вы должны знать имя файла *и* идентификатор процесса, который его создает (для выяснения этой информации удобно пользоваться утилитой ps).

Выводы

В данной главе мы рассмотрели пять команд, file, cat, less, head и tail, которые обеспечивают просмотр текстовых файлов разными способами. Команда file выдает информацию о виде файла, с которым вы работаете, так что вы сможете выяснить, какими командами его можно обрабатывать

в дальнейшем. Остальные четыре команды выдают информацию в режиме "только для чтения", но делают это по-разному. Команда `cat` выводит сразу весь файл, команда `less` отображает информацию по страницам. Команды `head` и `tail` представляют "две стороны одной медали" или, вернее, одного файла. Команда `head` отображает его начало, а команда `tail` — последние строки. Совместно эти команды дают возможность просмотреть любые части текстового файла.

7

Обработка текстовых файлов с помощью фильтров

Все команды в этой главе представляют собой *фильтры*. Фильтры получают данные на вход, как правило, из стандартного потока ввода, обрабатывают их, а затем записывают результаты в стандартный поток вывода. В этом механизме часто используются каналы, помогающие перемещать ввод и вывод от фильтра к фильтру, образуя конвейер (стандартные потоки `stdin`, `stdout` и каналы описаны в главе 5).

В этой главе рассматриваются фильтры `wc`, `nl`, `cut`, `sort`, `uniq`, `tr`, `sed` и `awk`. Более полный список фильтров UNIX приведен в статье Википедии `https://en.wikipedia.org/wiki/Filter_(software)`. Большинство команд из этого списка обсуждаются в этой книге. Кроме указанных выше команд, в главе 5 приведено описание команды `tee`; в главе 6 — `cat`, `head`, `less`, `more`, `tac`, и `tail`, в главе 9 упоминается команда `compress`, а в главе 10 — команда `grep`. Читая эту главу и упоминания о фильтрах в других главах, вы обнаружите большое количество перекрестных связей между этими командами. Например, команда `sort` может делать то же самое, что и команда `uniq`, команды `sed` и `awk` могут делать многое, что можно сделать с помощью других команд, а утилита `tr` способна выполнить ту же работу, которую можно сделать с помощью команд `sed` и `awk`, но быстрее. И так далее.

Это нормально! Часто одно и то же задание можно выполнить разными способами, и в этом нет ничего плохого. Чем больше вы будете узнавать о фильтрах, тем точнее будете выбирать наилучший инструмент для выполнения своей работы. Обширные знания — это всегда хорошо, особенно если эти знания относятся к системе Linux и ее командам.

Подсчет количества слов, строк и символов в файле

wc

Команда `wc` — это динозавр, который появился еще в первой версии системы UNIX в 1971 г. Но она все еще используется; даже сегодня, перед тем, как приступить к работе над этой главой, я выполнил команду `wc`! Этой команде 44 года, и она все еще необходима и полезна. Это одна из превосходных особенностей системы UNIX (и Linux, разумеется)!

Что делает команда `wc`? Иногда мне нужно подсчитать количество слов в документе. Намного чаще мне требуется знать количество строк в документе, а еще чаще необходимо подсчитывать количество символов. К счастью, существует способ подсчитать все это с помощью одной команды: `wc`.

```
$ wc "The Call of Cthulhu.txt"
192 11863 70246 The Call of Cthulhu.txt
```

Здесь по порядку указано количество строк, слов и символов в известном коротком рассказе Г.Ф. Лавкрафта[1]. А что если мне нужно знать только количество слов?

```
$ wc -w "The Call of Cthulhu.txt"
11863 The Call of Cthulhu.txt
```

Эти опции легко запомнить: `-l` (или `--lines`) означает количество строк (понятное дело!); `-w` (или `--words`) — количество слов (само собой!); `-m` (или `--chars`) — количество символов (что?!). А почему не `-c`? Потому что опция `-c` (или `--bytes`) используется для *подсчета* байтов в документе. А! Понятно! Ничего сложного!

[1] Говард Филлипс Лавкрафт (H.P. Lowcraft) (1890–1937) — американский писатель и поэт, писавший в жанре ужасов и мистики. — *Примеч. ред.*

А если у меня два документа? Тогда ситуация становится еще более интересной!

```
$ wc "Beyond the Walls of Sleep.txt" "The Call of
↳Cthulhu.txt"
 62  4307 24953 Beyond the Walls of Sleep.txt
192 11863 70246 The Call of Cthulhu.txt
254 16170 95199 total
```

Итак, теперь мы знаем не только количество строк, слов и символов в указанных рассказах Лавкрафта, но и их сумму. Превосходно!

Это очень полезно, но можно подсчитывать строки, слова и символы не только в документе, указанном в командой строке, но и тексте, полученном от другой команды (это же фильтр!).

Например, недавно я пожелал выяснить, сколько раз поисковый робот JerkyJerks заходил на мой сервер, пока я его не заблокировал (я буду очень вежлив и не стану называть настоящее имя робота). Я применил утилиту `grep` (см. главу 10), чтобы обнаружить имя JerkyJerks в файле журнала, а затем использовал команду `wc -l` , чтобы выяснить, сколько раз этот робот регистрировался на моей машине.

```
$ grep JerkyJerks server_access.log | wc -l
15428
```

Возмутительно! Заблокировать!

Рассмотрим еще один пример, к которому мы еще вернемся в главе 13.

```
$ ps aux | grep [/]usr/bin/cronolog | wc -l
83
```

Я хочу знать, сколько экземпляров программы `cronolog` было запущено, поэтому сначала создаю список всех процессов с помощью команды `ps aux` (см. главу 13), а затем применяю к этому списку утилиту `grep`, чтобы получить путь к команде `cronolog`. И наконец, с помощью команды `wc -l` я выясняю, сколько раз программа `cronolog` была запущена на моем сервере: 83. Отлично.

ЗАМЕЧАНИЕ

Вам интересно, почему я указал первую косую черту в квадратных скобках: `[/]usr/bin/cronolog`? Прочитайте в главе 10 раздел о поиске конкретных слов в результатах других команд, и вы поймете почему!

Перейдем к заключительному примеру, из которого вы узнаете кое-что важное о команде `wc`. Когда команда `wc` подсчитывает строки, она учитывает и пустые строки, потому что на самом деле подсчитывает количество символов перехода на новую строку в документе.

Если вы не хотите учитывать пустые строки, используйте команду `sed` (см. ниже в этой главе), которая удаляет пустые строки из документа и передает результат команде `wc -l`.

```
$ sed '/^$/d' "The Call of Cthulhu.txt" | wc -l
103
```

Выражение `/^$/d'` в команде `sed` расшифровывается как "найти все строки, в которых между началом (символ `^`) и концом (символ `$`) ничего нет, и удалить их (символ `d`)". Первая команда подсчитывает количество строк в рассказе "The Call of Cthulhu" (192); а без учета пустых строк мы получаем 103. Большая разница!

ЗАМЕЧАНИЕ

Еще один пример конвейерной передачи результата команде `wc` приведен в главе 8.

Количество строк в файле

```
nl
```

Работая с исходным кодом, необходимо видеть номера строк, чтобы ориентироваться самому и ориентировать

других. Для этого практически каждый текстовый редактор выводит на экран номера строк. А что если вы захотите пронумеровать строки рассказа "The Call of Cthulhu", чтобы сфокусировать внимание многих любителей творчества Г.Ф. Лавкрафта на нескольких наиболее страшных местах? В этом случае вам поможет команда `nl` (в следующем примере я использую эллипсис ... там, где слишком длинные строки обрываются).

```
$ nl "The Call of Cthulhu.txt"
1  The Call of Cthulhu by H.P. Lovecraft

2  (Found Among the Papers of the Late Francis…

3  "Of such great powers or beings there may be…

4  —Algernon Blackwood.

5  I.

6  The Horror in Clay.
[Листинг (192 строки) сокращен для экономии места!]
```

Внимательно просматривая результаты, можно обнаружить, что пустые строки не пронумерованы. Это свойство команды `nl`, заданное по умолчанию. Оно эквивалентно использованию опции `-b t`. Опция `-b` сообщает команде `nl`, какие строки текста (body, потому и `-b`) вы хотите пронумеровать, но после нее сразу же необходимо указать тип строки: `t` (непустая) или `a` (все строки). Существуют и другие варианты, но они вряд ли вам понадобятся; если вас интересуют эти возможности, откройте соответствующую справочную страницу `man`.

Итак, если вы хотите пронумеровать *все* строки, а не только непустые, выполните следующую команду:

```
$ nl -b a "The Call of Cthulhu.txt"
1  The Call of Cthulhu by H.P. Lovecraft
2
3 (Found Among the Papers of the Late Francis…
4
5  "Of such great powers or beings there may be…
6  —Algernon Blackwood.
```

```
7
8  I.
9  The Horror in Clay.
```

Если бы я захотел сохранить результат, то использовал бы перенаправление (см. главу 5), чтобы создать новый файл.

```
$ nl "The Call of Cthulhu.txt" > "Cthulhu
↳numbered.txt"
```

Не знаю, как вы, а я считаю команду `nl -b a` одной из самых загадочных. Что еще хуже, она слишком замысловатая и ее трудно запомнить.

Выбор отдельного столбца данных в размеченном файле

`cut`

Это очень старая команда, которая относится к тем временам, когда файлы размечались с помощью символа табуляции. Она до сих пор используется, но в настоящее время файлы имеют намного более разнообразное содержание. С другой стороны, даже если данные в файле не разделены символом табуляции, мы всегда используем для этого какой-нибудь символ: например, запятые, точки с запятыми, двоеточия и точки. Команда `cut` позволяет выбрать конкретный столбец из размеченного файла и вывести его на печать.

Начнем с символа табуляции, поскольку именно он задан в команде `cut` по умолчанию. Допустим, у вас есть файл `cool_movies.txt`, содержащий следующий текст (не обращайте внимание на то, что столбцы не выровнены, потому что значение имеет лишь наличие символа табуляции между ними):

```
Movie  Genre  Hero  Year
Die Hard  Action John McClane   1988
Star Wars Sci-Fi Luke Skywalker 1977
John Wick Action John Wick   2014
```

```
Aliens  Sci-Fi  Ellen Ripley 1986
The  Thing Horror  MacReady  1982
```

Для того чтобы извлечь первый и третий столбцы, выполним следующую команду:

```
$ cut -f 1,3 cool_movies.txt
Movie  Hero
Die Hard   John McClane
Star Wars  Luke Skywalker
John Wick  John Wick
Aliens  Ellen Ripley
The Thing  MacReady
```

Опция `-f` (или `--fields`) сообщает команде `cut`, какие столбцы необходимо извлечь. В данном примере извлекаются первый и третий столбцы: `-f 1,3`. Если бы я захотел извлечь только третий столбец, то использовал бы опцию `-f 1-3`. Если бы я хотел извлечь все столбцы, кроме второго, то использовал бы опцию `-f 1,3-4`.

Здесь есть одна тонкость, которая может создать неприятности, если вы не знаете, чего хотите. Допустим, сервер имеет `wu.images.granneman.com` и я хочу извлечь имена доменов верхнего (`com`), второго (`granneman`) и третьего уровней (`images`), но не четвертого (`wu`). Для этого я использую команду

```
$ echo wu.images.granneman.com | cut -d '.' -f 2-4
images.granneman.com
```

Вы видели это? Я получил имя домена третьего, второго и верхнего уровней, но они разделены точкой (`.`), а не символом табуляции или другим разделителем. Почему?

Поскольку я указал в качестве разделителя точку (`.`), то и получу в ответ поля, разделенные точками.

А если я захочу использовать другой разделитель, например, запятую? В таком случае необходимо использовать опцию `--output-delimiter` и сообщить команде `cut` о своем желании.

```
$ cut -f 1,3 --output-delimiter=',' cool_movies.txt
Movie,Hero
Die Hard,John McClane
```

```
Star Wars,Luke Skywalker
John Wick,John Wick
Aliens,Ellen Ripley
The Thing,MacReady
```

В примере, связанном с файлом `cool_movies.txt`, в качестве разделителя использовался символ табуляции. А если используется другой разделитель? В нашем случае для указания разделителя мы использовали опцию `-d` (или `--delimiter`).

Рассмотрим простой пример. В системе Linux есть переменная окружения `$HOSTNAME`, содержащая имя узла (сюрприз!), что-то вроде `perseus.websanity.com`. Однако в некоторых сценариях я использую настоящее имя компьютера (в данном случае `perseus`), а не доменное имя (`websanity.com`). Части имени домена разделены точками, которые можно интерпретировать как разделители.

```
$ echo $HOSTNAME
perseus.websanity.com
$ echo $HOSTNAME | cut -d '.' -f 1
perseus
```

ЗАМЕЧАНИЕ

Используя выражения "настоящее имя компьютера" и "доменное имя", я фактически говорил на жаргоне, но это не важно. Аспекты именования DNS описаны, например, в статье `https://en.wikipedia.org/wiki/Domain_name`.

А как извлечь полностью определенное доменное имя (`www.granneman.com`), если бы я использовал унифицированный указатель ресурса URL вроде `http://www.granneman.com/writing/books/`? В данном случае я использовал косую черту (/) и команду `cut`.

```
$ echo http://www.granneman.com/writing/books/ |
↳cut -d '/' -f 3
www.granneman.com
```

С помощью опции `-d '/'` я указал команде `cut`, что разделителем является символ `/` и что я хочу извлечь третье поле `-f 3`. Однако, если взглянуть на URL, выделить *третье* поле, отделенное косой чертой `/`, довольно нелегко. Первое поле предшествует первой косой черте `/`, т.е. `http:`. Второе поле стоит перед второй косой чертой `/`, которая следует сразу за первой косой чертой (иначе говоря, `//`), так что по существу второе поле пусто. Третье поле находится перед третьей косой чертой `/`. В итоге получаем `www.granneman.com`. Работает!

Сортировка содержимого файла

`sort`

Команда `sort` сортирует содержимое файла. Допустим, файл `cool_movies.txt` размечен символами табуляции и содержит следующий текст:

```
Movie  Genre  Hero  Year
Die Hard   Action John McClane    1988
Star Wars  Sci-Fi Luke Skywalker  1977
John Wick   Action John Wick 2014
Aliens Sci-Fi Ellen Ripley   1986
The Thing Horror MacReady    1982
```

Для того чтобы отсортировать этот файл по названиям фильмов, я применяю следующую команду:

```
$ sort cool_movies.txt
Aliens  Sci-Fi  Ellen Ripley    1986
Die Hard   Action John McClane     1988
John Wick   Action John Wick    2014
Movie   Genre   Hero   Year
Star Wars   Sci-Fi   Luke Skywalker  1977
The Thing   Horror   MacReady   1982
```

Хм... Почти получилось, но текст содержит заголовочную строку, чего я не хотел.

К счастью, я могу использовать команду sed (см. ниже в этой главе), чтобы удалить первую строку с помощью опции 1d, а затем передать результаты команде sort.

```
$ sed 1d cool_movies.txt | sort
Aliens  Sci-Fi  Ellen  Ripley   1986
Die Hard    Action   John McClane   1988
John Wick   Action   John Wick  2014
Star Wars   Sci-Fi   Luke Skywalker   1977
The Thing   Horror   MacReady   1982
```

А что если мы хотим упорядочить фильмы не по названиям, а по годам выпуска? В таком случае я использую опцию -k (или --key), чтобы выбрать разные столбцы для сортировки, т.е. в данном случае — четвертое поле, отделенное символом табуляции. Однако по умолчанию команда sort использует в качестве разделителя пробел, поэтому мы должны сообщить ей, что она должна искать символ табуляции.

```
$ sed 1d cool_movies.txt | sort -t ' ' -k 4
Star Wars  Sci-Fi  Luke Skywalker 1977
The Thing  Horror  MacReady   1982
Aliens  Sci-Fi  Ellen Ripley  1986
Die Hard   Action  John McClane   1988
John Wick  Action  John Wick  2014
```

Проще всего задать табуляцию с помощью опции -t в одиночных кавычках. Для этого надо набрать одиночную кавычку, нажать комбинацию клавиш <Ctrl+V>, затем <Tab> и, закрывающую одиночную кавычку.

А если мы хотим упорядочить список по годам, но в обратном порядке? Для этого используется опция -r (или --reverse).

```
$ sed 1d cool_movies.txt | sort -t ' ' -k 4 -r
John Wick   Action  John Wick  2014
Die Hard    Action  John McClane    1988
Aliens  Sci-Fi Ellen  Ripley   1986
The Thing   Horror  MacReady   1982
Star Wars   Sci-Fi  Luke Skywalker  1977
```

Мы упорядочили список от 2014 до 1977 гг.

Числовая сортировка содержимого файла

```
sort -n
```

Последняя крутая функциональная возможность команды `sort` — числовая сортировка. Вы можете подумать, что, когда мы сортировали фильмы по годам в предыдущем разделе, то использовали числовую сортировку, но на самом деле команда `sort` рассматривала годы как строки. Рассмотрим пример работы с моим сервером в другой день, когда демонстрация возможности числовой сортировки оказалась более яркой. Я захотел выяснить, что хранится в каталоге `/var`, поэтому начал с команды `du` (см. главу 13), которая выдает отчет об использовании диска в файловой системе или папке. Опция `-d 1` команды `du` задает глубину отчета; опция `-d 1` означает, что команда должна определить размер каталогов, содержащихся непосредственно в каталоге `/var` (т.е. глубина равна 1).

Ниже показано, как я использовал команду и сколько байтов занимают подкаталоги каталога `/var`. Эта информация передается команде `sort`.

```
$ cd /var
$ du -d 1 | sort
154272    ./cache
1724      ./backups
200976    ./lib
21624     ./mysql_import
2263216   ./log
34479828  ./www
[Листинг сокращен для экономии места]
```

На самом деле эти команды не дают желаемого результата, не так ли? Обратите внимание на то, что байты упорядочены как строки: сначала идут строки, начинающиеся с единицы, потом с двойки, а затем с тройки. Эти байты не упорядочены как числа! Попробуем снова, но на этот раз используем опцию `-n` (или `--numeric-sort`) команды `sort`.

```
$ du -d 1 | sort -n
1724      ./backups
21624     ./mysql_import
154272    ./cache
200976    ./lib
324204    ./shared_assets
2264292   ./log
34434904  ./www
[Листинг сокращен для экономии места]
```

Вот теперь все правильно — байты упорядочены в числовом порядке, но довольно трудно понять, что именно они означают. Если бы мы использовали опцию `-h` (или `--human-readable`) команды `du`, то получили бы отчет в кибибайтах, мебибайтах или гибибайтах, например. (Не совсем понимаете, что означают эти слова? Прочитайте главу 11.) Вот что мы получим в виде отчета:

```
$ du -d 1 -h | sort -n
1.7M  ./backups
2.2G  ./log
22M   ./mysql_import
33G   ./www
151M  ./cache
197M  ./lib
317M  ./shared_assets
[Листинг сокращен для экономии места]
```

И все-таки это не совсем то, к чему мы стремились! *Числа* упорядочены в числовом порядке, но их смысл непонятен. Оказывается, если мы используем опцию `-h` команды `du`, то такую же опцию мы должны использовать в команде `sort`, чтобы результаты были согласованными, но в этом случае нельзя использовать опцию `-n`, поскольку опции `-h` и `-n` несовместимы.

```
$ du -d 1 -h | sort -h
1.7M  ./backups
22M   ./mysql_import
151M  ./cache
197M  ./lib
317M  ./shared_assets
2.2G  ./log
```

```
33G   ./www
[Листинг сокращен для экономии места]
```

И все-таки мы своего добились! Иногда полезно достигать своего методом проб и ошибок, поэтому я так подробно описывал примеры. Я всегда говорю своим студентами, что на ошибках учатся — вот почему я такой ученый.

Удаление из файла строк-дубликатов

uniq

Любого читателя можно сравнить с маленькой снежинкой, которая отличается от всех других снежинок. Иначе говоря, мы все уникальны (да, я специально напустил немного тумана). Оказывается, система Linux может объединять все строки-дубликаты, содержащиеся в файле, в одну уникальную строку, соответствующую каждому набору дубликатов. Иными словами, три строки со словами `Linux Phrasebook` (и только `Linux Phrasebook`) превращаются в одну строку, содержащую слова `Linux Phrasebook`. Волшебство? Нет, просто команда `uniq`.

Несколько недель назад я решил выяснить, какие команды мы выполняем на сервере и как часто. Посмотрим, как я нашел ответ на этот вопрос. применяя команду `uniq`. (Все следующие примеры были сокращены, иначе они заняли бы тысячи строк!)

```
$ history
12291 man rsync
12293 ps aux | grep agent
12294 ps aux | grep ssh
12296 apt-cache search keychain
12297 cat ~/.ssh/authorized_keys
12300 cd /var/www
12302 ln -s /var/shared_assets ~/
12303 cd bin
12314 man dig
```

Команда history, рассматриваемая в главе 12, выдает нумерованный список команд, которые вы ввели в оболочку bash. Первый столбец представляет собой список номеров, который в данном случае к делу не относится. После номеров мы видим список команд, опций и файлов (например, cd /var/www/ или man rsync). Мне нужны только команды (например, cd или man). Поэтому я использую команду awk (которая описывается в этой главе ниже), чтобы вывести на печать с помощью команды history второй столбец, отделенный пробелами.

```
$ history | awk '{print $2}'
man
ps
ps
apt-cache
cat
cd
ln
cd
man
```

Хорошо, теперь я знаю, все команды, которые мы выполняли, но список получился слишком длинным (поверьте мне, он действительно длинный). Это не слишком удобно, потому что в списке есть команды-дубликаты: например, по две команды man, ps и cd. Выполним команду uniq, чтобы каждая команда вошла в список только по одному разу.

```
$ history | awk '{print $2}' | uniq
man
ps
apt-cache
cat
cd
ln
cd
man
```

Похоже, не получилось. Команды man и cd по-прежнему входят по два раза, но команда ps — только один раз. Что случилось? Оказывается, команда uniq пропускает повторяю-

щиеся строки, *только если они являются соседними*. В приведенном выше списке два экземпляра команды `man` расположены далеко друг от друга, между двумя экземплярами команды `cd` расположена команда `ln`, поэтому их невозможно объединить в одну строку. Однако два экземпляра команды `ps` расположены рядом, поэтому команда превращает их в одну строку.

Для того чтобы исправить эту ситуацию, сначала необходимо использовать команду `sort`, которую мы рассмотрели в предыдущем разделе, чтобы создать алфавитный список команд, а затем передать результаты сортировки команде `uniq`.

```
$ history | awk '{print $2}' | sort | uniq
apt-cache
cat
cd
ln
man
ps
```

Отлично, мы продвинулись вперед. Команда `sort` объединяет два экземпляра команд `cd`, `man` и `ps`. Поскольку они были расположены рядом, команда `uniq` оставила только один экземпляр каждой из них. Это уже лучше, но я хотел узнать, сколько раз мы выполнили каждую из команд. К счастью, если использовать опцию `-c` (или `--count`) команды `uniq`, то мы сможем определить количество всех экземпляров каждой команды до того, как из нее будут удалены дубликаты.

```
$ history | awk '{print $2}' | sort | uniq -c
14 apt-cache
50 cat
229 cd
249 ln
84 man
17 ps
```

Это хорошо — теперь мы знаем, сколько раз была использована каждая команда. Но они перечислены в неудобном порядке. Они все еще упорядочены по алфавиту командой `sort`, а нам нужен числовой порядок. Чтобы достичь цели,

снова применим команду sort, на этот раз с опцией -n (от слова numeric)!

```
$ history | awk '{print $2}' | sort | uniq -c |
↳sort -n
14 apt-cache
17 ps
50 cat
84 man
229 cd
249 ln
```

Итак, мы получили упорядоченный в числовом порядке пронумерованный список команд, которые были выполнены на сервере. Команды sort и uniq часто передают результаты работы друг другу по конвейеру, поскольку они отлично дополняют друг друга.

Замена заданных символов другими

tr

Команда tr используется для перевода (точнее, для *подстановки*) одного множества символов в другой. Например, можно использовать команду tr, чтобы преобразовать строчные символы в прописные.

```
$ echo "H.P. Lovecraft" | tr a-z A-Z
H.P. LOVECRAFT
```

После имени команды tr указывается, *что* мы хотим перевести и *во что*. В предыдущем примере параметр a-z представляет строчные символы, а A-Z — прописные. Впрочем, вместо параметров a-z или A-Z, которые можно считать устаревшими, следует использовать более современные и переносимые классы символов [:lower:] и [:upper:]. Классы символов представляют собой многочисленные наборы символов. Перечислим некоторые полезные классы, указав символы, которые они содержат.

- `[:alnum:]` Алфавитно-цифровые символы (A–Z, a–z и 0–9)
- `[:alpha:]` Алфавит (A–Z и a–z)
- `[:blank:]` Пробельные символы (пробел и табуляция)
- `[:digit:]` Числа (0–9)
- `[:lower:]` Строчной алфавит (a–z)
- `[:punct:]` Пунктуация и символы
- `[:space:]` Разделители (пробел, табуляция, переход на новую строку и вертикальная линия)
- `[:upper:]` Прописной алфавит (A–Z)

Просмотрев этот список, легко понять, что в предыдущем примере нам следовало использовать команду

```
$ echo "H.P. Lovecraft" | tr [:lower:] [:upper:]
H.P. LOVECRAFT
```

Надеюсь, что я не потревожил дух H.P. Мне не хотелось бы, чтобы он прислал за мной своих монстров!

Замена повторяющихся символов одним экземпляром

```
tr -s
```

Команда `tr` имеет много полезных опций, одна из них — опция `-s` (или `--squeeze-repeats`), позволяет оставить только один экземпляр повторяющихся символов. Допустим, некто прислал вам текстовый файл. Вы начали его читать и пришли в ужас от того, что автор после каждой точки вставил два пробела, что определенно является варварством. Используя опцию `-s`, вы можете сократить количество пробелов до одного, что, несомненно, является признаком интеллигентного, вежливого и прекрасно выглядящего человека.

```
$ cat "Written by a barbarian.txt"
I clearly do not know how to compose a letter.
↳Nope, I don't. In general, Conan is my
↳hero. What is best in life? Crush your
```

```
↳enemies. See them driven before you.
↳Hear the lamentations of their women.
$ tr -s [:blank:] < "Written by a barbarian.txt"
I clearly do not know how to compose a letter.
↳Nope, I don't. In general, Conan is my
↳hero. What is best in life? Crush your
↳enemies. See them driven before you. Hear
↳the lamentations of their women.
```

Как мы видим, файл с двумя пробелами после каждой точки (ужас!) превратился в файл с одним пробелом (ах!). Вы можете спросить, почему мы использовали опцию `[:blank:]`, а не `[:space:]`. Они похожи, но опция `[:blank:]` представляет только пробелы и символ табуляции, а опция `[:space:]`, помимо этого, подразумевает символы перехода на новую строку (и некоторые другие). Если бы файл содержал много абзацев, то опция `[:space:]` убрала бы лишние символы перехода на новую строку и все абзацы были бы записаны плотнее.

Следует знать об одной особенности команды `tr`: мы использовали символ `<`, чтобы перенаправить поток `stdin` на ввод из файла (см. главу 5), потому что команда `tr` относится к категории немногочисленных команд, в список аргументов которых не входят имена файлов. Иначе говоря, следующая команда невозможна: `tr -s [:blank:] Barbarian.txt`. Вместо этого необходимо перенаправить поток с помощью символов `>`, `|`, `<` и других методов (детали см. в главе 5): `echo "Too many spaces" | tr -s [:blank:]`.

А если мы хотим создать новый, правильно отформатированный файл из варварского оригинала? Легко. Можно просто перенаправить вывод в *новый файл*. (Напомним, что перенаправить файл в себя невозможно, иначе на вашем компьютере возникнет небольшая черная дыра.)

```
$ tr -s [:blank:] < "Written by a barbarian
↳txt" > Corrected.txt
```

Теперь можно послать файл, не рискуя своим добрым именем из-за ужасных двойных пробелов после точек!

Удаление заданных символов

tr -d

Команда `tr` с опцией `-d` (или `--delete`) удаляет символы (спорю, что вы ни за что бы не догадались). Помимо прочего, я покажу вам, как подготовить файл к использованию в веб. (Более подробную информацию можно найти в главе 12.) Следующая функция хранится в каталоге `~/.bash_functions` (я отформатировал строку 4, чтобы ее было легче читать):

```
web_ready () {
  [ "${1}" ] || return
  extension=$(echo "${1}" | awk -F . '{print $NF}')
  mv "${1}" $(echo "${1}" |
  ↳iconv -f utf-8 -t ascii//translit |
  ↳tr -d '[:punct:]' |
  ↳tr [:upper:] [:lower:] |
  ↳sed "s/${extension}$/.$extension/" | tr ' ' '-')
}
```

Прежде чем приступать к объяснениям, посмотрим на результат.

```
$ ls
What They Know, What You Don't.pdf
$ web_ready "What They Know, What You Don't.pdf"
$ ls
what-they-know-what-you-dont.pdf
```

Для того чтобы подготовить файл к публикации в веб, необходимо удалить знаки пунктуации, сделать все буквы строчными и заменить пробелы дефисами. Посмотрим, как работает эта функция.

Первая строка функции — `[ "${1}" ] || return` — проверяет, задан ли аргумент (имя файла) после команды `web_ready`, и если нет, то выполняется выход из функции.

Вторая строка — `extension=$(echo "${1}" | awk -F . '{print $NF}')` — немного сложнее. Мы создаем переменную с именем `extension`; значение, присваиваемое этой переменной,

является результатом подстановки, выполняемой в скобках `$(` и `)` (см. главу 5). Мы выводим на экран первый и единственный аргумент команды `web_ready`, представляющий собой имя файла, заданное в виде `"${1}"` (двойные кавычки используются потому, что имя файла может содержать пробелы).

Затем мы передаем имя файла команде `awk` (ее описание приводится в конце этой главы), которая знает, что точка (`.`) является разделителем, заданным опцией `-F`. Параметр `'{print $NF}'` сообщает команде `awk`, что она должна вывести последнее поле размеченного имени файла, который в данном случае будет расширением файла (в данном случае используется `pdf`, хотя оно может быть любым, например, `html`, `markdown` или `jpg`).

Теперь переменная `extension` содержит расширение файла: `pdf`. Очень скоро мы его используем.

Третья строка довольно длинная — `mv "${1}" $(echo "${1}" | iconv -f utf-8 -t ascii//translit | tr -d '[:punct:]' | tr [:upper:] [:lower:] | sed "s/$extension$/.$extension/" | tr ' ' '-')`, — но в ней легко разобраться, за небольшими исключениями. Команда `mv` переименовывает файл, заменяя его исходное имя, заданное первым аргументом (`"${1}"`), на его веб-имя, которое использует пятикратную подстановку.

Это интересная часть функции. Во-первых, мы задаем имя файла командой `echo "${1}"`, а затем немедленно передаем результат команде `iconv -f utf-8 -t ascii//translit`. Ничего себе! А это что такое? Внимательно посмотрите на имя файла (`What They Know, What You Don't.pdf`) и обратите внимание на то, что апостроф в слове `Don't` не прямой, а *типографский*, или фигурный. Иначе говоря, вместо слова `Don't` (в кодировке ASCII) в имени файла используется слово `Don't` (в кодировке UTF-8).

ЗАМЕЧАНИЕ

Я уже упоминал о кодировке UTF-8 в предыдущей главе, но не объяснил, что она собой представляет и почему считается такой важной. Я подготовил веб-страницу и несколько слайдов для моих студентов, чтобы

описать ASCII, UTF-8 и несколько других форматов. Если вас интересует этот вопрос, посетите эту веб-страницу по адресу `www.granneman.com/webdev/coding/characterencoding/` и посмотритте слайды (`http://files.granneman.com/presentations/webdev/Web-Dev-Intro.pdf`).

Проблема в том, что по состоянию на 2005 год команда `tr` еще не понимала кодировку UTF-8 (довольно запутанную). Следовательно, моя функция просто не работала. Ее можно было запускать, но функция спотыкалась на типографском апострофе. Итак, необходимо использовать команду `iconv`, конвертирующую файлы и тексты из одной кодировки (в данном случае `-f utf-8`) в другую (в данном случае `-t ascii`).

Параметр `//translit` сообщает команде `iconv`, что при обнаружении символов, не входящих в целевую кодировку (в данном случае ASCII, потому что указана опция `-t ascii`), она может автоматически заменить их похожими символами. Иначе говоря, когда команда `iconv` обнаруживает типографский апостроф, который входит в кодировку UTF-8, но не входит в более старую и простую кодировку ASCII, она заменяет его эквивалентным символом из кодировки ASCII, т.е. прямым апострофом (который называется *машинописным*). Без параметра `//translit` команда `iconv` начинает обрабатывать символ ’, сразу выдает ошибку и останавливается.

После замены типографского апострофа на машинописный имя файла превращается в строку `What They Know, What You Don't.pdf`. Оно передается команде `tr -d '[:punct:]'`, которая удаляет все знаки пунктуации из имени, в данном случае запятую, апостроф и точку перед расширением. Как же так? Нам же нужна эта точка!

Не беспокойтесь — она скоро вернется на место. А пока имя файла превращается в строку `What They Know What You Dontpdf` (обратите внимание на отсутствие точки перед расширением `pdf`!). Мы передаем эту строку команде `tr [:upper:]`

[:lower:], которая заменяет прописные буквы строчными, и теперь файл называется what they know what you dontpdf.

Настал момент решить проблему с расширением файла (я же говорил, что точка еще вернется!) с помощью команды sed "s/${extension}$/ .$extension/". Передадим текущее имя (what they know what you dontpdf) команде sed (которая описывается в конце главы) и используем нашу переменную, содержащую расширение: ${extension}. Тем самым мы даем команде sed указание заменить значение переменной extension (pdf) точкой и ее значением, т.е. .pdf.

ЗАМЕЧАНИЕ

Какая разница между выражениями ${extension} и $extension? Никакой — они означают одно и то же. Просто выражение ${extension} проще читать и интерпретировать как переменную.

Совершенно ясно, что ${extension} — это переменная, заменяющая расширение pdf, а что же означает выражение ${extension}$? Посмотрим на следующий пример:

```
$ extension=pdf
$ echo "this pdf is ready for youpdf" |
↳sed "s/${extension}/.$extension/"
this .pdf is ready for youpdf
```

После того как команда sed находит первую подстановку, она останавливается, поэтому работа прекращается на *первом* вхождении слова pdf (This .pdf), а расширение имени файла остается прежним (youpdf). Однако символ $ в регулярных выражениях всегда означает конец строки, поэтому второй символ $ в выражении ${extension}$ точно соответствует расширению, поскольку оно всегда находится после имени файла. По существу, выражение ${extension}$ указывает команде sed искать значение выражения ${extension} (pdf), но

только если оно появляется в конце строки, который обозначается символом $ (следовательно, расширение youpdf превращается в you.pdf).

Теперь имя файла имеет вид what they know what you dont.pdf, и пришло время разобраться с последней командой: tr ' ' '-'. Эта команда заменяет все пробелы в имени файла дефисами, выдавая искомый результат, файл с именем для использования в веб: what-they-know-what-youdont.pdf. Фух!

Преобразование текста в файле

sed

Позвольте мне быть откровенным: команда sed — чрезвычайно мощная, сложная и полезная, поэтому сама мысль о том, что я могу достаточно полно описать ее в коротком разделе, смехотворна. С другой стороны, я собираюсь ограничиться только основными приложениями команды sed, которые позволят читателям представить ее потенциальные возможности.

ЗАМЕЧАНИЕ

Команда sed встречается в книге несколько раз.

- В этой главе в разделах "Подсчет количества слов, строк и символов в файле", "Сортировка содержимого файла" и "Удаление заданных символов" (особенно).
- В главе 12 в разделе "Создание новой постоянно действующей функции".

Кроме моего скромного вклада, рекомендую обратиться к следующим источникам.

- Bruce Barnett's "Sed - An Introduction and Tutorial" (www.grymoire.com/Unix/Sed.html)

- Peteris Krumins, "Article series 'Sed One-Liners Explained'" (`www.catonmat.net/series/sed-one-liners-explained`)
- Peteris Krumins, Sed One-Liners Explained (`www.catonmat.net/blog/sed-book/`)
- Eric Pement, "Useful One-Line Scripts for sed" (`www.pement.org/sed/sed1line.txt`)
- Dale Dougherty & Arnold Robbins, sed & awk

Имя команды `sed` представляет собой сокращение от *stream editor* (редактор потоков). Следовательно, она позволяет направлять поток текста либо в файл, либо в другом направлении. Для того чтобы использовать всю мощь команды `sed`, необходимо разбираться в регулярных выражениям (синонимы: *regex*, или *regexp*), которые описываются в разделе "Общие сведения о шаблонах поиска" главы 10, а также в нескольких других разделах этой главы.

Первое, что следует знать о команде `sed`, — это то, что с ее помощью можно выполнять подстановки. Допустим, у вас есть автоматически сгенерированный файл, имя которого заканчивается `.markdown.txt`, а вам нужно расширение `.markdown` без `.txt`. Эту задачу решают следующие команды:

```
$ ls
The Cats of Ulthar.markdown.txt
$ title="The Cats of Ulthar.markdown.txt"
$ mv "$title" "$(echo "$title" |
↳sed 's/markdown.txt/markdown/')"
$ ls
The Cats of Ulthar.markdown
```

Команда `sed` в четвертой строке заменяет строку `markdown.txt` на `markdown`. Разумеется, если такую подстановку приходится выполнять часто, то нетрудно создать функцию, которая делает то же самое немного проще. Пример можно найти в предыдущем разделе.

Обратите внимание на то, что команда sed в предыдущем листинге останавливается, обнаружив первое совпадение в каждой строке. В нашем примере это не создает проблем, поскольку у нас есть всего лишь двойное расширение (.markdown.txt) в каждом имени файла (которому соответствует отдельная строка!). А если нам необходимо выполнять подстановку для всех совпадений независимо от количества строк?

```
$ cat twisters.txt
1. How much wud would a wudchuck chuck if a
↳wudchuck could chuck wud?
2. How much myrtle would a wud turtle hurdle if a
↳wud turtle could hurdle myrtle?
$ sed 's/wud/wood/g' twisters.txt
1. How much wood would a woodchuck chuck if a
↳woodchuck could chuck wood?
2. How much myrtle would a wood turtle hurdle if a
↳wood turtle could hurdle myrtle?
$ cat twisters.txt
1. How much wud would a wudchuck chuck if a
↳wudchuck could chuck wud?
2. How much myrtle would a wud turtle hurdle if a
↳wud turtle could hurdle myrtle?
```

Следует сделать два замечания об этом коде. Во-первых, флаг g в выражении s/wud/wood/g означает *глобально*. Когда вы добавляете флаг g, команда sed заменяет все экземпляры совпадающих слов или шаблонов в каждой строке ввода. Следовательно, в нашем примере каждое слово wud превратится в wood, даже если оно является частью другого слова (что может оказаться нежелательным в некоторых ситуациях). Во-вторых, обратите внимание на то, что исходный файл не изменился! На экран выводится другая информация, но сам файл остается прежним. Для того чтобы изменить файл, необходимо направить вывод в другой файл.

```
$ sed 's/wud/wood/g' twisters.txt >
↳twisters_fixed.txt
```

А еще лучше использовать опцию -i (или --in-place) команды sed. Она действительно изменяет файл в соответствии с внесенными изменениями, так что можете быть уверены, что команда sed делает именно то, что вы хотели. Посмотрите на результаты работы команды sed, чтобы убедиться, что все в порядке, а затем используйте опцию -i.

```
$ sed -i 's/wud/wood/g' twisters.txt
$ cat twisters.txt
1. How much wood would a woodchuck chuck if a
↳woodchuck could chuck wood?
2. How much myrtle would a wood turtle hurdle if a
↳wood turtle could hurdle myrtle?
```

Я уже писал, что, работая с командой sed, вы можете (и *должны*) использовать регулярные выражения. Они полезны для практически всех аспектов команды sed (регулярные выражения используются *в большей части* подстановок), а при удалениях оказывают просто незаменимую помощь.

Допустим, у вас есть файл, содержащий сценарий оболочки bash и комментарии: строки, начинающиеся символом #. Вы хотите удалить эти символы, потому что они вам больше не нужны, за исключением первого, который необходим в сценарии оболочки. Эту задачу легко решить с помощью команды sed. Вам необходимо использовать команду d (сокращение от слова *delete* — удалить) и регулярные выражения. Смысл сценария нас не интересует, мы просто изучаем работу команды sed.

```
$ cat handconv.sh
#!/bin/sh
# Set IFS to split on newlines, not spaces, but 1st
↳save old IFS
SAVEIFS=$IFS
IFS=$'\n'
# Convert files in folder
for i in * ; do
  # Get the extension of the file, so we can get
  ↳$filename
  extension=${i##*.}
  # Get the name of the file, sans extension
```

```
  filename=$(basename "$i" $extension)
  # Encode files with HandBrakeCLI
  HandBrakeCLI -i "$i" -o ~/Desktop/"$filename"m4v
  ↳ --preset="Normal"
done
# Restore IFS so it's back to splitting on <space>,
↳<tab>, & <newline>
IFS=$SAVEIFS
$ sed -i '1n; /^[[:blank:]]*#/d' handconv.sh
$ cat handconv.sh
#!/bin/sh

SAVEIFS=$IFS
IFS=$'\n'

for i in * ; do
  extension=${i##*.}
  filename=$(basename "$i" $extension)
  HandBrakeCLI -i "$i" -o ~/Desktop/"$filename"m4v
  ↳ --preset="Normal"
done

IFS=$SAVEIFS
```

Почти весь смысл команды заключен в выражении `1n; /^[[:blank:]]*#/d`. Символ `d` в конце означает удаление, а что значит все остальное? Напомню, что мы не хотим удалять первую строку, начинающуюся символом `#` (`#!/bin/sh`), поскольку она необходима для сценария оболочки. За эту часть команды отвечает параметр `1n;` — он указывает команду `sed`, что она должна пропустить первую строку и продолжать работу. Затем происходит сопоставление шаблона.

Мы хотим удалить оставшиеся строки, которые начинаются с символа `#`; в данном случае шаблон выглядит как `/^#/` (символ `^` означает начало строки). Команда обнаруживает все строки, начинающиеся символом `#`, но в этот момент следует проявить осторожность. Некоторые строки, которые кажутся соответствующими шаблону, на самом деле ему не соответствуют. Посмотрите на следующие две строки из сценария оболочки:

```
# Convert files in folder
  # Encode files with HandBrakeCLI
```

Первая строка начинается с символа #, расположенного в самом начале строки. Этому варианту соответствует выражение `/^#/`. Вторая строка тоже начинается с символа #, но перед знаком решетки стоят два пробела, поэтому она не соответствует шаблону.

Для того чтобы устранить эту проблему, используется выражение `/^[[:blank:]]*#/`. Символ ^ по-прежнему означает начало строки, но следующий класс символов `[[:blank:]]` (см. раздел "Замена заданных символов другими") обнаруживает пробел или символ табуляции. Однако нам требуется найти более одного пробела, поэтому после класса `[[:blank:]]` мы поставили знак *, означающий, что предшествующий символ может появляться несколько раз или вообще отсутствовать. Итак, выражение `/^[[:blank:]]*#/` означает следующее: начать поиск с первой позиции строки (^) и найти пробел или символ табуляции (`[[:blank:]]`), который может появляться перед символом # несколько раз или вообще отсутствовать.

Именно так работают регулярные выражения и команда sed. Я лишь слегка коснулся работы команды sed, но надеюсь, что у читателей появился интерес к этой команде. Если вы — компьютерный фанат, как и я, а даже если и нет, то, используя регулярные выражения, вы можете творить чудеса на своем компьютере. Изучайте команду sed и регулярные выражения — и будет вам счастье.

Вывод на печать конкретных полей из файла

awk

В начале предыдущего раздела я писал, что мы скоро перейдем к изучению команды awk. Команда awk — чрезвычайно

мощный, сложный и полезный инструмент. Описать ее в коротком разделе практически невозможно. Я поступлю так же, как и при описании команды sed, — сосредоточу основное внимание на нескольких приложениях команды awk, в данном случае на выборе столбцов и выводе их на печать. Это позволит вам получить первое впечатление о силе команды awk.

ЗАМЕЧАНИЕ

Команда awk используется в главе 12 в разделе "Создание новой постоянно действующей функции," поэтому целесообразно прочитать и этот раздел тоже. Кроме моих скромных усилий, используйте следующие источники.

- Bruce Barnett, "Awk" (www.grymoire.com/Unix/Awk.html)
- Peteris Krumins, "Article series 'Awk One-Liners Explained'" (www.catonmat.net/series/awk-one-liners-explained)
- Peteris Krumins, Awk One-Liners Explained (www.catonmat.net/blog/awk-book/)
- Eric Pement, "Handy One-Line Scripts for awk" (www.pement.org/awk/awk1line.txt)
- Dale Dougherty & Arnold Robbins, sed & awk
- Alfred Aho, Peter J. Weinberger, and Brian W.Kernighan, The AWK Programming Language (обратите внимание на инициалы авторов!)

Имя команды awk составлено из инициалов ее изобретателей: Ахо (Aho), Вайнбергера (Weinberger) и Кернигана (Kernighan). Она позволяет искать в файле и вводить шаблоны, а также фильтровать элементы и генерировать отчеты. Она напоминает команду sed, и в некоторых областях результаты их работы даже совпадают, но awk — это настоящий язык программирования, содержащий переменные, функции, массивы,

системные вызовы и многое другое, чего нет в команде sed. Означает ли это, что команду awk следует всегда применять вместо команды sed? Нет. Это значит лишь, что вы должны правильно выбирать команду для решения своей задачи.

На одном из моих серверов буферный файл почты иногда становится слишком большим, поэтому я периодически проверяю его размер и сортирую. Я начинал с простой команды wc (см. выше в этой главе), чтобы выяснить количество байтов в файле mail.

```
$ wc -c /var/mail/mail
2780599 /var/mail/mail
```

Это прекрасно, но мне нужно лишь количество байтов, а не имя файла. Для того чтобы вывести на печать только первый столбец, я указываю параметр $1 команды awk.

```
$ wc -c /var/mail/mail | awk '{print $1}'
2780599
```

Уже лучше, но... мне не хочется забивать голову дополнительными вычислениями, поэтому я хочу переложить эту обязанность на компьютер.

```
$ wc -c /var/mail/mail |
↳awk '{print "mail is " $1/1024 " MB"}'
mail is 2715.43 MB
```

Я указал команде awk, что она должна вывести строку "mail is " (обратите внимание на пробел в конце строки), а затем взять первый столбец, поделить его на 1024 и добавить строку " MB" (и снова обратите внимание на пробел, на этот раз в начале). Результат вы видите в листинге: mail is 2715.43 MB.

ПОДСКАЗКА

Было бы лучше превратить эту команду в функцию, поскольку она довольно длинная и набирать ее не очень удобно. О том, как это сделать, речь пойдет в разделе "Создание новой временной функции" главы 12.

Немного усложним задачу. Допустим, я хочу получить столбец данных и вывести его на экран как мини-отчет. Команда `df -h` (подробнее о ней — в главе 13) показывает степень занятости диска файловой системы компьютера (я удалил несколько строк для экономии места, чтобы сосредоточиться на главном).

```
$ df -h
Filesystem Size Used Avail Use% Mounted on
/dev/sda 4.9G 742M 4.1G 16% /
/dev/sdb 79G 36G 43G 46% /var
```

Мне не нужна вся эта информация, поэтому я использую команду `awk`, чтобы переписать данные с экрана.

```
$ df -h |
↳awk '{ print $6 " has " $4 " of " $2 " left" }'
Mounted has Avail of Size left
/ has 4.1G of 4.9G left
/var has 43G of 79G left
```

Ну и ну! Результат выглядит хорошо, за исключением первой строки! На самом деле команда `awk` сделала не совсем то, что я хотел (кстати, сортировка столбцов выполнена безупречно): она извлекла шестой, четвертый и второй столбцы и записала их вместе с заданными строками (обратите внимание на пробелы, которые добавлены для удобства чтения). Первая строка входит в эти столбцы, поэтому они появились на экране, хотя я этого не хотел. Вот как можно исправить эту ситуацию:

```
$ df -h | awk
↳'NR>1 { print $6 " has " $4 " of " $2 " left" }'
/ has 4.1G of 4.9G left
/var has 43G of 79G left
```

С помощью параметра `NR>1` я указал команде `awk`, что она должна пропустить первую строку. `NR` — это специальная переменная команды `awk`, которая означает *количество записей* (*number of records*), т.е. строк, которые команда `awk` уже просмотрела в файле. Эта переменная увеличивается на единицу

каждый раз, когда команда awk заканчивает обработку строки. Итак, выражение NR>1 означает, что, пропустив первую строку (>1), команда должна вывести на экран запрашиваемую информацию. Отлично, awk!

ЗАМЕЧАНИЕ

Некоторые из читателей, просматривая предыдущий пример, могли подумать, что можно было бы написать команду так:

```
$ df -h | grep '^/' |
awk '{ print $6 " has " $4 " of " $2 " left" }'
```

Вы абсолютно правы! Вместо параметра NR>1 в команде awk можно было бы указать команде grep пропустить все строки, не начинающиеся с символа /. Фактически это означает удаление первой строки из результатов работы команды df. Какой из этих вариантов следует использовать, решать вам. Это еще раз доказывает, что команды системы Linux часто позволяют выполнить одну и ту же работу по-разному.

Выводы

Выводы можно сформулировать в виде одного из законов программирования Тейлора ("Taylor's Law of Programming").

1. Никогда не пиши на языке C, если это можно сделать с помощью команды awk.
2. Никогда не вызывай команду awk, если это можно сделать с помощью команды sed.
3. Никогда не используй команду sed, если это можно сделать с помощью команды tr.
4. Никогда не вызывай команду tr, если достаточно команды cat.
5. По возможности избегай использования команды cat.

Прочитав эту и другие главы, посвященные командам, вы начнете понимать логику закона Тейлора. Он забавен, но в нем есть смысл: используйте правильную команду в подходящий момент.

Рассматривая команды `sed` и `awk`, мы начали путешествие по двум огромным странам. За пределами главы, да и книги в целом, осталось много информации, касающейся фильтров, но должно быть очевидным, что теперь вы знаете очень важную часть командной строки системы Linux. Система UNIX основана на использовании маленьких программ, предназначенных для решения отдельных задач, а фильтры идеально воплощают эту идею.

8

Владельцы файлов и права доступа

Система Linux изначально разрабатывалась как многопользовательская система (в отличие от нее, Windows была изначально ориентирована на одного пользователя, этим и объясняются проблемы с ее системой безопасности). Это означает, что с системой могут одновременно работать несколько пользователей: создавать файлы, удалять каталоги, просматривать текстовые файлы и выполнять другие, самые разнообразные действия. Для того чтобы пользователи не мешали друг другу и тем более не повредили операционную систему, был разработан механизм прав доступа. Понимание прав доступа существенно упрощает использование Linux, независимо от того, установлена ли эта система на рабочей станции, ориентированной на одного пользователя, или на сервере, посещаемом многими. Несмотря на то что данный инструмент довольно простой, он очень мощный. Попробуем разобраться с ним.

ЗАМЕЧАНИЕ

Готовя книгу ко второму изданию, я удалил раздел, посвященный командам `chgrp -v` и `chgrp -c`. Исходный текст можно найти на моем веб-сайте по адресу `www.granneman.com/linux-redactions`.

Вход под другим именем

```
su username
```

Команда `su` (сокращение от *switch user* (переключение пользователя), а не *super user* (администратор), как многие

считают) позволяет пользователю временно войти в систему под другим именем, но чаще всего она используется для того, чтобы быстро стать пользователем root в оболочке, выполнить одну или две команды, а затем снова стать обычным пользователем. Вспомните, как Кларк Кент[1] превращается в Супермена, побеждает парочку врагов, а затем принимает обычное состояние.

Вызвать команду su несложно: введите su, затем имя пользователя, идентичность которого хотите присвоить, и пароль этого пользователя (но не свой).

```
$ ls
/home/scott/libby
$ whoami
scott
$ su gromit
Password:
$ whoami
gromit
$ ls
/home/scott/libby
```

В этом примере встречается новая команда, которая используется не слишком часто: whoami. Она просто сообщает вам ваше имя, поскольку оно интересует оболочку. Здесь эта команда используется для того, чтобы проверить, что команда su работает так, как ожидалось.

Вход под другим именем и с другими переменными окружения

```
su -l
```

Команда su работает только в том случае, если вы знаете пароль целевого пользователя. Нет пароля — нет превращения. Если же команда работает, вы переключаетесь в оболочку,

[1] Герой американских комиксов. — *Примеч. ред.*

заданную пользователем в файле /etc/passwd: sh, zsh или bash, например. Большинство пользователей системы Linux по умолчанию используют оболочку bash, поэтому, возможно, вы не почувствуете никакой разницы. Обратите внимание на то, что в предыдущем примере мы не изменили каталоги при переключении пользователя. По существу, вы становитесь пользователем gromit, но продолжаете использовать переменные окружения пользователя scott. Это все равно, что вы нашли костюм Супермена и напялили его на себя. Возможно, вы похожи на Супермена (о, да!), но не обладаете его мощью.

Исправить эту ситуацию можно с помощью опции -l (или --login).

```
$ ls
/home/scott/libby
$ whoami
scott
$ su -l gromit
Password:
$ whoami
gromit
$ ls
/home/gromit
```

Внешне этот код выглядит примерно так же, как и в примере из раздела "Вход под другим именем", но за кулисами все происходит совершенно иначе. Тот факт, что теперь вы находитесь в домашнем каталоге пользователя gromit, должен продемонстрировать, что ситуация изменилась. Опция -l команды su означает, что должна использоваться оболочка с регистрацией, как будто пользователь gromit действительно зарегистрировался на компьютере.

Теперь вы — пользователь gromit не только по имени, но и по существу, поскольку используете его переменные окружения и находитесь в его домашнем каталоге (там, где оказывается пользователь gromit при первой регистрации на компьютере). Это значит, что, напялив на себя трусы супермена, вы действительно получаете возможность перепрыгивать через небоскребы!

Вход под именем пользователя root

```
su
```

В начале главы мы указали, что чаще всего команда su используется для входа под именем root. Для этого можно использовать команду su root, а еще лучше su -l root, которая работает намного быстрее.

```
$ whoami
scott
$ su
Password:
$ whoami
root
```

Если команда su не работает, попытайтесь выполнить команду sudo su. Разные пути к одной и той же цели — обычная ситуация!

Вход под именем пользователя root с его переменными окружения

```
su -
```

Ввод команды su без опций эквивалентен вводу команды su root — вы становитесь пользователем root по имени и по возможностям, но это все. На самом деле все ваши собственные переменные окружения (не root) продолжают действовать.

```
$ ls
/home/scott/libby
$ whoami
scott
$ su
Password:
$ whoami
```

```
root
$ ls
/home/scott/libby
```

Выполнив команду `su -`, вы не только становитесь пользователем `root`, но и начинаете использовать его переменные окружения.

```
$ ls
/home/scott/libby
$ whoami
scott
$ su -
Password:
$ whoami
root
$ ls
/root
```

Вот теперь все много лучше! Добавить параметр `-` после команды `su` равносильно выполнению команды `su -l root`, но при этом требуется набрать меньше символов. Теперь вы присвоили себе имя, права и окружение пользователя `root`, т.е. стали полноценным пользователем `root`. Теперь вы можете делать на компьютере все то, что может делать пользователь `root`. Наслаждайтесь своими сверхъестественными возможностями, но помните, что большая сила означает большую ответственность... ну, в общем, вы в курсе![2]

Изменение групп для файлов и каталогов

chgrp

По умолчанию почти во всех версиях системы Linux, создавая новый файл (или каталог), вы становитесь владельцем и файла, и его группы. Предположим, например, что вы собираетесь написать новый сценарий.

[2] Цитата человека-паука. — *Примеч. ред.*

```
$ touch new_script.sh
$ ls -l
-rw-r--r-- 1 scott scott script.sh
```

ЗАМЕЧАНИЕ

В данном случае и пользователь, и группа имеет имя scott, однако это не обязательно должно быть так. При создании файла идентификатор пользователя используется для обозначения владельца файла, а идентификатор группы — для обозначения группы.

Предположим также, что вы входите в группу, называемую admins, и хотите, чтобы этот сценарий могли запускать другие члены данной группы. Как добиться этого? Заменить группу scott на admins можно с помощью команды chgrp.

```
$ chgrp admins new_script.sh
$ ls -l
-rw-r--r-- 1 scott admins script.sh
```

ЗАМЕЧАНИЕ

Да, действительно, сценарий не будет работать, так как файл, в котором он содержится, не является исполняемым. Как исправить ситуацию, вы узнаете далее в этой главе при рассмотрении команды chmod.

Говоря о команде chgrp, необходимо отметить две особенности. Во-первых, при вызове команды chgrp можно указывать либо имя группы, либо ее идентификатор. Как же выяснить, какой идентификатор соответствует той или иной группе? Проще всего сделать это, применив команду cat к файлу /etc/group, содержащему информацию о группах.

```
$ cat /etc/group
bind:x:118:
scott:x:1001:
```

```
admins:x:1002:scott,alice,bob
[Листинг сокращен для экономии места]
```

Во-вторых, пытаясь применить данную команду, следует учитывать вопросы безопасности. Изменять принадлежность группе можно только в том случае, если вы являетесь ее членом. Другими словами, пользователи `scott`, `alice` и `bob` могут использовать команду `chgrp`, чтобы установить принадлежность файла или каталога группе `admins`, а команда `carol` не может сделать этого, поскольку она не является членом группы `admins`.

Рекурсивное изменение принадлежности каталога группе

`chgrp -R`

В ряде случаев нецелесообразно изменять принадлежность группе лишь для одного файла или каталога. Символы групповых операций позволяют сделать это сразу для нескольких файлов, содержащихся в каталоге. Если вы хотите изменить группу для каталога и всего начинающегося с него поддерева файловой системы, используйте опцию `-R` (или `--recursive`).

```
$ pwd
/home/scott/pictures/libby
$ ls -F
by_pool/ libby_arrowrock.jpg libby.jpg on_floor/
$ ls -lF *
-rw-r--r-- 1 scott scott libby_arrowrock.jpg
-rw-r--r-- 1 scott scott libby.jpg

by_pool/:
-rw-r--r-- 1 scott scott libby_by_pool_02.jpg
drwxr-xr-x 2 scott scott lieberman_pool

on_floor/:
-rw-r--r-- 1 scott scott libby_on_floor_01.jpg
```

```
-rw-r--r-- 1 scott scott libby_on_floor_02.jpg
$ chgrp -R family */*
$ ls -l *
-rw-r--r-- 1 scott family libby_arrowrock.jpg
-rw-r--r-- 1 scott family libby.jpg

by_pool:
-rw-r--r-- 1 scott family libby_by_pool_02.jpg
drwxr-xr-x 2 scott family lieberman_pool

on_floor:
-rw-r--r-- 1 scott family libby_on_floor_01.jpg
-rw-r--r-- 1 scott family libby_on_floor_02.jpg
```

ПРЕДУПРЕЖДЕНИЕ

Если вы выполните команду `chgrp -R family *`, она не затронет файлов, имена которых начинаются с точки (в данном примере речь идет о каталоге `/home/scott/pictures/libby`). Однако все произойдет по-другому, если команда будет выглядеть так: `chgrp -R family .*`. Она не только изменит принадлежность группе файлов текущего каталога, начинающихся с точки. Имя `..` также соответствует шаблону `.*`, поэтому изменения коснутся также файлов, находящихся в родительском каталоге. Вряд ли вы стремились получить такой результат!

Изменение владельцев файлов и каталогов

chown

Возможность изменить группу для файла достаточно важна, но гораздо чаще вы будете сталкиваться с задачей изменения владельца этого файла. Если для изменения группы применяется команда `chgrp`, то для изменения владельца надо

воспользоваться командой chown. Однако у нее есть одна особенность: владельца файла может изменить только пользователь root (в системе Ubuntu это может сделать программа sudo).

ЗАМЕЧАНИЕ

Еще один способ открыть другим пользователям доступ к вашим файлам — выполнить команды useradd или usermod от имени пользователя root, чтобы добавить новых пользователей в группу, контролирующую ваш файл. Но следует подчеркнуть, что и в этом случае команды выполняет пользователь root (соответственно, программа sudo).

```
$ ls -l
-rw-r--r-- 1 scott scott libby_arrowrock.jpg
-rw-r--r-- 1 scott family libby.jpg
-rw-r--r-- 1 scott scott libby_on_couch.jpg
$ chown denise libby.jpg
$ ls -l
-rw-r--r-- 1 scott scott ... libby_arrowrock.jpg
-rw-r--r-- 1 denise family ... libby.jpg
-rw-r--r-- 1 scott scott ... libby_on_couch.jpg
```

Некоторые сведения о команде chgrp, изложенные ранее, справедливы также и для команды chown. Команда chown использует либо имя пользователя, либо его числовой идентификатор. Выяснить числовой идентификатор можно с помощью команды cat /etc/passwd, которая предоставит информацию наподобие следующей:

```
bind:x:110:118::/var/cache/bind:/bin/false
scott:x:1001:1001:Scott,,,:/home/scott:/bin/bash
ntop:x:120:120::/var/lib/ntop:/bin/false
```

Первое из чисел, встречающихся в строке, — это числовой идентификатор пользователя (второе число — идентификатор основной группы, в состав которой входит пользователь).

Изменить владельца файла можно только в том случае, если вы зарегистрировались под именем владельца либо являетесь пользователем root. Хотя такое ограничение очевидно, некоторые пользователи забывают о нем.

ПРЕДУПРЕЖДЕНИЕ

Как и команда chgrp -R, команда chown -R используется для рекурсивного изменения владельца файлов и каталогов. Впрочем, есть одна тонкость. Если вы используете команду chown -R scott *, то файлы, имена которых начинаются с точки, в текущем каталоге не изменятся, и это хорошо. Однако команда chown -R scott .* не только изменит принадлежность этих файлов. Имя .. также соответствует шаблону .*, поэтому изменения коснутся и файлов, находящихся в родительском каталоге. Вряд ли вы стремились получить такой результат!

Изменение владельца и группы для файлов и каталогов

chown владелец:группа

Вы уже знаете, что команда chgrp позволяет изменить группу, а команда chown — владельца. С помощью команды chown можно также решить обе эти задачи одновременно. Для этого в качестве параметра chown надо задать имя пользователя, а за ним имя группы, разделив их двоеточием (такой формат команды chown — одна из причин, по которым не рекомендуется включать двоеточие в имя пользователя или группы). После этого, как обычно, следует имя файла или каталога.

```
$ ls -l
-rw-r--r-- 1 scott scott libby.jpg
$ chown denise:family libby.jpg
```

```
$ ls -l
-rw-r--r-- 1 denise family libby.jpg
```

Вы можете даже изменить с помощью команды chown только группу. Для этого не надо указывать перед двоеточием имя пользователя.

```
$ ls -l
-rw-r--r-- 1 scott scott libby.jpg
$ chown :family libby.jpg
$ ls -l
-rw-r--r-- 1 scott family libby.jpg
```

ПОДСКАЗКА

Что делать, если в имени пользователя или группы есть двоеточие? В этом случае надо предварять данный символ обратной косой чертой, что отменяет его специальное значение. Двоеточие, перед которым указана обратная косая черта, интерпретируется как обычный знак, а не как разделитель между именем пользователя и именем группы.

```
$ chown denise:family\:parents libby.jpg
```

Данный подход позволяет справиться с проблемой, но гораздо лучше будет, если вы вовсе откажетесь от использования двоеточия в имени пользователя или группы.

Поскольку команда chown позволяет выполнять те же функции, что и chgrp, последнюю можно не использовать вовсе.

ЗАМЕЧАНИЕ

Для разделения пользователя и группы можно применять либо точку, либо двоеточие, однако рекомендуется все же отдать предпочтение двоеточию. Точка в качестве разделителя считается устаревшим форматом и не рекомендована к применению.

Общие сведения о правах доступа

Перед тем как приступить к изучению команды chmod, позволяющей изменять права доступа, соответствующие файлу или каталогу, рассмотрим, как система Linux интерпретирует эти права.

ЗАМЕЧАНИЕ

В настоящее время готовится переход системы Linux на использование более гибкой и мощной системы прав доступа, называемой ACL (Access Control Lists). Сейчас ACL используется достаточно редко, поэтому мы не будем рассматривать здесь эту систему. Дополнительную информацию об ACL можно найти в статье *Access Control Lists*, опубликованной в *Linux Magazine* (она доступна по адресу `(www.linux-mag.com/id/1802/)` или в документе "FilePermisionsACLs" на сайте документации сообщества пользователей Ubuntu (`https://help.ubuntu.com/community/FilePermissionsACLs`). Существует также графический редактор среды GNOME GUI для ACL под названием Eiciel (`http://rofi.roger-ferrer.org/eiciel/`). Технические детали можно найти в статье Андреса Грюнбахера (Andreas Grünbacher "POSIX Access Control Lists on Linux" (`www.suse.de/~agruen/acl/linux-acls/online/`). Несмотря на то что она написана в 2003 году, изложенные в ней сведения еще не устарели.

Для каждого файла или каталога Linux различает три категории пользователей: владелец, группа и все остальные пользователи системы. Эти категории перечислены в табл. 8.1, там же указаны буквы, применяющиеся для их обозначения.

Таблица 8.1. Категории пользователей и их обозначение

Категория пользователей	Сокращенное обозначение
Владелец	u
Группа	g
Прочие пользователи	o

В главе 2 мы говорили о правах, указывающих, какие пользователи могут работать с файлами и каталогами. В этом разделе речь пойдет о трех атрибутах, соответствующих чтению, записи и выполнению. Для их обозначения применяются буквы r, w и x. Кроме того, существуют также признаки suid, sgid и специальный бит, называемый "битом закрепления в памяти" ("sticky bit")". Они представляются соответственно буквами s (или в некоторых случаях S), s (или S) и t (или T). Учтите, что эти признаки имеют различное значение в зависимости от того, относятся ли они к файлу или каталогу. В табл. 8.2 перечислены атрибуты, их сокращенное представление и значение.

Таблица 8.2. Атрибуты, определяющие доступ

Атрибут	Сокращенное представление	Значение для файла	Значение для каталога
Допускает чтение	r	Можно читать	Можно просматривать содержимое с помощью команды ls
Допускает запись	w	Можно редактировать	Можно удалять, переименовывать или добавлять файлы
Выполняемый	x	Можно запускать на выполнение	Можно читать файлы и каталоги и запускать файлы на выполнение
suid	s	Любой пользователь может запустить файл на выполнение с правами его владельца	Не применяется
sgid	s	Любой пользователь может запустить файл на выполнение с правами группы	Все файлы, вновь создаваемые в каталоге, принадлежат группе, владеющей каталогом

Окончание табл. 8.2

Атрибут	Сокращенное представление	Значение для файла	Значение для каталога
`suid` или `sgid`, но не `x`	`S`	Любой пользователь может выполнить файл с разрешения его владельца (если задан атрибут `suid`) или группы (если задан атрибут `sgid`), но файл не является исполняемым	Не применяется
"sticky bit"	`t`	Не применяется в системе Linux	Удалять или переименовывать файлы, находящиеся в каталоге, имеют право только их владельцы или владелец каталога
"sticky bit", но не `x`	`T`	Не применяется в системе Linux	Пользователь может только удалять или переименовывать файлы, но не имеет доступа для чтения файлов и подкаталогов

ЗАМЕЧАНИЕ

Пользователь `root` может выполнять любые действия с любым файлом или каталогом, поэтому сведения из табл. 8.2 не применимы к нему. (Конечно, пользователь `root` не может волшебным образом выполнить невыполняемые файлы, поэтому слово "любые" — это преувеличение, но вы меня понимаете?)

В последующих разделах мы подробнее рассмотрим указанные атрибуты. Теперь, когда вы знакомы с основами, обсудим использование команды `chmod` для изменения прав доступа к файлам и каталогам.

Изменения прав доступа к файлам и каталогам с использованием символьных обозначений

```
chmod [ugo][+-=][rwx]
```

Для команды chmod предусмотрены два типа обозначений: символьные и числовые. Оба имеют свои преимущества, однако для пользователя проще сначала изучить символьное представление. Для формирования символьных обозначений используется простой принцип: категория пользователей, на которую необходимо воздействовать, затем символ "плюс" (+) для назначения, "минус" (-) для удаления или знак равенства (=) для конкретной установки прав, а затем буквы (r, w, x, s, t), представляющие права, которые необходимо изменить. Предположим, например, что вам надо предоставить членам группы family право модифицировать изображение.

```
$ ls -l
-rw-r--r-- 1 scott family libby.jpg
$ chmod g+w libby.jpg
$ ls -l
-rw-rw-r-- 1 scott family libby.jpg
```

Как видите, это достаточно просто. А что делать, если вы хотите дать право на запись в файл не только членам группы family, но и всем остальным пользователям?

```
$ ls -l
-rw-r--r-- 1 scott family libby.jpg
$ chmod go+w libby.jpg
$ ls -l
-rw-rw-rw- 1 scott family libby.jpg
```

Предоставляя всем пользователям — владельцу, группе и остальным — права чтения и записи, вы можете также сделать это и по-другому.

```
$ ls -l
-rw-r--r-- 1 scott family libby.jpg
$ chmod a=rw libby.jpg
$ ls -l
-rw-rw-rw- 1 scott family libby.jpg
```

Предположим, вы заметили ошибку и хотите лишить группу `family` и всех остальных пользователей права изменять изображение, более того, вы хотите сделать так, чтобы пользователи, не являющиеся владельцем и членами группы, не могли даже просматривать картинку.

```
$ ls -l
-rw-rw-rw- 1 scott family libby.jpg
$ chmod go-w libby.jpg
$ ls -l
-rw-r--r-- 1 scott family libby.jpg
$ chmod o-r libby.jpg
$ ls -l
-rw-r----- 1 scott family libby.jpg
```

Вместо символа - можно использовать знак равенства.

```
$ ls -l
-rw-rw-rw- 1 scott family libby.jpg
$ chmod g=r libby.jpg
$ ls -l
-rw-r--rw- 1 scott family libby.jpg
$ chmod o= libby.jpg
$ ls -l
-rw-r----- 1 scott family libby.jpg
```

Заметьте, что в последней команде `chmod` после знака равенства, следующего за `o`, не указан никакой символ. Этим удаляются все права для остальных пользователей системы. Как видите, необходимый результат достигается быстро и эффективно.

Основной недостаток алфавитной системы виден из последнего примера: если вы хотите внести разные изменения для двух групп пользователей, вам надо запустить `chmod` два раза. Данную проблему можно устранить посредством числовых обозначений, которые будут рассмотрены в следующем разделе.

Изменения прав доступа к файлам и каталогам с использованием числовых обозначений

```
chmod [0-7][0-7][0-7]
```

Для числовых обозначений применяется восьмеричная система счисления. Мы не будем обсуждать, почему используются те или иные числа, к тому же изменить существующее положение дел не в наших силах. Рассмотрим лишь их значения. Праву на чтение (`r`) соответствует значение 4, праву на запись (`w`) — значение 2 и праву на выполнение (`x`) — значение 1. Как вы знаете, права доступа Linux предполагают наличие трех категорий пользователей: владелец, группа и все остальные. Для каждой категории можно разрешить чтение, запись и выполнение так, как показано в табл. 8.3.

Таблица 8.3. Права доступа и их числовое представление

	Владелец	Группа	Остальные пользователи
Символьное представление	`r; w; x`	`r; w; x`	`r; w; x`
Числовое представление	4;2;1	4;2;1	4;2;1

Назначение прав по данной схеме сводится к обычному сложению чисел. Рассмотрим несколько примеров.

- Пользователь имеет право на чтение и запись в файл или каталог. Чтению соответствует значение 4, записи — 2. Поскольку права на выполнение нет, соответствующее значение равно нулю. 4 + 2 + 0 = 6.
- Пользователь имеет право читать файл и запускать его на выполнение. Чтению соответствует значение 4, записи, поскольку она запрещена, — 0. Выполнение обозначается числом 1. 4 + 0 + 1 = 5.

- Пользователь имеет право на чтение, запись и выполнение для каталога. Чтению соответствует значение 4, записи — 2, а выполнению — 1. 4 + 2 + 1 = 7.

Нетрудно заметить, что максимально возможное значение для одной категории пользователей равно 7 (чтение, запись и выполнение разрешены), а минимальное — 0 (чтение, запись и выполнение запрещены). Поскольку существуют три категории пользователей, права доступа задаются тремя цифрами; каждая из них принимает значение от 0 до 7. В табл. 8.4 представлены все допустимые числа для одной категории и их значения.

Таблица 8.4. Числовое представление прав доступа, отображаемых с помощью команды `ls -l`

Числовое представление	Представление `ls -l`
0	`---`
1	`--x`
2	`-w-`
3	`-wx`
4	`r--`
5	`r-x`
6	`rw-`
7	`rwx`

Несмотря на то что допустимо чрезвычайно большое число конкретных сочетаний прав, некоторые из них встречаются чаще других. Эти права и их значение приведены в табл. 8.5.

Таблица 8.5. Часто встречающиеся сочетания прав

Команда `chmod`	Представление `ls -l`	Описание
chmod 400	`-r--------`	Пользователь имеет право чтения; никто другой не имеет права выполнять никакие действия
chmod 644	`-rw-r--r--`	Все пользователи имеют право чтения; владелец может редактировать

Команда chmod	Представление ls -l	Описание
chmod 660	`-rw-rw----`	Владелец и группа могут читать и редактировать; остальные не имеют права выполнять никакие действия
chmod 664	`-rw-rw-r--`	Все пользователи имеют право чтения: владелец и группа могут редактировать
chmod 700	`-rwx------`	Владелец может читать, записывать и запускать на выполнение; никто другой не имеет права выполнять никакие действия
chmod 744	`-rwxr--r--`	Каждый пользователь может читать; владелец имеет право редактировать и запускать на выполнение
chmod 755	`-rwxr-xr-x`	Каждый пользователь имеет право читать и запускать на выполнение; пользователь может редактировать
chmod 777	`-rwxrwxrwx`	Каждый пользователь может читать, редактировать и запускать на выполнение (устанавливать такой набор прав не рекомендуется)

ПРЕДУПРЕЖДЕНИЕ

В принципе для файла или каталога может быть выполнена команда `chmod 000`. В этом случае единственным пользователем, имеющим право на выполнение каких-либо действий, в том числе на вызов команды `chmod`, остается пользователь `root`. Однако и пользователь `root`, и владелец файла могут по-прежнему использовать команду `chmod` для отмены разрешений и возврата к обычной ситуации. Команда `chmod 000` изменяет состав пользователей, имеющих право *вносить изменения в файл*, но не может изменить состав пользователей, дающих разрешения *на изменение этих файлов*.

Возможно, вам не совсем понятно, чем же хороши числовые обозначения. Действительно, чтобы разобраться в них, надо приложить больше усилий, чем в случае с символьными

обозначениями, однако они позволяют одновременно задавать любой набор прав. Вернемся к примерам назначения прав для файлов и каталогов, рассмотренных ранее, но вместо символьных используем числовые обозначения.

Предположим, например, что вам надо предоставить членам группы `family` право изменять изображение.

```
$ ls -l
-rw-r--r-- 1 scott family libby.jpg
$ chmod 664 libby.jpg
$ ls -l
-rw-rw-r-- 1 scott family libby.jpg
```

А что делать, если вы хотите предоставить право на запись в файл не только членам группы `family`, но и всем остальным пользователям?

```
$ ls -l
-rw-r--r-- 1 scott family libby.jpg
$ chmod 666 libby.jpg
$ ls -l
-rw-rw-rw- 1 scott family libby.jpg
```

Теперь вы заметили ошибку и хотите лишить группу `family` и всех остальных пользователей права изменять изображение, а также хотите сделать так, чтобы пользователи, не являющиеся владельцами и членами группы, не могли даже просматривать картинку.

```
$ ls -l
-rw-rw-rw- 1 scott family libby.jpg
$ chmod 640 libby.jpg
$ ls -l
-rw-r----- 1 scott family libby.jpg
```

На этих примерах хорошо видно основное преимущество числовых обозначений. Там, где при использовании символьных обозначений установка прав происходит в два этапа (`chmod go-w`, а затем `chmod o-r`, или `chmod g-r` и `chmod o=`), числовые обозначения позволяют обойтись одной командой. По этой причине специалисты, хорошо разбирающиеся в системе

Linux, используют числовые обозначения, что позволяет им быстрее назначать требуемые права.

ЗАМЕЧАНИЕ

Если вы хотите использовать восьмеричные обозначения, то воспользуйтесь очень удобным калькулятором `permissions-calculator.org`.

Рекурсивное изменение прав

`chmod -R`

Вы, вероятно, заметили, что многие команды Linux можно рекурсивно применять к файлам и каталогам. Команда `chmod` — не исключение. Опция `-R` (или `--recursive`) позволяет мгновенно воздействовать на сотни объектов файловой системы, надо лишь быть уверенным, что вам действительно необходимо изменить права для них всех.

```
$ pwd
/home/scott/pictures/libby

$ ls -lF
drwxrw---- 2 scott scott    120 by_pool/
-rw-r--r-- 1 scott scott  73786 libby_arrowrock.jpg
-rw-r--r-- 1 scott scott  18034 libby.jpg
drwxrw---- 2 scott scott    208 on_floor/
$ ls -l *
-rw-r--r-- 1 scott scott  73786 libby_arrowrock.jpg
-rw-r--r-- 1 scott scott  18034 libby.jpg

by_pool:
-rw-r--r-- 1 scott scott 413929 libby_by_pool_02.jpg
-rwxr-xr-x 2 scott scott  64973 lieberman_pool.jpg

on_floor:
-rw-r--r-- 1 scott scott 218849 libby_on_floor_01.jpg
```

```
-rw-r--r-- 1 scott scott 200024 libby_on_floor_02.jpg
$ chgrp -R family *
$ chmod -R 660 *
chmod: 'by_pool' : Permission denied
chmod: 'on_floor' : Permission denied
```

Однако в данном примере было получено сообщение `Permission denied`. Что случилось? Обратите внимание на табл. 8.2. Если файл является исполняемым, это означает, что его можно запустить как программу; для каталога же соответствующий признак указывает на то, что пользователи могут обращаться к его содержимому, читать файлы и подкаталоги. Команда `chmod -R 660 *` удаляет право x и для файлов, и для каталогов, поэтому она не может сообщить о том, что задача выполнена: для каталогов уже сброшен признак x, и они недоступны.

Что же делать в этом случае? Найти ответ непросто. Можно использовать при вызове `chmod` символы групповых операций, но подбирать их так, чтобы команда воздействовала только на файлы определенного типа.

```
$ chmod -R 660 *.jpg
```

В данном примере права будут изменены только для изображений, но не для каталогов. Если в вашем распоряжении есть файлы различных типов, задача становится рутинной и отнимает много времени; вам придется выполнить команду `chmod` для каждого типа файлов.

Если в каталоге есть много подкаталогов или файлы различных типов и если вы приобрели достаточный опыт работы с Linux, используйте команду `find` для поиска файлов, не являющихся подкаталогами, а затем изменяйте права только для них. Команда `find` будет подробно рассмотрена в главе 11.

Пока что можно ограничиться лишь следующей рекомендацией: изменяя права доступа рекурсивно, будьте осторожны. Вы можете получить неожиданный результат и больше не сможете обратиться к файлу или каталогу.

Установка и сброс параметра suid

```
chmod [+-]u
chmod 4[0-7][0-7][0-7]
```

Ранее в этой главе мы рассматривали различные права доступа. Мы сосредоточили внимание на параметрах r, w и x, поскольку они применяются наиболее часто. Однако существуют и другие права, которые в ряде случаев могут быть очень полезными. Рассмотрим признак suid, который применяется для файлов, причем исполняемых. Для каталога признак suid не может быть указан.

Установленный признак suid означает, что пользователь может запустить файл на выполнение с правами пользователя, владеющего им, т.е. так, как будто этот файл был выполнен самим владельцем. В качестве примера применения suid можно привести права для команды passwd. Такой подход позволяет пользователям изменять собственные пароли.

```
$ ls -l /usr/bin/passwd
-rwsr-xr-x 1 root root ... /usr/bin/passwd
```

На тот факт, что для команды passwd установлен признак suid, указывает символ s, расположенный в той позиции, в которой располагается символ x для владельца файла. Владельцем passwd является пользователь root, но данная команда должна быть доступна для обычных пользователей, иначе они не смогут изменять свои пароли. С этой целью право x установлено для группы и для всех остальных пользователей. Однако этого недостаточно. Надо установить для passwd признак suid, чтобы любой пользователь, вызвавший эту команду, получал на время ее выполнения права root.

ЗАМЕЧАНИЕ

Для обозначения установленного признака suid может использоваться как строчная буква s, так и

прописная S. Строчная буква отображается в том случае, если перед установкой suid пользователь уже имел право на выполнение (x), а прописная буква S — если пользователь не имел такого права. Конечный результат будет один и тот же, но регистр символа дает дополнительную информацию об исходных установках.

Скажем честно, рядовому пользователю вряд ли придется часто изменять параметр suid. Это небезопасно, поэтому сначала стоит серьезно подумать, надо ли это делать! Впрочем, знания не бывают лишними, и, возможно, они вам когда-нибудь понадобятся.

ПРЕДУПРЕЖДЕНИЕ

Не хотите ли выяснить, сколько программ в вашем дистрибутивном пакете Linux устанавливают атрибут suid? Выполните следующую команду (на моем сервере только 22 программы делают это, так что все не так уж плохо).

```
$find / -xdev -perm -4000 -type f
↳ - print0 | xargs -0 ls -l | wc -l
```

Устанавливать и сбрасывать признак suid можно двумя способами: используя символьные либо числовые обозначения. Допустим, вы вошли в систему под именем root и инсталлировали программу foobar, которой необходимо устснавливать бит suid. Команда, в которой применены символьные обозначения, выглядит так:

```
$ ls -l
-rwxr-xr-- 1 admins foobar
$ chmod u+s foobar
$ ls -l
-rwsr-xr-- 1 admins foobar
```

После завершения команды каждый член группы admins получает возможность запускать программу foobar на выполнение от имени пользователя root. Заметьте также, что пользователи, не принадлежащие группе admins, не могут сделать этого; они имеют право только на чтение файла. Если вам надо, чтобы любой пользователь мог запускать программу foobar от имени пользователя root, установите права -rwsr-xr-x.

Для того чтобы сбросить признак suid, надо использовать параметр u- вместо u+.

```
$ ls -l
-rwsr-xr-- 1 scott admins foobar
$ chmod u-s foobar
$ ls -l
-rwxr-xr-- 1 root admins foobar
```

Использовать восьмеричные числа несколько сложнее, поскольку в данном случае вступают в действие дополнительные числовые признаки. До сих пор мы говорили о трехсимвольном наборе признаков, в котором первый символ представлял владельца файла, второй — группу, третий — остальных пользователей. Однако на самом деле в команде chmod предусмотрена четвертая цифра, расположенная слева от цифры, представляющей владельца файла. В большинстве случаев это цифра 0, поэтому указывать ее нет необходимости. Действительно, chmod 644 libby.jpg и chmod 0644 libby.jpg — одно и то же. Однако, когда возникает необходимость изменить признак suid (а также sgid или "sticky bit", которые будут рассмотрены в следующих разделах), эта цифра вступает в действие.

Для установки права suid используется значение 4, поэтому, если вы хотите изменить права для backup_data, используя числовые обозначения, команда будет выглядеть так, как показано ниже.

```
$ pwd
/home/scott/bin
$ ls -l
-rwxr-xr-- 1 root admins foorbar
$ chmod 4754 backup_data
```

```
$ ls -l
-rwsr-xr-- 1 root admins foorbar
```

Для сброса suid используется значение, равное 0.

```
$ ls -l
-rwsr-xr-- 1 root admins foorbar
$ chmod 0754 foobar
$ ls -l
-rwxr-xr-- 1 root admins foorbar
```

ПОДСКАЗКА

Следует знать об одной важной особенности атрибута suid: он не работает ни с одной *интерпретируемой* программой, т.е. ни с одним сценарием, который начинается с символов #!. Если вам все же потребуется такая возможность, будьте осторожны! Дополнительную информацию можно получить по адресу www.faqs.org/faqs/unix-faq/faq/part4/section-7.html.

Установка и сброс признака sgid

```
chmod [+-]s
chmod 2[0-7][0-7][0-7]
```

Признак sgid похож на suid, однако он применим не только к файлам, но и к каталогам. Для файлов признак sgid действует так же, как suid, за исключением того, что пользователь запускает файл на выполнение не с правами владельца, а с правами группы. Например, в вашей системе для файла crontab, вероятнее всего, установлен признак sgid, поэтому программы, заданные для cron, будут выполняться не с правами root, а с гораздо более ограниченными правами группы.

```
$ ls -l /usr/bin/crontab
-rwxr-sr-x 1 root crontab /usr/bin/crontab
```

Применительно к каталогам sgid делает следующее: для каждого очередного файла, создаваемого в каталоге, устанавливается принадлежность группе. Поясним сказанное на примере.

Предположим, в вашей системе зарегистрированы три пользователя, alice, bob и carol, которые являются членами группы admins. Для пользователя alice основной группой является alice. По тому же принципу формируются основные группы для пользователей bob и carol. Если alice создаст файл в каталоге, совместно используемом группой admins, то для него будут установлены владелец alice и группа alice. Это означает, что остальные члены группы admins не смогут записывать информацию в данный файл. Конечно же, alice может после создания файла выполнить команду chgrp admins document (или chown :admins document), однако поступать так каждый раз довольно утомительно.

Если же для каталога установлен признак sgid, каждый новый файл, созданный в этом каталоге, будет принадлежать пользователю, который создал файл, и той группе, которой принадлежит сам каталог, в данном случае admins. В результате alice, bob и carol смогут читать и редактировать любой файл, содержащийся в этом каталоге, причем для этого не потребуется прилагать дополнительные усилия.

Как и в случае suid, вы можете устанавливать признак sgid, используя либо символьные, либо числовые обозначения. Символьные обозначения применяются по тем же правилам, что и для признака suid, только вместо буквы u указывается g. Рассмотрим процесс установки признака sgid для каталога. Для файла этот признак задается точно так же.

```
$ ls -lF
drwxr-xr-x 11 scott admins bin/
$ chmod g+s bin
$ ls -lF
drwxr-Sr-x 11 scott admins bin/
```

ЗАМЕЧАНИЕ

Для обозначения установленного признака sgid может использоваться как строчная буква s, так и прописная S. Строчная буква отображается в том случае, если перед установкой sgid группа уже имела право на выполнение (x), а прописная S — если группа не имела такого права. Конечный результат будет один и тот же, но регистр символа дает дополнительную информацию об исходных установках.

Процесс удаления признака sgid противоположен процессу его установки.

```
$ ls -lF
drwxr-Sr-x 11 scott admins bin/
$ chmod g-s bin
$ ls -lF
drwxr-xr-x 11 scott admins bin/
```

Если вы не читали предыдущий раздел, посвященный признаку suid, прочитайте его, иначе вам будет непонятно назначение первой из четырех цифр, указанных в команде chmod. Если для suid это была цифра 4, то для sgid используется значение 2.

```
$ ls -lF
drwxr-xr-x 11 scott admins bin/
$ chmod 2755 bin
$ ls -lF
drwxr-Sr-x 11 scott admins bin/
```

Удаляется признак sgid так же, как и suid: первая цифра задается равной 0.

```
$ ls -lF
drwxr-Sr-x 11 scott admins bin/
$ chmod 0755 bin
$ ls -lF
drwxr-xr-x 11 scott admins bin/
```

ЗАМЕЧАНИЕ

Вы уже знаете, что происходит с новым файлом, который создается в каталоге, имеющем признак `sgid`. Однако необходимо заметить, что `sgid` влияет (причем по-разному) и на другие операции с файловой системой. Если вы скопируете файл в sgid-каталог посредством команды `cp`, он будет принадлежать той же группе, что и sgid-каталог. Если вы переместите в такой каталог файл, используя команду `mv`, он сохранит принадлежность прежней группе. И наконец, если вы создадите посредством команды `mkdir` новый каталог в составе каталога, имеющего признак `sgid`, он не унаследует группу sgid-каталога, но сам будет иметь признак `sgid`.

Установка и сброс признака "sticky bit"

```
chmod [+-]t
chmod 1[0-7][0-7][0-7]
```

Чем замечателен признак "sticky bit", помимо непереводимого названия? На заре Unix, если данный признак был установлен для исполняемого файла, система знала, что файл используется часто, поэтому хранила его в области подкачки, обеспечивая быстрый и эффективный доступ к нему. Linux — более современная система, она игнорирует "sticky bit", установленный для файлов.

В настоящее время "sticky bit" применяется только к каталогам. Если данный признак установлен для каталога, то удалять и переименовывать содержащиеся в нем файлы может только владелец файлов или самого каталога. В обычных условиях разрешение записи в каталог также означает право изменять и переименовывать содержимое каталога. Признак "sticky bit" отменяет такое право. В качестве примера каталога, в который каждый желающий может записывать файл, можно

привести /tmp. В этом же каталоге каждый файл и подкаталог защищен от остальных пользователей посредством признака "sticky bit".

```
$ ls -l /
drwxrwxrwt 12 root root ... tmp
[Результаты сокращены для экономии места]
```

ЗАМЕЧАНИЕ

Для обозначения установленного признака "sticky bit" может использоваться как строчная буква t, так и прописная T. Строчная буква отображается в том случае, если перед установкой "sticky bit" произвольный пользователь уже имел право на выполнение (x), а прописная T — если такого права у него не было. Конечный результат один и тот же, но регистр символа дает дополнительную информацию об исходных установках.

Подобно многим другим вариантам использования команды chmod, для установки признака "sticky bit" можно использовать либо буквенные, либо числовые обозначения.

```
$ ls -lF
drwxrwxr-x 2 scott family ... libby_pix/
$ chmod +t libby_pix
$ ls -lF
drwxrwxr-t 2 scott family ... libby_pix/
```

Однако процесс установки признака "sticky bit" имеет две существенные особенности. Во-первых, несмотря на то, что в предыдущих случаях необходимо было указать, для какой категории пользователей (u, g или o) задается соответствующее право, при установке "sticky bit" в этом нет необходимости. Достаточно указать +t. Во-вторых, символ t отображается в той позиции, в которой обычно выводится символ x для пользователей, не являющихся владельцами и не принадлежащих группе. Даже если запись в каталог разрешается только членам

группы, все равно удалять или переименовывать файл имеет право только его владелец.

Удаление "sticky bit" происходит так, как вы, вероятно, и ожидаете.

```
$ ls -lF
drwxrwxr-t 2 scott family libby_pix
$ chmod -t libby_pix
$ ls -lF
drwxrwxr-x 2 scott family libby_pix
```

Установка признака "sticky bit" с помощью числовых обозначений предполагает указание четвертой цифры в составе параметра, передаваемого команде `chmod`. Если для `suid` это 4, а для `sgid` — 2, то для "sticky bit" используется значение 1.

```
$ ls -lF
drwxrwxr-x 2 scott family libby_pix/
$ chmod 1775 libby_pix
$ ls -lF
drwxrwxr-t 2 scott family libby_pix/
```

Значение 0, как и в предыдущих случаях, сбрасывает "sticky bit".

```
$ ls -lF
drwxrwxr-t 2 scott family ... libby_pix/
$ chmod 0775 libby_pix
$ ls -lF
drwxrwxr-x 2 scott family ... libby_pix/
```

Вы вряд ли будете часто использовать признак "sticky bit" на рабочей станции, но для серверов он очень удобен. Без него некоторые задачи, связанные с контролем доступа, было бы достаточно трудно решить.

ПОДСКАЗКА

Для ускорения работы можно одновременно устанавливать из командной строки признаки `suid`, `sgid` и "sticky bit". Подобно тому, как вы объединяете путем сложения значения 4 (чтение), 2 (запись) и 1 (выполнение),

определяющие права пользователя, можно объединить suid, sgid и "sticky bit".

Числовое значение	Описание
0	Сбрасывает suid, sgid и "sticky bit"
1	Устанавливает "sticky bit"
2	Устанавливает sgid
3	Устанавливает "sticky bit" и sgid
4	Устанавливает suid
5	Устанавливает "sticky bit" и suid
6	Устанавливает sgid и suid
7	Устанавливает "sticky bit", sgid и suid

Заметьте также, что значение 0 одновременно удаляет suid, sgid и "sticky bit". Если вы использовали 0 для удаления suid, но хотите сохранить "sticky bit", вам надо повторно установить данный признак.

Выводы

Права доступа — чрезвычайно важный элемент системы защиты; по сути, он определяет работоспособность Linux. Зная принцип действия прав доступа, достаточно просто научиться устанавливать их в соответствии с собственными потребностями. Совместно команды chgrp (изменение принадлежности группе), chown (установка владельца, а также группы) и чрезвычайно мощная команда chmod формируют набор инструментов, позволяющих эффективно устанавливать права доступа.

9

Архивирование и сжатие данных

Некоторые пользователи считают, что архивация файлов и их сжатие — одно и то же, однако эти понятия необходимо различать. Если вы берете десять файлов и, не изменяя их размеры, объединяете в один файл, вы создаете архив. Пусть каждый из десяти файлов имеет размер 100 Кбайт, тогда размер полученного в результате архива будет равен 1000 Кбайт. Если вы выполните сжатие тех же десяти файлов, то получите по-прежнему десять файлов, размеры которых, в зависимости от типа исходных данных, будут составлять от нескольких до 100 Кбайт.

ЗАМЕЧАНИЕ

В принципе файл, получившийся в результате сжатия, может быть даже больше исходного файла. Если файл уже был сжат ранее, реального сжатия не произойдет, но в состав файла будет дополнительно включена служебная информация.

Все средства архивирования и сжатия, рассмотренные в данной главе (`zip`, `gzip`, `bzip2` и `tar`), широко используются на практике, однако самым популярным все же является инструмент `zip`. Он не только реализует стандартный формат для системы Windows, но и широко применяется в подавляющем большинстве операционных систем, поэтому zip-файл, созданный, например, в Linux, можно прочитать в Mac OS. Если ваш архив предназначен для некоторого пользователя и вы не знаете, с какой операционной системой он работает, лучше всего выбрать формат `zip`.

Программа gzip была разработана как проект с открытым исходным кодом и предназначалась на замену программы compress, которая использовалась ранее в системе Unix. Программа gzip имеется практически на каждом компьютере под управлением Unix, а также поддерживается в системе Mac OS X. В Windows она встречается значительно реже. Если вы собираетесь перемещать файлы между различными компьютерами, на которых установлена система Unix, можете смело использовать gzip.

Команда bzip2 сравнительно новая. Ее разработчики стремились получить результаты, лучшие по сравнению с gzip. Действительно, bzip2 создает файлы меньшего размера, но достигается это за счет увеличения времени обработки. В настоящее время быстродействие компьютеров настолько возросло, что большинство пользователей не заметят различия во времени при сжатии группы файлов посредством gzip и bzip2.

ЗАМЕЧАНИЕ

В Википедии опубликована хорошая статья, в которой сравниваются многочисленные форматы сжатия. Статья доступна по адресу http://www.wn.wikipedia.org/wiki/List_of_archive_formats.

Команды zip, gzip и bzip2 предназначены для сжатия данных (zip также может создавать архивы). Команда tar решает одну задачу — архивирование; именно для этого она разрабатывалась и используется длительное время. Она применяется в основном в системе Unix. Если в системе Linux вы копируете с сервера исходный код, он почти наверняка будет представлен в виде tar-архива.

ЗАМЕЧАНИЕ

В ходе подготовки книги ко второму изданию я удалил разделы о команде gzip -[0-9] (позволяющей

задавать степень сжатия с помощью утилиты gzip) и команде bzip2 -[0-9] (которая делает то же самое, только с помощью утилиты bzip2). Исходный текст можно найти на моем веб-сайте (www.granneman.com/linux-redactions).

Архивирование и сжатие файлов посредством программы zip

zip

Программа zip, которая с 1989 года стала практически вездесущей, предназначена как для архивирования, так и для сжатия файлов. Она удобна, например, для передачи нескольких файлов в составе почтового сообщения, формирования резервных копий или для хранения на диске данных, которые используются достаточно редко. Пользоваться программой zip очень просто. Предположим, вы хотите передать изображение в формате TIFF по электронной почте. Изображение TIFF хранится в несжатом виде, поэтому файл имеет большой размер. Обработав такой файл программой zip, вы получите файл меньшего размера, который можно присоединить к электронному письму.

ЗАМЕЧАНИЕ

Для экономии места из результатов, полученных по команде ls -l, представлены только те, которые имеют отношение к рассматриваемому примеру.

```
$ ls -lh
-rw-r--r-- scott scott 1006K young_edgar_scott.tif
$ zip grandpa.zip young_edgar_scott.tif
  adding: young_edgar_scott.tif (deflated 19%)
$ ls -lh
-rw-r--r-- scott scott 1006K young_edgar_scott.tif
-rw-r--r-- scott scott 819K grandpa.zip
```

В данном примере мы сэкономили около 200 Кбайт или, по сообщению zip, 19% объема. Не так плохо. То же самое можно сделать для нескольких изображений.

```
$ ls -l
-rw-r--r--  scott  scott  251980 edgar_intl_
↳shoe.tif
-rw-r--r--  scott  scott  1130922 edgar_baby.tif
-rw-r--r--  scott  scott  1029224 young_edgar_scott.tif
$ zip grandpa.zip edgar_intl_shoe.tif edgar_
↳baby.tif young_edgar_scott.tif
  adding: edgar_intl_shoe.tif (deflated 4%)
  adding: edgar_baby.tif (deflated 12%)
  adding: young_edgar_scott.tif (deflated 19%)
$ ls -l
-rw-r--r--  scott  scott  251980 edgar_intl_
↳shoe.tif
-rw-r--r--  scott  scott  1130922  edgar_baby.tif
-rw-r--r--  scott  scott  2074296  grandpa.zip
-rw-r--r--  scott  scott  1029224  young_edgar_scott.tif
```

Указывать программе zip отдельные файлы — не лучшее решение. Для трех файлов это приемлемо. Пользователь, выполнив команду unzip grandpa.zip, получит всего три файла. Если же в архиве содержится пятьдесят файлов, то после разархивирования файлы, извлеченные из архива, смешаются с теми файлами, которые уже находились в каталоге, и разобраться с ними будет непросто. Гораздо лучше поместить эти пятьдесят файлов в отдельный каталог, который пользователь и получит в результате разархивирования.

```
$ ls -lF
drwxr-xr-x  scott  scott  edgar_scott/
$ zip grandpa.zip edgar_scott
  adding: edgar_scott/ (stored 0%)
  adding: edgar_scott/edgar_baby.tif (deflated 12%)
  adding: edgar_scott/young_edgar_scott.tif
↳(deflated 19%)
  adding: edgar_scott/edgar_intl_shoe.tif
↳(deflated 4%)
$ ls -lF
```

```
drwxr-xr-x scott scott        160 edgar_scott/
-rw-r--r--  scott  scott  2074502  grandpa.zip
```

Независимо от того, архивируется ли файл, несколько файлов или каталог, выполняемые действия одинаковы: задается команда `zip`, за ней указывается имя zip-архива, а после него объект (или объекты), который необходимо добавить к архиву.

Повышение уровня сжатия с помощью программы zip

```
zip -[0-9]
```

Программе `zip` можно задавать различные степени сжатия данных. Эти степени обозначаются числами от 0 до 9, причем 0 означает отсутствие сжатия (в этом случае `zip` работает подобно программе `tar`, которую мы рассмотрим ниже), 1 задает большую производительность и малую степень сжатия, а 9 соответствует максимальной степени сжатия, достигаемой за счет снижения скорости обработки данных. По умолчанию принимается степень 6, однако быстродействие современных компьютеров велико, и всегда можно использовать степень 9.

Предположим, вас заинтересовало произведение Мелвилла "Моби Дик" и вы хотите собрать тексты, которые помогут вам понять эту книгу: само произведение "Моби Дик", "Утерянный рай" Мильтона и Библию. Сравним результаты при различных степенях сжатия.

```
$ ls -l
-rw-r--r--  scott  scott   102519  job.txt
-rw-r--r--  scott  scott  1236574  moby-dick.txt
-rw-r--r--  scott  scott   508925  paradise_lost.txt
$ zip -0 moby.zip *.txt
  adding: job.txt (stored 0%)
  adding: moby-dick.txt (stored 0%)
  adding: paradise_lost.txt (stored 0%)
$ ls -l
```

```
-rw-r--r--  scott  scott   102519  job.txt
-rw-r--r--  scott  scott  1236574  moby-dick.txt
-rw-r--r--  scott  scott  1848444  moby.zip
-rw-r--r--  scott  scott   508925  paradise_lost.txt
$ zip -1 moby.zip *txt
updating: job.txt (deflated 58%)
updating: moby-dick.txt (deflated 54%)
updating: paradise_lost.txt (deflated 50%)
$ ls -l
-rw-r--r--  scott  scott   102519  job.txt
-rw-r--r--  scott  scott  1236574  moby-dick.txt
-rw-r--r--  scott  scott   869946  moby.zip
-rw-r--r--  scott  scott   508925
  paradise_lost.txt
$ zip -9 moby.zip *txt
updating: job.txt (deflated 65%)
updating: moby-dick.txt (deflated 61%)
updating: paradise_lost.txt (deflated 56%)
$ ls -l
-rw-r--r--  scott  scott   102519  job.txt
-rw-r--r--  scott  scott  1236574  moby-dick.txt
-rw-r--r--  scott  scott   747730  moby.zip
-rw-r--r--  scott  scott   508925  paradise_lost.txt
```

В виде таблицы результаты можно представить следующим образом.

Книга	zip -0	zip -1	zip -9
Моби Дик	0%	54%	61%
Утерянный рай	0%	50%	56%
Библия	0%	58%	65%
Общий размер (в байтах)	1848444	869946	747730

Степень сжатия существенно зависит от типа файла (тексты обычно сжимаются очень хорошо) и размеров исходных файлов. Очевидно, что лучше использовать степень сжатия, равную 9. Другое решение целесообразно, если ваш компьютер имеет очень низкое быстродействие, либо если сами вы крайне нетерпеливы.

ЗАМЕЧАНИЕ

Целесообразно создать в файле `.bashrc` псевдоним для команды `zip`. Соответствующая запись будет выглядеть следующим образом:

```
alias zip='zip -9'
```

Теперь при каждом вызове `zip` будет автоматически задаваться максимальная степень сжатия.

Архивирование и сжатие файлов конкретного типа в каталогах и подкаталогах

```
zip -l
zip -r
```

Команду `zip` легко использовать для сжатия файлов и каталогов. А что если вы хотите архивировать файлы определенного типа? А если эти файлы находятся в подкаталогах? Команду `zip` можно применять для решения обеих задач. Рассмотрим их по-отдельности. В любом случае предположим, что структура каталога представлена в следующем коде:

```
$ ls reading/*
Authors_and_Texts.txt
reading/lovecraft:

Beyond the Wall of Sleep.jpg
Beyond the Wall of Sleep.txt
The Call of Cthulhu.jpg
The Call of Cthulhu.txt

reading/machen:
The Great God Pan.jpg
The Great God Pan.txt
```

Обратите внимание на то, что в каталоге lovecraft находятся файлы двух видов: txt и jpg. Если хотите заархивировать только файлы типа txt, выполните следующие команды:

```
$ cd reading/lovecraft
$ zip lovecraft.zip . -i \*.txt
  adding: Beyond the Wall of Sleep.txt (deflated 57%)
  adding: The Call of Cthulhu.txt (deflated 58%)
$ ls
Beyond the Wall of Sleep.jpg
Beyond the Wall of Sleep.txt
The Call of Cthulhu.jpg
The Call of Cthulhu.txt
lovecraft.zip
```

Стоит обратить внимание на три аспекта.

- Для того чтобы указать команде zip, где следует выполнить архивирование, используйте точку (.), которая означает текущий каталог (см. главу 2).
- Для того чтобы указать команде zip, что мы хотим архивировать только файлы определенного типа, следует использовать опцию -i (или --include).
- Для того чтобы указать конкретные файлы, необходимо ввести строку *.txt, которая на первый взгляд выглядит странно. Не следовало ли ввести *.txt? Оказывается, обратная косая черта необходима для команды zip, а не для оболочки bash.

Этот способ вполне работоспособен, но что если вы хотите архивировать все без исключения файлы txt во всех подкаталогах? Для этого перейдите в родительский каталог, который содержит каталог, в котором находится ваш файл, и выполните следующую команду:

```
$ ls -F
reading/
$ zip -r weird_fiction.zip reading/ -i \*.txt
  adding: reading/Authors_and_Texts.txt (deflated
↳.58%)
  adding: reading/Lovecraft/Beyond the Wall of
```

```
↳Sleep.txt (deflated 57%)
  adding: reading/Lovecraft/The Call of Cthulhu.txt
↳(deflated 58%)
  adding: reading/Machen/The Great God Pan.txt
↳(deflated 60%)
$ ls -F
reading/
weird_fiction.zip
```

На этот раз мы внесли значительное изменение — добавили опцию `-r` (или `--recurse-paths`), которая указывает утилите `zip`, что архивирование следует начинать с текущего каталога и продолжать в его подкаталогах.

Обратите внимание на то, что вместо точки `.` в качестве стартовой точки я указал `reading/`. Почему? Потому что я хотел, чтобы итоговой zip-файл находился на одном уровне с каталогом `reading`. Если бы я хотел, чтобы файл `weird_fiction.zip` оказался *в* каталоге `reading`, то выбрал бы его в качестве стартового каталога и использовал команду `zip -r weird_fiction.zip. -i \*.txt`.

Защита zip-архивов паролем

```
zip -P
zip -e
```

Программа `zip` позволяет защищать паролем создаваемые zip-архивы. Для этой цели предусмотрена опция `-P`. Однако использовать ее не рекомендуется, так как она чрезвычайно уязвима с точки зрения безопасности. Это хорошо видно из следующего примера (в данном случае задан пароль 12345678):

```
$ zip -P 12345678 moby.zip *.txt
```

Поскольку пароль приходится задавать в командной строке, каждый желающий, просматривая предысторию работы с оболочкой (а сделать это очень просто), увидит его. Итак, не используйте опцию `-P`!

Лучше применять для этой цели опцию -e, которая шифрует содержимое zip-файла и требует ввода пароля. В этом случае пароль не задается в командной строке, а вводится в ответ на приглашение. В результате он не попадает в список предыстории.

```
$ zip -e moby.zip *.txt
Enter password:
Verify password:
  adding: job.txt (deflated 65%)
  adding: moby-dick.txt (deflated 61%)
  adding: paradise_lost.txt (deflated 56%)
```

В этом списке сохранится только вызов команды `zip -e moby.zip *.txt`. Реальный пароль останется в секрете.

ПРЕДУПРЕЖДЕНИЕ

Уровень безопасности, обеспечиваемый программой `zip`, невысок. В глобальной сети можно найти немало инструментов, позволяющих быстро взломать zip-архив, защищенный паролем. Архив с паролем можно сравнить с письмом в конверте. Для обычного человека оно остается закрытым, но, попав в руки злоумышленника, конечно же, будет прочитано.

Кроме того, версия программы `zip`, поставляемая в составе некоторых дистрибутивных пакетов Linux, не поддерживает шифрование. В этом случае вы получите сообщение об ошибке `encryption not supported`. Единственным решением остается перекомпиляция программы `zip` из исходного текста.

Разархивирование файлов

unzip

Разархивирование zip-архива осуществляется очень просто. Если для того, чтобы создать архив, используется команда `zip`, то для разархивирования служит команда `unzip`.

```
$ unzip moby.zip
Archive: moby.zip
  inflating: job.txt
  inflating: moby-dick.txt
  inflating: paradise_lost.txt
```

Команда `unzip` подробно сообщает о всех выполняемых действиях. Чтобы получить еще более детальную информацию, следует задать опцию `-v` (сокращение от слова `verbose`).

```
$ unzip -v moby.zip
Archive: moby.zip
Length  Method  Size  Ratio CRC-32  Name
------- ------ ------ ----- ------  ----
 102519 Defl:X 35747   65% fabf86c9 job.txt
1236574 Defl:X 487553  61% 34a8cc3a moby-dick.txt
 508925 Defl:X 224004  56% 6abe1d0f paradise_lost.t
------- ------ ------ ---           -------
1848018        747304  60%           3 files
```

Как видите, в этом случае программа сообщает о методе, использованном для сжатия файлов, размере сжатого файла в процентах от исходного размера и значении CRC (оно используется для коррекции ошибок).

Проверка файлов, предназначенных для разархивирования

```
unzip -t
```

Иногда файл архива оказывается поврежденным. Хуже всего, если вы разархивируете файлы и удалите архив, а лишь потом обнаружите, что некоторые или даже все файлы невозможно открыть. Гораздо лучше проверить архив до разархивации. Для этой цели предусмотрена опция `-t` (сокращение от слова `test`).

```
$ unzip -t moby.zip
Archive: moby.zip
    testing: bible/                OK
    testing: bible/genesis.txt     OK
    testing: bible/job.txt         OK
    testing: moby-dick.txt         OK
    testing: paradise_lost.txt     OK
No errors detected in compressed data of moby.zip.
```

Если вас беспокоит целостность zip-файла, то для его проверки следует использовать опцию `-t`. Это очень благоразумно, хотя и требует дополнительных затрат времени.

ПОДСКАЗКА

Существует аналогичная опция `-l` (от слова list — список). Просматриваете zip-file и не можете вспомнить, что он собой представляет? Хотите убедиться, что искомый файл находится в zip-файле? Волнуетесь, что получите 100 файлов в текущем каталоге, а не каталог, который просто содержит 100 файлов? Сначала выполните команду `zip -l`!

Желательно каждый раз при работе с zip-архивом использовать опцию `-t`. Несмотря на то что это потребует дополни-

тельного времени, в конце концов вы компенсируете затраты за счет повышения надежности своей работы.

Сжатие файлов посредством программы gzip

gzip

Использовать программу `gzip` несколько проще, чем `zip`. Работая с программой `zip`, вы должны указать имя создаваемого zip-файла. Программа `gzip` требует лишь, чтобы вы ввели имя команды и имя файла, подлежащего сжатию.

```
$ ls -l
-rw-r--r--  scott  scott  508925  paradise_lost.txt
$ gzip paradise_lost.txt
$ ls -l
-rw-r--r--  scott  scott  224425  paradise_lost.txt.gz
```

ЗАМЕЧАНИЕ

Как было указано ранее, команда `zip` позволяет задавать степень сжатия цифрой от 0 до 9. Команда `gzip` также предоставляет эту возможность.

Необходимо принимать во внимание существенное различие между программами `zip` и `gzip`. Если вы обрабатываете файл с помощью `zip`, он остается в неприкосновенности, т.е. вы получаете и исходный файл, и вновь созданный zip-архив. Программа `gzip` удаляет исходный файл и оставляет только сжатый файл.

Если вы хотите, чтобы после обработки `gzip` исходный файл не удалялся, используйте опцию `-c` (или `--stdout` или `--to-stdout`), которая выводит результаты работы программы `gzip` в стандартный выходной поток, и перенаправьте вывод в

другой файл. Если вы зададите опцию -c и забудете перенаправить вывод, то получите на экране странный набор символов, подобный приведенному ниже.

```
$ gzip -c paradise_lost.txt
w'
  I
ulu,(3эug_ui'+uuMuS3utl*f%eYu'[q?uu
Duu}d]C%gu      Ru@,r?ektrB3+3/uu|*uu0Djhs
BAqnuu,Y8*#"]]RU
```

Перенаправление в файл выглядит следующим образом:

```
$ ls -l
-rw-r--r-- 1 scott scott 508925 paradise_lost.txt
$ gzip -c paradise_lost.txt > paradise_lost.txt.gz
$ ls -l
-rw-r--r-- 1  scott  scott  497K paradise_lost.txt
-rw-r--r-- 1  scott  scott  220K paradise_lost.txt.gz
```

Теперь результаты значительно лучше. В вашем распоряжении есть и исходный файл, и его сжатая версия.

ПОДСКАЗКА

Если вы случайно задали опцию -c и не перенаправили вывод, нажмите несколько раз комбинацию клавиш <Ctrl+C>, чтобы прервать работу программы gzip.

Рекурсивная обработка файлов посредством программы gzip

```
gzip -r
```

Если вы хотите обработать командой gzip несколько файлов, содержащихся в одном каталоге, используйте символы групповых операций. Однако результаты могут отличаться от тех, которые вы ожидаете. Это демонстрирует следующий пример:

```
$ ls -F
bible/ moby-dick.txt paradise_lost.txt
$ ls -l *
-rw-r--r--  scott  scott  1236574  moby-dick.txt
-rw-r--r--  scott  scott   508925  paradise_lost.txt

bible:
-rw-r--r--  scott  scott  207254  genesis.txt
-rw-r--r--  scott  scott  102519  job.txt
$ gzip *
gzip: bible is a directory -- ignored
$ ls -l *
-rw-r--r--  scott  scott  489609  moby-dick.txt.gz
-rw-r--r--  scott  scott  224425  paradise_lost.txt.gz

bible:
-rw-r--r--  scott  scott  207254  genesis.txt
-rw-r--r--  scott  scott  102519  job.txt
```

Заметьте, что файлы в каталоге `bible` остались необработанными; по умолчанию `gzip` не затрагивает содержимого подкаталогов. Для того чтобы задать рекурсивную обработку, надо использовать не только символ групповой операции, но и опцию `-r` (или `--recursive`).

```
$ ls -F
bible/ moby-dick.txt paradise_lost.txt
$ ls -l *
-rw-r--r--  scott  scott  1236574  moby-dick.txt
-rw-r--r--  scott  scott   508925  paradise_lost.txt

bible:
-rw-r--r--  scott  scott  207254  genesis.txt
-rw-r--r--  scott  scott  102519  job.txt
$ gzip -r *
$ ls -l *
-rw-r--r--  scott  scott  489609  moby-dick.txt.gz
-rw-r--r--  scott  scott  224425  paradise_lost.txt.gz

bible:
-rw-r--r--  scott  scott  62114  genesis.txt.gz
-rw-r--r--  scott  scott  35984  job.txt.gz
```

На этот раз сжатию подверглись все файлы, даже те, которые содержались в подкаталоге. Однако сжатый вариант каждого файла сохраняется независимо от других. В отличие от `zip`, команда `gzip` не объединяет все файлы в один большой файл. Для того чтобы сделать это, надо использовать команду `tar`, которую мы рассмотрим далее в этой главе.

Распаковка файлов, сжатых с помощью программы gzip

`gunzip`

Для распаковки файлов, сжатых посредством `gzip`, используется команда `gunzip`.

```
$ ls -l
-rw-r--r--  scott  scott  224425 paradise_lost.txt.gz
$ gunzip paradise_lost.txt.gz
$ ls -l
-rw-r--r--  scott  scott  508925 paradise_lost.txt
```

Подобно тому, как программа `gzip` удаляет исходный файл, оставляя лишь результат его обработки, `gunzip` удаляет файл `.gz`, предоставляя пользователю только распакованный файл. Если вы хотите сохранить оба файла, вам надо задать опцию `-c` (или `--stdout`, или `--to-stdout`) и перенаправить вывод в файл.

```
$ ls -l
-rw-r--r--  scott  scott  224425  paradise_lost.txt.gz
$ gunzip -c paradise_lost.txt.gz  > paradise_lost.txt
$ ls -l
-rw-r--r--  scott  scott  508925  paradise_lost.txt
-rw-r--r--  scott  scott  224425  paradise_lost.txt.gz
```

Опция `-c` удобна в случае, если вам надо оставить у себя распакованный файл и передать кому-нибудь сжатый.

Конечно, вы снова сможете использовать `gzip`, но зачем выполнять лишнюю работу?

ЗАМЕЧАНИЕ

Если вам по каким-то причинам не нравится команда `gunzip`, используйте вместо нее команду `gzip` с опцией `-d` (или `--decompress`, или `--uncompress`).

Проверка файлов, предназначенных для распаковки с помощью программы gunzip

```
gunzip -t
```

Перед тем как распаковывать файл, желательно проверить, не поврежден ли он. Для этого можно использовать опцию `-t` (или `--test`).

```
$ gzip -t paradise_lost.txt.gz
$
```

Если архив в порядке, `gzip` не выведет никакого сообщения. При наличии проблем программа оповестит вас об этом. Возможно, поведение программы при отсутствии проблем покажется кому-то странным, но таков общий принцип работы утилит Unix: информация выводится только в том случае, когда от пользователя требуются специальные действия.

ЗАМЕЧАНИЕ

Опцию `-t` для проверки создаваемого архива можно также применять и при работе с утилитой `gzip`.

Сжатие файлов посредством программы bzip2

bzip2

Работать с программой bzip2 очень легко. Если вы не испытываете затруднений при использовании программы gzip, то обязательно справитесь и с bzip2. Разработчики этой утилиты позаботились о том, чтобы набор ее опций и поведение были максимально похожи на gzip.

```
$ ls -l
-rw-r--r--  scott  scott  1236574  moby-dick.txt
$ bzip2 moby-dick.txt
$ ls -l
-rw-r--r--  scott  scott   367248 moby-dick.txt.bz2
```

ЗАМЕЧАНИЕ

Как уже указывалось, команды zip и gzip позволяют задавать степень сжатия от 0 до 9. Команда bzip2 работает точно так же. Этого требует совместимость.

Подобно gzip, программа bzip2 оставляет только файл .bz2, а исходный файл удаляется. Для того чтобы сохранить его, надо задать опцию -k (или --keep).

```
$ ls -l
-rw-r--r--  scott  scott  1236574  moby-dick.txt
$ bzip2 -k moby-dick.txt
$ ls -l
-rw-r--r--  scott  scott  1236574  moby-dick.txt
-rw-r--r--  scott  scott   367248  moby-dick.txt.bz2
```

Если вы прочитали разделы, посвященные программе gzip, то заметите, что gzip и bzip2 ведут себя одинаково.

Распаковка файлов, сжатых с помощью программы bzip2

bunzip2

Если bzip2 разработана так, чтобы быть максимально похожей на gzip, то bunzip2 ведет себя практически так же, как и gunzip.

```
$ ls -l
-rw-r--r--  scott  scott   367248
  moby-dick.txt.bz2
$ bunzip2 moby-dick.txt.bz2
$ ls -l
-rw-r--r--  scott  scott  1236574  moby-dick.txt
```

У команд bunzip2 и gunzip есть еще одно общее свойство. Обе команды удаляют сжатый файл, оставляя только результат его обработки. Если вы хотите сохранить оба файла, укажите опцию -k (или --keep), как при работе с командой bzip2.

```
$ ls -l
-rw-r--r--  scott  scott   367248 moby-dick.txt.bz2
$ bunzip2 -c moby-dick.txt.bz2 > moby-dick.txt
$ ls -l
-rw-r--r--  scott  scott  1236574  moby-dick.txt
-rw-r--r--  scott  scott   367248  moby-dick.txt.bz2
```

Опции этих команд очень легко запомнить.

ЗАМЕЧАНИЕ

Если вам по каким-то причинам не нравится команда bunzip2, используйте для распаковки файлов команду bzip2 с опцией -d (или -decompress, или --uncompress).

Проверка файлов, предназначенных для разархивирования с помощью программы bunzip2

```
bunzip2 -t
```

Перед тем как распаковывать файл, желательно проверить, не поврежден ли он. Для этого можно использовать опцию `-t` (или `--test`).

```
$ bunzip2 -t paradise_lost.txt.gz
$
```

Подобно программе `gunzip`, если ошибки в архиве не обнаружены, `bunzip2` не отображает никаких сведений. При наличии проблем программа оповестит вас об этом.

Архивирование файлов с помощью программы tar

```
tar -cf
```

Как уже упоминалось в этой главе, программа `tar` не осуществляет сжатие, она лишь создает архивы. Для сжатия сформированных ею архивов `tar` использует другие программы, например `gzip` или `bzip2`. Если вы не собираетесь сжимать архив, используйте при его создании основные опции: `-c` (или `--create`), которая сообщает `tar` о том, что создается архив, и `-f` (или `--file`), посредством которой задается имя архивного файла.

```
$ ls -l
scott  scott   102519  job.txt
scott  scott  1236574  moby-dick.txt
scott  scott   508925  paradise_lost.txt
$ tar -cf moby.tar *.txt
```

```
$ ls -l
scott  scott   102519  job.txt
scott  scott  1236574  moby-dick.txt
scott  scott  1853440  moby.tar
scott  scott   508925  paradise_lost.txt
```

Работая с программой `tar`, необходимо учитывать две особенности. Во-первых, объем архива больше, чем суммарный объем входящих в него файлов. Например, общий размер файлов `job.txt`, `moby-dick.txt` и `paradise_lost.txt` составляет 1848018 байт. Сравнив это значение с размером файла `moby.tar`, вы увидите, что архив на 5422 байта больше. Не забывайте, что `tar` выполняет архивацию, а не сжатие, поэтому к суммарному размеру файлов, помещенных в архив, добавляется служебная информация. Во-вторых, заметьте, что `tar`, в отличие от `gzip` и `bzip2`, не удаляет исходные файлы. Это неудивительно, если учесть, что первоначально программа `tar` разрабатывалась как инструмент для создания резервных копий. (Имя *tar* происходит от слов *tape archive* (архив магнитных лет) и восходит к тем временам, когда резервные копии хранились на многочисленных магнитных лентах.)

Способность объединять большое количество файлов и каталогов в один большой файл — основное преимущество программы `tar` по сравнению с программами `gzip` и `bzip2`.

```
$ ls -lF
drwxr-xr-x  scott  scott   168  moby-dick/
$ ls -l moby-dick/*
scott  scott   102519  moby-dick/job.txt
scott  scott  1236574  moby-dick/moby-dick.txt
scott  scott   508925  moby-dick/paradise_lost.txt

moby-dick/bible:
scott  scott  207254  genesis.txt
scott  scott  102519  job.txt
$ tar -cf moby.tar moby-dick/
$ ls -lF
scott  scott     168  moby-dick/
scott  scott 2170880  moby.tar
```

Программа tar постоянно используется с ранних версий Unix и до настоящего времени. Причину ее популярности понять нетрудно: она очень удобна. Однако данная команда становится еще удобнее, если использовать ее совместно с инструментом сжатия. Этот вопрос будет рассмотрен в следующем разделе.

ЗАМЕЧАНИЕ

Команда tar имеет (и заслуженно) темную репутацию. Веб-комикс xkcd содержит множество анекдотов об утилите tar на странице http://xhcd.com/1168/.

Создание архивов и сжатие файлов посредством программ tar и gzip

```
tar -pzcvf
```

Если вы помните материал разделов, посвященных программам gzip и bzip2, то легко поймете суть проблемы. Предположим, вам надо сжать каталог, содержащий сто файлов, распределенных по различным подкаталогам. Если вы используете для этой цели программу gzip или bzip2 с опцией -r, то получите сто отдельных сжатых файлов, находящихся в тех же подкаталогах. Но такой результат вряд ли устроит вас. Если, например, все эти данные надо переслать по почте, как присоединить к письму сто файлов?

На помощь приходит программа tar. Сначала надо использовать tar для архивирования каталога и его содержимого (сто файлов в различных подкаталогах), а затем сжать полученный архив с помощью программы gzip или bzip2. Поскольку gzip используется совместно с tar чаще, чем другие программы, мы сосредоточим внимание именно на ней.

Задачу архивации большого количества файлов можно решить следующим образом:

```
$ ls -l moby-dick/*
scott  scott   102519  moby-dick/job.txt
scott  scott  1236574  moby-dick/moby-dick.txt
scott  scott   508925  moby-dick/paradise_lost.txt

moby-dick/bible:
scott  scott  207254  genesis.txt
scott  scott  102519  job.txt
$ tar -cf moby.tar moby-dick/ | gzip -c >
↳moby.tar.gz
$ ls -l
scott  scott  168  moby-dick/
scott  scott   20  moby.tar.gz
```

Такой подход даст нужный результат, но при этом выражение в командной строке получается достаточно длинным. Существует более простой способ. Надо использовать следующие опции `tar`: `-p` (или `--preserve-permissions` или `--same-permissions`), которая сохраняет права доступа; `-z` (или `--gzip`), которая вызывает `gzip` и применяет ее к данным, сгенерированным `tar`, и `-v` (или `--verbose`), которая оповестит вас о выполняемых действиях. Заметьте, что указывать опцию `-v` не обязательно, но предоставляемые ею сведения позволяют убедиться в том, что все идет так, как вы предполагали.

```
$ ls -l moby-dick/*
scott  scott   102519  moby-dick/job.txt
scott  scott  1236574  moby-dick/moby-dick.txt
scott  scott   508925  moby-dick/paradise_lost.txt

moby-dick/bible:
scott  scott   207254  genesis.txt
scott  scott   102519  job.txt
$ tar -zcvf moby.tar.gz moby-dick/
moby-dick/
moby-dick/job.txt
moby-dick/bible/
moby-dick/bible/genesis.txt
moby-dick/bible/job.txt
moby-dick/moby-dick.txt
```

```
moby-dick/paradise_lost.txt
$ ls -l
scott  scott     168  moby-dick
scott  scott  846049  moby.tar.gz
```

Обычно файл, полученный в результате выполнения программ `tar` и `gzip`, оканчивается символами `.tar.gz`, однако вы можете задать суффикс `.tgz` или `.tar.gzip`.

ЗАМЕЧАНИЕ

Вместо `gzip` совместно с программой `tar` можно использовать `bzip2`. При этом команда будет выглядеть так (обратите внимание на опцию `-j`, которая задает вызов `bzip2`):

```
$ tar -pjcvf moby.tar.bz2 moby-dick/
```

В данном случае имя файла будет оканчиваться символами `.tar.bz2`, но вы можете также задать окончания `.tar.bzip2`, `.tbz2` или `.tbz`. Применяя `bzip2` совместно с `tar`, желательно воздерживаться от использования суффикса `.tbz`, поскольку он очень похож на `.tgz`, генерируемый при работе с программой `gzip`.

Проверка файлов, предназначенных для распаковки и разархивирования

```
tar -zvtf
```

Перед тем как разархивировать файлы, содержащиеся в tar-архиве (независимо от того, сжат он или нет), желательно убедиться в том, что они не были повреждены. Во-первых, зная, что архив не в порядке, вы не будете тратить время, пытаясь организовать работу с содержащимися в нем файлами. Во-вторых, это позволит выявить ситуацию, когда пользователь, создававший архив, вместо каталога, содержащего сто файлов,

включил в него сто отдельных файлов. При разархивировании подобного архива необходимо принять меры, чтобы находящиеся в нем файлы не смешались с файлами, расположенными в вашем текущем каталоге.

Для проверки tar-архива (предположим, что он был также сжат) используется опция `-t` (или `--test`).

```
$ tar -zvtf moby.tar.gz
scott/scott        0  moby-dick/
scott/scott   102519  moby-dick/job.txt
scott/scott        0  moby-dick/bible/
scott/scott   207254  moby-dick/bible/genesis.txt
scott/scott   102519  moby-dick/bible/job.txt
scott/scott  1236574  moby-dick/moby-dick.txt
scott/scott   508925  moby-dick/paradise_lost.txt
```

В результате вы получите информацию о правах доступа, владельце, размере и времени создания каждого файла. Поскольку каждая строка начинается с `moby-dick/`, вы можете сделать вывод, что в архиве находится каталог, содержащий файлы и каталоги.

При вводе команды убедитесь, что опция `-f` расположена последней, так как после нее указывается имя файла `.tar.gz`. Если вы зададите `-f` в середине последовательности опций, программа `tar` сообщит об ошибке.

```
$ tar -zvft moby.tar.gz
tar: You must specify one of the '-Acdtrux' options
Try 'tar --help' or 'tar --usage' for more information.
```

После того как вы убедились, что ваш файл `.tar.gz` не был поврежден, надо извлечь его содержимое. Как это сделать, вы узнаете в следующем разделе.

ЗАМЕЧАНИЕ

Если вы проверяете tar-архив, сжатый посредством программы `bzip2`, используйте следующую команду:

```
$ tar -jvtf moby.tar.bz2
```

Распаковка и разархивирование файлов

```
tar -pzxf
```

Для того чтобы создать файл .tar.gz, мы использовали набор опций -zcvf. Для распаковки и разархивирования полученного файла необходимо лишь заменить опцию -c (или --create) опцией -x (или --extract).

```
$ ls -l
rsgranne rsgranne 846049 moby.tar.gz
$ tar -pzxvf moby.tar.gz
moby-dick/
moby-dick/job.txt
moby-dick/bible/
moby-dick/bible/genesis.txt
moby-dick/bible/job.txt
moby-dick/moby-dick.txt
moby-dick/paradise_lost.txt
$ ls -l
rsgranne  rsgranne     168  moby-dick
rsgranne  rsgranne  846049  moby.tar.gz
```

Перед разархивированием файла его следует проверить. Это означает, что последовательность выполняемых команд должна выглядеть так:

```
$ tar -zvtf moby.tar.gz
$ tar -pzxvf moby.tar.gz
```

ЗАМЕЧАНИЕ

Если вы разархивируете tar-архив, сжатый посредством программы bzip2, используйте следующую команду:

```
$ tar -pjxvf moby.tar.bz2
```

Выводы

В то время, когда скорость модемов была низкой, а объем жестких дисков маленьким, архивирование и сжатие файлов имело огромное значение. Сейчас это лишь вопрос удобства в работе, однако большинство пользователей постоянно выполняют подобные действия. Например, если вы когда-либо копировали исходный код, предназначенный для компиляции, он почти наверняка был расположен в сжатом tar-архиве. В будущем, вероятно, вам будут часто встречаться файлы, имена которых оканчиваются символами `.tar.bz2`. Если же вы обмениваетесь файлами с пользователями Windows, то, скорее всего, применяете для этой цели zip-архивы. Знать, как работают средства архивирования и сжатия, и уметь пользоваться ими гораздо важнее, чем может показаться на первый взгляд.

10

Поиск файлов, каталогов, слов и фраз

С каждым годом объем жестких дисков возрастает, а цена на них снижается. Появление новых технических средств — цифровых фотоаппаратов, видеокамер, mp3-плееров — приводит к генерации большого объема данных, которыми заполняется дополнительное дисковое пространство. Увеличение объема доступной информации порождает новые проблемы: найти нужные данные оказывается все труднее. Непросто отыскать фотоснимок среди 10000 изображений или текстовый файл среди 600 других документов. К счастью, в системе Linux есть мощные средства поиска, позволяющие быстро и эффективно найти требуемую информацию.

ПОДСКАЗКА

Если вас интересует команда `find`, переходите к следующей главе.

ЗАМЕЧАНИЕ

Пересматривая книгу для второго издания, я удалил информацию о команде `locate -n` (которая позволяет задавать количество результатов, возвращаемых программой `locate`). Исходный текст можно найти на моем веб-сайте (`www.granneman.com/linux-redactions`).

Поиск в базе имен файлов

`locate`

Представьте себе, что вы знаете имя файла или часть имени, но не представляете, в каком месте файловой системы он находится. Решить эту проблему вам поможет команда `locate`. Она ищет файлы и каталоги по заданному имени. Результаты поиска выводятся на терминал.

ЗАМЕЧАНИЕ

Для экономии места первая часть пути — `/home/scott` — заменена многоточием.

```
$ locate haggard
.../txt/rider haggard
.../txt/rider haggard/Queen of the Dawn.txt
.../txt/rider haggard/Allan and the Ice-Gods.txt
.../txt/rider haggard/Heu-Heu or The Monster.txt
```

Результаты отображаются быстро, однако программа `locate` не выполняет реального поиска. Вместо этого она просматривает базу данных имен файлов, которая ежедневно автоматически обновляется (подробнее этот вопрос будет рассмотрен далее в данной главе). Поскольку `locate` ищет информацию в предварительно созданной базе, она выполняет поставленную перед ней задачу практически мгновенно.

На вашем компьютере, возможно, вместо `locate` установлена программа `slocate`. Команда `slocate` (secure locate) не предпринимает попыток поиска в тех каталогах, которые пользователь, вызвавший ее, не имеет права просматривать (например, если вы не являетесь пользователем `root`, вам недоступно содержимое каталога `/root`). До появления `slocate` программа `locate` генерировала множество ошибок, связанных с правами доступа. С началом использования `slocate` эти ошибки — дело прошлого.

ЗАМЕЧАНИЕ

Все чаще в дистрибутивных пакетах системы Linux используется программа `mlocate` (сокращение от *merging locate* — поиск с помощью слияния), а не `slocate`. Вместо повторной индексации всего жесткого диска программа `mlocate` использует существующие базы данных, пытаясь ускорить поиск и не проверять жесткий диск каждую ночь. К счастью, программа `mlocate` обеспечивает обратную совместимость с программой `slocate`, поэтому все, что я напишу, относится и к ней.

Для проверки работы программы `slocate` попробуйте выполнить следующие команды. Обратите внимание на то, что первый поиск, запущенный от имени обычного пользователя, а не пользователя `root`, результатов не дает. После этого вы становитесь пользователем `root`, выполнив команду `su` (или `sudo`), снова запускаете программу `locate` и получаете результат! (Кстати, `slocate.db` — это файл базы данных, используемый программой `slocate`.)

```
$ locate slocate.db
$ su -
# locate slocate.db
/var/lib/slocate/slocate.db.tmp
/var/lib/slocate/slocate.db
```

Поскольку пользователь может и не знать, что вызывает именно `slocate`, а также потому, что в имени `locate` на одну букву меньше и ее быстрее вводить, мы будем в данной книге говорить о `locate`, не задумываясь о том, какая программа выполняется реально.

Поиск в базе имен файлов без учета регистра

```
locate -i
```

В предыдущем примере мы искали файлы или каталоги, содержащие в своем имени последовательность символов haggard (таким образом, мы просматривали коллекцию романов Г. Райдера Хаггарда). Полученные результаты выглядели следующим образом:

```
$ locate -i haggard
.../txt/rider haggard
.../txt/rider haggard/Queen of the Dawn.txt
.../txt/rider haggard/Allan and the Ice-Gods.txt
.../txt/rider haggard/Heu-Heu or The Monster.txt
```

Это объясняется тем, что программа просматривала каталоги, имя которых содержит слово haggard. Но если бы каталог назывался H Rider Haggard, поиск не дал бы результатов, потому что система Linux чувствительна к регистру символов (см. главу 1). Если указана опция -i, то регистр символов при поиске не учитывается. В этом случае были бы найдены файлы, в названии пути к которым содержалось бы haggard или Haggard (а также HAGGARD и даже HaGgArD).

```
$ locate -i haggard
.../txt/rider_haggard
.../txt/rider_haggard/Queen_of_the_Dawn.txt
.../txt/rider_haggard/Allan_and_the_Ice-Gods.txt
.../txt/rider_haggard/Heu-Heu_or_The_Monster.txt
.../txt/Rider_Haggard
.../txt/Rider_Haggard/King_Solomons_Mines.txt
.../txt/Rider_Haggard/Allan_Quatermain.txt
```

Оказывается, в системе есть два каталога, удовлетворяющие новому критерию поиска. Если, вызывая программу locate, вы хотите получить максимальное количество результатов, не

забывайте задавать опцию -i, в противном случае вы можете пропустить именно те файлы или каталоги, которые ищете.

ЗАМЕЧАНИЕ

Читателям, которых интересует, кто такой Г. Райдер Хаггард, рекомендую прочитать статью `http://en.wikipedia.org/wiki/Rider_Haggard`. Это писатель, писавший увлекательные приключенческие романы (правда, очень старомодные).

Обновление базы, используемой программой locate

```
updatedb
```

В первом разделе этой главы, где лишь начинался разговор о `locate`, было упомянуто, что причина столь быстрой работы команды в том, что поиск реально выполняется в базе данных, содержащей имена файлов и каталогов. При инсталляции программа `locate` автоматически настраивается на просмотр жесткого диска и обновление базы. Обычно такое обновление происходит ночью. Это очень удобно, но такой подход не позволяет найти файлы, которые лишь недавно были помещены в файловую систему.

Предположим, например, что вы инсталлировали Rootkit Hunter (программу, которая выявляет средства, используемые злоумышленниками) и хотите получить информацию об установленных файлах. Команда `locate` не поможет вам, поскольку она ничего не знает об этих файлах и не будет знать о них до тех пор, пока база данных не обновится. При необходимости вы можете в любое время вручную задать обновление базы, используемой `locate`. Для этого надо вызвать команду `updatedb`.

Поскольку данная команда индексирует практически каждый файл и каталог на вашем компьютере, для вызова ее вам необходимы права root (в системах типа Ubuntu можно также задать sudo).

```
# apt-get install rkhunter
# exit
$ locate rkhunter
$ su -
# updatedb
# exit
$ locate rkhunter
/usr/local/rkhunter
/usr/local/rkhunter/bin
/usr/local/rkhunter/etc
```

В предыдущем примере мы сначала установили rkhunter (пакет Rootkit Hunter), а затем завершили сеанс пользователя root. После этого был выполнен поиск rkhunter, но он не дал результатов. Мы снова выступили под именем root, запустили updatedb для просмотра жесткого диска, в результате чего locate получила информацию об изменениях, а затем завершили сеанс root. И наконец, мы снова выполнили поиск rkhunter с помощью программы locate, и на этот раз он был успешным.

Запуская updatedb, необходимо учитывать следующее: время работы этой программы прямо пропорционально числу файлов на вашем жестком диске и быстродействию компьютера. Если у вас быстрый процессор, быстрый диск и немного файлов, то updatedb завершится быстро. А что если быстродействие процессора мало, скорость обмена с жестким диском невелика, а на диске миллионы файлов? Тогда придется запастись терпением. Если вы хотите узнать, сколько времени потребовалось для индексирования содержимого файловой системы, задайте перед updatedb команду time так, как показано ниже.

```
# time updatedb
```

Когда updatedb завершит работу, time сообщит о затраченном времени. Эти сведения будут полезны на случай, если вы не имеете достаточного запаса времени и вам надо решить, имеет ли смысл вызывать updatedb.

ЗАМЕЧАНИЕ

Если в дистрибутивном пакете системы Linux используется более современная программа mlocate, то команда updatedb работает значительно быстрее, чем в сочетании с программой slocate, потому что в первом случае просматриваются уже существующие базы данных. Если в вашем дистрибутивном пакете системы Linux по-прежнему используется программа slocate, то команда updatedb даст точно такие же результаты, что и запуск slocate с опцией -u, поскольку updatedb — это всего лишь ссылка на slocate. При использовании программы mlocate команда updatedb представляет собой отдельную, совершенно самостоятельную программу.

Поиск фрагментов текстового файла

grep

Команда locate позволяет организовать поиск имен файлов и каталогов, но не дает возможности искать информацию в составе файлов. Для того чтобы решить эту задачу, надо использовать команду grep. При работе с этой командой ей задается шаблон поиска и файл или группа файлов (или даже весь жесткий диск), в которых надо найти фрагмент, соответствующий шаблону. В результате выполнения grep отображает строки, в которых присутствует интересующий вас фрагмент.

```
$ grep pain three_no_more_forever.txt
all alone and in pain
```

В данном случае мы используем команду grep для проверки, содержится ли в указанном файле слово pain. Как видите, результат проверки положительный: grep выводит на экран строку, содержащую данное слово. А можно ли выполнить поиск в нескольких файлах? Сделать это позволяют символы групповых операций.

```
$ grep pain *
fiery inferno in space.txt:watch the paint peel,
three_no_more_forever.txt:all alone and in pain
the speed of morning.txt:of a Chinese painting.
8 hour a day.txt:nice paint job too
ghost pain.txt:Subject: ghost pain
```

Заметьте, что команда grep нашла все вхождения ключевого слова pain, в том числе paint и painting. Также обратите внимание на то, что данная команда выводит не только строки, содержащие искомое слово, но и имена файлов. До сих пор мы не испытывали трудностей, выполняя поиск посредством grep. Пора усложнить задачу.

Общие сведения о шаблонах поиска

Из предыдущего раздела вы узнали, что программа grep позволяет находить указанное слово в группе файлов. Это одно из самых простых применений grep. Теперь попробуем глубже разобраться в структуре шаблонов, используемых для поиска. При формировании этих шаблонов применяется один из самых мощных инструментов Linux: регулярные выражения. Для того чтобы в полной мере воспользоваться возможностями, предоставляемыми программой grep, надо изучить механизм регулярных выражений. Однако, для того, чтобы подробно рассмотреть этот вопрос, потребовалась бы отдельная книга, поэтому здесь мы обсудим лишь общие положения.

ПОДСКАЗКА

Если хотите получить дополнительную информацию о регулярных выражениях, ее легко найти в глобальной сети. Но на мой взгляд, самым лучшим пособием будет книге Бена Форта *Освой самостоятельно регулярные выражения. 10 минут на урок.*

Одна из особенностей, затрудняющих начинающим пользователям изучение `grep`, состоит в том, что существует несколько версий данной команды. Сведения о них приведены в табл. 10.1.

Таблица 10.1. Различные версии программы `grep`

Интерпретация шаблона	Опция команды grep	Отдельная команда
Основные регулярные выражения	`grep -G` (или `--basic-regexp`)	`grep`
Расширенные регулярные выражения	`grep -E` (или `--extended-regexp`)	`egrep`
Набор фиксированных строк, для каждой из которых может быть установлено соответствие	`grep -F` (или `--fixed-strings`)	`fgrep`
Регулярные выражения Perl	`grep -P` (или `--perl-regexp`)	Отсутствует

Как видно из таблицы, сама по себе команда `grep` поддерживает основные регулярные выражения. Если вы зададите опцию `-E` (или `--extended-regexp`) либо команду `egrep`, то сможете использовать расширенные регулярные выражения. Именно такие выражения вам потребуются в большинстве случаев, за исключением разве что очень простых критериев поиска. Более сложные варианты команды — это `grep` с опцией `-F` (или `--fixed-strings`) или команда `fgrep`, позволяющая задавать множественные условия поиска, и `grep` с опцией `-P` (или `--perl-regexp`), которая дает возможность применять конструкции, типичные для языка Perl.

ЗАМЕЧАНИЕ

В данной книге мы будем в основном применять обычную команду `grep`, поддерживающую основные регулярные выражения. Отклонения от этого правила будут оговариваться отдельно.

Перед тем как продолжить изучение материала, надо обсудить некоторые особенности, которые могут затруднить его восприятие.

Символы групповых операций и регулярные выражения — не одно и то же. Например, и в том и в другом случае используется символ `*`, однако он интерпретируется по-разному. Если в групповых операциях знаки `?` и `*` означают произвольные символы, то в групповых операциях они управляют повторением предшествующей им конструкции. Например, символ групповой операции `?` в строке `c?t` заменяется одним и только одним символом. Этому выражению соответствуют слова `cat`, `cot` и `cut`, но не `ct`. В регулярных выражениях символ `?` в конструкции `c[a-z]?t` указывает на то, что буква из диапазона `a-z` может присутствовать один раз или отсутствовать вовсе. Такому шаблону соответствуют последовательности символов `cat`, `cot`, `cut` и `ct`.

ПОДСКАЗКА

Чтобы лучше понять различия между символами групповых операций и регулярными выражениями, обратитесь к документам *Regular Expressions Explained* (`http://www.castaglia.org/proftpd/doc/contrib/regexp.html`) и *Wildcards Gone Wild* (`http://www.linux-mag.com/id/1528`).

Источником проблем при работе с командой `grep` могут также стать символы, имеющие специальное назначение. Например, выражение `[a-e]` обозначает диапазон символов, ему

соответствует только одна из следующих букв: a, b, c, d или e. Используя [или] в регулярном выражении, надо различать случаи, когда они обозначают границы диапазона или непосредственно входят в последовательность символов, предназначенную для поиска. Символы, имеющие специальное значение, приведены ниже.

```
. ? [ ] ^ $ | \
```

И наконец, одинарная и двойная кавычки в регулярных выражениях существенно отличаются друг от друга. Одинарная кавычка сообщает о том, что должна быть найдена строка символов, а двойная кавычка указывает на использование переменных оболочки. Рассмотрим строку `hey you!`, заданную в качестве условия поиска.

```
$ grep hey you! *
grep: you!: No such file or directory
txt/pvzm/8 hours a day.txt:hey you! let's run!
txt/pvzm/friends & family.txt:in patience they wait
txt/pvzm/speed of morning.txt:they say the force
```

Поскольку в данном случае последовательность символов `hey you` была включена в состав команды без ограничителей, команда `grep` неправильно обработала запрос. Она интерпретировала первое слово, `hey`, как ключевое, а второе слово, `you!`, как имя файла. Такого файла не существует, поэтому поиск в нем не может быть выполнен. Затем, восприняв символ *, она стала искать слово `hey` в каждом файле, содержащемся в текущем каталоге, и вернула три строки результатов. В первой из этих строк содержится фраза, которую требовалось найти, но сказать о том, что поиск был выполнен корректно, нельзя. Повторим попытку.

Ограничим последовательность символов для поиска двойными кавычками. Будет ли устранена проблема, возникшая в предыдущем случае?

```
$ grep "hey you!" *
bash: !" *: event not found
```

Стало еще хуже! Включив в выражение двойные кавычки, мы допустили большую ошибку и в результате не получили даже тех строк, которые команда `grep` нашла в прошлый раз. Что случилось? Символ `!` — это команда оболочки, ссылающаяся на список предыстории. По правилам за ним должен следователь идентификатор процесса, представляющий команду, выполненную ранее, например `!264`. Таким образом, `bash` ожидает идентификационный номер после символа `!` и сообщает, что не может найти команду `" *` (двойная кавычка, пробел и звездочка).

Дело в том, что двойная кавычка указывает на использование в шаблоне поиска переменных оболочки, хотя мы совсем не собирались применять их. Таким образом, двойные кавычки в данном случае неуместны. Попробуем использовать одинарные кавычки.

```
$ grep 'hey!' *
txt/pvzm/8 hours a day.txt:hey you! let's run!
```

Теперь все намного лучше. Одинарные кавычки сообщают `grep` не о переменных оболочки, а лишь о том, что критерием поиска является обычная строка символов. Результатом является строка, содержащая указанное выражение, чего мы и добивались.

Какие можно сделать выводы? Необходимо знать, когда использовать одинарные кавычки, когда двойные, а когда можно обойтись без них. Если вы ищете конкретную последовательность символов, ее надо поместить в одинарные кавычки. Если же вы хотите включить в эту последовательность переменную оболочки (такая необходимость возникает крайне редко), следует использовать двойные кавычки. Если же вы ищете одно слово, содержащее лишь числа и буквы, вполне можно обойтись без кавычек. Желая перестраховаться, поместите слово в одинарные кавычки — вреда от этого не будет.

Рекурсивный поиск фрагментов текста в файлах

```
grep -R
```

Символ групповой операции * позволяет указать несколько файлов в одном каталоге, но для поиска в нескольких подкаталогах вам понадобится опция `-R` (или `--recursive`). Попробуем найти слово `hideous`, которое любили употреблять авторы романов ужасов в XIX и начале XX столетия. Поиск будем производить в наборе файлов, содержащем литературные произведения.

```
$ grep -R hideous *
Machen/The Great God Pan.txt:know, not in
↳your most fantastic, hideous dreams can you have
Machen/ The Great God Pan.txt:hideously
↳contorted in the entire course of my practice
Lovecraft/Beyond the Wall of Sleep.txt:some hideous
↳but unnamed wrong, which
Lovecraft/Beyond the Wall of Sleep.txt:blanket over
↳the hideous face, and awakened the nurse.
Lovecraft/The Call of Cthulhu.txt:hideous a chain. I
↳think that the professor, too, intended to
Lovecraft/The Call of Cthulhu.txt:voodoo meeting;
↳and so singular and hideous were the rites
```

ПОДСКАЗКА

Если вы получите слишком много результатов, можете передать их средствами конвейерной обработки программе `less`.

```
$ grep -R hideous * | less
```

Можно также перенаправить вывод команды в текстовый файл, а потом просмотреть его содержимое с помощью текстового редактора.

```
$ grep -R hideous * > hideous_in_horror.txt
```

Этот способ хорош тем, что вы сможете в любой момент вернуться к полученным результатам, не выполняя снова команду grep.

Поиск слов и выделение результатов

```
grep --color=auto
```

Визуальные подсказки — очень полезная вещь. В частности, цветом можно выделить важную часть информации. Если вы ищете слова с помощью утилиты grep, то, добавив опцию --color=auto, можете найти результаты быстро и без труда (поскольку книга печатается в черно-белом варианте, я выделил окрашенные слова полужирным шрифтом).

```
$ grep -R hideous *
Machen/The Great God Pan.txt:know, not in
↳your most fantastic, hideous dreams can you have
Machen/The Great God Pan.txt:hideously
↳contorted in the entire course of my practice
Lovecraft/Beyond the Wall of Sleep.txt:some hideous
↳but unnamed wrong, which
Lovecraft/Beyond the Wall of Sleep.txt:blanket over
↳the hideous face, and awakened the nurse.
Lovecraft/The Call of Cthulhu.txt:hideous a chain. I
↳think that the professor, too, intended to
Lovecraft/The Call of Cthulhu.txt:voodoo meeting;
.and so singular and hideous were the rites
```

Видите? Теперь искомое слово легко найти, потому что утилита grep выделила его для вас!

ПОДСКАЗКА

Иногда утилита grep окрашивает слова, даже если опция --color=auto не указана. В моей версии системы Ubuntu, например, я получил такие результаты (поскольку книга печатается в черно-белом варианте, я выделил окрашенные слова полужирным шрифтом):

```
$ grep Finny family.txt
My son's name is Finny.
$ grep --color=auto Finny family.txt
My son's name is Finny.
```

Почему это происходит? Я подозреваю, что система Ubuntu по умолчанию задает опцию `--color=auto` для утилиты `grep`. Для проверки этого предположения я выполнил следующую команду, которая подтвердила свою правоту:.

```
$ type grep
grep is aliased to 'grep --color=auto'
```

Как указано в главе 4, команда `type` демонстрирует, как оболочка `bash` интерпретирует команду. В данном случае имя `grep` оказалось псевдонимом. Если вы предполагаете, что какая-то команда имеет скрытые опции, заданные для ее псевдонима, применяйте команду `type`.

Поиск фрагментов текста в файлах без учета регистра

```
grep -i
```

По умолчанию при поиске, осуществляемом посредством команды `grep`, учитывается регистр символов. В предыдущем разделе мы искали слово `hideous` в рассказах Г.Ф. Лавкрафта. А как насчет `Hideous`?

```
$ grep Hideous h_p_lovecraft/*
Lovecraft/The Whisperer in Darkness.txt: them.
↳Hideous though the idea was, I knew...
```

Поиск по слову `hideous` дал 463 результата (ого!), а по слову `Hideous` — один. Можно ли задать поиск обоих слов? Это позволяет опция `-i` (или `--ignore-case`), которая указывает, что при поиске не должен учитываться регистр, поэтому будут найдены не только `hideous` и `Hideous`, но также `HiDeOuS` и

HIDEOUS и другие сочетания строчных и прописных букв, если они, конечно, будут встречаться в тексте.

```
$ grep -i hideous h_p_lovecraft/*
Lovecraft/The Call of Cthulhu.txt:voodoo meeting;
↳and so singular and hideous were the rites
Lovecraft/The Call of Cthulhu.txt:stated, a very
↳crude bas-relief of stone, comprising a hideous
Lovecraft/The Whisperer in Darkness.txt: them.
↳Hideous though the idea was, I knew…
```

При этом надо учитывать, что в общем случае при использовании опции -i число результатов может увеличиться в несколько раз и даже на порядки. В конце предыдущего раздела были приведены рекомендации, как справиться со слишком большим объемом информации, возвращаемой командой grep.

Поиск слов в файлах

grep -w

Вспомним раздел, в котором мы только начали изучать команду grep. Мы искали слово pain, а команда grep вернула нам список строк, в которых встречалась данная последовательность символов.

```
$ grep pain *
fiery inferno in space.txt:watch the paint peel,
three_no_more_forever.txt:all alone and in pain
the speed of morning.txt:of a Chinese painting.
8 hour a day.txt:nice paint job too
ghost pain.txt:Subject: ghost pain
```

По умолчанию утилита grep находит все вхождения указанной строки, в данном случае pain, в том числе слова paint и painting. Если бы в тексте присутствовали слова painless, Spain или painstaking, они также были бы учтены. Но что делать, если необходимы только те строки, в которые входит именно слово pain? Для этой цели предназначена опция -w (или --word-regexp).

```
$ grep -w pain *
three_no_more_forever.txt:all alone and in pain
ghost pain.txt:Subject: ghost pain
```

Данная опция позволяет сузить поиск и соответственно уменьшить набор результатов.

Отображение номеров строк, в которых слово содержится в файле

```
qrep -n
```

Команда `grep` отображает каждую строку, содержащую заданную последовательность символов, но не сообщает о том, в какой части файла она расположена. Чтобы получить номер строки, надо использовать опцию `-n` (или `--line-number`).

```
$ grep -n pain *
fiery inferno in space.txt:56:watch the paint peel,
three_no_more_forever.txt:19:all alone and in pain
the speed of morning.txt:66:of a Chinese painting.
8 hour a day.txt:78:nice paint job too
ghost pain.txt:32:Subject: ghost pain
```

Теперь, когда мы знаем номера строк, содержащих последовательность символов `pain`, достаточно просто, используя любой текстовый редактор, обратиться непосредственно к нужной строке.

Поиск слов в выходных данных других команд

```
[команда] | grep
```

Команда `grep` сама по себе является мощным средством поиска, но ее также можно использовать как фильтр для

обработки выходных данных, сгенерированных другими программами. Предположим, например, что вы храните mp3-файлы с записями Джона Колтрейна, выделив для каждого альбома отдельный подкаталог, причем имя подкаталога начинается с года, в котором соответствующий альбом был выпущен. Часть информации, полученной с помощью команды `ls -1`, приведена ниже (опция `-1` приводит к тому, что в каждой строке отображается только одно имя).

```
$ ls -1
1956_Coltrane_For_Lovers
1957_Blue_Train
1957_Coltrane_[Prestige]
1957_Lush_Life
1957_Thelonious_Monk_With_John_Coltrane
[Результаты сокращены]
```

Предположим теперь, что вам необходимо получить список альбомов, выпущенных в 1960 году. Решить эту задачу можно, передав результаты выполнения `ls -1` команде `grep`.

```
$ ls -1 | grep 1960
1960_Coltrane_Plays_The_Blues
1960_Coltrane's_Sound
1960_Giant_Steps
1960_My_Favorite_Things
```

Немного поразмыслив, вы придумаете сотни вариантов применения команды `grep`. Рассмотрим еще один пример. Команда `ps` выводит список выполняющихся процессов, причем опция `-f` указывает `ps` на то, что необходимо отобразить полный список с подробной информацией о каждом процессе, а опция `-U`, за которой следует пользовательское имя, ограничивает вывод лишь процессами, принадлежащими конкретному пользователю. Задав `-fU scott`, мы, вероятнее всего, получим длинный список, в котором очень сложно будет найти информацию о требуемом процессе. Команда `grep` позволяет сократить объем выходных данных.

ЗАМЕЧАНИЕ

Для экономии места часть информации, обычно отображаемой по команде ps, здесь не приводится.

```
$ ps -fU scott | grep firefox
scott 17623 /bin/sh /opt/firefox/firefox
scott 17634 /opt/firefox/firefox-bin
scott  1601 grep firefox
```

Команда ps выводит информацию обо всех процессах, принадлежащих пользователю scott (в данном случае их 64), но выходные данные посредством механизма конвейерной обработки передаются программе grep, отсеивающей все строки, в которых не содержится слово firefox. К сожалению, последняя строка лишняя: нам нужна информация о самой программе firefox, а не о ее имени, переданном команде grep. Скрыть эту строку можно следующим образом:

```
$ ps -fU scott | grep [f]irefox
scott 17623 /bin/sh /opt/firefox/firefox
scott 17634 /opt/firefox/firefox-bin
```

Теперь первая буква слова, передаваемого grep, лежит в диапазоне от f до f. Этому критерию соответствуют строки, сгенерированные программой ps, которые содержат слово firefox. Последовательность [f]irefox, переданная grep, не удовлетворяет условиям поиска. поскольку в ней содержатся символы [и]. Сама же команда grep использует шаблон [f]irefox, которому соответствует только слово firefox. Такой способ выглядит несколько сложно, но он дает нужные результаты. Возьмите его на вооружение!

ПОДСКАЗКА

Если вы по-настоящему ленивы, то попытайтесь использовать команду pgrep, которая объединяет несколько предыдущих абзацев в один простой пакет.

Просмотр контекста для слов, имеющихся в файлах

grep [-ABC]

Когда речь идет о работе с данными, важность контекста трудно переоценить. Как вы знаете, программа `grep` выводит строку, содержащую заданную последовательность символов, но мы также можем указать `grep` выводить предыдущие и последующие строки. В предыдущем разделе `grep` использовалась для обработки списка альбомов Джона Колтрейна. Одним из наилучших считается "A Love Supreme". Какие три альбома были выпущены перед ними? Чтобы получить ответ, зададим опцию `-B` (или `--before-context=#`).

```
$ ls -1 | grep -B 3 A_Love_Supreme
1963_Impressions
1963_John_Coltrane_&_Johnny_Hartman
1963_Live_At_Birdland
1964_A_Love_Supreme
```

Если вас интересует, какие альбомы последовали за `A Love Supreme`, задайте опцию `-A` (или `--after-context=#`).

```
$ ls -1 | grep -A 3 "A_Love_Supreme"
1964_A_Love_Supreme
1964_Coltrane's_Sound
1964_Crescent
1965_Ascension
```

Для того чтобы получить полный контекст, можно использовать опцию `-C` (или `--context=#`), которая является сочетанием двух опций, рассмотренных выше.

```
$ ls -1 | grep -C 2 "A_Love_Supreme"
1963 Impressions
1963 John_Coltrane & Johnny_Hartman
1963 Live At Birdland
1964 A Love Supreme
```

```
1964 Coltrane's Sound
1964 Crescent1965 Ascension
```

Такая информация в некоторых случаях может быть сложной для восприятия, поскольку при обнаружении соответствия условиям поиска программа отображает сразу несколько строк. Так, например, в названиях ряда альбомов Колтрейна присутствует слово "live", и если вы зададите вывод предшествующих и последующих альбомов, то результаты могут оказаться несколько неожиданными.

```
$ ls -1 | grep -C 1 Live
1963 John Coltrane & Johnny Hartman
1963 Live At Birdland
1964 A Love Supreme
--
1965 Last Trane
1965 Live in Seattle
1965 Major Works of John Coltrane
--
1965 Transition
1966 Live at the Village Vanguard Again!
1966 Live in Japan
1967 Expression
1967 Olatunji Concert Last Live Recording
1967 Stellar Regions
```

Символы -- отделяют одну группу результатов от другой. Первые две группы очевидны — рядом с альбомом со словом "Live" в названии находятся другие альбомы, однако последняя группа выглядит гораздо сложнее. Несколько альбомов со словом "Live" непосредственно следуют друг за другом, поэтому результаты объединены. Это может показаться странными, но если вы ищете каждое вхождение слова "Live", то заметите, что отображается информация о предшествующем и последующем альбомах.

Результаты станут еще более информативными, если мы дополнительно укажем при вызове команды опцию `-n`, которая задает вывод номеров строк.

```
$ ls -1 | grep -n -C 1 Live
37-1963 John Coltrane & Johnny Hartman
```

```
38:1963 Live At Birdland
39-1964 A Love Supreme
--
48-1965 Last Trane
49:1965 Live in Seattle
50-1965 Major Works of John Coltrane
--
52-1965 Transition
53:1966 Live at the Village Vanguard Again!
54:1966 Live in Japan
55-1967 Expression
56:1967 Olatunji Concert Last Live Recording
57-1967 Stellar Regions
```

Теперь опция `-C` предоставляет еще больше информации о каждой строке: отображает после номера строки разные символы. Символ `:` указывает на то, что строка соответствует условиям поиска, а символ `-` означает, что это предшествующая или последующая строка. Строка `54:1966 Live in Japan` выполняет две функции: следует за строкой `53:1966 Live at the Village Vanguard Again!` (по этому критерию после номера должен быть указан символ `-`), и сама содержит последовательность символов, заданную при вызове команды, т.е. в ней должно быть двоеточие. Поскольку соответствие критерию поиска является более важным признаком, предпочтение отдается символу `:`.

Отображение строк, не содержащих указанных слов

`grep -v`

С момента смерти Джона Колтрейна прошло около сорока лет, но его альбомы по-прежнему популярны. Знатоки искусства также высоко ценят творчество группы "Led Zeppelin", которая выпустила девять альбомов; в названиях многих из них присутствует имя группы (вы скажете, что четвертый альбом

вовсе не имеет названия, но большинство критиков называют его "Led Zeppelin IV"). Предположим, вы хотите получить список каталогов, в которые поместили альбомы "Led Zeppelin", но в названии которых не содержатся сами слова "Led Zeppelin". Опция `-v` (или `--invert-match`) позволяет вам отобразить только те результаты, которые не соответствуют заданному шаблону.

```
$ ls -l
1969 Led Zeppelin
1969 Led Zeppelin II
1970 Led Zeppelin III
1971 Led Zeppelin IV
1973 Houses Of The Holy
1975 Physical Graffiti
1976 Presence
1979 In Through The Out Door
1982 Coda
$ ls -l | grep -v "Led Zeppelin"
1973 Houses Of The Holy
1975 Physical Graffiti
1976 Presence
1979 In Through The Out Door
1982 Coda
```

Опция `-v` дает возможность сократить набор результатов и отобразить только те из них, которые вам действительно нужны. Вряд ли вам придется использовать данную опцию очень часто, но в некоторых случаях она может оказаться полезной.

Отображение списка файлов, содержащих указанное слово

```
grep -l
```

Как уже неоднократно говорилось, команда `grep` выводит строки, содержащие слово, заданное для поиска. Однако в некоторых случаях строки бывают излишни; нужны лишь имена

файлов, в которых находится слово. В одном из предыдущих разделов мы искали вхождение слова hideous. Опция -l (или --files-with-matches) дает возможность получить список файлов (опция -i, указанная в той же строке, задает поиск без учета регистра символов).

```
$ grep -il hideous Lovecraft/*
h_p_lovecraft/Call of Cthulhu.txt
h_p_lovecraft/From Beyond.txt
h_p_lovecraft/The Case of Charles Dexter Ward.txt
[Результаты сокращены]
```

Подобные результаты в особенности полезны в сочетании с другими командами. Например, если вы хотите вывести на печать список файлов из конкретного каталога, в которых встречается слово hideous, объедините команды grep и lpr следующим образом:

```
$ grep -il hideous Lovecraft/* | lpr
```

Помните, что данная команда выведет список имен файлов, а не их содержимое (если вы хотите распечатать сами тексты, используйте команду cat).

Поиск слов среди результатов

```
grep | grep
```

Английский мистический поэт XVIII века Уильям Блейк (William Blake) как-то сказал о поэме Мильтона "Потерянный рай" (Milton, *Paradise Lost*), что тот "был сторонником Дьявола, сам того не зная". Иначе говоря, персонаж Сатаны в поэме "Paradise Lost" был более интересным, чем персонаж Бога. Для того чтобы стать полноценным участником дебатов о комментарии Блейка, вам необходимо многое прочитать, начиная с самой поэмы "Paradise Lost" (я так и сделал, когда работал над докторской диссертацией по английской литературе XVII века).

Даже если у вас нет времени прочитать все, что нужно, вы можете сделать свой вклад в дискуссию, хотя и в ограниченном объеме. Применим опцию `-c` (или `--count`) команды `grep`, чтобы выяснить, сколько раз Мильтон упоминает слова "Satan" и "God" в поэме "Paradise Lost".

```
$ grep -c Satan "Paradise Lost.txt"
12
$ grep -c God "Paradise Lost.txt"
327
```

Вот те раз! Только 12 упоминаний Сатаны в поэме "Paradise Lost"? Не может быть. Я еще не забыл ее содержание. Погодите секунду... она была написана в XVII веке. Тогда прописные буквы использовались намного свободнее. Попытаемся выполнить команду без учета регистра символов.

```
$ grep -ci Satan "Paradise Lost.txt"
72
$ grep -ci God "Paradise Lost.txt"
329
```

Теперь результаты выглядят правдоподобнее! Эти подсчеты показывают, что Мильтон вовсе не был таким уж "сторонником Дьявола (получи, Уильям Блейк!).

ЗАМЕЧАНИЕ

Опция `-c` сообщает количество *строк*, в которых появляется слово, а не количество *появлений* слова в файле. Иначе говоря, если вы ищете слово "hideous" в строке вроде "The old hideous man was hideously hideous", то результат будет равен 1, а не 3.

Выводы

В данной главе внимание было уделено двум командам, которые часто используются на практике: `locate` и `grep`. Несмотря на то что эти команды принадлежат одной группе,

реализующей средства поиска информации на компьютере, они решают разные задачи. Команда `locate` выполняет поиск по именам файлов, используя для ускорения работы специальную базу данных, а команда `grep` просматривает в реальном времени содержимое файлов и извлекает фрагменты, удовлетворяющие условиям поиска.

И команда `locate`, и команда `grep` — удобные средства, но возможности поиска в файловой системе далеко не исчерпываются ими. Следующая глава посвящена одной из самых мощных и гибких команд системы Linux, которая дополняет `locate` и `grep`, а также может работать совместно с ними. Это команда `find`. Переходите к следующей главе, и начнем знакомство с ней.

11

Команда find

В предыдущей главе мы рассматривали команды, позволяющие выполнять поиск файлов (`locate`) и данных внутри файлов (`grep`). Третья команда из той же группы — `find`. Программа `locate` выполняет поиск в базе данных имен файлов, что позволяет ей быстро работать, но ставит в зависимость от времени обновления базы, а команда `find` непосредственно ищет файлы по заданным условиям. Поскольку команда `find` осуществляет реальный обход дерева каталогов, она работает намного медленнее, чем `locate`, однако предоставляет возможности, недоступные посредством команды `locate`.

В этой главе мы будем выполнять поиск на внешнем жестком диске, смонтированном в каталоге `/media/music`. Обратите внимание на пробелы в именах файлов, которые объясняют наличие двойных кавычек во многих командах. Вы увидите, что команда `find` позволяет обрабатывать самые разнообразные запросы.

Поиск файлов по имени

`find -name`

Чаще всего команда `find` используется для поиска файлов по имени или его части. Для этой цели предусмотрена опция `-name`. По умолчанию поиск осуществляется рекурсивно в дереве каталогов. Давайте найдем mp3-файлы с записями группы `Shaggs`.

```
$ cd /media/music
$ find . -name Shaggs
./Outsider/Shaggs
```

Что-то не так! Команда `find` нашла каталог, а не файлы, содержащиеся в нем. Почему это произошло? Дело в том, что мы не указали символы групповых операций, поэтому команда `find` искала файлы по точному совпадению имени со строкой `Shaggs`. Такое имя имеет только один файл — каталог, содержащий mp3-файлы (как вы, наверное, помните, каталог — это файл специального типа).

Нам надо использовать символ групповой операции, но? чтобы предотвратить специальную интерпретацию его оболочкой, поместим имя, содержащее этот символ, в кавычки. Попробуем выполнить поиск снова.

```
$ find . -name "*Shaggs*"
./Outsider/Shaggs
./Outsider/Shaggs/Gimme Dat Ting (Live).mp3
./Outsider/Shaggs/My Pal Foot Foot.mp3
./Outsider/Shaggs/I Love.mp3
./Outsider/Shaggs/You're Somethin' Special To Me.mp3
./Outsider/Shaggs/Things I Wonder.mp3
```

Теперь мы получили информацию не только о каталоге, но и о файлах.

ЗАМЕЧАНИЕ

Не подозревая о том, мы использовали еще одну опцию: `-print`. Эта опция сообщает программе `find`, что результаты поиска надо вывести на терминал. По умолчанию эта опция включена, и ее не надо указывать явно при вызове `find`.

Важно отметить еще одну особенность программы `find`: формат представления результатов зависит от пути, указанного при вызове команды. В предыдущем примере мы использовали относительный путь — указали текущий каталог, поэтому результаты также были представлены относительно текущего каталога. Что произойдет, если мы зададим абсолютный путь, начинающийся с символа `/`?

```
$ find / -name "*Shaggs*"
/media/music/Outsider/Shaggs
/media/music/Outsider/Shaggs/Gimme Dat Ting
↳ (Live).mp3
/media/music/Outsider/Shaggs/My Pal Foot Foot.mp3
/media/music/Outsider/Shaggs/I Love.mp3
/media/music/Outsider/Shaggs/You're Somethin'
↳ Special To Me.mp3
/media/music/Outsider/Shaggs/Things I Wonder.mp3
```

При поиске по абсолютному пути результаты также представляются посредством абсолютного пути. Другие применения этого принципа мы увидим далее в данной главе. Сейчас же просто запомним эту особенность.

ЗАМЕЧАНИЕ

Более подробную информацию о группе Shaggs см. по адресам `http://www.allmusic.com/artist/the-shaggs-mn0000418794` и `http://en.wikipedia.org/wiki/The_Shaggs`.

Поиск файлов по имени владельца

```
find -user
find -group
```

Помимо поиска по имени файла, можно также осуществлять поиск по имени владельца. Предположим, что вы хотите найти файлы, принадлежащие пользователю `scott`. Задайте при вызове команды `find` опцию `-user`, введите после нее имя пользователя (или числовой идентификатор, который можно найти в файле `/etc/passwd`).

```
$ find . -user scott
```

Количество результатов слишком велико! Возможно, будет проще искать файлы, которые *не* принадлежат пользователю

scott. Сделать это можно, указав символ ! перед опцией, действие которой вы хотите изменить на обратное.

```
$ find . ! -user scott
./Punk/Stooges/Fun House/01 Down on the Street.mp3
$ ls -l "Punk/Stooges/Fun House/01 Down on the
↳Street.mp3"
gus music ./Punk/Stooges/Fun House/01 Down on the
↳Street.mp3
```

Один из файлов с записью композиций *The Stooges* принадлежит пользователю gus, а не scott. Помните, что при необходимости вы всегда можете использовать символ ! в качестве оператора отрицания. Так, только что мы указали команде find найти файлы, *не* принадлежащие пользователю scott.

Если вы хотите найти файлы, принадлежащие определенной группе, то вам необходимо указать опцию -group и ввести после нее имя или номер группы. На диске, используемом при поиске, владельцем файлов является пользователь scott, а в качестве группы указана music. Попробуем найти файлы, не принадлежащие группе music.

```
$ find . ! -group music
./Disco/Brides of Funkenstein/Disco to Go.mp3
./Disco/Sister Sledge/He's The Greatest Dancer.mp3
./Disco/Wild Cherry/Play_That_Funky_Music.mp3
./Electronica/New Order/Bizarre Love Triangle.mp3
```

Из огромного числа файлов только четыре не принадлежат данной группе. Заметьте, что мы снова использовали символ !, чтобы выполнить поиск файлов, не принадлежащих группе music.

Поиск файлов по размеру

```
file -size
```

В некоторых случаях приходится использовать в качестве критерия для поиска файлов их размер. Команда find

позволяет решить и эту задачу. Для того чтобы указать размер, надо задать опцию `-size` и ввести после нее букву, которая представляет используемую вами схему размера. Если вы не зададите букву, будет использовано значение по умолчанию, но следует иметь в виду, что оно не обязательно будет соответствовать вашим намерениям. По умолчанию, если после числа вы не введете букву, принимается значение размера в байтах, деленное на 512 и округленное вверх до ближайшего целого числа. Некоторым пользователям покажется, что здесь слишком много математики и проще указать суффикс, представляющий единицу изменения размера. Набор допустимых суффиксов представлен в табл. 10.1.

Таблица 10.1. Суффиксы, используемые при поиске файлов по размеру

Суффикс	Значение
`b`	512-байтовые блоки (по умолчанию)
`c`	Байты
`k`	Килобайты
`M`	Мегабайты
`G`	Гигабайты

ЗАМЕЧАНИЕ

Строго говоря, несмотря на то, что на страницах справочной системы `man`, посвященных команде `find`, используются термины килобайт, мегабайт и гигабайт, эти термины неправильные (см. главу 6). Множество программ по-прежнему используют старые термины, которые в настоящее время стали некорректными. За подробной информацией обращайтесь по адресу `http://en.wikipedia.org/wiki/Mebibyte`.

Предположим, нам надо найти файлы с записью песен группы “Clash” из знаменитого альбома “London Calling”, причем размер файлов должен составлять 10 Мбайт (конечно же,

использовалось кодирование, обеспечивающее наивысшее качество). Эта задача легко решается с помощью команды find.

```
$ cd Punk/Clash/1979_London_Calling
$ find . -size 10M
./07 - The Right Profile.mp3
./08 - Lost In The Supermarket.mp3
./09 - Clampdown.mp3
./12 - Death Or Glory.mp3
```

Результаты выглядят странно. Только четыре файла? Чтобы понять причины такого поведения команды find, надо учесть следующую ее особенность: если вы задаете значение 10 Мбайт, команда find ищет файлы, размер которых в точности равен 10 Мбайт (конечно же, с учетом округления). Если вам нужны файлы размером больше 10 Мбайт, то надо перед значением размера ввести символ +, если же размеры файлов должны быть меньше 10 Мбайт — то знак –.

```
$ find . -size +10M
./03 - Jimmy Jazz.mp3
./15 - Lover's Rock.mp3
./18 - Revolution Rock.mp3
```

И снова возникла проблема. Когда мы указали размер для поиска, равный 10 Мбайт, были найдены файлы, размер которых равен этому значению, но не больше его, а когда задали +10M, получили список файлов, размер которых превышает 10 Мбайт, но файлы, размер которых в точности равен 10 Мбайт, не были учтены. А как получить сведения об обоих наборах файлов? Ответ на этот вопрос вы найдете ниже в данной главе.

ПОДСКАЗКА

Если вы хотите отыскать большие текстовые файлы, используйте после числового значения букву c. Как показано в табл. 10.1, этот символ задает измерение размера в байтах. Каждый символ в текстовом файле занимает один байт, поэтому ясно, что символ c — это первая буква в слове characters.

Например, чтобы найти очень большой текстовый файл, можно использовать выражение

```
$ find /home/scott/documents -size +500000c
```

Поиск файлов по типу

find -type

Одна из самых полезных опций программы `find` — это `-type`, позволяющая указать тип объекта для поиска. В главе 1 было сказано, что все объекты в системе Unix являются файлами, поэтому если вы укажете программе `find` тип файла, то она найдет для вас нужный объект. В табл. 11.2 перечислены типы файлов, поддерживаемые `find`.

Таблица 11.2. Обозначения при поиске файлов по типу

Буква, представляющая тип файла	Значение
f	Обычный файл
d	Каталог
l	Символьная ссылка
b	Специальный файл блочного типа
c	Специальный файл символьного типа
p	FIFO
s	Сокет

Предположим, нам надо получить список разных вариаций легендарной песни Фрэнка Синатры “Come Fly With Me”, которые записаны на музыкальном диске. Для этого надо выполнить ряд команд (результаты приводятся в сокращенном виде — на самом деле их 14).

```
$ cd "Jazz - Vocal/Frank Sinatra"
$ find . -name "*Come Fly With Me*"
./1962 Live In Paris/26 - Come Fly With Me.mp3
./1957 Come Fly With Me
```

```
./1957 Come Fly With Me/01 - Come Fly With Me.mp3
./1966 Sinatra At The Sands/01 - Come Fly With. Me.mp3
```

Обратите внимание на второй результат. Это каталог, в котором хранится знаменитый альбом Фрэнка Синатры 1957 года под таким же названием: "Come Fly With Me". Но нам нужны только файлы, а не каталоги, поэтому с помощью опции `-type f` ограничим результаты только файлами.

```
$ find . -name "*Come Fly With Me*" -type f
./1962 Live In Paris/26 - Come Fly With Me.mp3
./1957 Come Fly With Me/01 - Come Fly With Me.mp3
./1966 Sinatra At The Sands/01 - Come Fly With. Me .mp3
```

Этот список удобен, но поскольку имя каждого альбома начинается с года его выпуска, его можно передать команде `sort` (см. главу 7) и проследить за изменениями, которые Синатра постепенно вносил в эту песню.

```
$ find . -name "*Come Fly With Me*" -type f | sort
./1957 Come Fly With Me/01 - Come Fly With Me.mp3
./1962 Live In Paris/26 - Come Fly With Me.mp3
./1966 Sinatra At The Sands/01 - Come Fly With. Me.mp3
```

Передача результатов команды `find` для последующей фильтрации может оказаться очень полезной; по мере освоения команды `find` вы научитесь создавать все более сложные фильтры.

Поиск файлов по времени

```
find -amin|-cmin|-mmin
find -atime|-ctime|-mtime
find -anewer|-cnewer|-newer|-newerXY
```

До сих пор мы вели речь об использовании команды `find` для создания списка файлов по именам, владельцам, размерам и типам. А как насчет времени? Команда `find` решает эту задачу блестяще.

Например, однажды я захотел выявить все файлы в каталоге и его подкаталогах, созданные не менее чем четыре часа назад. С помощью команды `find` это оказалось очень просто.

```
$ find . -mmin +240
```

Эта команда означает: "Создать список всех файлов, модифицированных более чем 240 минут назад". Когда вы используете команду `find` таким образом, вы должны задать число (в большинстве случаев). Это число можно выразить тремя способами, показанными в табл. 11.3.

Таблица 11.3. Числовые аргументы для поиска файлов по времени

Numeric Argument	Meaning
`+n`	Больше, чем `n`
`-n`	Меньше, чем `n`
`n`	Равно `n`

Следует помнить, что число `n` может быть количеством минут или часов в зависимости от конкретного случая. Кроме того, команда `find` может проверять, когда был последний *доступ* к файлу (accessed), когда он был *изменен* (changed) и когда *модифицирован* (modified). Несмотря на то что эти термины могут показаться синонимами, с точки зрения системы Linux в них вкладывается разный смысл.

- **Доступ к файлу** означает, что его содержимое читали, но не изменяли, например, с помощью команды `less`.
- **Изменение файла** означает изменение метаданных (или статуса файла), но не его содержимого, например, с помощью команд `chmod`, `chown`, `link` и `rename`.
- **Модификация файла** означает, что данные редактировали.

ПОДСКАЗКА

Вы можете проверить эту информацию с помощью очень полезной команды `stat` (см. пример):

```
$ stat foobar.txt
Access: 2013-04-14 16:57:24.768011000 -0500
Modify: 2012-12-01 22:27:24.424000023 -0600
Change: 2012-12-01 22:27:24.424000023 -0600
```

Итак, зная разницу между доступом, изменением и модификацией, мы можем, наконец, выполнить полную проверку файлов. Они сгруппированы логически в табл. 11.4. Напоминаю, что параметр `n` в табл. 11.4 может означать любое число из табл.11.3.

Таблица 11.4. Поиск файлов по времени

Тест	Описание
Минуты	
`-amin n`	Был доступ `n` минут назад
`-cmin n`	Статус был изменен `n` минут назад
`-mmin n`	Данные были изменены `n` минут назад
Часы (дробные части суток игнорируются)	
`-atime n`	Был доступ `n*24` минут назад
`-ctime n`	Статус был изменен `n*24` минут назад
`-mtime n`	Данные были изменены `n*24` минут назад

ЗАМЕЧАНИЕ

Может возникнуть вопрос: почему в качестве опции поиска не используется время создания файла? Причина этого очевидна: ядро системы Linux не отслеживает время создания файлов.

Хотите найти файлы, доступ к которым был выполнен более 45 минут назад? Используйте опцию `-amin +45`.

Хотите найти файлы, статус которых был изменен в течение последних 24 часов? Используйте опцию `-ctime 0`. Помните, что команда `find` отбрасывает дробные доли суток, поэтому, если вам нужно найти файлы, которые изменялись в

течение последних суток, используйте параметр `0` (это вызывает замешательство, я знаю).

А как найти файлы, которые были модифицированы, например, от двух до четырех суток назад? Используйте опцию `-mtime +2 -a -mtime +4`. (Мы объединили две опции с помощью оператора AND.) Вероятно, вы получите несколько вчерашних файлов, но их можно удалить с помощью команды `grep -v` (см. главу 10). Помните, что опции `-atime`, `-ctime` и `-mtime` означают `n*24` часов, отсчитывая от текущего момента времени, так что результаты могут трактоваться в широком смысле.

ЗАМЕЧАНИЕ

Существуют и другие выражения, использующие время, которые я не стал описывать из-за недостатка места. В частности, вам могут понадобиться опции `-anewer`, `-cnewer`, `-newer` и `-newerXY`.

Отображение результатов при выполнении всех выражений (AND)

`find -a`

Главной особенностью команды `find` является возможность объединять несколько опций, чтобы конкретизировать поиск. Опция `-a` (или `-and`) позволяет связать вместе столько опций, сколько вам необходимо. Предположим, например, что вы хотите найти песню “Let it Bleed” группы “Rolling Stones”. Вы можете использовать опцию `-name "*Let It Bleed*"`, но она не даст необходимых результатов. Группа “Rolling Stones” записала альбом с таким же названием, поэтому нам надо отделить файлы от каталогов. Следовательно, надо также задать опцию `-type f`. Объединим вместе две опции.

```
$ cd Rolling_Stones
$ find . -name "*Let It Bleed*" -a -type f
./1972 More Hot Rocks/17 - Let It Bleed.mp3
./1995 Stripped/08 - Let It Bleed.mp3
./1969 Let It Bl eed/5 - Let It Bleed.mp3
```

Вроде бы все в порядке, но сколько песен группы "Rolling Stones" мы получили реально? Передадим результаты работы команды `find` программе `wc` (ее имя является сокращением от *word count*; см. главу 7) и зададим опцию `-l`, чтобы подсчитывать не слова, а строки.

```
$ cd Rolling_Stones
$ find . -name "*mp3*" -a -type f | wc -l
1323
```

Мы получили 1323 песни группы "Rolling Stones".

Отображение результатов при выполнении любого из выражений (OR)

```
find -o
```

Ранее мы использовали команду `find` для поиска всех записей группы "Clash" из альбома "London Calling", которые содержались в файлах размеров 10 Мбайт. Там же выполнялся поиск файлов, размер которых превышает 10 Мбайт, но объединить результаты опция `-size` не позволяла. Из предыдущего раздела вы узнали, что опция `-a` дает возможность объединять другие опции в виде логического выражения AND. Сейчас мы используем опцию `-o` (или `-or`) для реализации операции OR.

Итак, чтобы найти записи из альбома "London Calling", которые содержатся в файле размером не менее 10 Мбайт, воспользуемся следующей командой:

```
$ cd Clash
$ find . -size +10M -o -size 10M
./1977 The Clash/01 - Clash City Rockers.mp3
./1977 The Clash/13 - Police And Thieves.mp3
```

```
./1979 London Calling/15 - Lover's Rock.mp3
./1979 London Calling/18 - Revolution Rock.mp3
./2007 The Sandinista Project/04 - Jason
↳Ringenberg - Ivan Meets G.I. Joe.m4a
./1980 Sandinista!/01 - The Magnificent Seven.mp3
./1980 Sandinista!/02 - Hitsville U.K.mp3
[Results greatly truncated for length]
```

И снова проблема. Мы получили также сведения об альбоме "The Sandinista Project" группы "Clash", а это не входило в наши планы. Нам надо сделать следующее: во-первых, исключить данные о данном альбоме, а во-вторых, убедиться, что выражение `OR` работает корректно.

Чтобы убрать записи, посвященные альбому "The Sandinista Project", допишем в конец команды конструкцию `! -name "*Project*"`, чтобы получилась команда `find . -size +10M -o -size 10M ! -name "*Project"`. Однако она тоже не работает по нескольким причинам.

Опция `-name` является неправильной, потому что она ищет только имена файлов, игнорируя обратную косую черту перед ними. Слово, которое мы хотим исключить из списка результатов, — `Project` — представляет имя каталога, поэтому опция `-name` его никогда не обнаружит. Вместо этого следует использовать опцию `-wholename`, которая ищет весь путь, включая имя файла.

Другая проблема — способ организации команды. У нас есть два выражения, объединенных оператором `OR` с другим выражением для опции `wholename`. Чтобы проверить, что оператор `OR` делает именно то, что нам нужно, необходимо заключить его в скобки, которые объединят инструкции (как в алгебре!). Однако сочетания скобок и обратной косой черты следует избегать, потому что оболочка может их неправильно интерпретировать. По этой причине необходимо до и после инструкции вставить пробелы, иначе она не будет работать. В результате получаем следующую комбинацию:

```
$ find . \( -size +10M -o -size 10M \) -a
↳ ! -wholename "*Project*"
./1977 The Clash/01 - Clash City Rockers.mp3
```

```
./1977 The Clash/13 - Police And Thieves.mp3
./1979 London Calling/15 - Lover's Rock.mp3
./1979 London Calling/18 - Revolution Rock.mp3
./1980 Sandinista!/01 - The Magnificent Seven.mp3
./1980 Sandinista!/02 - Hitsville U.K.mp3
```

Альбом "Sandinista Project" прекрасен, но нам нужны только песни группы "Clash". Благодаря команде `find` мы нашли их.

ЗАМЕЧАНИЕ

Альбому "London Calling" посвящена страница сайта `Allmusic.com` (`www.allmusic.com/album/london-callingmw0000189413`) и статья в Википедии (`http://en.wikipedia.org/wiki/London_calling`).

Кроме того, с помощью опции `-o` можно выяснить, сколько песен записано на музыкальном диске. Можно начать со следующей команды, которую надо выполнить из каталога `/media/music`, используя опцию `-a` (терпение, мы еще доберемся до опции `-o`):

```
$ find . -name "*mp3" -a -type f | wc -l
105773
```

Сто пять тысяч? Не может быть. А, теперь понятно. Мы ищем только файлы `mp3`, а некоторые песни записаны в формате M4A. Поищем файлы `mp3` или `m4a`, а затем подсчитаем количество результатов с помощью команды `wc -l`:

```
$ find . \( -name "*mp3" -o -name "*.m4a" \) -a
↳-type f | wc -l
106666
```

Восемьсот девяносто три файла в формате m4a — вполне правдоподобный результат, но я забыл о файлах FLAC! Добавим еще одну опцию `-o`:

```
$ find . \( -name "*mp3" -o -name "*.m4a*" -o - name
↳"*.flac" \) -a -type f | wc -l
109709
```

Вот теперь кажется все: мы нашли около 110000 песен!

ЗАМЕЧАНИЕ

В предыдущем издании я получил такие результаты: 23407 файлов формата mp3, 0 файлов формата m4a, 18244 файла формата ogg (теперь их нет) и 556 файлов формата flac. Всего на диске было 42187 музыкальных записей. Очевидно, раньше у меня было меньше свободного времени!

Отображение результатов, если выражение не выполняется (NOT)

find -n

Мы уже использовали ранее символ `!` для того, чтобы инвертировать значение выражения. Вернемся к рассмотрению этого оператора. В предыдущем разделе с помощью команды `find` мы определили, сколько файлов `mp3`, `mp4a`, `ogg` и `flac` находится на диске. А сколько всего файлов на диске?

```
$ find . | wc -l
122255
```

Из 122255 файлов только 109709 имеют форматы mp3, m4a или flac. А зачем остальные? Сформируем команду `find`, которая исключит из поиска файлы, имена которых заканчиваются на `.mp3`, `.m4a` или `.flac`. Также нам не следует учитывать каталоги. Поместим эти четыре условия в круглые скобки и зададим перед всем выражением символ `!`. Таким образом мы отобразим лишь объекты, не удовлетворяющие четырем описанным выше условиям.

```
$ find . ! \( -name "*mp3" -o -name "*m4a" -o
↳-name "*flac" -o -type d \)
./Rock - Folk/Band/1970 Stage Fright/Art/
```

```
↳Cover.png
./Rock - Folk/Bob Dylan/2007 Dylan Hears a Who/
↳Art/Cover.jpg
./Rock - British Blues/Rolling Stones/1973 0908
↳ Wembley, London/Info.txt
./Classical - Opera/Puccini/Turandot/1972
.↳Sutherland, Pavarotti, Caballe/Libretto.pdf
./Rock - Pop/ABBA/1993 Abba Gold Greatest Hits/
↳01 - Dancing Queen.MP3
./Rock/Neil Young/1970 0301 Tea Party, Boston/
↳Info.htm
./Rock - Punk/Clash/2007 The Sandinista Project/
↳Booklet.pdf
./Jazz - Vocal/Frank Sinatra/1963 The Concert
.↳Sinatra/07 - This Nearly Was Mine.Mp3
[Результаты сокращены для экономии места]
```

Мы получили ответ, или, вернее, сведения, позволяющие его сформулировать. У нас есть файлы PDF, текстовые файлы, файлы HTML, а также файлы с расширением `MP3` вместо `mp3` (не считая файлов JEG, PNG и видеофайлов). Как и все программы Linux, команда `find` чувствительна к регистру символов (см. главу 1), поэтому при поиске по `"*mp3*"` не будут найдены файлы с суффиксом `.MP3`. Существует ли простой способ изменить имена файлов так, чтобы их суффиксы были представлены в нижнем регистре? Такой способ есть, и он предполагает использование команды `find`.

ПОДСКАЗКА

Выражение `-name` чувствительно к регистру, а выражение `-iname` — нет (буква `i` именно это и означает — *insensitive* (нечувствительный)). Следовательно, если нужно найти все файлы mp3, независимо от регистра их расширения, следует использовать команду `find . -iname "*mp3*"`, которая найдет файл с расширениями `mp3`, `MP3`, `Mp3` и `mP3`.

Выполнение действий над каждым найденным файлом

find -exec

Теперь вы готовы ознакомиться с той областью, в которой команда `find` реально проявляет свои огромные возможности. Над каждым файлом, найденным посредством команды `find`, можно выполнять произвольные действия. После опций, сужающих поиск (например, `-name`, `-type` или `-user`), можно задать опцию `-exec`, а затем ввести команду, которую необходимо применить к файлам. Для представления каждого файла используются символы {}, а команду надо завершить обратной косой чертой, чтобы отменить специальное действие точки с запятой, иначе оболочка воспримет ее как разделитель при объединении команд (см. главу 5).

В предыдущем разделе мы выяснили, что имена некоторых файлов на диске оканчиваются последовательностью символов `MP3`. Поскольку мы предпочитаем суффиксы, составленные из символов нижнего регистра, преобразуем все вхождения `MP3` в `mp3`. Для этой цели используем опцию `-exec` команды `find`. Сначала убедимся, что файлы с суффиксом `.MP3` действительно существуют.

```
$ find . -name "*MP3"
./Blues - Delta/Robert Johnson/1990 Complete
↳ Recordings/29 - Malted Milk.MP3
./Blues - Delta/Robert Johnson/1990 Complete
.↳Recordings/16 - Dead Shrimp Blues.MP3
./Blues - Delta/Robert Johnson/1990 Complete
↳ Recordings/11 - Terraplane Blues.MP3
```

Затем сформируем команду для изменения суффиксов. В качестве значения опции `-exec` зададим программу `rename`, которая позволяет изменять имя файла или его часть.

```
$ find . -name " *MP3"
↳ -exec rename 's/MP3/mp3/g' {} \;
```

ПОДСКАЗКА

Существует по крайней мере две разные версии команды `rename`, каждая из которых имеет свои синтаксис и опции. Я использовал пример, написанный Ларри Уоллом (Larry Wall), изобретателем языка программирования Perl, который — вполне ожидаемо — использует для подстановки синтаксис, похожий на язык Perl. Лучшим сообщением на эту тему является статья "A Tale of Two Renames", написанная Тимом Хини (Tim Heaney) и размещенная по адресу `http:// chnsa.ws/68`.

Для проверки версии команды `rename` откройте справочную страницу `man rename` и прокрутите ее вниз до раздела AUTHOR. Если в вашем дистрибутивном пакете Linux нет команды `rename` (что очень даже возможно), вам придется инсталлировать ее, следуя инструкциям, изложенным в главе 14.

Команде `rename` передаются инструкции по изменению имени в формате `s/`*`старое_имя`*`/`*`новое_имя`*. (В данном случае `s` — первая буква слова *substitute* (подставить).) Ознакомимся с результатами выполнения команды.

```
$ find . -name "*MP3"
./Blues - Delta/Robert Johnson/1990 Complete
↳ Recordings/29 - Malted Milk.mp3
./Blues - Delta/Robert Johnson/1990 Complete
↳ Recordings/16 - Dead Shrimp Blues.mp3
./Blues - Delta/Robert Johnson/1990 Complete
↳ Recordings/11 - Terraplane Blues.mp3
```

ПОДСКАЗКА

В нашем примере мы использовали формат `s/`*`старое_имя`*`/`*`новое_имя`*, но вместо него часто применяется формат `s/`*`старое_имя`*`/`*`новое_имя`*`/g` (буква `g` означает global (глобальный)). Если каждое имя файла встречается и заменяется только один раз, как файл "`Terraplane`

Blues.mp3, то команда s/MP3/mp3 работает просто отлично. А что делать с песней Роберта Джонсона под названием Corrupted MP3 Blues.MP3". Для того чтобы изменить оба имени MP3 на mp3, необходимо использовать инструкции s/MP3/mp3/g. Благодаря этому команда rename будет знать, что мы хотим заменить MP3 на mp3 не только в первом файле (как это делает инструкция s/MP3/mp3/, но и во втором.

Итак, все расширения MP3 заменены на mp3. Применим тот же подход для решения другой задачи. В предыдущем разделе мы выяснили, что на диске содержатся файлы m3u. Так получилось, что в именах многих из них содержатся символы подчеркивания, которые хотелось бы заменить пробелами. Сгенерируем список файлов m3u, содержащих символы подчеркивания. Используем шаблон поиска * *_*m3u. В нем слева и справа от символа подчеркивания помещены символы групповой операции, т.е. таким образом выявляются файлы, в имени которых содержится хотя бы один символ подчеркивания.

```
$ find . -name "*_*m3u"
./Holiday_-_Christmas/Christmas_With_The_Rat_
↳Pack.m3u
./Holiday_-_Christmas/Boots_For_Your_Stockings.m3u
./Classical_-_Baroque/Handel/Chamber_Music.m3u
./Classical_-_Opera/Famous_Arias.m3u
./R&B_-_Doo_Wop/Doo_Wop_Box.m3u
./Electronica/Aphex_Twin/I_Care_Because_You_Do.m3u
```

Теперь к каждому найденному файлу применим команду rename. Символ для подстановки имеет вид "\ "; при этом команда rename получает информацию о символе подчеркивания, а оболочка не обрабатывает его нежелательным для нас образом. Знак подчеркивания заменяем пробелом.

```
$ find . -name "* *m3u"
↳-exec rename 's/\ /_/g' {} \;
$ find . -name "* *m3u"
$
```

Команда работает так, как мы и ожидали.

ЗАМЕЧАНИЕ

Обратите внимание на то, что, перед тем, как применять к файлам команду изменения имени, необходимо выяснить, какие файлы затронет эта операция. Не стоит пренебрегать такой проверкой, в противном случае можно получить нежелательные результаты.

Более эффективный поиск файлов

```
find +
find | xargs
```

Недавно я работал на клиентском сервере, который из-за неправильной конфигурации оказался заполненным 1,5 миллиона временных файлов. Да, полтора миллиона файлов. Можно предположить, что эта проблема решается простым сочетанием команды `find` и `rm`, но, попытавшись сделать это, вы получите следующее:

```
$ ls | less
sess_000000001.txt
sess_000000002.txt
sess_000000003.txt
[Результаты сокращены с 1.5 млн до трех]
$ find . -name "sess_*" -exec rm {} \;
/bin/rm Argument list too long
```

Из-за ограничений ядра системы Linux задать такое громадное количество аргументов команды физически невозможно. Необходимо как-то избавиться от всех этих файлов, но как?

Новомодное решение заключается в использовании команды `find` (а затем надо терпеливо ждать, пока не исчезнут 1,5 млн файлов).

```
$ find . -name "sess_*" -exec rm {} +
```

В отличие от предыдущего случая, опция `-exec` здесь используется двумя способами: отсутствуют символы `\;` и используется символ `+` (фактически именно символ `+` в конце позволяет отбросить символы `\;`). Обычно команда `find` применяет команду `rm` к каждому найденному файлу, что очень неэффективно и приводит к слишком длинному списку аргументов, но символ `+` указывает команде `find`, что указанную порцию файлов необходимо объединить в одно целое и применять команду `rm` к каждой порции, а не к отдельным файлам.

Почему бы не использовать символ `+` с командой `find` во всех случаях? Потому что эта возможность появилась в системе UNIX относительно недавно, в 2006 г., и не все версии команды `find` ее поддерживают. Для проверки можно выполнить безопасную команду вроде `find . -exec ls {} +`. Если вместо сообщения об ошибке вы получите результаты, то ваша версия команды `find` поддерживает символ `+` (и приготовьтесь нажать комбинацию клавиш <Ctrl+C>, чтобы остановить прокрутку огромного списка файлов!).

Классический принцип "что сработало один раз, должно работать постоянно", подсказывает нам, что следует использовать команду `xargs`. Эта команда считывает аргументы из потока STDIN, а не из командной строки. Успех такого подхода объясняется тем, что команда `xargs` обрабатывает данные порциями, а не по одному (напоминает сочетание символа `+` и команды `find`, не так ли?). По этой причине команда `xargs` выполняется реже и позволяет избежать сообщений о слишком длинном списке аргументов. Насколько большими должны быть порции? Выполните команду `xargs --show-limits` (чтобы остановить выполнение команды `xargs`, нажмите комбинацию клавиш <Ctrl+D>).

ЗАМЕЧАНИЕ

Конвейеры и команда `xargs` похожим образом обрабатывают и используют результаты команд (причем команда `xargs` почти всегда использует конвейеры!), но

между ними есть существенная разница. Конвейер получает результат первой команды и использует его как ввод для *второй*. В то же время команда `xargs` получает результаты первой команды (как в конвейере) и использует его как *аргументы* второй.

Например, команда `ls -1 | wc -l` получает результат команды `ls -1` и использует его как ввод для команды `wc -l`, потому что команда `wc` использует поток `STDIN`, а команда `ls -1 | echo` ничего не выводит, потому что команда `echo` ожидает аргументы, а не данные из потока `STDIN`. Поскольку команда `echo` получает аргументы, команда `ls -1 | xargs echo` работает, потому что команда `xargs` превращает результаты работы команды `ls -1` в аргументы команды `echo`. (Да, повторять, как эхо, результаты команды `ls` довольно глупо, но я это делаю только для иллюстрации!)

Вернемся к 1,5 млн файлов и применим действие `-exec` с командой `find`, затем передадим результаты команды `find` команде `xargs` и удалим эти файлы.

```
$ find . -name "sess_*" | xargs rm
$ ls
[Все! 1.5 млн файлов удалены!]
```

Удаление 1,5 млн файлов занимает довольно много времени, но все же оно выполняется, а это уже хорошо.

ПОДСКАЗКА

Несмотря на то что команда `xargs` часто образует конвейер с командой `find`, ее можно использовать с разными командами. Однако в 90% примеров, которые можно найти в веб, используется комбинация команд `xargs` и `find`!

Поиск файлов, имена которых содержат пробелы

```
find -print0 | xargs -0
```

В предыдущем разделе был описан процесс удаления 1,5 млн файлов, имена которых выглядели как `sess_000000001.txt`, поэтому мы использовали сочетание команд `find` и `xargs`, например:

```
$ find . -name "sess_*" | xargs rm
```

Этот вариант работает, так как имя `sess_000000001.txt` и ему подобные не содержат пробелов. Если бы имена файлов выглядели, как, например, `sess 000000001.txt`, то мы получили бы такие результаты:

```
$ find . -name "sess_*" | xargs rm
rm: cannot remove './sess': No such file or
↳directory
rm: cannot remove '000000001.txt': No such file or
↳directory
rm: cannot remove './sess': No such file or
↳directory
rm: cannot remove '000000002.txt': No such file or
↳directory
^C
```

Символы `^C` в конце позволяют прекратить вывод трех миллионов строк с помощью комбинации клавиш <Ctrl+C> (каждому файлу соответствуют две строки, поскольку имена файлов содержат один пробел). Очевидно, что пробелы в именах файлов создают проблему. Но почему?

Команда `xargs` использует пробелы для разделения результатов, поэтому, когда команда `find` передает ей имя `sess 000000001.txt`, команда `xargs` видит их как `rm sess` и `rm 000000001.txt`. Поскольку таких файлов не существует, команда `rm` выдает ошибку. Кстати, это может оказаться опасным — если файл с именем `000000001.txt` существовал, то он будет удален.

Проблемы создают не только пробелы в именах файлов. Такие символы, как кавычки, апострофы, символы перехода на новую строку и обратная косая черта, также приводят к неожиданных результатам и сбоям. Для того чтобы решить эти проблемы, необходимо выполнить команду

```
$ find . -name "sess_*" -print0 | xargs -0 rm
```

Как правило, команда `find` передает каждое имя файла с символом перехода на новую строку (именно поэтому вы видите все результаты команды `find` в отдельных строках). Действие `-print0` указывает команде `find` сопровождать каждое имя файла не символом перехода на новую строку, а нулевым символом. Между прочим, опция `-0` указывает команде `xargs`, что для разделения результатов она должна использовать нулевой символ вместо пробела. Поскольку команды `find` и `xargs` работают вместе, вы не получите никаких сообщений об ошибке. Кроме того, опция `-0` указывает команде `xargs` игнорировать кавычки и любые другие символы, которые могут вызвать сбой.

Выводы

Как вы уже знаете, возможности команды `find` огромны. Чтобы получить подробную информацию о них, выполните команду `man find` или прочитайте руководства, доступные в веб. Разнообразные опции `find` можно использовать для получения списка файлов и каталогов, а опция `-exec` делает данную программу поистине бесценной, позволяя применять произвольные команды к результатам поиска. Можно также передавать выходные данные `find` другим командам. Команда `find` — одна из самых полезных в системе Linux. Используйте ее!

12

Оболочка

До сих пор в данной книге речь шла о выполнении команд в оболочке, но совсем не уделялось внимания самой оболочке. В этой главе мы рассмотрим две команды, оказывающие влияние на работу с оболочкой: `history`, которая отображает список команд, введенных ранее в командной строке, и `alias`, которая создает псевдонимы для других команд. Обе эти команды часто используются и позволяют сэкономить много времени при работе с командной строкой. Лень — положительное качество пользователя, а две команды, о которых пойдет речь, дают возможность проявить свою лень в полной мере.

Просмотр списка предыстории

`history`

Каждый раз, когда вы вводите в оболочке очередную команду, она сохраняется в файле `.bash_history`, расположенном в вашем рабочем каталоге (точка в начале имени указывает на то, что файл скрытый, и, чтобы увидеть его, надо использовать команду `ls -a`). По умолчанию в файле сохраняется 500 строк. Если вы хотите увидеть список команд, введенных вами ранее, выполните команду `history`.

```
$ history
496 ls
497 cd rsync_ssh
498 ls
499 cat linux
500 exit
```

ПРЕДУПРЕЖДЕНИЕ

Теперь вы понимаете, почему не следует вводить пароль и другие важные данные в командной строке. Каждый, кто может просмотреть содержимое файла `.bash_history`, получит доступ к ним.

Если вы хотите найти конкретную команду в своей истории, то это можно сделать несколькими способами. Во-первых, это можно сделать вручную. Поскольку в файле содержится 500 строк, они быстро промелькнут на экране и вы не сможете рассмотреть их. Для того чтобы организовать поэкранный вывод, используйте уже известную вам конструкцию

```
$ history | less
```

Теперь просматривать данные будет намного удобнее.

Если вы не хотите использовать команду `less`, то выполните команду `history`, указав количество предыдущих команд, которые вы хотите вывести на экран. Например, команда `history 50` выведет на экран последние 50 команд, что упрощает просмотр результатов.

Просмотр списка ранее выполненных команд — это прекрасная, но утомительная возможность. Для того чтобы автоматизировать эту функцию, необходимо передать результаты команды `history` команде `grep` (см. главу 10).

```
$ history | grep pandoc
105 pandoc -o temp.html temp.md
```

Отлично! Вот если бы эту команду можно было бы выполнить повторно! Впрочем, такая возможность тоже существует.

ПОДСКАЗКА

Если вам приходится часто выполнять конструкцию `history | grep` (как мне, например), то лучше создать псевдоним в файле `.bash_aliases`:

```
alias histg="history | grep"
```

Что за псевдоним? Ответ ищите в следующем разделе!

Повторное выполнение последней команды

```
!!
```

Если возникнет необходимость снова выполнить команду, которая была вызвана последней, введите два восклицательных знака. В результате произойдет обращение к списку предыстории и будет вызвана команда, расположенная последней в списке.

```
$ pwd
/home/scott
$ !!
pwd
/home/scott
```

Заметьте, что на экране сначала отображается сама команда, а затем результаты ее выполнения. Этот способ позволяет вам избавиться от части рутинной работы, которую приходится выполнять при пользовании системой. По умолчанию команда выводится просто для сведения. Если команда выполняется очень быстро, то кроме нее на экран выводятся и результаты ее работы.

Очевидно, это может привести к неожиданным ситуациям. Что если на самом деле выполняется не та команда, о которой вы *думали* (из-за того, что последняя команда отделена от текущего приглашения многими страницами результатов, или потому, что вы указали оболочке bash не сохранять определенные команды в истории и забыли об этом)?

Впрочем, есть способ устранить эту проблему. Добавьте в файл .bashrc следующие строки и перезагрузите этот файл:

```
# Verify substituted history expansion before running
shopt -s histverify
```

Первая строка — это комментарий, вторая — команда, указывающая оболочке bash выводить на экран из истории любые команды, не выполняя их. Да, это значит, что вы должны сами нажать клавишу <Enter>, но это небольшая цена за возможность увидеть будущее.

Если команда `!!` вам не нравится, нажмите клавишу <↑>. После того, как вы увидите историю и найдете то, что искали, нажмите клавишу <Enter>.

ПРЕДУПРЕЖДЕНИЕ

Все это хорошо. Но если вы выполнили многострочную команду, то оболочка bash по умолчанию сохранит каждую строку как отдельную запись. Это значит, что вы не сможете просто выполнить многострочную команду с помощью клавиши <↑> или команды `!!`. Однако, если добавить в файл `.bashrc` следующие две строки (первая — комментарий, в принципе необязательный, но полезный), проблема исчезнет:

```
# Save each line of a multi-line command in.the
same history entry
shopt -s cmdhist
```

Перезагрузите файл `.bashrc` (см. главу 1), и вы сможете повторно выполнять многострочные команды.

Это действительно прекрасная возможность заставить компьютер выполнять за вас утомительную работу.

Вызов предыдущей команды путем указания ее номера

```
![##]
```

Команда `history` автоматически помещает перед именем каждой предыдущей команды номер. Если вы хотите вызвать одну из команд, выполненных ранее, и знаете ее номер в списке предыстории, то достаточно ввести восклицательный знак и указать после него номер команды. В результате команда будет выполнена повторно.

```
$ pwd
/home/scott
$ whoami
scott
$ !499
pwd
/home/scott
```

Если вы не помните номер, вам надо снова вызвать команду `history`. Заметьте, что в приведенном примере команда `pwd` сначала имела номер 499, но после того, как она будет вызвана с помощью выражения `!499`, ее номер изменится на 498, поскольку содержимое списка будет сдвинуто в результате включения в нее новой команды.

Вызов предыдущей команды путем указания строки символов

```
![строка]
```

Возможность вызова команды путем указания ее номера удобна, но для этого надо запомнить номера команд в списке предыстории, что достаточно трудно (можно, конечно, обработать выходные данные команды `history` посредством

программы grep, но такое решение нельзя назвать оптимальным). Проще ссылаться на команду, выполненную ранее, по ее имени. Если вы введете после восклицательного знака несколько первых букв из имени команды, оболочка вызовет первую из подходящих команд, содержащихся в списке предыстории (просмотр .bash_history осуществляется от конца к началу).

```
$ cat /home/scott/todo
Buy milk
Buy dog food
Renew Linux Magazine subscription
$ cd /home/scott/pictures
$ !cat
cat /home/scott/todo
Buy milk
Buy dog food
Renew Linux Magazine subscription
```

Предположим, в списке предыстории команда cat присутствует трижды: по номерами 35 (cat /home/scott/todo), 412 (cat /etc/apt/sources.list) и 496 (cat /home/scott/todo). При вводе !cat будет запущена команда с номером 496. Если вы хотите выполнить команду, имеющую номер 412, введите либо !412, либо после восклицательного знака введите информацию, достаточную для распознавания требуемой команды.

```
$ !cat /etc
cat /etc/apt/sources.list
deb http://us.archive.ubuntu.com/ubuntu precise main
deb-src http://us.archive.ubuntu.com/ubuntu precise
↳main
```

Поскольку запомнить слово значительно проще, чем число, вы, вероятнее всего, предпочтете описанный здесь метод повторного вызова предыдущих команд. К тому же в любой момент вы можете вызвать команду history и ознакомиться со списком предыстории.

Поиск и выполнение предыдущей команды

```
^-r (Ctrl-r)
^-s (Ctrl-s)
^-g (Ctrl-g)
```

Мы рассмотрели несколько способов выполнения предыдущих команд, и все они были очень удобными, но лучший из них я отложил на десерт. Допустим, ваша история содержит следующие строки:

```
1471 ssh hlp@azathoth.miskatonic-expedition.com
1472 cd /var/www
1473 chmod 755 ~/bin/backup.sh
1474 ssh hlp@nyarlathotep.miskatonic-expedition.com
1475 find /var/www -name "bootstrap.js"
1476 cd ~/bin
1477 ssh hlp@cthulhu.miskatonic-expedition.com
```

Предположим, что история содержит 500 строк и мы хотим, чтобы оболочка SSH (см. главу 16) вернулась на сайт `nyarlathotep.miskatonicexpedition.com`, но не хотим набирать все эти команды заново. Разумеется, мы не хотим сотни раз нажимать клавишу <↑>, а выполнять команду `history | grep ssh`, чтобы найти соответствующий номер строки с требуемой командой, нам лень. Что делать? Искать!

Нажмите комбинацию клавиш <Ctrl+R> (в документации по Linux ее иногда обозначают как ^-r или C-r), и вы увидите внизу окна терминала строку

```
(reverse-i-search)'':
```

Начинайте набирать символы `ss`, и после этого оболочка `bash` начнет обратный просмотр истории (об этом свидетельствуют слова “reverse” (обратный) и “search” (поиск) в приглашении). По мере того как вы будете набирать эти буквы, на экране будут появляться соответствующие команды,

независимо от того, в каком месте стоят символы ss. Например, вы можете увидеть строку

```
(reverse-i-search)'ss': less /etc/inputrc
```

Однако, когда вы наберете символ h, настроив свой поиск на команду ssh, на экране появится первый результат обратного просмотра истории (буква i в приглашении означает "incremental" (пошаговый)). Если в последних 500 строках истории нет свидетельства того, что оболочка SSH не соединялась с какими-нибудь другими серверами, то вы увидите следующее:

```
(reverse-i-search)'ssh': ssh hlp@cthulhu.
↳miskatonic-expedition.com
```

Это не совсем то, что мы искали, но мы на правильном пути. Для того чтобы найти такую же команду в истории (в данном случае — ssh), еще раз нажмите комбинацию клавиш <Ctrl+R>. Буква "R" означает "reverse" (обратный), поэтому мы увидим то, что хотели:

```
(reverse-i-search)'ssh': ssh hlp@nyarlathotep.
↳miskatonic-expedition.com
```

Однако предположим, что мы поторопились и нажали комбинацию клавиш <Ctrl+R> *лишний* раз. Это значит, что мы найдем еще одну предыдущую команду ssh в еще более ранней истории.

```
(reverse-i-search)'ssh': ssh hlp@azathoth.
↳miskatonic-expedition.com
```

Не страшно. Мы зашли слишком далеко в прошлое и теперь хотим вернуться назад, поэтому нажмем <Ctrl+S>, чтобы выполнить прямой просмотр истории (конечно, эти клавиши следует нажимать столько раз, сколько нужно). Сделав это, мы вернемся в нужную точку.

```
(reverse-i-search)'ssh': ssh hlp@nyarlathotep.
↳miskatonic-expedition.com
```

Теперь нажмите клавишу <Enter>. Если вы точно следовали моим указаниям, то на экране появится команда, для выполнения которой придется еще раз нажать клавишу <Enter>. Если же вы не следовали моим указаниям, то нажатие клавиши <Enter> приведет к немедленному выполнению команды.

ПОДСКАЗКА

Если нажатие комбинации клавиш <Ctrl+S> не приводит к результату, значит, возник конфликт с конкретной привязкой клавиш. На многих компьютерах с системой Linux команда <Ctrl+S> посылает последовательность XOFF для контроля потока (более подробную информацию можно найти на странице `https://en.wikipedia.org/wiki/Software_flow_control`). Это довольно старый и очень редко используемый механизм. Для того чтобы временно отключить его, введите следующую команду и нажмите клавишу <Enter>:

```
$ stty -ixon
```

Команда <Ctrl+S> должна заработать, но, как только вы выйдете из системы и вернетесь снова, флажок XOFF снова будет сброшен. Для того чтобы он был установлен постоянно, отредактируйте файл `.bashrc`, вставив в него следующие строки:

```
# Disable XOFF flow control so that Ctrl-s
↳does forward history searching
stty -ixon
```

Сохраните файл, перезагрузите его (`source ~/.bashrc`), и все будет хорошо.

А что, если, увидев искомую команду, вы скажете себе: "Это не совсем не то, что я хотел сделать. Я не только не собирался выполнять эту команду, но и вообще не хотел искать команды!" В таком случае прервите процесс обратного просмотра истории, нажав комбинацию клавиш <Ctrl+G>.

Очень хорошо, но нельзя ли сделать еще лучше? Да! Вспомните, что команды <Ctrl+R> и <Ctrl+S> выполняют *пошаговый* просмотр истории. Совершенно очевидно, что этот поиск может дать массу ложноположительных результатов. Допустим, мы работаем на сервере `azathoth` и хотим запустить оболочку SSH на сервере `nyarlathotep`. Мы нажимаем клавиши <Ctrl+R>, одновременно набирая команду `ssh`, а затем получаем следующую информацию (на самом деле в обратном порядке, но это не имеет значения):

```
cat ~/.ssh/config
ssh hlp@nyarlathotep.miskatonic-expedition.com
cd ~/.ssh
man ssh
less intern-ssh-instructions.txt
```

Обратите внимание на то, что пошаговый поиск обнаруживает строку `ssh`, где бы она ни находилась. Это превосходно, но мы хотели зайти в оболочку SSH на сервере `nyarlathotep`, а не разбираться со всеми обнаруженными совпадениями. Посмотрим, как это сделать.

Отредактируйте файл `~/.inputrc` (см. главу 1), добавив в него следующие строки:

```
# When pressing up or down arrows, show only
# history that matches what was already typed
"\e[A":history-search-backward
"\e[B":history-search-forward
```

Сохраните этот файл и перезагрузите: введите команду `bind -f ~/.inputrc` и нажмите клавишу <Enter>. (В следующий раз, когда вы зайдете в систему через другую оболочку, оболочка `bash` автоматически откроет этот файл, так что достаточно ввести эту команду, если вы хотите изменить настройки.) Теперь вернитесь в командную строку и не нажимайте клавиши <Ctrl+R>, чтобы начать поиск команды в истории, а просто наберите строку `ssh` и нажмите клавишу <↑> (в нашем примере символы `\e[A` соответствуют клавише <↑>,

а \e[B — клавише <↓>). Посмотрите, какая команды появится на экране первой!

```
$ ssh hlp@nyarlathotep.miskatonic-expedition.com
```

Нажмите клавишу <Enter>. Теперь мы соединились с сервером nyarlathotep. Почему? Потому что, когда мы нажали клавишу <↑>, была найдена команда, которую мы ввели (аналогично команде <Ctrl+R>), но на этот раз на экран выводились только те результаты, в которых набранные нами символы находились *в начале строки*. Больше не нужно просматривать каждую строку, содержащую символы ssh, — вы увидите только строки, в которых символы ssh стоят в начале строки. Проверьте и убедитесь, насколько это удобно!

ЗАМЕЧАНИЕ

Для того чтобы убедиться в том, что перезагрузка файла .inputrc приводит к нужному результату, введите следующую команду, нажмите клавишу <Enter>, и вы увидите следующее:

```
$ bind -P | grep history-search
history-search-backward can be found on "\e[A".
history-search-forward can be found on "\e[B".
```

Это сообщение означает, что обратный поиск связан с клавишей <↑>, а прямой поиск — с клавишей <↓>. Это именно то, чего мы хотели. Если бы мы получили что-то другое, то возникла бы проблема:

```
$ bind -P | grep history-search
history-search-backward is not bound to any keys
history-search-forward is not bound to any keys
```

Проще всего открыть новую оболочку и выполнить ее проверку. Если проверка пройдет успешно, закройте старую оболочку и начинайте работать с новой. Если проверка закончится неудачей, то внимательно проверьте файл ~/.inputrc — в нем есть ошибка.

Отображение псевдонимов команд

`alias`

Если вы часто используете какую-либо команду, достаточно длинную и трудную для ввода, целесообразно создать для нее псевдоним. После этого, указав псевдоним, вы выполните ту команду, для которой он был создан. Конечно, если команда слишком сложная или занимает несколько строк, желательно оформить ее в виде сценария или функции. Но для относительно небольших команд псевдонимы — вполне приемлемое решение.

Как вы вскоре убедитесь, псевдонимы могут быть или временными и храниться в оперативной памяти до конца сеанса работы с оболочкой, или "постоянными" и храниться в файле в вашем домашнем каталоге. (Я заключил слово "постоянные" в кавычки, потому что вы всегда можете удалить эти псевдонимы.) Псевдонимы можно найти в файле `.bashrc`, но на самом деле следует использовать файл `bash_aliases`. (Более подробно о файлах `.bashrc` и `.bash_aliases` см. в конце главы 1.)

Псевдонимы хранятся в файле, находящемся в рабочем каталоге каждого пользователя. Имя этого файла `.bashrc` либо (что вероятнее) `.bash_aliases`. В большинстве дистрибутивных версий Linux уже определены некоторые псевдонимы. Для того чтобы просмотреть их список, надо ввести в командной строке команду `alias`.

```
$ alias
alias la='ls -a'
alias ll='ls -l'
```

Число предопределенных псевдонимов обычно невелико, и при желании можно задать новые. Как это сделать, вы узнаете ниже.

Просмотр псевдонима конкретной команды

```
alias [имя псевдонима]
```

Определив несколько псевдонимов, легко забыть, какой из них принадлежит какой команде. По команде `alias` может отображаться слишком много информации, и, для того чтобы разобраться в ней, может потребоваться время. Если вас интересует, какие действия будут выполнены при указании конкретного псевдонима, введите его имя после команды `alias`.

```
$ alias wgetpage
alias wgetpage='wget --html-extension -recursive
  --convert-links --page-requisites --no-parent $1'
```

В данном случае вы быстро и без труда получили сведения о том, какой команде соответствует псевдоним `wgetpage`.

ЗАМЕЧАНИЕ

Программа `wget` будет подробно рассмотрена в главе 16.

Создание нового временного псевдонима

```
alias [псевдоним]='[команда]'
```

Если вы обнаружили, что вам слишком часто приходится вводить одну и ту же команду, пора определить для нее псевдоним. Так, для просмотра подкаталогов текущего каталога используется команда `ls -d */`. Для нее можно создать временный псевдоним следующим образом:

```
$ ls -d */
by_pool/   libby_pix/   on_floor/
$ alias lsd='ls -d */'
```

```
$ lsd
by_pool/   libby_pix/   on_floor/
```

Используя таким образом механизм псевдонимов, необходимо учитывать некоторые особенности. В имени псевдонима не должно быть символа =, так как справа от него указывается команда, которой соответствует псевдоним. Сама команда может содержать этот символ. Кроме того, псевдоним, созданный таким способом, действует лишь до завершения сеанса работы с оболочкой.

Если вы хотите создать псевдоним, который будет действовать и в последующих сеансах, вы сможете это сделать, прочитав следующий раздел.

Создание нового постоянно действующего псевдонима

```
alias [имя псевдонима]='[команда]'
```

Если вы хотите, чтобы созданный вами псевдоним не удалялся по окончании сеанса, надо включить его в файл, который оболочка использует для хранения псевдонимов. Обычно это либо `.bashrc`, либо `.bash_aliases`. Предположим, в вашей системе используется `.bash_aliases`. Независимо от того, с каким файлом приходится работать, редактируя его, будьте внимательны, иначе при следующей регистрации могут возникнуть проблемы. А лучше всего перед редактированием файла создать его резервную копию.

ЗАМЕЧАНИЕ

Как определить, какой файл следует использовать? Для этого достаточно ввести выражение `ls -a ~`. Если вы увидите имя `.bash_aliases`, используйте этот файл, в противном случае найдите файл `.bashrc` и проверьте, определены ли в нем другие каталоги. Если в этом файле

нет определений псевдонимов, попробуйте найти файл .profile. В некоторых случаях используется именно он. Эти файлы описывались в конце главы 1.

Для того чтобы включить псевдоним в файл .bash_aliases, откройте файл с помощью текстового редактора и добавьте строку наподобие следующей:

```
alias lsd='ls -d */'
```

То же правило, которое использовалось при создании временных псевдонимов, применимо и здесь: псевдоним не должен иметь знака равенства. После того как вы включили псевдоним в .bash_aliases, сохраните файл и закройте редактор. Вы увидите, что псевдоним еще не работает. Чтобы вновь созданный псевдоним вступил в действие, файл .bash_aliases (равно как и .bashrc) требует перезагрузки. Сделать это можно двумя способами. Вы можете либо завершить сеанс работы, а затем снова зарегистрироваться, что неудобно и поэтому не рекомендуется, либо ограничиться перезагрузкой файла с помощью следующей команды:

```
$ . .bash_aliases
```

Теперь новый псевдоним должен работать.

ЗАМЕЧАНИЕ

Некоторые специалисты избегают использовать псевдонимы, которые изменяют поведение обычных команд, например, alias rm="rm -i", потому что это, дескать, вырабатывает вредную привычку и при работе на другом компьютере может возникнуть опасная ситуация. Я считаю, что это несерьезный аргумент. Если псевдонимы облегчают работу, то их следует использовать. Я считаю, что пользователи — достаточно умные люди, чтобы, работая на другом компьютере, не использовать на нем свои псевдонимы.

Удаление всех псевдонимов

unalias

Задачи, которые приходится выполнять, работая на компьютере, меняются, и может случиться так, что псевдоним будет больше не нужен. Для того чтобы удалить его, используется команда `unalias`.

```
$ ls -d */
by_pool/ libby_pix/ on_floor/
$ alias lsd='ls -d */'
$ lsd
by_pool/ libby_pix/ on_floor/
$ unalias lsd
$ lsd
$
```

Заметьте, что данная команда позволяет безвозвратно удалить лишь временные псевдонимы, например, псевдоним `lsd` из предыдущего примера. Если же вы примените эту команду к псевдониму, определенному в файле `.bash_aliases`, он также будет недоступен, но лишь в течение текущего сеанса. Когда вы повторно зарегистрируетесь в системе, его снова можно будет использовать.

Для того чтобы исключить псевдоним из файла `.bash_aliases`, надо открыть этот файл с помощью редактора и вручную удалить соответствующую строку. Если же вы считаете, что псевдоним может потребоваться вам в дальнейшем, ограничьтесь включением в начало этой строки символа #, как показано в следующем примере:

```
# alias lsd='ls -d */'
```

Сохраните файл `.bash_aliases`, перезагрузите его с помощью команды `.bash_aliases`, и псевдоним не будет работать. Если у вас снова возникнет необходимость в данном псевдониме, откройте файл `.bash_aliases`, удалите символ # и снова загрузите файл.

Создание новой временной функции

```
function [function name] { }
[function name] () { }
```

Если вам нужно нечто более сложное, чем псевдоним, но менее сложное, чем сценарий, значит, вам нужна функция.

В официальном справочнике *GNU Bash Reference Manual* функция оболочки описывается как "способ группирования команд для последующего выполнения с помощью отдельного имени для группы", и это абсолютно правильно. Во многих ситуациях функция вызывается так же, как псевдоним. Вы просто набираете имя функции и аргументы, которые хотите ей передать. Большая разница, которую мы продемонстрируем позже, заключается в том, что функция позволяет делать то, чего не могут псевдонимы.

Есть два способа объявить функцию. Во-первых, можно использовать команду `function`, за которой следует имя функции, в сочетании с командами, которые вы хотите выполнить в функции, заключив их в фигурные скобки (`{` и `}`). Во-вторых, можно использовать метод, более привычный для программистов: набрать имя функции, пустую пару круглых скобок, а затем пустую пару фигурных скобок. Выбор метода — дело вкуса, но я предпочитаю второй вариант, поэтому именно его мы будем использовать в последующих примерах.

Временные функции можно создавать в одной строке. В следующем примере мы сначала создаем новый каталог с помощью команды `mkdir -p`, а затем переходим в него, выполняя команду `cd`:

```
$ mkcd () { mkdir -p "$1"; cd "$1"; }
$ mkcd lovecraft
$ pwd
/home/scott/lovecraft
```

Если функция задается в одной строке, то после каждой команды, в том числе последней, *необходимо* ставить точку с запятой.

ЗАМЕЧАНИЕ

Возможно, вы уже догадались, что `$1` — это *позиционный параметр* функции, который соответствует первому аргументу, следующему за именем функции. Более подробно смысл этого параметра будет описан ниже.

Кроме того, функции могут занимать несколько строк. Если вы создаете функцию в командной строке (она же временная, в конце концов), то приготовьтесь увидеть то, чего еще не видели. Когда вы наберете первую строку новой функции и нажмете клавишу <Enter>, на экране появится *второе приглашение* >. Тем самым оболочка bash сообщает вам, что для завершения команды необходимо ввести дополнительные данные. В нашем случае, поскольку мы создаем функцию, оболочка ждет закрывающую фигурную скобку } и нажатия клавиши <Enter>. Пока вы не введете строки функции и не нажмете клавишу <Enter>, каждая последующая строка будет начинаться с символа >, означая, что оболочка ожидает данные.

Этот процесс трудно проиллюстрировать в книге, но следующий пример должен разъяснить эту идею. Я нажал клавишу <Enter> после каждой из четырех строк, приведенных ниже; после первого нажатия появилось приглашение, которое не исчезло с экрана, пока я не ввел закрывающую фигурную скобку } и не нажал клавишу <Enter>.

```
$ mkcd () {
> mkdir -p "$1"
> cd "$1"
> }
```

Каким бы способом вы ни создали функцию в командной строке, она будет действовать лишь до завершения активной сессии оболочки. Когда вы выходите из сеанса, функция исчезает.

Если вы хотите создать функцию, которая продолжит существование даже после окончания сеанса, читайте следующий раздел.

ПОДСКАЗКА

Очень полезные и поучительные функции можно найти с помощью любой поисковой системы, набрав фразу "полезные функции bash" или что-нибудь похожее.

Создание новой постоянной функции

```
function [function name] { }
[function name] () { }
```

Если вы хотите создать функции, которые будут существовать между сеансами оболочки, то необходимо записать их в файл, используемый оболочкой. Чаще всего для этого используется файл `.bashrc` или `.bash_aliases`; я отдаю предпочтение файлу `.bash_aliases`, потому что он уже существует и я включил в файл `.bashrc` ссылку на него.

У некоторых людей возникает когнитивный диссонанс, когда они слышат о хранении функций в файле `.bash_aliases`. Если с вами это случилось, то вы всегда можете создать файл с именем `.bash_functions` в своем домашнем каталоге и сослаться на него в файле `.bashrc`:

```
if [ -f ~/.bash_functions ]; then
source ~/.bash_functions
fi
```

Если создано *слишком много* функций или файл функций стал слишком большим, то его можно разделить на несколько файлов в скрытом каталоге и сослаться на них из файла `.bashrc`:

```
if [ -d ~/.bash_functions ]; then
for file in ~/.bash_functions/*; do
```

```
source "$file"
done
fi
```

Обратите внимание на опцию `-d`, которая используется вместо опции `-f`, как в предыдущем примере; она, очевидно, означает слово "directory" (каталог), в отличие от опции `-f`, означающей файл "file" (файл). Для того чтобы добавить функцию в файл `.bash_aliases`, откройте его в текстовом редакторе и добавьте свою функцию. Если функция короткая и ее длина не превышает одной строки, то можно сделать так:

```
mkcd () { mkdir -p "$1"; cd "$1"; }
```

Если функция занимает несколько строк, что часто объясняется желанием повысить ее читабельность, то можно сделать следующее:

```
knownhosts () {
[ "${1}" ] || return
local date=$(date +%Y-%m-%d-%H.%M.%S)
sed -i_${date} -e "${1}d" ~/.ssh/known_hosts
}
```

Попробуем разобраться в том, что здесь происходит. Я считаю эту функцию очень полезной для протокола SSH (подробнее об этом — в главе 16). Периодически открытый SSH-ключ сервера может изменяться. Для этого необходимо отредактировать файл `known_hosts` в каталоге `~/.ssh`, чтобы удалить строку, содержащую ссылку на этот сервер, и получить для него другой открытый ключ.

К счастью, протокол SSH уведомляет в своем сообщении об ошибке, в какой строке обнаружена ошибка:

```
Offending key in /home/scott/.ssh/known_hosts:8
```

Я должен обслуживать много серверов, и мне показалось утомительным постоянно открывать файл `known_hosts`, чтобы удалить строки, поэтому я написал функцию. Для того чтобы использовать ее, достаточно набрать имя и задать номер строки, указанный в сообщении протокола SSH.

```
$ knownhosts 8
```

Восьмая строка удалена, и теперь можно установить новое соединение с требуемым сервером. Все просто. Как же это работает?

Первая строка — `[ "${1}" ] || return` — проверяет, что аргумент (в нашем случае это номер строки) указан после имени команды. Если нет, то происходит выход из функции (кстати, очень полезная возможность!).

Вторая строка — `local date=$(date +%Y-%m-%d-%H.%M.%S)` — создает локальную переменную с именем `date`, содержащую результат работы команды `date`. По умолчанию все переменные, созданные в функции, являются глобальными; иначе говоря, они продолжают существовать после выхода из функции. В противоположность этому, локальная переменная доступна лишь в функции, и после выхода из функции она исчезает. В данном случае переменная `date` предназначена для хранения текущей даты, например `2015-07-25-22.21.41`.

Третья строка — `sed -i_${date} -e "${1}d" ~/.ssh/known_hosts` — использует команду `sed` (см. главу 7), чтобы выполнить несколько действий. Опция `-i` указывает команде `sed` отредактировать файл, но сначала создать резервную копию и добавить значение переменной `date` в нее, например, так: `known_hosts_2015-07-25-22.21.41`. (Это означает, что со временем в каталоге `~/.ssh` накопится множество резервных файлов, но это не проблема: просто не забывайте периодически их удалять.) Опция `-e` удаляет проблематичную запись о сервере из файла. Номер, заданный как аргумент функции, — в нашем случае `8` — передается функции с помощью параметра `${1}`.

ЗАМЕЧАНИЕ

Обратите внимание на фигурные скобки `{}` вокруг переменных. Это не обязательно, но помогает выделить переменные в окружающем тексте, чтобы избежать неоднозначности Сравните, например, `${1}d` и `$1d`. Эти выражения означают одно и то же, но первое намного понятнее.

Не имеет значения, как вы внесете функцию в файл .bash_aliases. Просто сохраните файл и закройте его, когда закончите работу над функцией. Теперь необходимо перезагрузить файл .bash_aliases, чтобы можно было вызывать новую функцию. Для этого следует выполнить команду

```
$ source ~/.bash_aliases
```

Теперь новая функция готова к работе. Надеюсь, она повысит эффективность вашей работы!

Вывод на экран списка всех функций

Если вы хотите увидеть список всех своих псевдонимов, вы просто выполняете команду alias. К сожалению, для функций нет такого инструмента. Вместо этого можно написать маленькую, но очень полезную функцию, которая выводит на экран список всех ваших функций!

```
listfunc () {
for func in $(compgen -A function | grep -v _);
do
declare -f $func;
echo -e "\r";
done
}
```

Здесь используется цикл for, команда подстановки (см. главу 5), команда grep (см. главу 10) и две команды (compgen и declare), которые я оставляю читателям в качестве домашней работы. Включите эту функцию в файл .bash_aliases. Когда вам понадобится вызвать ее, наберите имя listfunc в командной строке. Вот и все!

ЗАМЕЧАНИЕ

Некоторые пользователи думают, что просмотреть список всех своих функций можно с помощью команды declare -f. Это правда, но вы также увидите в списке

имена всех других функций, доступных в вашей оболочке, включая все функции, поставляемые вместе с дистрибутивным пакетом. Когда я выполнил функцию `listfunc`, то выяснилось, что все мои функции занимают 59 строк. Когда я выполнил команду `declare -f`, все остальные функции, не созданные мною, расширили этот список до 2213 строк!

Удаление функции

```
unset -f [function name]
```

Удалить псевдоним очень просто — достаточно выполнить команду `unalias`. Однако команды `unfunction` не существует. Вместо этого, закончив работу с функцией, необходимо выполнить команду `unset` с опцией `-f` (что означает — кто бы мог подумать! — "function" (функция)).

```
$ mkcd () { mkdir -p "$1"; cd "$1"; }
$ mkcd lovecraft
$ pwd
/home/scott/lovecraft
$ unset -f mkcd
$ mkcd cthulhu
mkcd: command not found
```

Впрочем, эта команда применяется только к временным функциям оболочки, рассмотренным выше. Функция `mkcd` из предыдущего примера будет успешно удалена. Применение команды `unset` к функции, указанной в файле `.bash_aliases` (или `.bash_functions`), тоже приведет к их удалению, но только на время сеанса. Когда вы завершите сеанс и выйдете из системы или перезагрузите файл `.bash_aliases`, функция появится вновь.

Для того чтобы удалить функцию из файла `.bash_aliases`, необходимо отредактировать этот файл и вручную удалить строки, содержащие эту функцию. Если вы предполагаете, что

в дальнейшем эта функция может вам снова понадобиться, то просто поставьте символ решетки (обозначающий комментарий) в начале строки, содержащей функцию:

```
# mkcd () { mkdir -p "$1"; cd "$1"; }
```

Сохраните файл `.bash_aliases`, перезагрузите его с помощью команды `source ~/.bash_aliases`, и функция больше не будет работать. Когда в будущем она понадобится вновь, откройте файл `.bash_aliases`, удалите знак решетки, сохраните и перезагрузите файл, и все будет хорошо.

Когда следует использовать псевдоним, а когда функцию?

Итак, мы рассмотрели псевдонимы и функции, но когда целесообразно применять те или другие? У псевдонимов и функций есть несколько общих свойств. И псевдонимы, и функции оболочка загружает в оперативную память. (Современные компьютеры являются настолько надежными, а функции занимают так мало места в памяти, что об этом факторе можно даже не думать.) И псевдонимы, и функции можно использовать в текущей оболочке, а значит, они влияют только на окружение текущей оболочки (в отличие от сценариев).

Однако между ними есть важные различия. В официальном справочнике *GNU Bash Reference Manual* о псевдонимах сказано следующее: "Практически во всех ситуациях предпочтительнее использовать функции, а не псевдонимы". Что это значит? С одной стороны, оболочка `bash` всегда выполняется *после* функций, а с другой стороны, это практически незаметно.

А что можно сказать о предназначении псевдонимов и функций? В основном, но не всегда, псевдонимы используются для превращения длинных команд в короткое и запоминающееся выражение (например, `alias wgetpage="wget --html-extension--recursive --convert-links --page-re-`

quisites--no-parent") или для добавления требуемых аргументов к командам (например, alias lsd="ls -d */").

Функции — это намного более длинные и сложные последовательности команд, часто содержащие логические операторы и циклы. Например, следующие действия можно оформить в виде функции (см. предыдущий раздел), но невозможно представить как псевдоним:

```
listfunc () {
for func in $(compgen -A function | grep -v _);
do
declare -f $func;
echo -e "\r";
done
}
```

Кроме того, псевдонимы и функции совершенно поразному обрабатывают аргументы. Как будет показано далее, псевдонимы передают аргументы командам в том порядке, в котором вы их вводите. В то же время функции оболочки позволяют запомнить аргументы, а затем переупорядочить, проверить, удалить и обработать их.

ЗАМЕЧАНИЕ

Аргументы — это входные данные, которые передаются команде. Например, рассмотрим функцию `mkcd () { mkdir -p "$1"; cd "$1"; }`. Когда вы вводите строку `mkcd cthulhu` и нажимаете клавишу <Enter>, строка `cthulhu` считается аргументом. Параметры определяются для функции или программы; например, в предыдущих примерах строка `$1` определяет параметр. (Точнее говоря, `$1` — это *позиционный параметр*, соответствующий первому аргументу; для ссылки на другие аргументы можно использовать другие позиционные параметры.) Это можно интерпретировать так: то, что вы определяете, является параметром, а то, что вы фактически передаете, — аргументом.

Можно попытаться использовать позиционные параметры с псевдонимами, но это пустые хлопоты. Аргумент псевдонима всегда добавляется в конец.

```
$ alias lstest="ls -l $1 | grep tar"
$ lstest /bin
grep: /bin: Is a directory
$ unalias lstest
```

Чем объяснить эту ошибку? Дело в том, что оболочка `bash` автоматически добавляет ваш аргумент — `/bin` — в конец, сразу после команды `grep tar`. Следовательно, команды `ls -l / bin | grep tar` вы на самом деле выполняете команду `ls -l | grep tar /bin`. По этой причине утилита `grep` выдает сообщение об ошибке: вы отдали команду искать слово "tar" в имени каталога, что неправильно, а не в имени файла или в списке файлов, что было бы правильно.

Итак, оболочка `bash` помещает аргументы псевдонима в конце команды (команд). Теперь рассмотрим, как ту же саму задачу можно решить с помощью функции:

```
$ lstest () { ls -l $1 | grep tar; }
$ lstest /bin
-rwxr-xr-x root root tar*
-rwxr-xr-x root root unicode_start*
```

Теперь все в порядке, потому что функция позволяет поместить аргумент туда, куда вы хотите, а данном случае — сразу после первой команды, `ls -l $1`. В своем инструментарии вы обнаружите и псевдонимы, и функции и по мере освоения системы Linux постепенно научитесь правильно их использовать.

Выводы

Вы, как пользователь системы Linux, заинтересованы в том, чтобы, решая конкретную задачу, вводить с клавиатуры как можно меньше символов. В этом наверняка были убеждены авторы Unix, именно потому они назвали команду `ls`, а не `list`,

`mkdir` и не `makedirectory`. Команды `history` и `alias`, которые были рассмотрены в этой главе, позволяют сократить объем работы по вводу команд. Если вы устали вводить сложную и длинную команду, найдите ссылку на нее в списке предыстории или создайте псевдоним. В любом случае вы сэкономите время и усилия, да и клавиатура прослужит дольше.

13

Контроль использования системных ресурсов

Хороший пользователь должен обладать некоторыми навыками системного администратора, т.е. ему следует проверять состояние машины, чтобы убедиться, что она работает корректно. Для того чтобы способствовать этой деятельности, в Linux предусмотрено несколько команд, предназначенных для отслеживания системных ресурсов. В данной главе мы рассмотрим несколько таких команд. Многие из них предназначены для выполнения различных действий, но для каждой из них есть задача, которую она решает лучше других (в этом авторы данных инструментов следуют лучшим традициям Unix — создавать небольшие программы, ориентированные на отдельные задачи). Диапазон задач, решаемых инструментами мониторинга, простирается от контроля программ, выполняющихся на компьютере (`ps` и `top`), до удаления нежелательных процессов (`kill`), вывода списка всех открытых файлов (`lsof`) и предоставления сведений об использовании оперативной памяти (`free`) и дискового пространства (`df` и `du`). Хорошо зная эти несложные инструменты, вы сможете предотвратить возникновение серьезных проблем.

Выяснение того, как долго работает компьютер

```
uptime
```

Если вы хотите выяснить, как долго работает ваш компьютер, воспользуйтесь командой `uptime`. Это очень просто.

```
$ uptime
22:07:10 up 22 days, 23:29, 1 user, load average:
.0.46, 0.63, 0.52
```

Зачем использовать команду `uptime`, чтобы выяснить, как долго работает компьютер? Ну, например, чтобы похвалиться. Двадцать два дня непрерывной работы — неплохой рекорд, но я знаю и более высокие числа. Многие люди забавляются тем, что сравнивают, кто проработал дольше.

Более серьезная причина — желание выяснить, успешно ли произошла перезагрузка. Например, иногда я запускаю оболочку SSH (подробнее о ней — в главе 16) на удаленном сервере, выполняю перезагрузку, жду несколько секунд, а затем запускаю ее снова. Есть ли быстрый способ выяснить, успешно ли прошла перезагрузка? Мне кажется есть — это команда `uptime`!

Вывод информации о процессах, выполняемых в системе

```
pa aux
```

Иногда бывает, что запущенная программа "зависает" и перестает реагировать на ваши действия. При этом необходимо найти способ закрыть ее. Иногда приходится выяснять, какие из программ, запущенных некоторым пользователем, имеются в системе. В ряде случаев вам надо знать обо всех процессах,

выполняемых на машине в текущий момент. В каждой из этих ситуаций (а также во множестве других) надо использовать команду ps, отображающую информацию о процессах на вашем компьютере.

К сожалению, на данный момент существуют различные версии ps, поддерживающие разные типы опций. Более того, действие некоторых опций зависит от наличия или отсутствия дефиса перед ней, так что u и -u приводят к различным результатам. До сих пор все рассмотренные опции предварялись дефисом, однако, говоря о команде ps, нам придется отступить от этого правила.

Для того чтобы отобразить информацию обо всех процессах, запущенных в системе любыми пользователями, надо задать после ps следующие опции: a (что означает "all users", или все пользователи), u ("user-oriented", или ориентированная на пользователя, т.е. команда должна отображать информацию о пользователе-владельце процесса) и x (processes without controlling ttys — процессы, не контролируемые ttys; еще одна категория процессов, необходимость отображения которых приходится задавать явно). Приготовьтесь получить длинный список процессов; в следующем примере, например, он насчитывает 132 строки:

```
$ ps aux
USER    PID %CPU %MEM     VSZ   RSS TTY STAT   START
↳TIME COMMAND
scott 24224  4.1  4.2 1150440 44036  ?     R    11:02
↳12:03 /home/scott/.cxoffice/bin/wine-preloader
scott  5594  0.0  0.1    3432  1820 pts/6  S+   12:14
↳0:00 ssh scott@humbug.machine.com
scott 14957  0.3  7.5 171144  78628  ?     Sl   13:01
↳0:35 /usr/lib/openoffice2/program/soffice.bin
↳-writer
scott 12369  0.0  0.0      0     0  ?      Z   15:43
↳0:00 [wine-preloader] <defunct>
scott 14680  0.0  0.1   3860  1044  pts/5   R+ 15:55
↳0:00 ps aux
```

ПРЕДУПРЕЖДЕНИЕ

Если выполнить команду `ps aux`, то вы заметите, что по умолчанию вывод обрезается на границе терминального окна. Разумеется, это приводит к тому, что часть информации, которая могла быть очень полезной, оказывается утерянной. Для того чтобы увидеть всю строку вывода, добавьте опцию `-w` (от слова "wrap" — свернуть): `ps aux -w`.

Команда `ps` предоставляет подробную информацию: отображает сведения о пользователе-владельце процесса, идентификатор процесса (`PID` — Process ID), загрузку центрального процессора (`%CPU`) и использование памяти (`%MEM`) в процентах, текущее состояние процесса (`STAT`), а также имя процесса.

В столбце `STAT` содержится несколько букв. Наиболее важные из них приведены ниже.

Буква в столбце STAT	Значение
R	Выполняющийся
S	Спящий
T	Остановленный
Z	Зомби

Появление буквы Z — плохая новость. Она означает, что процесс "завис" и его нельзя завершить. Если у вас возникла проблема с программой и `ps` отображает для нее состояние Z, вам, вероятно, придется перезагрузить машину, чтобы избавиться от этой программы.

Поскольку команда `ps aux` предоставляет большой объем данных, бывает трудно найти в списке информацию о конкретной программе. Передавая выходные данные `ps aux` программе `grep` для обработки, можно достаточно просто ограничить полученные результаты конкретной командой.

```
$ ps aux | grep [f]irefox
scott 25205  0.0  0.0   4336    8 ?     S
↳Feb0 8  0:00 /bin/sh /opt/firefox/firefox
scott 25213 1.1 10.9 189092 113272 ?    Rl
↳Feb08  29:42 /opt/firefox/firefox-bin
```

На данный момент мы имеем сведения об экземплярах Firefox, имеющихся на компьютере, а именно: пользователь, запустивший программу, какую нагрузку на компьютер создает эта программа и как долго она выполняется. Эта информация может оказаться очень полезной.

ПОДСКАЗКА

Почему мы выполнили поиск `[f]irefox`, а не `firefox`? Ответ на этот вопрос см. в главе 10.

Просмотр дерева процессов

```
ps axjf
```

В системе Linux все процессы так или иначе взаимосвязаны. Некоторые программы в процессе выполнения запускают другие. Общим предком для всех процессов в системе является `init`, идентификатор которого всегда равен 1. Команда `ps` выводит текстовое представление дерева процессов, и вы можете выяснить, каким процессом был порожден тот или иной процесс. Для отображения дерева процессов предназначены опции `a` и `x`, которые уже встречались в предыдущем разделе, а также опции `j` (BCD job control format — двоично-десятичный формат управления заданиями) и `f` (art forest — "искусственный лес").

ЗАМЕЧАНИЕ

Как правило, программа ps отображает следующие столбцы:

```
PPID PID PGID SID TTY TPGID STAT UID TIME COMMAND
```

Для того чтобы упростить восприятие дерева команд, в листингах некоторые столбцы не приводятся.

```
$ ps axuf
 PPID  PID COMMAND
    1  7558 /usr/sbin/gdm
 7558  7561 \_ /usr/sbin/gdm
 7561  7604 \_ /usr/X11R6/bin/X :0
 7561  8225 \_ /bin/sh /usr/bin/startkde
 8225  8279 \_ /usr/bin/ssh-agent
 8225  8341 \_ kwrapper ksmserver
    1  8316 kdeinit Running...
 8316 10842 \_ konqueror [kdeinit] --silent
 8316 29663 \_ quanta
 8316 30906 \_ /usr/bin/kwrite /home/scott/analysis
 8316 17893 \_ /usr/lib/opera/9.0-20060206.1/opera
17893 17960 | \_ /usr/lib/opera/pluginwrapper
17893 17961 | \_ /usr/lib/opera/plugincleaner
```

Заметьте, что набор опций axjf приводит к появлению нового столбца PPID, представляющего идентификатор родительского процесса — процесса, породившего данный. Зная значения PID и PPID, вы сможете завершить процесс, вышедший из под контроля. Этот вопрос будет рассматриваться далее в данной главе.

ЗАМЕЧАНИЕ

Другой способ просмотра дерева процессов — команда pstree. Она предусмотрена не во всех системах, поэтому, возможно, вам придется инсталлировать ее самостоятельно (как это сделать, вы узнаете в главе 14).

Отображение процессов, принадлежащих конкретному пользователю

```
ps U [имя пользователя]
```

До сих пор мы рассматривали программу ps в режиме отображения списка всех процессов в системе. Если вы хотите ограничить набор результатов процессами, принадлежащими одному пользователю, задайте опцию U и введите после нее имя или числовой идентификатор этого пользователя.

```
$ ps U scott
PID    TTY      STAT   TIME   COMMAND
14928  ?        S      0:00   /opt/ooo2/program/soffice-writer
14957  ?        Sl     0:42   /opt/ooo2/program/soffice.bin-writer
4688   pts/4    S+     0:00   ssh scott@humbug.machine.com
26751  ?        Z      0:00   [wine-preloader] <defunct>
27955  pts/5    R+     0:00    ps U scott
```

Как видите, ps не включает в состав выходных данных пользовательское имя, но оно фигурирует в команде.

ПОДСКАЗКА

Если вам неизвестен числовой идентификатор пользователя, обратитесь к файлу /etc/passwd. Достаточно найти пользовательское имя, и в третьем столбце вы увидите числовой идентификатор.

Завершение выполняющегося процесса

```
kill
killall
```

Иногда программа начинает работать некорректно и не реагирует на попытки завершить ее обычным способом. Если эта программа предоставляет графический пользовательский

интерфейс, то для ее завершения предусматривается специальная кнопка, а программа, выполняемая в режиме командной строки, должна завершаться после нажатия комбинации клавиш <Ctrl+C>). Если программа выходит из под контроля, соответствующие действия пользователя не дают никакого результата. В этом случае необходимо применить команду `kill`.

С помощью программы `kill` процессу можно передать различные сигналы: от "завершить работу, по возможности приведя в порядок ресурсы", до "завершить работу немедленно". Рассмотрим три наиболее важных сигнала. Вызывая `kill`, вы можете задать "интенсивность" завершения процесса, указав одно из числовых или символьных значений, приведенных в табл. 13.1.

Таблица 12.1. Сигналы, передаваемые процессу посредством команды `kill`

Номер сигнала	Символьное обозначение	Действие
-1	-HUP	Процесс должен быть завершен. Прекратить его выполнение. (Применительно к системным службам он означает перезагрузку конфигурационных файлов и перезапуск соответствующей программы.)
-15	-TERM	"Мягкое завершение" с удалением порожденных процессов и закрытием файлов
-9	-KILL	Прекратить все выполняющиеся действия и завершить работу

Обычно следует сначала попытаться установить значение -15 (если вы не зададите опции при вызове `kill`, данное значение будет принято по умолчанию). Этим вы даете программе шанс завершить зависимые процессы, закрыть файлы и выполнить другие действия по освобождению ресурсов. Если вы выждали некоторое время (какое именно — зависит от того, насколько вы терпеливы) и процесс по-прежнему выполняется и неуправляем или не отвечает, надо привести в действие "тяжелую артиллерию" и задать значение -9. Этим вы "выбиваете почву из под ног" работающей программы; она вынуждена

бросить решаемую задачу в текущем состоянии, оставить порожденные процессы выполняющимися, а используемые файлы открытыми. Такой режим завершения нежелателен, но в некоторых случаях нет другого выхода.

Значение -1, или -HUP, предназначено в основном для служб, подобных Samba. Вы, вероятнее всего, будете использовать его достаточно редко, но должны знать, какие действия при этом выполняются.

Предположим, например, что процесс grim "завис". (Обычно он выполняется без проблем, но это лишь пример.)

```
$ ps U scott
  PID  TTY      STAT   TIME  COMMAND
14928  ?        S      0:00  /opt/ooo2/program/soffice-writer
14957  ?        Sl     0:42  /opt/ooo2/program/soffice.bin-writer
 4688  pts/4    S+     0:00  ssh scott@humbug.machine.com
26751  ?        Z      0:00  [wine-preloader] <defunct>
27921  ?        Ss     0:00  /usr/bin/gvim
27955  pts/5    R+     0:00  ps U scott
$ kill 27921
$ ps U scott
PID    TTY      STAT   TIME  COMMAND
14928  ?        S      0:00  /opt/ooo2/program/soffice-writer
14957  ?        Sl     0:42  /opt/ooo2/program/soffice.bin-writer
 4688  pts/4    S+     0:00  ssh scott@humbug.machine.com
26751  ?        Z      0:00  [wine-preloader] <defunct>
27955  pts/5    R+     0:00  ps U scott
```

Для того чтобы завершить работу gvim, мы сначала должны выяснить его PID, вызвав команду ps. В данном случае значение идентификатора равно 27921. Далее следует передать PID программе kill, помня при этом, что по умолчанию она использует сигнал TERM, а затем снова проверить состояние процессов с помощью команды ps. Как видите, процесса gvim уже не существует.

ЗАМЕЧАНИЕ

Почему мы не удалим с помощью команды `kill` процесс с идентификатором 26751, состояние которого обозначено буквой Z? Дело в том, что для зомби-процесса даже значение -9 не действует. Процесс уже "мертв", поэтому "убить" его невозможно. Единственный способ избавиться от него — перезагрузить систему.

Для того чтобы выполнить команду `kill`, необходимо знать идентификатор процесса PID. А что если программа, выполнение которой вы хотите прекратить, имеет несколько или даже много таких идентификаторов? Применять команду `kill` к каждому идентификатору PID может оказаться утомительным и даже неприятным занятием. В таких ситуациях следует использовать команду `killall`, а не `kill`. Команда `killall` использует не идентификаторы процессов, а их имена, что в таких ситуациях намного легче.

Допустим, что `cronolog` (программа, позволяющая управлять созданием файлов системного журнала) стала работать неправильно и ее необходимо остановить. На одном из моих серверов с программой `cronolog` связано 84 процесса, и мне определенно не хочется останавливать каждый из них отдельно с помощью команды `kill`! Посмотрите, как можно остановить все эти процессы одним махом с помощью команды `killall` (это следует делать только в случае крайней необходимости, так что, пожалуйста, выполняйте команды `killall` и `kill` ответственно).

```
$ ps aux | grep /usr/bin/cronolog| wc -l
84
$ killall cronolog
$ ps aux | grep /usr/sbin/cronolog | wc -l
0
```

Мы использовали команду `ps`, описанную ранее в этой главе, а затем передали результаты ее работы утилите `grep`

(глава 10), чтобы остановить только действительно активные процессы программы cronolog. Затем передали результаты команде wc -l (см. главу 7), подсчитывающей количество строк вывода. Это число равно 84. Мы выполнили команду killall, указав имя процесса программы cronolog, а затем снова выполнили команду ps. Что теперь? Нуль. Ни одного процесса не осталось, чего мы и добивались.

Отображение динамически обновляемого списка выполняющихся процессов

top

Иногда пользователь обнаруживает, что быстродействие системы Linux без видимой причины внезапно уменьшилось. "Нечто" выполняется на компьютере настолько интенсивно, что занимает все процессорное время. Бывает, что, запустив программу, вы обнаруживаете, что она потребляет намного больше ресурсов центрального процессора, чем положено. Для того чтобы выяснить причины проблемы или хотя бы узнать, какие процессы выполняются в системе, можно вызвать программу ps, но после вывода данных она не обновляет их. Вы получаете лишь "мгновенный снимок" процессов в системе.

В отличие от ps, команда top представляет динамически обновляемые сведения о процессах и о том, какой объем системных ресурсов использует каждый из них. Работу команды top трудно проиллюстрировать в книге, так как листинги по своей природе статические. Можно лишь представить работу данной команды в один конкретный момент времени.

```
$ top
top - 18:05:03 up 4 days, 8:03, 1 user, load
↳average: 0.83, 0.82, 0.97
Tasks: 135 total, 3 running, 126 sleeping, 2
↳stopped, 4 zombie
Cpu(s): 22.9% us, 7.7% sy, 3.1% ni, 62.1% id,
```

```
↳3.5% wa, 0.1% hi, 0.7% si
Mem:  1036136k total, 987996k used, 48140k
↳free,   27884k buffers
Swap: 1012084k total, 479284k used, 532800k
↳free,   179580k cached
  PID   USER    PR   NI   VIRT  RES   SHR  S  %CPU  %MEM
↳TIME+     COMMAND
25213  scott   15   0    230m  150m  15m  S  11.1  14.9
↳33:39.34  firefox-bin
 7604  root    15   0    409m  285m  2896 S  10.8  28.2
↳749:55.75 Xorg
 8323  scott   15   0    37396 10m   7456 S   1.0   1.1
↳13:53.99  kded
 8523  scott   15   0    69416 13m   3324 S   0.7   1.3
↳0:25.30  nautilus
 8378  scott   15   0    37084 10m   7456 S   1.0   1.1
↳13:53.99 kicker
```

В первых пяти строках команда `top` отображает подробную информацию о системе, а затем описывает каждый выполняющийся процесс. Заметьте, что `top` автоматически сортирует выходные данные по уровню нагрузки на центральный процессор (`%CPU`), поэтому если интенсивность использования процессора программой изменится, то изменится также положение соответствующей строки в списке.

ПРЕДУПРЕЖДЕНИЕ

Для того чтобы использовать команду `top` с максимальной эффективностью, нажмите клавишу <?> и прочитайте справку. Она действительно содержит полезную информацию!

Если, работая с `top`, вы захотите удалить программу, вам достаточно ввести символ `k`. В начале списка, после строки, начинающейся со `Swap:`, вы увидите следующее сообщение:

```
PID to kill:
```

Введите идентификатор процесса, который собираетесь завершить (например, 8026), нажмите клавишу <Enter>, и программа запросит у вас номер сигнала.

```
Kill PID 8026 with signal [15]:
```

По умолчанию `top` использует сигнал с номером 15. Если это значение вам подходит, нажмите клавишу <Enter>, если нет — задайте номер сигнала и опять же завершите ввод нажатием клавиши <Enter>. Примерно через секунду информация о процессе исчезнет из списка, отображаемого программой `top`.

Для того чтобы завершить работу `top`, введите символ `q`.

Команда `top` чрезвычайно полезна, и пользователи часто используют ее для того, чтобы выяснить, что случилось в системе. Если у вас возникнут вопросы, связанные с активностью программ, то `top`, вероятнее всего, даст ответы на них.

ЗАМЕЧАНИЕ

Со временем команда `htop` стала популярнее, чем устаревшая команда `top`. Она обеспечивает прокрутку, раскраску, меню и прекрасную графическую информацию об использовании памяти. Более подробную информацию можно найти на странице `http://hisham.hm/htop/`. Команда `htop` на самом деле намного лучше, чем команда `top` (лично я предпочитаю именно ее). Однако обычно она не входит в стандартный дистрибутивный пакет системы Linux, поэтому ее приходится инсталлировать отдельно (подробно об этом — в главе 14).

Получение списка открытых файлов

```
lsof
```

В главе 1 говорилось о том, что любой объект на компьютере под управлением Linux — каталог, сетевое соединение,

устройство — представляет собой файл. Это означает, что, хотя с вашей точки зрения открыт лишь один файл, на самом деле в системе в каждый момент присутствуют тысячи открытых файлов. Это не ошибка. Действительно тысячи. Для того чтобы увидеть полный список открытых файлов, надо использовать команду `lsof` (сокращение от "list open files").

Если вы вызовете команду `lsof` без параметров (для этого вам надо выступать от имени пользователя `root`), то получите список, содержащий тысячи строк. В моей системе в данный момент открыто и используется 5497 файлов. Таким образом, `lsof` предоставляет дополнительную информацию о состоянии компьютера в определенный момент времени.

Передавая выходные данные `lsof` программе `less`, можно организовать поэкранный просмотр результатов.

```
# lsof | less
COMMAND   PID USER   FD   TYPE DEVICE    SIZE  NODE
↳NAME
init        1 root cwd    DIR    3,1     656     2 /
init        1 root rtd    DIR    3,1     656     2 /
init        1 root txt    REG    3,1   31608  2072
↳/sbin/init
init        1 root mem    REG    0,0       0
↳[heap] (stat: No such file or directory)
init        1 root mem    REG    3,1 1226096 27220
↳/lib/tls/i686/cmov/libc-2.3.5.so
init        1 root mem    REG    3,1   86596 27536
↳/lib/ld-2.3.5.so
init        1 root 10u   FIFO   0,12    3915
↳/dev/initctl
ksoftirqd   2 root cwd    DIR    3,1     656     2 /
```

Однако это не выход. Сколько страниц займут 5497 результатов? Для обработки выходных данных можно также использовать программу `grep`, но это не всегда нужно. Как вы узнаете из последующих разделов, сама программа `lsof` предоставляет возможность фильтрации данных так, что вы сможете сосредоточить внимание на том подмножестве открытых файлов, которое действительно интересует вас.

ЗАМЕЧАНИЕ

Команда `lsof` имеет огромное количество опций, поэтому перед одной и той же буквой может стоять как знак "минус", так и "плюс". Иначе говоря, существуют, например, опции `+c` и `-c`, `+d` и `-d`, которые имеют разный смысл. Чтобы не запутаться, сначала необходимо подготовиться.

Отображение файлов, открытых конкретным пользователем

```
lsof -u
```

Если вы хотите получить список файлов, которые открыл конкретный пользователь (заметьте, что набор этих файлов включает также сетевые соединения и устройства), задайте при вызове команды `lsof` опцию `-u`, а затем введите пользовательское имя. (Помните, что сделать это вы сможете, лишь обладая правами `root`.)

```
# lsof -u scott
COMMAND     PID USER   FD  TYPE      DEVICE    SIZE
↳NODE NAMEP
evolution  8603 scott 37u  unix  0xe73dc680
↳13518 socket
evolution  8603 scott 38r  FIFO         0,5
↳13520 pipe
evolution  8603 scott 39w  FIFO         0,5
↳13520 pipe
evolution  8603 scott 40u  REG          3,2    49152
↳181419 /home/scott/.evolution/addressbook/
↳local/system/addressbook.db
opera     11638  scott cwd  DIR          3,2     7096
↳6 /home/scott
opera     11638  scott rtd  DIR          3,1      656
↳2 /
opera     11638  scott txt  REG          3,1 13682540
```

```
↳109226 /usr/lib/opera/9.0-20060206.1/opera
opera    11638  scott mem  REG          0,0
↳0 [heap] (stat: No such file or directory)
opera    11638  scott mem  REG          3,1   34968
↳162978 /usr/lib/X11/locale/common/
↳xomGeneric.so.2.0.0
opera    11638  scott mem  REG          3,1  111724
↳109232 /usr/lib/opera/plugins/libnpp.so
opera    11638  scott mem  REG          3,1   14776
↳27361 /lib/tls/i686/cmov/libnss_dns-2.3.5.so
opera    11638  scott mem  REG          3,1  286620
↳103029 /usr/share/fonts/truetype/
↳msttcorefonts/Arial_Bold.ttf
[Листинг сокращен для экономии места]
```

Даже когда мы ограничились информацией, соответствующей только одному пользователю, в списке все еще осталось 3039 строк. Некоторые пункты этого списка вызывают интерес. Во-первых, оказывается, что Evolution — программа обработки почты и персональной информации — постоянно выполняется. Во-вторых, для отображения одной из страниц, отображаемых браузером Opera, требуется шрифт Arial Bold.

Если на компьютере, который вы администрируете, работают несколько пользователей, желательно просматривать информацию, касающуюся каждого из них, посредством команды `lsof -u`. Возможно, вы обнаружите, что они запускают программы, которые не должны использовать. Если вы — обычный пользователь системы Linux, применяйте эту команду для просмотра информации о своих файлах. Не исключено, что вы выявите программы, о наличии которых даже не догадывались.

Получение списка пользователей для конкретного файла

lsof [файл]

В предыдущем разделе вы узнали, как отображать сведения о файлах, открытых конкретным пользователем. Решим обратную задачу — выясним, кто использует конкретный файл. Для этого надо лишь ввести после `lsof` путь к этому файлу. Предположим, вы хотите узнать, кто использует демон SSH, предназначенный для удаленного взаимодействия с компьютером (опять же не забывайте, что команда `lsof` в этом режиме доступна только пользователю `root`).

```
# lsof /usr/sbin/sshd
COMMAND   PID   USER  FD   TYPE   DEVICE  SIZE   NODE
↳NAME
sshd     7814   root  txt  REG      3,1  278492  67453
↳/usr/sbin/sshd
sshd    10542   root  txt  REG      3,1  278492  67453
↳/usr/sbin/sshd
sshd    10548  scott  txt  REG      3,1  278492  67453
↳/usr/sbin/sshd
```

Вы получили сведения о двух пользователях: `root` и `scott`. Если вы встретите неизвестное вам имя, например `4ackordood`, это означает, что система, вероятнее всего, подверглась атаке.

ЗАМЕЧАНИЕ

С вашей точки зрения, `sshd` — это программа, но для системы Linux `/usr/sbin/sshd` — это всего лишь еще один файл.

Отображение сведений о процессах, соответствующих конкретной программе

```
lsof -c [программа]
```

В предыдущем примере вы узнали, кто использует /usr/sbin/sshd. Однако полученные результаты не дают полной картины происходящего. Каждая программа предполагает взаимодействие с несколькими (иногда очень многими) другими процессами, применение сокетов и устройств. Все они являются для системы Linux лишь файлами. Для получения более полных сведений о файлах, имеющих отношение к выполняемой программе, задайте при вызове команды lsof опцию -c, а после нее — имя выполняющейся (следовательно, открытой) программы. Например, команда lsof /usr/sbin/sshd сообщает лишь о двух пользователях и трех открытых файлах. Но что еще можно сказать о команде sshd?

```
# lsof -c sshd
COMMAND   PID  USER  FD   TYPE      DEVICE    SIZE
↳NODE NAME
sshd    10542  root  mem   REG         3,1   37480
↳27362 /lib/tls/i686/cmov/libnss_files-2.3.5.so
sshd    10542  root  mem   REG         3,1   34612
↳27396 /lib/tls/i686/cmov/libnss_nis-2.3.5.so
sshd    10542  root  mem   REG         3,1   86596
↳27536 /lib/ld-2.3.5.so
sshd    10542  root   0u   CHR         1,3
↳1237 /dev/null
sshd    10542  root   1u   CHR         1,3
↳1237 /dev/null
sshd    10542  root   2u   CHR         1,3
↳1237 /dev/null
sshd    10542  root   3u  IPv6       28186
↳TCP 192.168.0.170:ssh->192.168.0.100:4577
↳(ESTABLISHED)
sshd    10542  root   5u  unix  0xf2961680
```

```
↳28227 socket
sshd    10548 scott  cwd   DIR          3,1      656
↳2 /
sshd    10548 scott  rtd   DIR          3,1      656
↳2 /
sshd    10548 scott  txt   REG          3,1   278492
↳67453 /usr/sbin/sshdp
sshd    10548 scott   3u  IPv6       28186
↳TCP 192.168.0.170:ssh->192.168.0.100:4577
↳(ESTABLISHED)
[Листинг сокращен для экономии места]
```

В предыдущем листинге представлена лишь часть из 94 строк. Они представляют 94 открытых файла, так или иначе связанных с программой `sshd`. Среди них встречаются файлы `.so` (shared object — аналог DLL в Windows), а некоторые из файлов соответствуют сетевым соединениям между вашим компьютером и другой машиной (на самом деле удаленный компьютер посредством SSH обращается по сети к вашему (подробно этот вопрос рассмотрен в главе 15).

Применить команду `lsof` можно к самым разнообразным командам. Помимо получения полезной информации, вы узнаете, насколько сложными могут быть современные программы. Используйте `lsof` с программами, которыми пользуетесь ежедневно, и вы оцените труд программистов, благодаря которым эти средства стали доступны вам.

ЗАМЕЧАНИЕ

Команда `lsof` поддерживает большое число опций; мы рассмотрели лишь малую часть из них. В поставке исходного кода `lsof` содержится файл `00QUICKSTART` (имя начинается с двух нулей). Это руководство по наиболее важным применениям команды. Найдите с помощью Google этот файл и прочитайте его.

Отображение информации об оперативной памяти системы

free

Большинство современных компьютеров содержит сотни мегабайт и даже гигабайты оперативной памяти. Несмотря на это, нередко бывает, что работа компьютера замедляется вследствие слишком интенсивного использования памяти. Текущее состояние системной памяти позволяет выяснить команда `free`.

```
$ free
        total     used   free shared buffers cached
Mem:  1036136  995852  40284      0   80816 332264
-/+ buffers/cache:    582772 453364
Swap: 1012084  495584 516500
```

По умолчанию команда `free` представляет результаты в килобайтах, так, как будто вы задали опцию `-k`. Однако ее поведение можно изменить. Опция `-b` задает отображение сведений о памяти в байтах, а опция `-m` (именно она используется чаще всего) — в мегабайтах.

```
$ free -m
         total  used  free  shared  buffers
cached
Mem:      1011   963    48       0       78  316
-/+ buffers/cache:     569     442
Swap:      988   483   504
```

Из выходных данных команды `free` мы видим, что на машине есть 1011 Мбайт доступной оперативной памяти (реально объем памяти равен 1024 Мбайт, но `free` отображает значение 1011, потому что остальные 13 Мбайт заняты ядром системы и недоступны для других пользователей). Далее мы видим, что используется 963 Мбайт памяти, а свободными остаются лишь 48 Мбайт. Также в системе насчитывается почти гигабайт

виртуальной памяти, около половины которой, а именно 483 Мбайт, также заняты. Таково положение дел в данный момент.

Однако это еще не все. Очень важна информация в строке, начинающейся с `-/+ buffers/cache`. Для ускорения работы жестких дисков используются буфера и кеш. Если эта память понадобится, она может быть быстро освобождена для программ. С точки зрения приложений, выполняющихся на машине под управлением Linux, 442 Мбайт памяти свободны и могут использоваться, а 569 Мбайт можно освободить в случае необходимости. Это немалое дополнение к виртуальной памяти. Система Linux вряд ли была бы столь популярной, если бы не обеспечивала эффективного управления памятью.

ПОДСКАЗКА

Прекрасная веб-страница, содержащая более подробную информацию о команде `free` и управлении памятью в системе Linux, доступна по адресу `http://linux-mm.org`. Менее техническую информацию можно найти на странице `www.linuxatemyram.com`.

Отображение информации об использовании дискового пространства

`df`

Команда `free` предоставляет сведения об оперативной памяти, имеющейся в системе. Подобно ей, команда `df` (disk free) сообщает об объеме доступного дискового пространства. Вызовите команду `df`, и вы получите список дисков, данные о свободной области на них и смонтированных файловых системах.

```
$ df
Filesystem  1Kblocks   Used    Available  Use%   Mounted on
/dev/hda1    7678736  5170204    2508532   68%   /
```

```
tmpfs          518068         0     518068    0%
↳/dev/shm
tmpfs          518068     12588     505480    3%
↳/lib/modules/2.6.12-10-386/volatile
/dev/hda2    30369948 24792784    5577164   82%  /home
```

Перед тем как подробно рассматривать результаты выполнения команды, обсудим их структуру. По умолчанию команда `df` выводит данные в килобайтах, однако они становятся проще для восприятия, если вы используете опцию `-h` (или `--human-readable`).

```
$ df -h
Filesystem    Size   Used   Avail   Use%   Mounted on
/dev/hda1     7.4G   5.0G    2.4G    68%   /
tmpfs         506M      0    506M     0%   /dev/shm
tmpfs         506M    13M    494M     3%   /lib/modules/
↳2.6.12-10-386/volatile
/dev/hda2      29G    24G    5.4G    82%   /home
```

Термин "human-readable" означает, что килобайты представляются буквой K, мегабайты — M, а гибатайты — G. Буквы M и G присутствовали в последнем листинге.

Что же означают эти результаты? В данном случае на жестком диске есть два раздела: `/dev/hda1`, смонтированный в каталоге /, и `/dev/hda2`, смонтированный в разделе `/home`. Раздел `/home` имеет объем 29 Гбайт, 82% которого используется в данный момент. Свободным остается около 5,4 Гбайт. Пока еще нет оснований для беспокойства, но если в данный раздел будет записано содержимое еще нескольких компакт-дисков, то придется подумать о том, чтобы удалить ненужные файлы.

В корневом разделе, или /, объем свободного пространства несколько меньше: 2,4 из 7,4 Гбайт. Этот раздел вряд ли будет быстро заполняться, поскольку в нем содержатся программы и другие относительно статические файлы. В нем вы найдете каталог `/var`, в котором размещаются данные, используемые при инсталляции (процесс инсталляции будет описан в главе 14), а также файлы, размер и содержимое которых постоянно изменяются (например, файлы протоколов). Несмотря на это,

общий объем данных в каталоге изменится мало, если, конечно, вы не планируете устанавливать сложные приложения или разместить на данной машине веб-сервер или сервер баз данных.

Остальные два раздела помечены как tmpfs, это означает, что они содержат временные файловые системы, используемые виртуальной памятью, или областью подкачки. При выключении компьютера данные в этих "разделах" теряются.

ПОДСКАЗКА

Дополнительную информацию о tmpfs можно найти в статье Википедии по адресу http://en.wikipedia.org/wiki/TMPFS.

Определение размера области, занятой содержимым каталога

du

Команда df сообщает о состоянии всего жесткого диска. Но что делать, если вам надо узнать, какой объем дисковой памяти занимают каталог и файлы, содержащиеся в нем? Ответ на этот вопрос даст команда du (сокращение от "disk usage"). Сначала надо сделать текущим интересующий вас каталог, а затем выполнить команду du.

```
$ cd music
$ du
36582   ./Donald_Fagen
593985  ./Clash
145962  ./Hank_Mobley/1958_Peckin'_Time
128200  ./Hank_Mobley/1963_No_Room_For_Squares
108445  ./Hank_Mobley/1961_Workout
2662185 .
[Листинг сокращен для экономии места]
```

Как и в случае команды df, результаты, сгенерированные du, представляются в килобайтах, и она также поддерживает опцию -h (или --human-readable), позволяющую вывести данные в формате, более удобном для восприятия.

```
$ cd music
$ du -h
36M     ./Donald_Fagen
581M    ./Clash
143M    ./Hank_Mobley/1958_Peckin'_Time
126M    ./Hank_Mobley/1963_No_Room_For_Squares
106M    ./Hank_Mobley/1961_Workout
2.6G    .
```

Полученные результаты позволяют определить объем дисковой памяти, занимаемой каждым подкаталогом. Обратите внимание на каталог Hank_Mobley, который занимает 374 Мбайт. Этот объем получен путем суммирования объема всех подкаталогов, содержащихся в составе Hank_Mobley (поскольку реальный объем в килобайтах при представлении в мегабайтах округлятся, значения объемов несколько различаются). Если объем Hank_Mobley существенно больше, чем объем трех подкаталогов, это значит, что каталог Hank_Mobley содержит обычные файлы, расположенные за пределами подкаталогов.

В конце списка приводится общий размер всего каталога music/. Он составляет 2,6 Гбайт.

ПОДСКАЗКА

Если необходимо увидеть размеры как каталогов, так и файлов, используйте команду du -a (или --all). Если вы хотите, чтобы информация была упорядочена по размеру файлов, используйте команду du |sort -rn (или --reverse или --numeric-sort). Команда du выдает размеры файлов, которые затем упорядочиваются командой sort (см. главу 7). Опция -r задает порядок по убыванию, а опция -n — по возрастанию. В качестве дополнительной возможности можно направить вывод

в файл (> toomuchcrap.txt), который можно вставить в электронную таблицу, например LibreOffice.

И наконец, если вы хотите применить команду du ко всему жесткому диску, не забудьте передать результаты команде grep (см. главу 10) между вызовами команд du и sort, чтобы удалить нежелательные каталоги. Например: du | grep -v /dev -v /proc | sort -rn.

Ограничение вывода общим размером пространства, занятого каталогом

du -s

Если вам не нужна информация о подкаталогах, задайте du, чтобы вывести лишь общий объем. Для этого используется опция -s.

```
$ cd music
$ du -hs
2.6G    .
```

Результаты получаются достаточно компактными.

Выводы

Для каждой из команд, рассмотренных в данной главе, существует вариант, оснащенный графическим пользовательским интерфейсом. Однако если система не поддерживает графических средств, или если вы обращаетесь к машине посредством оболочки SSH (этот вопрос будет рассмотрен в главе 16) и не можете использовать графический интерфейс, или если хотите воспользоваться наиболее быстродействующим инструментом, то вам придется работать с командной строкой. Благодаря удачно выбранным именам команды, описанные в данном разделе, легко запомнить.

- `ps` (view running **p**rocesses — просмотр информации о выполняющихся процессах);
- `kill` (завершение работы процессов);
- `top` (верхняя часть списка выполняющихся процессов);
- `lsof` (list [**ls**] **o**pen **f**iles — список открытых файлов);
- `free` (свободная, или доступная, память);
- `df` (**d**isk **f**ree space — свободное место на диске);
- `du` (**d**isk **u**sage — информация об использовании дискового пространства).

Применяйте эти команды при работе на своем компьютере и не забывайте читать справочную информацию. В данной главе была рассмотрена лишь незначительная часть их возможностей. Команды `ps`, `top` и `lsof` поддерживают большое количество опций, что позволяет в деталях контролировать поведение компьютера, на котором вы работаете.

14

Инсталляция программного обеспечения

В состав дистрибутивных пакетов Linux входят тысячи программ. Сразу же после установки системы вы можете обращаться к веб-страницам из глобальной сети, подготавливать отчеты, создавать электронные таблицы, просматривать изображения, воспроизводить звуковые файлы и выполнять многие другие действия. Но, несмотря на обилие программ, очень часто возникает необходимость в установке новых пакетов. К счастью, Linux позволяет без труда инсталлировать программное обеспечение на компьютер.

Раньше считалось, что любую программу, которая должна быть установлена в системе Linux, необходимо скомпилировать. В принципе никто не запрещает делать этого (так поступают многие), но в настоящее время реальная необходимость в компиляции возникает достаточно редко. Вы можете ограничиться инсталляцией пакетов, используя для этой цели простые инструменты.

Приступая к реальной установке программ, необходимо учесть следующее. В системе Linux используются различные форматы программных пакетов, но два из них, RPM и DEB, встречаются чаще других (именно их мы и рассмотрим в этой главе). RPM применяется в системах типа Red Hat (да и само название формата расшифровывается как Red Hat Package Manager), Fedora Core, SUSE и некоторых других. DEB ориентирован на системы Debian, Ubuntu, а также на их многочисленных потомков — Linux Mint, Knoppix и многих других. Необходимо представлять себе работу обоих форматов, но ориентироваться надо, естественно, на системы, соответствующие вашей версии Linux.

ПОДСКАЗКА

Подробный анализ разнообразных систем управления пакетами и версий Linux, использующих их, приведен в документе *DistroWatch "Package Management Cheatsheet"*, доступном по адресу `http://distrowatch.com/dwres.php?resource=package-management`.

Инсталляция программных пакетов в RPM-системах

```
rpm -ihv [пакет]
rpm -Uhv [пакет]
```

Команда `rpm` инсталлирует пакеты, имена которых заканчиваются символами `.rpm` (что вполне логично). Для установки RPM-пакета его сначала надо скопировать на локальный компьютер. Рассмотрим в качестве примера сканер портов `nmap`. RPM-пакет `nmap` можно скопировать, обратившись по адресу `http://www.insecure.org/nmap/download.html`. После того как пакет окажется в вашей системе, вам достаточно будет вызвать `rpm`, задав три опции: `-i` (install), `-h` (для отображения хода процесса инсталляции) и `-v` (verbose, т.е. вывод подробной информации о выполняемых действиях). Данная команда должна быть вызвана от имени пользователя `root`.

```
# rpm -ihv nmap-4.01-1.i386.rpm
```

Однако эту команду нельзя рекомендовать для использования. Гораздо лучше установить набор опций `-Uhv`, где `U` — сокращение от `upgrade`. Почему же `-U` лучше, чем `-i`? Дело в том, что `-i` задает только инсталляцию, а `-U` — обновление и инсталляцию. Если пакет уже инсталлирован в системе и вы хотите установить его новый вариант, то опция `-U` лишь дополнит его до новой версии. Если же пакет в системе отсутствует,

то опция -U приведет к его инсталляции. Таким образом, лучше всего задавать опцию -U; независимо от того, собираетесь ли вы обновить версию или инсталлировать программу, данная опция выполнит нужные действия.

```
# rpm -Uhv nmap-4.01-1.i386.rpm
Preparing...                ########################################### [100%]
Updating / installing...
1:nmap-2:6.40-4.el7         ########################################### [100%]
```

Если вы хотите инсталлировать несколько RPM-пакетов, их имена надо разделить пробелами.

```
# rpm -Uhv nmap-6.40-4.el7.x86_64.rpm nmap-
↳frontend-6.40-4.el7.noarch.rpm
```

Если же число пакетов велико, вы даже можете использовать символы групповых операций. Предположим, например, что в вашем распоряжении есть двадцать файлов .rpm, расположенных в подкаталоге software. Вы можете установить все пакеты с помощью следующей команды:

```
# rpm -Uhv software/*.rpm
```

ПРЕДУПРЕЖДЕНИЕ

Опции -U следует отдавать предпочтение всегда, за исключением установки ядра. В этом случае надо задавать опцию -i. Если вы выполните обновление с помощью -U и новое ядро не будет работать, вы столкнетесь с серьезными проблемами. С другой стороны, если вы зададите опцию -i, то старое ядро останется на вашей машины в качестве резервной копии. Если новое ядро не будет работать, вы сможете воспользоваться старым.

Удаление программных пакетов из RPM-систем

```
rpm -e [пакет]
```

Удалить инсталлированный пакет RPM даже проще, чем установить его. Вместо `-Uhv` надо задать опцию `-e` (erase).

```
# rpm -e nmap
```

Вот и все. Опция `-e` не отображает данных. Обратите внимание на то, что, когда вы инсталлировали программу, используя опцию `-Uhv`, вам надо было указать имя файла, иначе `rpm` не смогла бы узнать, где находятся данные для инсталляции. При удалении программного обеспечения вы указываете имя пакета. Программа `rpm` распознает пакет по его имени, а не по имени файла, так как файлы, использованные для инсталляции, уже могли давно удалить.

ЗАМЕЧАНИЕ

Если вы хотите выяснить, какие пакеты инсталлированы в вашей системе и как пакет RPM идентифицирует их, используйте команду `rpm -qa`. Обратите внимание на то, что кроме имени пакета вы узнаете и номер его версии. Эта информация может оказаться очень полезной.

Инсталляция зависимых программных пакетов в RPM-системах

```
yum install [пакет]
```

Команда `rpm` предоставляет большие возможности, но, поработав с ней, вы рано или поздно столкнетесь с проблемой установки зависимых пакетов. Для того чтобы установить

пакет A, вам также надо скопировать и инсталлировать пакеты B и C, но чтобы инсталлировать C, необходимо скопировать и установить пакеты D и E, а чтобы установить пакет E ... и т.д.

В системе Debian и других подобных ей (о них пойдет речь далее в этой главе) данная проблема была решена давно путем использования системы управления пакетом APT. Его можно применять и в RPM-системах, но в них гораздо чаще используется сравнительно новая команда `yum`. Первоначально система управления пакетом YUM была разработан для Yellow Dog Linux (отсюда имя, которое расшифровывается как *Yellow Dog Updater, Modified*). В настоящее время она широко используется.

Команда `yum` инсталлирует, обновляет и удаляет программные пакеты, действуя как оболочка для пакета RPM. Кроме того, система YUM автоматически отслеживает зависимости. Например, если вы пытаетесь установить пакет A, о котором шла речь в начале данного раздела, система YUM скопирует и инсталлирует A, B и C. Если впоследствии вы решите, что пакет A вам не нужен, система YUM удалит не только его, но также B и C (при условии, что другие программные пакеты не используют их).

ЗАМЕЧАНИЕ

Система управления пакетом YUM в итоге будет вытеснена системой DNF ...возможно. Аббревиатура DNF расшифровывается как Dandified YUM (украшенный YUM). Эта система была задумана как усовершенствованный вариант YUM, особенно с точки зрения производительности. В момент подготовки данного издания (лето 2015 года) система DNF была внедрена только в одном крупном дистрибутивном пакете — Fedora. Последуют ли этому примеру остальные? Поживем — увидим. Если это произойдет, я опишу систему DNF в третьем издании книги, а пока вы можете почитать о ней на сайте `http://dnf.baseurl.org`.

Инсталлировать программы с помощью системы управления пакетом YUM очень просто. Предположим, вы хотите установить программу управления графическими файлами Shotwell. Для этого необходимо также инсталлировать несколько зависимых пакетов. Система YUM существенно упрощает этот процесс по сравнению с инсталляцией посредством RPM. Для того чтобы начать работу, вам не надо самостоятельно искать и копировать пакет Shotwell, система YUM сама скопирует не только его, но и все зависимые пакеты.

К сожалению, в процессе работы система YUM выводит слишком много информации. Приведенные ниже данные были существенно сокращены, и все равно объем листинга остается большим. Ознакомившись с этим листингом, вы можете составить представление о том, что вы увидите при работе с командой yum.

```
# yum install shotwell
Resolving Dependencies
--> Running transaction check
---> Package shotwell.x86_64 0:0.14.1-5.el7 will
↳be installed
--> Processing Dependency: libraw.so.5()(64bit)
.for package: shotwell-0.14.1-5.el7.x86_64
--> Processing Dependency: libgexiv2.so.1()(64bit)
↳for package: shotwell-0.14.1-5.el7.x86_64
--> Running transaction check
---> Package LibRaw.x86_64 0:0.14.8-5 will be
↳installed
---> Package libgexiv2.x86_64 0:0.5.0-9.el7 will
.be installed
--> Finished Dependency Resolution

Package Version Repository Size
Installing:
shotwell 0.14.1-5.el7 base 2.8 M
Installing for dependencies:
LibRaw 0.14.8-5 base 250 k
libgexiv2 0.5.0-9.el7 base 61 k
```

```
Transaction Summary
Install 1 Package (+2 Dependent packages)
Total download size: 3.1 M
Installed size: 13 M
Is this ok [y/d/N]:
```

После того как вы введете y, система YUM начнет копирование и инсталляцию пакетов, продолжая комментировать каждый шаг своей работы.

```
Downloading packages:
(1/3): LibRaw-0.14.8-5.el7.x86_64.rpm | 250 kB
(2/3): libgexiv2-0.5.0-9.el7.x86_64.rpm | 61 kB
(3/3): shotwell-0.14.1-5.el7.x86_64.rpm | 2.8 MB

Total                          1.9 MB/s | 3.1 MB
Running transaction check
Running transaction test
Transaction test succeeded
Running transaction
 Installing : LibRaw-0.14.8-5.el7.x86_64 1/3
 Installing : libgexiv2-0.5.0-9.el7.x86_64 2/3
 Installing : shotwell-0.14.1-5.el7.x86_64 3/3
 Verifying : libgexiv2-0.5.0-9.el7.x86_64 1/3
 Verifying : LibRaw-0.14.8-5.el7.x86_64 2/3
 Verifying : shotwell-0.14.1-5.el7.x86_64 3/3

Installed:
 shotwell.x86_64 0:0.14.1-5.el7

Dependency Installed:
 LibRaw.x86_64 0:0.14.8-5.el7
 libgexiv2.x86_64 0:0.5.0-9.el7

Complete!
```

Наконец инсталляция программы Shotwell завершилась, и она доступна для использования. Теперь рассмотрим, как в случае необходимости избавиться от нее.

Удаление зависимых программных пакетов из RPM-систем

```
yum remove [пакет]
```

Система YUM обладает важной положительной особенностью: ее система команд дружественна по отношению к пользователю. Хотите инсталлировать пакеты? Введите команду `yum install`. Хотите удалить его? Введите команду `yum remove`. Так, если вам надо избавиться от программы Shotwell, задайте следующую команду:

```
# yum remove shotwell
Resolving Dependencies
--> Running transaction check
---> Package shotwell.x86_64 0:0.14.1-5.el7
↳will be erased
--> Finished Dependency Resolution

Package   Version       Repository   Size
Removing:
shotwell  0.14.1-5.el7  @base        11 M

Transaction Summary
Remove 1 Package
Installed size: 11 M
Is this ok [y/N]:
```

Даже в таком простом деле, как удаление программного пакета, система YUM продолжает надоедать вам подробными комментариями. Нажмите клавишу <Y>, чтобы подтвердить готовность к удалению программы, и вы увидите дополнительные сообщения.

```
Downloading packages:
Running transaction check
Running transaction test
Transaction test succeeded
Running transaction
 Erasing    : shotwell-0.14.1-5.el7.x86_64  1/1
```

```
 Verifying : shotwell-0.14.1-5.el7.x86_64  1/1

Removed:
 shotwell.x86_64 0:0.14.1-5.el7

Complete!
```

Теперь программа Shotwell удалена. Заметьте, что зависимые пакеты, инсталлированные с помощью программы `yum`, остались в неприкосновенности. Они нужны были для работы Shotwell, но ими могут пользоваться другие программы на вашем компьютере, поэтому они остаются на месте (как вы увидите далее, по умолчанию такое же поведение демонстрирует система APT).

ЗАМЕЧАНИЕ

Если вы хотите выяснить, какие пакеты были инсталлированы в вашей системе и как система YUM идентифицирует их, используйте команду `yum list installed`. Обратите внимание на то, что кроме имени пакета вы узнаете и номер его версии. Эта информация может оказаться очень полезной.

Обновление программных пакетов в RPM-системах

`yum update`

Обычно в системе Linux установлены сотни, а то и тысячи программных пакетов. И постоянно приходится обновлять то один, то другой. Если бы вам пришлось вручную отслеживать появление новых версий программ и устанавливать необходимые дополнения, это заняло бы все ваше время. К счастью, есть средства, позволяющие упростить этот процесс. Простая команда `yum update` говорит программе `yum` о том, что

необходимо искать обновления к управляемым ею программам. Если новые пакеты доступны, программа yum оповещает вас о появившихся возможностях и запрашивает подтверждение на продолжение инсталляции.

```
# yum update
Resolving Dependencies
--> Running transaction check
---> Package kernel.x86_64 0:3.10.0-229.11.1.el7
↳will be installed
---> Package openssl.x86_64 1:1.0.1e-42.el7 will
.be updated
---> Package openssl.x86_64 1:1.0.1e-42.el7.9 will
↳be an update
---> Package unzip.x86_64 0:6.0-13.el7 will be
↳updated
---> Package unzip.x86_64 0:6.0-15.el7 will be an
↳update
--> Running transaction check
--> Finished Dependency Resolution

Package  Version                 Repository    Size

Installing:
kernel   3.10.0-229.11.1.el7     updates       31 M
Updating:
openssl  1:1.0.1e-42.el7.9       updates      711 k
unzip    6.0-15.el7              updates      166 k

Transaction Summary
Install 1 Package
Upgrade 2 Packages
Total size: 32 M
Is this ok [y/d/N]:
```

Если на данном этапе вы нажмете клавишу <Y>, то дадите этим согласие на копирование и инсталляцию двенадцати пакетов. После вывода пространных сообщений система YUM завершит свое выполнение, и ваш компьютер будет готов к дальнейшей работе. Хотите всегда быть на переднем крае? Выполняйте команду yum update ежедневно. Если вы не

испытываете необходимости всегда иметь в своем распоряжении последние версии, выполняйте команду `yum update` реже, но делайте это регулярно. Дополнения, предназначенные для повышения уровня защиты, появляются очень часто, поэтому желательно иметь их на вооружении.

Поиск пакетов, готовых к копированию на RPM-системы

```
yum search [строка]
yum list available
```

Теперь вы знаете, как инсталлировать и удалять программное обеспечение с помощью системы YUM, но как найти его? Предположим, вас интересует пакет для анализа трафика Wireshark. Вы хотите знать, есть ли пакеты, имеющие отношение к Wireshark, готовые к инсталляции посредством системы YUM. Можно выполнить команду `yum search wireshark`, но это далеко не идеальное решение. Данная команда будет искать соответствие условиям поиска в именах всех пакетов, в аннотациях и даже в списках имен программы для работы с пакетами. В результате вы получите список, включающий чуть ли не все программы, известные в мире.

Лучшим решением будет запросить список пакетов, доступных посредством системы YUM (размеры списка и в этом случае будут невероятно большими), а затем средствами конвейерной обработки передать результаты для обработки программе `grep`.

```
$ yum list available | grep wireshark
wireshark.i686          1.10.3-12.el7_0 base
wireshark.x86_64        1.10.3-12.el7_0 base
wireshark-devel.i686    1.10.3-12.el7_0 base
wireshark-devel.x86_64 1.10.3-12.el7_0 base
wireshark-gnome.x86_64 1.10.3-12.el7_0 base
```

В данном примере мы получили пять результатов — вполне приемлемое количество. Если вы действительно хотите выполнить полномасштабный поиск, применяйте команду `yum search`, в противном случае используйте `yum list available` и программу `grep`. В большинстве случаев второй способ оказывается наиболее приемлемым.

Инсталляция программных пакетов в системе Debian

```
dpkg -i [пакет]
```

Инсталляция нового программного обеспечения — одно из самых приятных занятий при работе в системе Linux. Как вы узнаете из последующих разделов, в системе Debian используется система APT — самый мощный и простой в применении инструмент установки программ. Эта система предоставляет богатые возможности, но большинство из них она реализует, выступая в роли оболочки программы `dpkg` (подобно тому, как YUM является оболочкой для RPM). Программа `dpkg` выполняет черновую работу по инсталляции и удалению программ на машине под управлением Debian. Перед тем как рассматривать систему APT, научимся работать с `dpkg`, поскольку APT не обладает универсальными возможностями по инсталляции любых программ.

Рассмотрим следующий пример. В настоящее время одной из самых популярных программ VoIP (Voice over IP) является Skype. Однако лицензионные соглашения не позволяют включать Skype в состав дистрибутивного пакета операционной системы. Если вы хотите воспользоваться Skype, сначала скопируйте эту программу с сервера компании, а затем установите ее вручную. Обратимся к странице с информацией о пакетах, доступных для копирования (`http://www.skype.com`), и найдем файл для нашей версии системы. В данном случае нам нужен пакет для Debian — `skype_1.2.0.18-1_i386.deb`.

После того как пакет будет скопирован в систему, надо инсталлировать его. Прежде всего с помощью команды cd сделайте текущим каталог, содержащий пакет, а затем инсталлируйте его посредством программы dpkg.

ЗАМЕЧАНИЕ

В большинстве систем, подобных Debian, команды, предполагающие обращение к dpkg, должны выполняться с правами root. С другой стороны, в популярной версии Ubuntu данное правило не действует. Вместо этого команды предваряются sudo. Другими словами, в Debian используется следующая команда:

```
# dpkg -i skype_debian-4.3.0.37-1_i386.deb
```

В Ubuntu и других sudo-системах выражение в командной строке должно иметь такой вид:

```
$ sudo dpkg -i skype_debian-4.3.0.371.2.0.18-1_.
  ↳i386.deb
```

При написании этой книги использовались машины под управлением Ubuntu, поэтому, если увидите команду sudo, знайте, зачем она нужна.

```
$ ls
skype_debian-4.3.0.3-1_i386.deb
$ sudo dkpg -i skype_debian-4.3.0.3-1_i386.deb
Selecting previously deselected package skype.
(Reading database ... 97963 files and directories
↳currently installed.)
Unpacking skype (from skype_debian-4.3.0.3-1_i386.
↳deb) ...
Setting up skype (4.3.0.3-1) ...
```

Вот и все. Команда dpkg отличается краткостью и сообщает вам только самые важные сведения и ничего сверх этого.

Удаление программных пакетов из системы Debian

```
dpkg -r [пакет]
```

Опция `-i`, использующаяся для инсталляции программного обеспечения на машине под управлением Debian, означает *install*. Аналогично опция `-r`, удаляющая программы, означает *remove*. Если вам больше не нужна программа Skype, вы можете легко удалить ее со своего компьютера.

```
# dpkg -r skype
(Reading database ... 98004 files and directories
currently installed.)
Removing skype ...
```

Когда вы инсталлировали программу, используя `dpkg -i`, вам надо было указать имя файла, иначе `dpkg` не смогла бы узнать, где находятся данные для инсталляции. При удалении программного обеспечения посредством команды `dpkg -r` надо ввести имя пакета. Программа `dpkg` распознает пакет по его имени, а не по имени файла, так как файлы, использованные для инсталляции, уже давно могли быть удалены.

ЗАМЕЧАНИЕ

Если вы хотите выяснить, какие пакеты были инсталлированы в вашей системе и как программа `dpkg` идентифицирует их, используйте команду `dpkg -l`. Обратите внимание на то, что кроме имени пакета вы узнаете номер его версии и архитектуру (обычно 32- или 64-битовую), а также краткое описание.

Инсталляция зависимых пакетов в системе Debian

```
apt-get inatall [пакет]
```

Команда dkpg предоставляет большие возможности, но, поработав с ней, вы рано или поздно столкнетесь с проблемой установки зависимых пакетов. Для того чтобы установить пакет A, вам также надо скопировать и инсталлировать пакеты B и C, но, чтобы инсталлировать C, необходимо скопировать и установить пакеты D и E, а чтобы установить пакет E... и т.д. Вам нужна система APT!

Команда apt, как и средства, рассмотренные ранее, позволяет инсталлировать, обновлять и удалять программные пакеты, кроме того, система APT автоматически обрабатывает зависимые пакеты. Например, если вы пытаетесь установить пакет A, о котором шла речь в начале данного раздела, система APT скопирует и инсталлирует пакеты A, B и C. Если впоследствии вы решите, что пакет A вам не нужен, система APT удалит не только его, но также пакеты B и C (при условии, что другие программные пакеты не используют их).

Команда apt изначально была разработана для использования в системе Debian в качестве интерфейса для программы dpkg. В настоящее время она имеется в каждом дистрибутивном пакете на базе Debian — в самой системе Debian, в Ubuntu, Linspire, Xandros и во многих других. Это один из тех инструментов, благодаря которым система Debian столь мощна и проста в использовании. Специалисты, использующие системы, отличные от Debian, по достоинству оценивают преимущества команды apt и стараются применить ее для работы в RPM-системах. В данной главе мы сосредоточим внимание лишь на использовании системы APT в сочетании с Debian.

ЗАМЕЧАНИЕ

Несмотря на то что в этой главе описывается работа системы APT в сочетании с системой Debian, она часто используется в комбинации с дистрибутивными RPM-пакетами. Более подробную информацию об этом можно найти на сайте `http://apt-rpm.org`. Даг Виирс (Dag Wieers) разработал прекрасный учебный материал "Using `apt` in an RPM world" (`http://dag.wieers.com/blog/using-apt-in-an-rpm-world`). И наконец, в 2003 г. я написал статью для журнала *Linux Magazine* под названием "A Very Apropos `apt`", которая, конечно, во многих отношениях устарела, но все еще содержит массу полезной информации. Ее можно найти на странице `www.linux.org.com/id/1476/`.

Предположим, вы хотите инсталлировать с помощью `apt` сложный инструмент `sshfs`. Чтобы сделать это, выполните следующую последовательность команд (помните, что для вызова этих команд надо обладать правами `root`):

```
# apt-get update
Get:1 http://security.ubuntu.com trusty-security
↳Release.gpg [933 B]
Ign http://extras.ubuntu.com trusty InRelease
Get:2 http://security.ubuntu.com trusty-security
↳Release [63.5 kB]
Get:3 http://us.archive.ubuntu.com trusty-updates
↳Release.gpg [933 B]Fetched 3,353 kB in 7s (445 kB/s)
Reading package lists... Done
[Листинг сокращен для экономии места]
# apt-get install bluefish
Reading package lists... Done
Building dependency tree
The following extra packages will be installed:
 bluefish-data bluefish-plugins
Suggested packages:
 bluefish-dbg libxml2-utils php5-cli tidy weblint
The following NEW packages will be installed:
```

```
 bluefish bluefish-data bluefish-plugins
0 upgraded, 3 newly installed, 0 to remove and 0
↳not upgraded.
Need to get 2,548 kB of archives.
After this operation, 9,484 kB of additional disk
↳space will be used.
Do you want to continue? [Y/n] y
Get:1 http://us.archive.ubuntu.com/ubuntu/ trusty/
↳universe bluefish-data all 2.2.5-1 [2,134 kB]
Get:2 http://us.archive.ubuntu.com/ubuntu/ trusty/
↳universe bluefish-plugins amd64 2.2.5-1 [172 kB]
Get:3 http://us.archive.ubuntu.com/ubuntu/ trusty/
↳universe bluefish amd64 2.2.5-1 [243 kB]
Fetched 2,548 kB in 3s (831 kB/s)
(Reading database ... 223602 files and directories
↳currently installed.)
Unpacking bluefish-data (2.2.5-1) ...
Unpacking bluefish-plugins (2.2.5-1) ...
Unpacking bluefish (2.2.5-1) ...
Setting up bluefish-data (2.2.5-1) ...
Setting up bluefish-plugins (2.2.5-1) ...
Setting up bluefish (2.2.5-1) ...
```

Рассмотрим подробно выполненные действия. В данном случае были вызваны две команды. По команде `apt-get update` произошла загрузка списка текущих программных пакетов с серверов (говоря об установке программ, принято использовать для их обозначения термин "хранилище"), указанных в конфигурационном файле `apt /etc/apt/sources.list`. (Если вы хотите узнать, что это за серверы, выполните команду `cat /etc/apt/sources.list`.) Если в начале строки, следующей за `apt-get update`, вы увидите слово `Get`, то это означает, что состояние хранилища изменилось и надо снова скопировать список. Слово `Ign` означает, что информация на компьютере соответствует состоянию хранилища и выполнять копирование не надо. Вызов `apt-get update` перед другими действиями позволяет убедиться, что список пакетов корректен и вовремя обновлен.

ЗАМЕЧАНИЕ

Для того чтобы добавить новые репозитории, не обязательно редактировать файл `sources.list`, поскольку ваши добавления в любой момент могут быть перезаписаны при обновлении программного обеспечения. Лучше создать отдельный файл конфигурации для каждого нового репозитория в каталоге `/etc/apt/sources.d`. Например, хранилище Dropbox можно загрузить как deb, но я предпочитаю выполнять добавления с помощью системы APT, чтобы автоматически получать обновления. Инструкции по добавлению хранилища Dropbox в качестве репозитория находятся на его сайте (наберите в поисковой системе Google слова "dropbox apt repo"). В качестве системного администратора или с помощью программы `sudo` создайте файл `dropbox.list` в каталоге `/etc/apt/sources.list.d`. Запишите в этот файл команду `deb http://linux.dropbox.com/ubuntu trusty main`. (Слово `Trusty` относится к конкретной версии системы Ubuntu; если вы не знаете, какую именно версию системы Ubuntu вы собираетесь использовать, введите `cat /etc/os-release`.) Сохраните и закройте файл, а затем, прежде чем делать что-нибудь еще, импортируйте ключи GPG хранилища Dropbox, выполнив команду `sudo apt-key adv --keyserverpgp.mit.edu --recv-keys 5044912E`. Сделав это, выполните команду `sudo apt-get update`. По завершении попытайтесь выполнить команду `apt-cache search dropbox`, и вы увидите результат — Dropbox. Когда вы захотите добавить другие репозитории, заслуживающие внимания, включите ссылку на них в файл `sources.list.d`.

Команда `apt-get install bluefish` загружает конкретный пакет, а также все необходимые зависимости (в данном случае `bluefish-data` и `bluefish-plugins`). После того как они появятся на вашем компьютере, система APT (на самом деле

программа dpkg, действующая от имени системы APT) инсталлирует все программное обеспечение.

Помните, что использовать нужно имя пакета, а не файла. Иначе говоря, используйте команду `apt-get installbluefish`, а не `apt-get install bluefish_2.2.5-1_amd64.deb`. Если система APT выявит дополнительные зависимости для требуемого пакета, как это было в случае с пакетом `bluefish`, вы должны будете подтвердить, что хотите их инсталлировать, прежде чем система APT приступит к их загрузке.

Если вы хотите одновременно загрузить несколько пакетов, перечислите их в командной строке. Например, если хотите инсталлировать пакеты `bluefish` и `pandoc`, сделайте следующее:

```
# apt-get install bluefish pandoc
```

При этом будут выявлены любые зависимости, связанные с пакетами `bluefish` или `pandoc`, и вы получите запрос о том, хотите ли вы их инсталлировать. Да, все просто.

ПОДСКАЗКА

Вы не знаете, что такое `pandoc`? Это недопустимо! Начните с официального веб-сайта `http://pandoc.org`. Свой опыт использования пакета `pandoc` я описал в блоге *Chainsaw on a Tire Swing* по адресу `www.chainsawonatireswing.com` (здесь можно найти много интересного).

Удаление зависимых пакетов из системы Debian

```
apt-get remove [пакет]
```

Если пакет больше не нужен, система APT позволяет без труда удалить его; для этого вместо команды `apt-get install` надо выполнить команду `apt-get remove`. Данная команда

выполняет действия, противоположные `apt-get install`, — удаляет указанные, а также зависимые пакеты. Как и ранее, при вызове команды надо задавать не имя файла, а имя пакета, например `apt-get remove sshfs`, а не `apt-get remove sshfs_2.5-1ubuntu1_amd64.deb`.

```
# apt-get remove sshfs
Reading package lists... Done
Building dependency tree... Done
The following packages will be REMOVED:
sshfs
0 upgraded, 0 newly installed, 1 to remove and 54
↳not upgraded.
Need to get 0B of archives.
After unpacking 98.3kB disk space will be freed.
Do you want to continue [Y/n]?
```

Следует заметить, что процедура удаления пакета не затрагивает некоторые файлы, необходимые для его работы. Так, например, в системе остаются конфигурационные файлы удаленных пакетов. Если вы уверены, что хотите удалить все элементы, задайте опцию `--purge`.

```
# apt-get --purge remove sshfs
Reading package lists... Done
Building dependency tree... Done
The following packages will be REMOVED:
  sshfs*
0 upgraded, 0 newly installed, 1 to remove and 54
↳not upgraded.
Need to get 0B of archives.
After unpacking 98.3kB disk space will be freed.
Do you want to continue [Y/n]?
```

Опцию `--purge` при удалении пакетов можно сравнить с символом звездочки при обычной операции с файлами. Удаляются все элементы пакета, включая конфигурационные файлы.

ЗАМЕЧАНИЕ

Если хотите выяснить, какие пакеты были инсталлированы в вашей системе и как система APT идентифицирует их, используйте команду `apt --installed list`. Обратите внимание на то, что кроме имени пакета вы узнаете номер его версии и архитектуру (обычно 32- или 64-битовую), а также краткое описание.

Обновление зависимых пакетов в системе Debian

`apt-get upgrade`

В системах Linux существуют тысячи пакетов, и можно поручиться, что в любой момент времени по крайней мере один из них нуждается в обновлении. Система APT позволяет достаточно просто следить за программными пакетами и обновлять их. Процедура обновления выглядит следующим образом (напоминаем: эта команда должна выполняться либо с полномочиями пользователя `root`, либо с помощью программы `sudo`):

```
$ sudo apt-get update
Get:1 http://security.ubuntu.com trusty-security
↳Release.gpg [933 B]
Get:2 http://security.ubuntu.com trusty-security
↳Release [63.5 kB]
Get:3 http://extras.ubuntu.com trusty Release.gpg
↳[72 B]
Get:4 http://us.archive.ubuntu.com trusty-updates
↳Release.gpg [933 B]
Fetched 4,275 kB in 6s (694 kB/s)
Reading package lists... Done
[Results greatly truncated for length]
$ sudo apt-get upgrade
Reading package lists... Done
Building dependency tree
Reading state information... Done
```

```
Calculating upgrade... Done
The following packages have been kept back:
  system-config-printer-gnome
The following packages will be upgraded:
  firefox firefox-locale-en fonts-droid
511 upgraded, 0 newly installed, 0 to remove and 8
↳not upgraded.
Need to get 349 MB of archives.
After this operation, 45.9 MB of additional disk
↳space will be used.
Do you want to continue? [Y/n]
```

Рассмотрим данный процесс более подробно. В данном случае мы, как и прежде, вызываем `apt-get update`, чтобы привести информацию на компьютере в соответствие с репозиториями системы APT. Команда `apt-get upgrade` выявляет различия между инсталлированным программным обеспечением и данными в хранилище. Если такие различия имеют место, система APT отображает список всех пакетов, которые надо скопировать и инсталлировать на компьютер. Реальное состояние списка пакетов зависит от того, как давно обновлялись программные средства. В данном примере с тех пор прошло достаточно много времени, и 511 пакетов требуют обновления.

Если вы нажмете клавишу <Y>, то система APT загрузит 511 пакетов в каталог `/var/cache/apt/archives` и установит необходимые дополнения. Если не хотите обновлять программы, нажмите клавишу <N>.

Описанные действия достаточно просты, но система APT позволяет еще эффективнее решить поставленную задачу. Вы можете объединить команды следующим образом:

```
# apt-get update && apt-get upgrade
```

Символы `&&` указывают на то, что команда `apt-get upgrade` должна выполняться только в том случае, если команда `apt-get update` завершилась без ошибок. Можно также создать псевдоним в файле `.bash_aliases` (о создании псевдонимов см. в главе 11).

```
alias upgrade='apt-get update && apt-get upgrade'
```

Перезагрузите файл .bash_aliases, и теперь, чтобы пакеты обновились, вам достаточно ввести upgrade, нажать клавишу <Enter>, а затем клавишу <Y>. Если сравнить данную процедуру с соответствующими средствами в системе Windows, оно будет явно не в пользу Windows Update.

ЗАМЕЧАНИЕ

Когда вы используете команду apt-get upgrade, вы получаете информацию о пакетах, которые еще не были обновлены или инсталлированы. Если хотите автоматически инсталлировать новое программное обеспечение и удалять ранее инсталлированные пакеты, которые больше не нужны, то выполните команду apt-get dist-upgrade. Не выполняйте команду dist-upgrade бездумно! Внимательно подумайте о последствиях для вашей системы.

Поиск пакетов, доступных для копирования в систему Debian

apt-cache search

Мы много говорили об инсталляции системы с помощью системы APT, но как узнать о доступных программных пакетах? Это также позволяет сделать система APT, предоставляя инструмент apt-cache search, который ищет списки доступных пакетов в хранилище. Для вызова команды apt-cache search не обязательно иметь права root.

```
$ apt-cache search dvdcss
brasero - CD/DVD burning application for GNOME
ubuntu-restricted-extras - Commonly used restricted
↳packages for Ubuntu
arista - multimedia transcoder for the GNOME Desktop
gxine - the xine video player, GTK+/Gnome user
```

```
↳interface
libdvdread4 - library for reading DVDs
python3-dvdvideo - Video DVD reader library
```

Поиск, выполняемый данной командой, имеет одну особенность. Команда ищет совпадение последовательности символов (в данном случае — `dvdcss`), а не конкретных слов. Кроме того, шаблон поиска может присутствовать в имени пакета или описании. И наконец, `apt-cache search` просматривает весь список пакетов, как инсталлированных, так и удаленных, поэтому не исключено, что отображаемый пакет уже установлен у вас.

ПРЕДУПРЕЖДЕНИЕ

Существует множество графических пользовательских интерфейсов для систем управления пакетами, которые делают практически все, что было описано в главе, — просмотр, управление, обновление, инсталляцию, деинсталляцию и многое другое, — но с помощью кнопок и мыши, а не клавиш. Ниже приведен список некоторых из таких инструментов, но прежде необходимо отметить несколько важных моментов.

Во-первых, графические системы управления пакетами, созданные для сред GNOME или KDE, конечно, будут работать и с другими настольными средами, если библиотеки были установлены правильно. Во-вторых, изучение этих систем стоит начинать с Википедии. Итак, перечислим их в алфавитном порядке.

- **Apper** (бывший KPackageKit). Приложение для среды KDE, обеспечивающее внешний интерфейс для службы PackageKit и потому совместимое практически со всеми пакетами Linux: YUM, APT, Entropy, Portage, urpmi, Pacman, zypp, Hif и др. Разработан в 2008 г. Адаптирован для систем Fedora, Red Hat и OpenSUSE.

- **GNOME PackageKit**. Внешний интерфейс сервера PackageKit (см. Apper), основанный на инструментарии GTK. Предназначен для опытных пользователей, в то время как пакет GNOME Software (см. ниже) — для рядовых пользователей.
- **GNOME Software**. Обеспечивает внешний интерфейс службы PackageKit (см. Apper), основанный на инструментарии GTK. Как и все программы для среды GNOME, использует совершенно непримечательные имена, которые невозможно найти в системе Google. Выпущен в 2013 г. Используется до сих пор.
- **Muon Discover**. Внешний интерфейс для системы APT, основанный на инструментарии Qt. Используется в системе Kubuntu. Предназначен для рядовых пользователей.
- **Octopi.** Внешний интерфейс для системы управления пакетами Pacman в операционной системе Arch Linux, основанный на инструментарии Qt. Выпущен в 2013 г. Используется до сих пор.
- **Smart.** Амбициозный проект, основанный на инструментарии GTK, совместимый с пакетами RPM, DPKG и Slackwares. Предназначен для тонкого управления процессами инсталляции и обновления программного обеспечения (этим и объясняется его название — smart (умный)).
- **Synaptic**. Внешний интерфейс системы APT (или RPM!), основанный на инструментарии GTK. Очень популярен и до сих пор широко используется, но больше не включается в некоторые дистрибутивные наборы как графическая система управления по умолчанию. Содержит очень много функций.
- **Ubuntu Software Center.** Внешний интерфейс для среды GNOME, основанный на инструментарии GTK. Совместим с системой APT и программой `dpkg`. Выпущен в 2009 г. и в настоящее время используется

в системе Ubuntu по умолчанию. Предназначен для рядовых пользователей (опытные пользователи должны использовать пакет Synaptic).

- **Yumex** (сокращение YUM Extender). Программа, основанная на инструментарии GTK и обеспечивающая внешний интерфейс для системы YUM, а в настоящее время — и DNF (система управления пакетами, на которую перешла операционная система Fedora).
- **Zero Install**. Программа, основанный на инструментарии GTK и загружающая программы в соответствующие каталоги, а затем использующая переменные окружения для того, чтобы эти программы могли найти свои библиотеки. Работает в системах Linux, Mac OS X, UNIX и Windows. Доступна только одна тысяча пакетов.

Удаление ненужных инсталляционных пакетов из системы Debian

```
apt-get clean
```

Пакеты, скопированные на локальный компьютер и инсталлированные на нем, система Debian сохраняет в каталоге `/var/cache/apt/archives/`. Со временем ненужные инсталляционные пакеты начинают занимать существенную часть дискового пространства. Удалить все не используемые файлы `.deb` позволяет команда `apt-get clean` (должна выполняться с правами `root`).

```
$ ls -1 /var/cache/apt/archives/
bluefish_2.2.5-1_amd64.deb
bluefish-data_2.2.5-1_all.deb
bluefish-plugins_2.2.5-1_amd64.deb
lock
partial/
```

```
$ sudo apt-get clean
$ ls -1 /var/cache/apt/archives/
lock
partial/
```

Опция `clean` команды `apt-get` является невероятно полезной, но существуют еще две связанные с ней опции, о которых следует помнить. Если вы выполняете команду `apt-get autoclean`, будут удалены только устаревшие пакеты. Аналогично команда `apt-get autoremove` ограничивается пакетами, инсталлированные как зависимости, но которые больше не нужны, например старыми ядрами. Все это помогает системе освобождаться от пакетов, которые больше не нужны.

Если по каким-то причинам процесс копирования файлов будет прерван, скопированную часть пакета вы найдете в каталоге `/var/cache/apt/archives/partial/`. Если вы знаете, что все обновления и дополнения установлены, можете удалить содержимое данного каталога.

Устранение проблем с помощью системы APT

В процессе работы никто не застрахован от возникновения проблем. Некоторые из них описаны ниже, там же приводятся рекомендации, как справиться с ними посредством системы APT.

Предположим, вы попытались выполнить команду `apt-get`, но получили следующее сообщение об ошибке:

```
E: Could not open lock file /var/lib/dpkg/lock -
↳open (13 Permission denied)
E: Unable to lock the administration directory
↳(/var/lib/dpkg/), are you root?
```

Причина проста: вы зарегистрированы не как пользователь `root`! Завершите сеанс работы, укажите при регистрации пользовательское имя `root`, и все будет работать.

ЗАМЕЧАНИЕ

Если вы используете Ubuntu или другую систему, в которой вместо работы с правами `root` указывается `sudo`, то же сообщение об ошибке означает, что вы не задали `sudo` перед командой. Другими словами, команда, вызванная вами, имеет следующий вид:

```
$ apt-get upgrade
```

Чтобы не было ошибки, она должна выглядеть так:

```
$ sudo apt-get upgrade
```

Бывает, что система APT сообщает о разрушенных зависимостях и предлагает выполнить команду `apt-get -f install`. Таким образом программа сообщает о том, что в системе есть разрушенные зависимости, не позволяющие завершить работу.

Существует несколько способов решения этой проблемы. Вы можете последовать совету системы APT и запустить команду `apt-get -f install`; она предпримет попытки разрешить ситуацию путем копирования и установки необходимых пакетов. Обычно этого достаточно, чтобы привести все в порядок.

Если не хотите поступать подобным образом, попробуйте вызвать команду `apt-get -f remove`. Она удалит пакеты, которые, по мнению `apt`, являются источником некорректной работы. Такой шаг потенциально опасен, поэтому надо быть очень внимательным. Программа сообщает о предлагаемых изменениях и ожидает вашего подтверждения. Перед тем как ответить "yes", внимательно прочитайте сообщение `apt`.

И наконец, вы можете получить предупреждающее сообщение о том, что некоторые пакеты `have been kept back`. Этим команда `apt` информирует вас о том, что обнаружен конфликт между запрашиваемым пакетом (или одним из зависимых) и пакетом, уже инсталлированным в вашей системе. Чтобы решить проблему, попробуйте инсталлировать пакет с опцией `-u`, которая сообщит о том, какие средства требуют обновления.

Выводы

Несмотря на то что системы на базе RPM и Debian имеют ряд различий, их средства управления пакетами все же во многом похожи. В обоих случаях разработчики позаботились об упрощении процедуры инсталляции и удаления программ. В RPM-системах для инсталляции, обновления и удаления программного обеспечения используется команда `rpm`, а в Debian для той же цели служит команда `dpkg`. Для решения проблем, связанных с зависимостями в системе RM, используется команда `yum` — оболочка команды `rpm`, а в системе Devian — система APT, которая служит оболочкой команды `dpkg`.

В любом случае и `apt`, и `yum` значительно лучше, чем инструменты подобного назначения, доступные пользователям Windows. Windows Update обновляет только программное обеспечение Microsoft и несколько драйверов независимых производителей, в то время как `apt` и `yum` поддерживают практически любые программы, способные выполняться в системе Linux. Пользователи Linux имеют в своем распоряжении средства, лучшие, чем те, которые доступны пользователям Microsoft, и это заслуга разработчиков данной системы.

15

Сетевое взаимодействие

Средства сетевого взаимодействия доступны в системе Linux с момента ее появления. Они были включены в ядро небольшой группой программистов, и без них система, конечно же, не была бы столь популярной. Поддержка сетевого взаимодействия осуществляется большую часть времени, и соответствующие средства можно настраивать для решения самых разнообразных задач.

Переход от старого к новому происходит на протяжении многих лет, и в данный момент мы находимся на середине пути. Некоторые из инструментов, рассматриваемых в этой главе, существуют в системе Linux с незапамятных времен и сегодня считаются устаревшими: `ifconfig`, `iwconfig`, `host`, `ifup`, `ifdown`, и `route` (утилиты `ping` и `traceroute` существовали всегда, но и они уже ожидают замены). Например, в ядрах системы 3.x Linux утилита `route` объявлена устаревшей, и в пакет инструментов `iproute2` включена ее замена — команда `ip route show`.

Это относится и к остальным устаревшим командам, перечисленным выше. Проблема заключается в том, что большинство дистрибутивных пакетов (возможно, даже подавляющее большинство) по-прежнему содержит утилиты `route`, `ifconfig`, `ifup` и др. Многие из этих дистрибутивов (возможно, также большинство) наряду с этими командами содержат пакет инструментов `iproute2`, а остальные позволяют его загрузку и инсталляцию. По этой причине по мере изложения будут описаны оба варианта — старый и новый. Таким образом, читатели получат двойной объем за те же деньги — везет же людям!

В любом случае мы коснемся вопросов тестирования, измерения и управления сетевыми устройствами и соединениями. Когда все работает корректно (а так происходит почти всегда),

вы можете использовать инструменты, описанные в данной главе, для контроля соединений. Если же вы столкнетесь с проблемой, они помогут вам решить ее, взяв на себя наиболее утомительную часть работы.

ПОДСКАЗКА

Материал этой главы основан на предположении о том, что вы используете систему адресации IPv4, следовательно, адрес представляется в формате `xxx.xxx.xxx.xxx`. В конце концов, IPv6 заменит IPv4, но это дело будущего. Тогда `route` и многие другие команды изменятся. На данный момент сведения, приведенные в настоящей главе, вполне могут быть использованы в работе. Дополнительная информация о IPv6 содержится в статье *IPv6* Википедии (`http://en.wikipedia.org/wiki/Ipv6`).

ЗАМЕЧАНИЕ

Пересматривая книгу для второго издания, я удалил текст, посвященный команде `iwconfig` для настройки конфигурации интерфейсов беспроводной сети. Этот раздел сохранен лишь для того, чтобы объяснить, почему я удалил из него старое содержание. Старый текст можно найти на моем веб-сайте (`www.granneman.com/linuxredactions`).

Определение состояния сетевых интерфейсов

```
ifconfig
ip addr show
```

Данная глава полностью посвящена сетевым соединениям. В конце главы мы поговорим о том, как устранять проблемы,

связанные с обменом по сети. Сейчас же выясним, как определить состояние сетевых средств.

Для того чтобы получить информацию обо всех сетевых устройствах, независимо от того, работают они или нет, используйте команду `ifconfig` (interface configuration — конфигурация интерфейса) и задайте при ее вызове опцию `-a` (all). На экране отобразится информация наподобие следующей (заметьте, что в некоторых системах, для того, чтобы вызвать команду `ifconfig`, надо зарегистрироваться с правами `root`):

```
$ ifconfig
ath0 Link encap:Ethernet HWaddr 00:14:6C:06:6B:FD
     inet addr:192.168.0.101 Bcast:192.168.0.255
↳Mask:255.255.255.0
UP BROADCAST RUNNING MULTICAST MTU:1500
↳Metric:1
lo  Link encap:Local Loopback
    inet addr:127.0.0.1 Mask:255.0.0.0
UP LOOPBACK RUNNING NTU:16436 Metric:1
[Листинг сокращен для экономии места]
```

В данном примере приводится информация о двух интерфейсах: `ath0` (карта беспроводной связи) и `lo` (интерфейс обратной петли; он будет рассмотрен несколько позже). Для каждого из них, помимо всего прочего, указываются тип соединения, MAC-адрес (Media Access Control — аппаратный адрес), IP-адрес, широковещательный адрес, маска подсети и информация о принятых и переданных пакетах. Если задать опцию `-a`, чтобы увидеть все устройства, то информация об отсоединенных устройствах (в данном случае `eth0`), например, IP-адрес и слово `UP` в четвертой полной строке будут пропущены.

Рассмотрим интерфейсы в порядке, обратном тому, в котором они представлены в листинге. Интерфейс `lo` называется интерфейсом обратной петли и позволяет компьютеру обращаться к самому себе. Интерфейс имеет IP-адрес 127.0.0.1 и необходим для нормальной работы системы. Если этот интерфейс присутствует, то в процессе работы вы не заметите этого, проблемы начнутся, если он по каким-то причинам исчезнет.

ПОДСКАЗКА

Дополнительную информацию об интерфейсе обратной петли и его адресе вы найдете в статье *Loopback* Википедии (`http://en.wikipedia.org/wiki/Loopback`).

Интерфейс `eth0` — это карта Ethernet, к которой можно подключить кабель. В текущий момент кабель не подключен, поэтому интерфейс не активен и для него не отображаются IP-адрес, широковещательный адрес и маска подсети. Интерфейсы проводной и беспроводной связи могут работать одновременно, однако в большинстве случаев в этом нет необходимости.

Кроме них, существует еще `ath0` — PCMCIA-карта беспроводной связи. Имя `eth0` представляет первичный сетевой интерфейс, а `eth1` — вторичный. Когда беспроводная карта имеется в компьютере, Ubuntu автоматически распознает ее и конфигурирует систему для работы с ней, присваивая ей идентификатор `ath0`. Поскольку беспроводной интерфейс по сути представляет собой Ethernet-систему с дополнительными возможностями, информация о `ath0` и `eth0`, полученная посредством `ifconfig`, имеет в основном одинаковый формат.

ЗАМЕЧАНИЕ

Для сетевых устройств могут использоваться другие имена, в частности, карта беспроводной связи может иметь имя `wlan0`.

Команда `ifconfig` — это быстрый способ проверки статуса сетевых интерфейсов, особенно если требуется быстро определить IP-адрес. Однако это — устаревший метод. В настоящее время предпочтительнее использовать команду `ip`.

Команда `ip` никогда не используется сама по себе. Рядом с ней всегда указывается объект — сетевое устройство или информация, которую вы хотите увидеть или изменить, а далее задаются команды для этого объекта. Например, для того

чтобы получить ту же информацию с помощью команды `ifconfig`, пришлось бы выполнить следующие действия:

```
$ ip addr show up
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 state
   ↳UNKNOWN
   link/loopback 00:00:00:00:00:00 brd
   ↳00:00:00:00:00:00
   inet 127.0.0.1/8 scope host lo
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500
   ↳state UP
   link/ether 00:14:6C:06:6B:FD brd
   ↳ff:ff:ff:ff:ff:ff
   inet 192.168.0.101/24 scope global dynamic eth0
[Листинг сокращен для экономии места]
```

Сначала задается команда `ip` и уточняется объект: `addr` (или `address`, или просто `a`). После этого мы указываем команду `show`, чтобы увидеть объект, а затем уточняем, какой именно объект мы хотим увидеть — `up` —, и выполняем команду. В результате мы получим ту же информацию, которую могли получить с помощью команды `ifconfig`.

Проверка способности компьютера принимать запросы

```
ping
ping -c
```

Команда `ping` передает на указанный адрес пакет специального типа — сообщение ICMP `ECHO_REQUEST`. Если компьютер по этому адресу принимает ICMP-сообщения, он отвечает пакетом ICMP `ECHO_REPLY`. (Заметьте, что брандмауэр может блокировать ICMP-сообщения, и в этом случае команда `ping` становится бесполезной, однако в большинстве случаев она работает без проблем.) Успешное выполнение команды `ping` означает, что между двумя компьютерами поддерживается сетевое соединение.

```
$ ping www.google.com
ping www.google.com
PING www.l.google.com (72.14.203.99) 56(84)
↳bytes of data.
64 bytes from 72.14.203.99: icmp_seq=1 ttl=245
↳time=17.1 ms
64 bytes from 72.14.203.99: icmp_seq=2 ttl=245
↳time=18.1 ms

--- www.l.google.com ping statistics ---
6 packets transmitted, 5 received, 16% packet loss,
↳time 5051ms
rtt min/avg/max/mdev=16.939/17.560/18.136/0.460 ms
```

Выполнение команды `ping` не прекратится до тех пор, пока вы не нажмете комбинацию клавиш <Ctrl+C>. Если вы забудете о том, что вызывали `ping`, эта команда будет выполняться до завершения работы компьютера или до разрыва соединения. (Однажды я забыл о ней, и она работала 18 дней, передав за это время на мой сервер 1,4 миллиона пакетов!)

Если вы хотите ограничить время работы `ping`, задайте число пакетов, которые должна передать данная программа. Для этого надо задать опцию `-c` и ввести число. После того как `ping` передаст указанное количество пакетов, она прекратит работу, отобразив перед этим сообщение о результатах своего выполнения.

```
$ ping -c 3 www.granneman.com
PING granneman.com (216.23.180.5) 56(84) bytes of
↳data.
64 bytes from 216.23.180.5: icmp_seq=1 ttl=44
↳time=65.4 ms
64 bytes from 216.23.180.5: icmp_seq=2 ttl=44
time=64.5 ms
↳64 bytes from 216.23.180.5: icmp_seq=3 ttl=44
time=65.7 ms

--- granneman.com ping statistics ---
3 packets transmitted, 3 received, 0% packet loss,
↳time 4006ms
rtt min/avg/max/mdev=64.515/65.248/65.700/0.510 ms
```

Команда `ping` — стандартное средство, позволяющее быстро определить наличие сетевого соединения. Указав опцию `-c`, вы застрахуете себя от неожиданностей и через несколько дней не обнаружите, что она все еще работает. Однако будьте осторожными, если вы "пингуете" некоторые правительственные или военные серверы, они могут решить, что вы их атакуете. В лучшем случае вас занесут в черный список и запретят доступ к этому серверу, а в худшем — к вам придут люди в черном.

Использование команды `ping` для диагностирования сетевых соединений будет рассмотрено далее в этой главе.

Контроль прохождения пакета между двумя узлами

traceroute

Команда `traceroute` отображает сведения о каждом шаге на пути пакета от вашей машины к указанному узлу. Предположим, вы хотите знать, почему вам не удается связаться с сайтом `www.granneman.com`. Еще вчера все было нормально, но сегодня каждая попытка загрузить веб-страницу оканчивается по тайм-ауту. В чем проблема?

```
$ traceroute www.granneman.com
traceroute to granneman.com (216.23.180.5), 30 hops
↳max, 38 byte packets
 1 192.168.0.1 (192.168.0.1) 1.245 ms 0.827 ms
↳0.839 ms
 2 10.29.64.1 (10.29.64.1) 8.582 ms 19.930 ms
↳7.083 ms
 3 24.217.2.165 (24.217.2.165) 10.152 ms 25.476 ms
↳36.617 ms
 4 12.124.129.97 (12.124.129.97) 9.203 ms 8.003 ms
↳11.307 ms
 5 12.122.82.241 (12.122.82.241) 52.901 ms
↳53.619 ms 51.215 ms
```

```
 6 tbr2-p013501.sl9mo.ip.att.net (12.122.11.121)
51.625 ms 52.166 ms 50.156 ms
 7 tbr2-cl21.la2ca.ip.att.net (12.122.10.14)
↳50.669 ms 54.049 ms 69.334 ms
 8 gar1-p3100.lsnca.ip.att.net (12.123.199.229)
↳50.167 ms 48.703 ms 49.636 ms
 9 * * *
10 border20.po2-bbnet2.lax.pnap.net
↳(216.52.255.101) 59.414 ms 62.148 ms 51.337 ms
↳11 intelenet-3.border20.lax.pnap.net
(216.52.253.234) 51.930 ms 53.054 ms 50.748 ms
↳12 v8.core2.irv.intelenet.net
(216.23.160.66) 50.611 ms 51.947 ms 60.694 ms
↳13 * * *
↳14 * * *
↳15 * * *
```

Что означают символы * * *? Каждое появление этой последовательности соответствует пятисекундному тайм-ауту для соответствующего узла. В некоторых случаях это может означать, что машина вследствие сбоя попросту не может понять, как обработать пакет, но постоянное повторение символов * * * указывает на то, что источником проблемы является один из маршрутизаторов, в данном случае тот, которому узел `v8.core2.irv.intelenet.net` должен передавать пакеты. В такой ситуации вам надо оповестить администратора `v8.core2.irv.intelenet.net` о происходящем (не помешает также послать администратору `gar1-p3100.lsnca.ip.att.net` сообщение о том, что при попытке доступа к `border20.po2-bbnet2.lax.pnap.net` возникали проблемы, но в первую очередь, конечно, надо разобраться с узлом, находящимся за `v8.core2.irv.intelenet.net` по пути следования пакета).

Некоторые проблемы можно разрешить, увеличив число переходов, отслеживаемых `traceroute`. По умолчанию их количество равно 30, но вы можете увеличить это значение с помощью опции `-m`, например, задав команду `traceroute -m 40 www.bbc.co.uk`.

ПОДСКАЗКА

Самым лучшим инструментом для отслеживания прохождения пакетов, наверное, является `mtr` (имя этой программы расшифровывается как Matt's traceroute). Его можно представить себе как сочетание `ping` и `traceroute`. Если `mtr` может выполняться в вашей версии Linux, скопируйте эту программу и попробуйте поработать с ней. Дополнительную информацию вы найдете по адресу `http://en.wikipedia.org/wiki/MTR_(software)`.

Выполнение DNS-преобразования

```
host
dig
```

Система доменных имен (Domain Name System — DNS) предназначена для того, чтобы упростить доступ пользователей к ресурсам глобальной сети. Компьютер лучше всего работает с числами (это неудивительно, ведь содержимое его памяти представляет собой именно числа), однако людям гораздо проще оперировать словами. Веб-сайт может находиться по адресу 74.125.227.243, но большинству пользователей запомнить его будет очень трудно. Гораздо проще удержать в памяти имя `www.google.com`. DNS, по сути, представляет собой огромную базу данных, которая содержит информацию о соответствии миллионов IP-адресов доменным именам, в частности, о том, что 74.125.227.243 — это тот же сайт, что и `www.google.com`.

ПОДСКАЗКА

DNS — сложная, но в то же время прекрасно организованная система. Общие сведения о ней приведены

в статье Domain Name System Википедии по адресу `http://en.wikipedia.org/wiki/Domain_Name_System`, а более подробные — в книге Пола Албица (Paul Albitz) и Крикета Лью (Cricket Liu) *DNS and BIND*.

Для того чтобы быстро найти IP-адрес, соответствующий доменному имени, надо использовать команду `host`.

```
$ host chainsawonatireswing.com
chainsawonatireswing.com has address 68.65.123.160
chainsawonatireswing.com mail is handled by 20
↳alt2.aspmx.l.google.com.
chainsawonatireswing.com mail is handled by 10
↳aspmx.l.google.com.
$ host www.chainsawonatireswing.com
www.chainsawonatireswing.com is an alias for
↳gawain.websanity.com.
gawain.websanity.com has address 23.239.25.194194
```

Обратите внимание на то, что очень похожие запросы приводят к совершенно разным ответам. Можно подумать, что сайты `chainsawonatireswing.com` и `www.chainsawonatireswing.com` — это одно и то же, однако это определенно не так.

Этот процесс можно обратить, выполнив так называемый *обратный DNS-поиск* (reverse DNS lookup), и найти доменное имя, связанное с указанным IP-адресом. Однако этот прием не срабатывает. Если с IP-адресом не связано никакой DNS-записи, вы не получите полезного ответа.

```
$ host 23.239.25.19
19.25.239.23.in-addr.arpa domain name pointer
↳li708-19.members.linode.com
$ host 68.65.123.160
Host 160.123.65.68.in-addr.arpa. not found:
↳3(NXDOMAIN)
```

Кроме того, легко заметить, что в ответе IP-адреса перечислены в обратном порядке. Это абсолютно нормально и ожидаемо.

Команда host используется уже долгое время и вполне успешно, но недавно была разработана новая команда, которая в итоге должна заменить команду host — команда dig (сокращение от *domain information groper* — средство сбора информации о доменах). Команда dig может делать все, что и команда host, а также многое другое.

С помощью команды dig можно выполнять разные виды поиска. Для этого достаточно указать после доменного имя желаемый вид поиска. Помимо прочих, в частности, предусмотрены опции a (адрес IPv4), aaaa (адрес IPv6), cname (каноническое имя, указывающее на запись A), mx (почтовый сервер), soa (запись Start of Authority, в которой хранится официальная информация о домене, такая, как имя основного сервера DNS, почтовый адрес администратора и расписание обновления доменных записей), any (вся информация, включая A, AAAA, CNAME, MX, SOA и многое другое).

```
$ dig www.chainsawonatireswing.com any
;; ANSWER SECTION:
www.chainsawonatireswing.com. 600 IN CNAME
↳gawain.websanity.com.
;; AUTHORITY SECTION:
chainsawonatireswing.com. 1800 IN NS dns1.registrar-
↳servers.com.
chainsawonatireswing.com. 1800 IN NS dns2.registrar-
↳servers.com.
```

На самом деле я оборвал листинг. В этом примере только пять строк вывода, а в исходном их было 21! Однако с помощью команды dig можно выделить самую важную информацию.

```
$ dig +noall +answer www.chainsawonatireswing.com any
www.chainsawonatireswing.com. 600 IN CNAME
.gawain.websanity.com.
```

Опция +noall сообщает команде dig, что она должна сбросить все другие флаги при выводе результатов (этих флагов может быть много), а опция +answer приказывает команде dig выводить на экран только ответ и ничего больше. Вместе они

позволяют получить намного более короткий и ясный ответ на запрос.

А что если мы пропустим символы www в нашей команде?

```
$ dig chainsawonatireswing.com any
;; ANSWER SECTION:
chainsawonatireswing.com. 3601 IN SOA dns1.
↳registrar-servers.com. hostmaster.registrar-
↳ servers.com. 2015072302 43200 3600 604800 3601
chainsawonatireswing.com. 1800 IN NS
↳dns1.registrar-servers.com.
chainsawonatireswing.com. 1800 IN NS
↳dns2.registrar-servers.com.
chainsawonatireswing.com. 583 IN A
↳68.65.123.160
chainsawonatireswing.com. 600 IN MX 20
↳alt1.aspmx.l.google.com.
chainsawonatireswing.com. 600 IN MX 10
↳aspmx.l.google.com.
chainsawonatireswing.com. 600 IN TXT
↳"v=spf1 include:_spf.google.com ~all"
```

Исходный список результатов содержал 31 строку, а мы сократили его до 8. Если вы хотите получить как можно больше информации о домене, то выполните команду `dig` с опцией `any`.

Как легко догадаться, команда `dig` может выполнять обратный поиск. Эту возможность реализует опция `-x`, но она будет описана ниже, вместе с остальными опциями.

```
$ dig +noall +answer -x 23.239.25.194
194.25.239.23.in-addr.arpa. 86384 IN PTR
↳gawain.websanity.com.
```

Как уже говорилось, в данном примере команда работает, но это возможно не всегда. Обе команды, `host` и `dig`, чрезвычайно полезны. В конце данной главы будут рассмотрены примеры того, как команда `host` может помочь в разрешении проблем, связанных с обменом по сети.

Настройка сетевого интерфейса

```
ifconfi9
ip addr add
ip link set
```

В первом разделе данной главы мы рассматривали использование команд `ifconfig` и `ip` для получения информации о состоянии сетевых интерфейсов. Однако команды `ifconfig` и `ip` предоставляют более широкие возможности, например, позволяют настраивать сетевые интерфейсы.

ЗАМЕЧАНИЕ

С помощью команд `igconfig` и `ip` вы можете выполнять с интерфейсом самые разнообразные действия, но здесь мы рассмотрим лишь несколько из них (более подробная информация представлена на справочных страницах `man ifconfig` и `man ip`).

Для того чтобы изменить IP-адрес Ethernet-карты, соответствующей интерфейсу `eth0`, на 192.168.0.125, выполните приведенную ниже команду (практически все действия по команде `ifconfig` требуют полномочий `root`).

```
# ifconfig eth0 192.168.0.125
```

При использовании команды `ip` надо выполнить следующее (имея полномочия `root`):

```
# ip addr add 192.168.0.125 dev eth0
```

Для того чтобы запустить некоторые из инструментов сбора информации, передаваемой по сети, например, печально известный WireShark (см. сайт `www.wireshark.org`), необходимо сначала отключить режим фильтрации данных в сетевой карте. По умолчанию карта, соответствующая интерфейсу `eth0`, принимает только те пакеты, которые были направлены именно

ей, но чтобы организовать сбор всей информации, передаваемой по сети, надо сообщить карте, чтобы она принимала и остальные пакеты.

```
# ifconfig eth0 promisc
```

После того как вы выполните эту команду, ваша карта будет принимать любой пакет, который попал в ее поле зрения. Обратите внимание на ключевое слово PROMISC в четвертой строке.

```
# ifconfig eth0
eth0 Link encap:Ethernet HWaddr 00:02:8A:36:48:8A
     inet addr:192.168.0.143 Bcast:192.168.0.255
↳Mask:255.255.255.0
     inet6 addr: fe80::202:8aff:fe36:488a/64
Scope:Link
     UP BROADCAST PROMISC MULTICAST MTU:1500
↳Metric:1
[Листинг сокращен для экономии места]
```

Если вы используете программу Wireshark, не забудьте отключить режим приема всех сетевых пакетов (promiscuous mode).

```
# ifconfig eth0 -promisc
# ifconfig eth0
eth0 Link encap:Ethernet HWaddr 00:02:8A:36:48:8A
     inet addr:192.168.0.143 Bcast:192.168.0.255
↳Mask:255.255.255.0
     inet6 addr: fe80::202:8aff:fe36:488a/64
↳Scope:Link
     UP BROADCAST MULTICAST MTU:1500 Metric:1
```

Если вы работаете с командой ip, то можете включить режим приема всех сетевых пакетов с помощью следующей команды (обратите внимание на строку с именем eth0):

```
# ip link set dev eth0 promisc on
# ip addr show up
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 state
   ↳UNKNOWN
...
```

```
2: eth0: <BROADCAST,MULTICAST,PROMISC,UP,LOWER_UP>
   ↳mtu 1500 state UP
```

Отключим этот режим снова.

```
# ip link set dev eth0 promisc off
# ip addr show up
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 state
   ↳UNKNOWN
...
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500
   ↳state UP
```

При желании можете даже изменить аппаратный MAC-адрес вашего сетевого устройства. Это бывает необходимо при попытках связать сетевую службу с конкретной машиной. Заменяя MAC-адреса, будьте внимательны: ошибка может привести к конфликтам с другими сетевыми устройствами (впрочем, это *крайне* маловероятно).

Если вы решите изменить аппаратный MAC-адрес, то сначала с помощью команды `ifconfig` или `ip addr show up` выясните MAC-адрес, заданный по умолчанию, чтобы можно было выполнить откат операции. Затем необходимо отключить устройство, внести изменения и включить его снова. Не волнуйтесь, если не сможете узнать в листинге команду, включающую или выключающую устройство, — она будет описана позднее. В заключение команда `ifconfig` или `ip addr show up` снова используется для того, чтобы убедиться в том, что MAC-адрес был действительно изменен. MAC-адрес в этой команде вымышленный, так что не пытайтесь использовать ее.

```
# ifdown eth0
# ifconfig eth0 hw ether 12:34:56:78:90:aa
# ifup eth0
```

С помощью команды `ip` эту операцию можно выполнить так:

```
# ip link set dev eth0 down
# ip link set dev eth0 address 12:34:56:78:90:aa
# ip link set dev eth0 up
```

Команды `ifconfig` и `ip` — это "краеугольные камни" сетевых интерфейсов. Внимательно разберитесь в работе этих команд и попытайтесь в полной мере использовать их возможности.

Получение информации о состоянии сетевого интерфейса беспроводной связи

```
iwconfiq
nmcli
```

Команда `ifconfig` отображает состояние всех сетевых интерфейсов, в том числе беспроводной связи. Однако все данные, относящиеся к интерфейсам беспроводной связи, получить не удастся, поскольку команда `ifconfig` попросту не имеет сведений о них. Для получения максимально возможных сведений о картах, предназначенных для установления беспроводных соединений, надо вместо команды `ifconfig` использовать `iwconfig`.

```
$ iwconfig
lo     no wireless extensions.
eth0   no wireless extensions.

ath0 IEEE 802.11g  ESSID:"einstein"
     Mode:Managed  Frequency:2.437 GHz   Access Point:
↳00:12:17:31:4F:C6
     Bit Rate:48 Mb/s  Tx-Power:18 dBm
↳Sensitivity=0/3
     Retry:off  RTS thr:off    Fragment thr:off
     Power Management:off
     Link Quality=41/94 Signal level=-54 dBm
↳Noise level=-95 dBm
     Rx invalid nwid:1047 Rx invalid crypt:0
↳Rx invalid frag:0
     Tx excessive retries:73 Invalid misc:73
↳Missed beacon:21
```

Теперь вы видите специальную информацию, предоставляемую `iwconfig`, — данные, относящиеся исключительно к беспроводной связи, например, тип карты (в данном случае 802.11g), ESSID, или сетевое имя (`einstein`), тип сети, к которой подсоединен компьютер, MAC-адрес точки доступа (00:12:17:31:4F:C6) и различные сведения о качестве соединения.

Команды `ifconfig` и `iwconfig` совместно сообщают пользователю всю необходимую информацию о сетевом интерфейсе беспроводной связи. Для настройки карт такого типа можно использовать либо команду `ifconfig`, либо, как вы увидите в следующем разделе, команду `iwconfig`.

Команда `ifconfig` была объявлена устаревшей по сравнению с командой `ip`. Команду `iwconfig` постигла та же участь, несмотря на то, что ее все еще можно найти почти в каждом дистрибутивном пакете. Предполагается, что на замену ей придет команда `iw`, но пока работа над ней еще *очень* далека от завершения. На справочной странице команды `iw` можно найти даже такое сообщение: "Не загружайте этот инструмент, мы считаем его работу неустойчивой" (так-то вот!). Я отложу рассмотрение команды `iw` до третьего издания книги, предполагая, что за эти годы она не сойдет со сцены. Если вас интересует информация об этой команде, зайдите на страницу `https://wireless.wiki.kernel.org/en/users/documentation/iw`.

Что же целесообразно использовать в настоящее время? В большинстве дистрибутивных пакетов можно найти программу NetworkManager с графическим пользовательским интерфейсом, которая упрощает подсоединение к проводным и беспроводным сетям и управление ими. В этом интерфейсе есть команда `nmcli` (аббревиатура от *NetworkManager command line interface*), которая отлично подходит для наших целей.

Для того чтобы выяснить статус беспроводного устройства, выполните команду `nmcli` с опцией `-p` (от слова *pretty*, т.е. ориентированная на удобство пользователя) и укажите объект `device` с командой `status`:

```
$ nmcli -p device status
=====================================
Status of devices
=====================================
DEVICE TYPE     STATE        CONNECTION
wlan0 wifi      connected    Home
eth0  ethernet  unavailable --
lo    loopback  unmanaged    --
```

Теперь мы знаем, что сеть Wi-fi подключена, но не знаем деталей. Можно было бы выполнить команду `nmcli -p connection show Home` (указав объект `connection` с командой `show`, за которыми следует имя соединения, в данном случае `Home`), но на самом деле это не лучшая идея. Нет, с детализацией все в порядке — просто вы получите в ответ 126 строк!

Вместо этого попробуйте сузить информацию (учтите, что я сократил листинг, который все еще содержит 36 строк).

```
$ nmcli -p -f GENERAL,IP4,WIFI-PROPERTIES dev show
↳wlan0
GENERAL.DEVICE          wlan0
GENERAL.TYPE            wifi
GENERAL.VENDOR          Intel Corporation
GENERAL.PRODUCT         PRO/Wireless 5100 AGN [Shiloh]
GENERAL.DRIVER          iwlwifi
GENERAL.HWADDR          12:34:56:78:90:ab
GENERAL.STATE           100 (connected)
GENERAL.CONNECTION      Home
IP4.ADDRESS             ip=192.168.1.100/24,
↳gw=192.168.1.1
IP4.DNS[1]              192.168.1.1
IP4.DNS[2]              208.67.220.220
WIFI-PROPERTIES.WEP     yes
WIFI-PROPERTIES.WPA2    yes
```

В данном случае опция `-f` означает, что команда `show` будет относиться только к выбранным полям — `GENERAL`, `IP4` и `WIFI-PROPERTIES` объекта `device`, в данном случае `wlan0`. Здесь содержится очень полезная информация: производитель и идентификатор продукта, драйвер, адрес аппаратного устройства

(MAC), состояние соединения, имя соединения, IP-адрес и связанные с ним адреса, а также типы поддерживаемого шифрования.

Настройка сетевого интерфейса беспроводной связи

В предыдущем издании книги я описал команду `iwconfig` и показал, как ее настроить для беспроводной сетевой карты в системе Linux. Как уже говорилось в начале главы, я решил удалить этот текст из второго издания. Старый текст читатели могут найти на моем веб-сайте.

Я исключил этот раздел по нескольким причинам, которые уже кратко перечислил. Когда книга впервые вышла в 2005 г., настройка беспроводного соединения в системе Linux все еще оставалась проблематичной. В этом разделе я попытался помочь читателям решить эту болезненную проблему. Однако в 2015 г. ситуация кардинально изменилась.

В настоящее время у пользователей настольных и переносных систем Linux нет необходимости настраивать беспроводное соединение с помощью командной строки (спасибо Ричарду Шталлману!). Если вы работаете на сервере, то скорее всего используете программу CLI, но и в этом случае вы также не используете Wi-Fi! Вам нужна сеть Ethernet, которая настраивается без помощи GUI.

Как уже было сказано в предыдущем разделе, в настоящее время практически все дистрибутивы настольных систем Linux поставляются вместе с программой NetworkManager, графическим пользовательским интерфейсом, который решает все проблемы, связанные с конфигурацией соединений Wi-Fi. Что еще важнее, он поддерживает WPA2 — текущий стандарт шифрования беспроводных соединений. Настройка соединения Wi-Fi вместе с WPA2 с помощью командной строки все еще остается сложной и неприятной задачей. Инструмент

nmcli не слишком полезен в этой области. В данном случае достаточно просто запустить программу NetworkManager (или ее дистрибутивный эквивалент) и работать с ней. Это намного удобнее.

Получение адресов средствами DHCP

dhclient

В большинстве внутренних сетей организаций для выделения вновь подключаемым компьютерам IP-адресов и предоставления им другой информации о сети используются средства DHCP (Dynamic Host Control Protocol). Без DHCP соответствующие данные должны были бы быть жестко закодированы. При наличии DHCP новую машину достаточно присоединить к сети; DHCP-серверу будет передан запрос на предоставление IP-адресов, а ответ сервера будет автоматически учтен при формировании конфигурации сети.

ЗАМЕЧАНИЕ

При изложении дальнейшего материала данной главы предполагается, что ваше сетевое устройство уже сконфигурировано для использования DHCP. В различных версиях Linux эта информация хранится в разных конфигурационных файлах. В системе Debian в файле `/etc/network/interfaces` содержится строка `iface [интерфейс] inet dhcp`. В системе Red Hat для этой цели в файл `/etc/sysconfig/network-scripts/ifcfg-[интерфейс]` включается выражение `BOOTPROTO=dhcp`. В наших примерах параметр `[интерфейс]` следует заменить именем вашего интерфейса. Дополнительную информацию по этому вопросу можно найти, выполнив средствами Google поиск по ключевым словам `dhcp` *версия_Linux*.

В некоторых случаях при загрузке машина не может обратиться к DHCP-серверу, и DHCP-запрос приходится инициировать вручную. Иногда возникают проблемы, для разрешения которых нужен новый IP-адрес. Независимо от того, для чего он потребовался, следует обратиться к одному из доступных DHCP-серверов посредством команды `dhclient` (она должна быть запущена от имени пользователя `root`).

```
# dhclient eth0
Internet Systems Consortium DHCP Client V3.0.2
Copyright 2004 Internet Systems Consortium.
All rights reserved.
For info, please visit http://www.isc.org/products/
↳DHCP

Listening on LPF/eth0/00:0b:cd:3b:20:e2
Sending on   LPF/eth0/00:0b:cd:3b:20:e2
Sending on   Socket/fallback
DHCPDISCOVER on eth0 to 255.255.255.255 port 67
↳interval 8
DHCPOFFER from 192.168.0.1
DHCPREQUEST on eth0 to 255.255.255.255 port 67
DHCPACK from 192.168.0.1
bound to 192.168.0.104 -- renewal in 37250 seconds.
# ifconfig eth0
eth0 Link encap:Ethernet HWaddr 00:0B:CD:3B:20:E2
     inet addr:192.168.0.104 Bcast:192.168.0.255
.Mask:255.255.255.0
     inet6 addr: fe80::20b:cdff:fe3b:20e2/64
↳Scope:Link
[Листинг сокращен для экономии места]
```

Для того чтобы освободить выделенный вам адрес, надо указать при вызове команды `dhclient` опцию `-r` (release).

```
# dhclient -r eth0
Internet Systems Consortium DHCP Client V3.0.2
Copyright 2004 Internet Systems Consortium.
All rights reserved.
For info, please visit http://www.isc.org/products/
↳DHCP
sit0: unknown hardware address type 776
```

```
sit0: unknown hardware address type 776
Listening on LPF/eth0/00:0b:cd:3b:20:e2
Sending on LPF/eth0/00:0b:cd:3b:20:e2
Sending on Socket/fallback
```

В идеале команда dhclient должна автоматически выполняться при загрузке компьютера, установке новой PCMCIA-карты беспроводной связи или присоединении к обычной Ethernet-карте сетевого кабеля. Однако в некоторых случаях этого не происходит. Если средства DHCP работают не так, как положено, приходится явным образом вызывать dhclient. В процессе выполнения данная программа отображает подробную информацию о своих действиях; она помогает понять происходящее и выявить причины проблемы.

ЗАМЕЧАНИЕ

В некоторых версиях Linux вместо dhclient для работы с DHCP используется более старая программа pump. Для того чтобы получить информацию о ней, выполните команду man pump либо обратитесь к документу *HOWTO for Red Hat and Mandrake*, доступному по адресу http://www.faqs.org/docs/Linux-mini/DHCP.html#REDHAT6.

Активизация сетевого соединения

```
ifup
ip link set
```

В процессе работы вы систематически используете команду ifup, даже не зная об этом. Если ваш компьютер успешно подключился к Интернету, то это произошло благодаря ifup. Если вы подключили Ethernet-кабель к разъему на карте и через несколько секунд имеете возможность получать электронную почту, это значит, что команда ifup выполнила большой

объем работы. Реально ifup вызывается при возникновении события, имеющего отношение к сетевому взаимодействию, например, перезагрузке компьютера, подключении кабеля и т.д. После запуска эта команда выполняет инструкции, описанные в конфигурационном файле (в предыдущем разделе упоминались имена и расположение этих файлов).

Иногда при возникновении проблем, связанных с обменом по сети, приходится вызывать команду ifup вручную. Сделать это просто: зарегистрируйтесь в системе как пользователь root, затем введите в командной строке ifup и имя сетевого интерфейса, который хотите активизировать.

```
# ifconfig
lo Link encap:Local Loopback
   inet addr:127.0.0.1  Mask:255.0.0.0
   inet6 addr: ::1/128 Scope:Host
   UP LOOPBACK RUNNING  MTU:16436  Metric:1
# ifup eth0
# ifconfig
eth0 Link encap:Ethernet HWaddr 00:0B:CD:3B:20:E2
     inet addr:192.168.0.14 Bcast:192.168.0.255
↳Mask:255.255.255.0
     inet6 addr: fe80::20b:cdff:fe3b:20e2/64
↳Scope:Link
     UP BROADCAST RUNNING MULTICAST MTU:1500
↳Metric:1
lo   Link encap:Local Loopback
     inet addr:127.0.0.1 Mask:255.0.0.0
     inet6 addr: ::1/128 Scope:Host
     UP LOOPBACK RUNNING MTU:16436 Metric:1
[Листинг сокращен для экономии места]
```

Команда ifup не сообщает об успешном завершении. Как и все Unix-программы, если все идет как надо, ifup "сохраняет молчание" и информирует пользователя лишь при возникновении ошибки. Для того чтобы получить сведения о действиях, выполненных ifup, надо использовать команду ifconfig, как это было сделано в предыдущем примере.

ЗАМЕЧАНИЕ

Для активизации сетевого соединения по кабелю или беспроводного соединения можно также использовать команду `ifconfig [интерфейс] up` или `iwconfig [интерфейс] up`. При работе с командной строкой программы NetworkManager можно использовать команды `nmcli connection up id nameof-connection` или `nmcli connection down id nameof-connection`.

В пакете `iproute2` включение интерфейса обеспечивает новая команда `ip link set eth0 up` (разумеется, для ее запуска необходимо иметь права пользователя `root`).

```
# ip addr show up
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 state
   ↳UNKNOWN
# ip link set eth0 up
# ip addr show up
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 state
   ↳UNKNOWN
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500
   ↳state UP
```

Вот и все!

Перевод сетевого интерфейса в неактивизированное состояние

```
ifdown
ip link set
```

Если команда `ifup` активизирует сетевой интерфейс, то `ifdown` деактивизирует его. Зачем это может понадобиться? Предположим, например, что вы хотите активизировать интерфейс, а команда `ifconfig` сообщает вам, что он уже активизирован. Тогда надо перевести его в неактивизированное состояние, а затем снова активизировать.

```
# ifup eth0
ifup: interface eth0 already configured
# ifdown eth0
# ifup eth0
```

Как и `ifup`, команда `ifdown` в случае успешного выполнения не выводит информацию. Если вы не увидите никакого сообщения, это значит, что команда `ifdown` выполнила свою задачу и интерфейс деактивизирован.

ЗАМЕЧАНИЕ

Для деактивизации сетевого интерфейса кабельной или беспроводной связи можно также использовать команду `ifconfig eth0 down` или `iwconfig ath0 down`.

В пакете `iproute2` отключение интерфейса обеспечивает новая команда `ip link set eth0 down` (разумеется, для ее запуска необходимо иметь права пользователя `root`).

```
# ip addr show up
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 state
   ↳UNKNOWN
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500
   ↳state UP
# ip link set eth0 down
# ip addr show up
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 state
   ↳UNKNOWN
```

Неплохо, не так ли? Лично я считаю, что команду `ifup` использовать (и запомнить) проще, но это дело вкуса.

Отображение таблицы маршрутизации

```
route
ip route
```

Пытаясь установить SSH-соединение с другим узлом (к этому вопросу мы еще вернемся в главе 16), необходимо как-то

сообщить компьютеру, должен ли он пересылать пакеты только по локальной сети или обращаться к маршрутизатору, чтобы они были переданы в глобальную сеть. Аналогично, если ваш браузер обращается по адресу `www.ubuntu.com`, система должна знать, надо ли передавать запрос маршрутизатору

Решить эту задачу позволяет таблица маршрутизации, которая содержится в ядре Linux. Для того чтобы просмотреть текущее содержимое этой таблицы, надо ввести в командной строке команду `route` (просмотреть данные, содержащиеся в таблице маршрутизации, может любой пользователь, но, чтобы изменить их, нужны права `root`).

```
$ route
Kernel IP routing table
Destination  Gateway    Genmask    Flags Metric
↳Ref        Use Iface
192.168.0.0  *          255.255.255.0 U   0
↳         0   0   eth0
default      192.168.0.1 0.0.0.0      UG  0
↳0        0   eth0
```

ЗАМЕЧАНИЕ

Эта команда приводит почти к тем же результатам, что и команда `route`: `netstat -nr` (или `--numeric` и `--route`). Команда `netstat` слишком подробная и может показаться несколько запутанной, поэтому дополнительную информацию следует искать в веб.

IP-адрес состоит из четырех октетов и записывается в формате `xxx.xxx.xxx.xxx`, например 192.168.0.124. Когда вы передаете пакет на другую машину, целевой IP-адрес сравнивается с данными в столбце `Destination` таблицы маршрутизации. Совместно со столбцом `Destination` используется столбец `Genmask`, который позволяет определить, какие из четырех октетов в целевом адресе следует учитывать.

Предположим, например, что вы ввели в командной строке команду `ping 192.168.0.124`. Маска в столбце `Genmask` имеет

вид 255.255.255.0, а это значит, что имеет значение лишь последний октет; он представлен числом 0. Другими словами, в адресе 192.168.0.124 для выполнения маршрутизации важен только адрес .124. Пакеты, имеющие адреса от 192.168.0.1 до 192.168.0.255 (пределы IP-адресации), соответствуют данным в столбцах `Genmask` и `Destination`, поэтому они должны пересылаться в пределах локальной сети. Именно поэтому в столбце `Gateway` указано значение *после адреса 192.168.0.0. При передаче по локальной сети шлюз не нужен.

Все остальные пакеты по умолчанию должны передаваться маршрутизатору, который имеет адрес 192.168.0.1; этот адрес указан в столбце `Gateway`. В этом случае маска, указанная в столбце `Genmask`, имеет значение 0.0.0.0. Другими словами, каждый пакет, для которого целевой адрес лежит вне диапазона 192.168.0.1–192.168.0.255, должен быть передан через компьютер с адресом 192.168.0.1 (он выполняет роль шлюза). Например, адреса 72.14.203.99, 82.211.81.166 и 216.23.180.5 соответствуют маске 0.0.0.0, поэтому они должны передаваться в глобальную сеть через компьютер, адрес которого указан в столбце `Gateway`.

Среди данных, отображаемых по команде `route`, есть столбец `Flags`. Он предоставляет дополнительную информацию о маршруте. Возможно несколько значений флагов, но чаще всего встречаются `U` (маршрут активизирован) и `G` (использовать шлюз). В приведенном выше примере вы видите, что оба маршрута активизированы, но лишь второй из них предполагает передачу данных через шлюз.

Рассмотрим новый способ: просто выполним команду `route` на той же самой машине.

```
$ ip route show
default via 192.168.0.1 dev eth0
192.168.0.0/24 dev eth0 proto kernel scope link
↳src 192.168.0.10
```

Первая строка довольно простая: вся исходящая из сети `eth0` информация должна по умолчанию идти на маршрутизатор с адресом 192.168.0.1.

Вторая строка более сложная. Запись 192.168.0.0/24 означает то же самое, что и 192.168.0.0 в команде `route` с маской 255.255.255.0, — это просто другой способ выражения того же самого факта (по-моему, более трудный). Для того чтобы убедиться в этом, поищите в веб информацию по ключевым словам "ip subnet calculator" и введите эти числа.

Далее, `dev eth0` — это устройство, а выражение `proto kernel` описывает процесс инсталляции маршрутизатора. В данном случае слово `kernel` означает, что ядро инсталлировало данный маршрутизатор в процессе автоконфигурации. Затем выражение `scope link` сообщает, что область, которой принадлежит адрес, является корректной, и в данном случае слово `link` означает, что адрес является корректным только для данного устройства. В заключение выражение `src 192.168.0.10` определяет адрес источника, который будет задавать ваш компьютер при отсылке пакетов по адресу назначения. Лично я считаю, что команду `route` намного проще понять. Выбор за вами, но я бы начал изучение с команды `ip route show`.

Внесение изменений в таблицу маршрутизации

```
route
ip route
```

Команда `route` может использоваться не только для просмотра данных, содержащихся в таблице маршрутизации, но и для их изменения. Модифицируя таблицу, надо соблюдать осторожность, так как этим вы можете нарушить работу сети.

Предположим, например, что ваша машина не обращается к шлюзу, в результате чего пакеты не покидают локальную сеть (однажды это произошло на моей машине). Чтобы устранить неисправность, вам надо выполнить команду `route`, выяснить, что в столбце `Gateway` нет адреса маршрутизатора, а затем

изменить данные с помощью той же команды route (несмотря на то, что команда route в режиме просмотра информации доступна любому пользователю, модифицировать таблицу маршрутизации может только пользователь root).

```
# route
Kernel IP routing table
Destination Gateway Genmask          Flags Metric Ref
↳Use Iface
192.168.0.0 *       255.255.255.0     U     0      0
↳0 eth0
# route add -net default gw 192.168.0.1 dev eth0
# route
Kernel IP routing table
Destination Gateway Genmask          Flags Metric
↳Ref Use Iface
192.168.0.0 *       255.255.255.0    U     0
↳0     0 eth0
default 192.168.0.1 0.0.0.0 UG 0
↳0     0 eth0
```

Рассмотрим команду, приведенную в листинге. В ее составе имеется ключевое слово add, которое задает новый маршрут (удалить маршрут позволяет ключевое слово del). Опция -net сообщает ядру об адресе целевой сети; в данном случае имеется в виду целевой адрес, заданный по умолчанию, который определяется посредством ключевого слова default. Ключевое слово gw указывает на то, что пакеты, соответствующие данному пункту назначения, должны передаваться через шлюз с адресом 129.168.0.1 (поскольку здесь речь идет о маршруте по умолчанию, то значение маски равно 0.0.0.0). И наконец, выражение dev eth0 определяет используемое устройство, в данном случае это Ethernet-карта с интерфейсом eth0.

С помощью новых инструментов это можно было бы сделать так:

```
# ip route show
192.168.0.0/24 dev eth0 proto kernel scope link
↳192.168.0.101
# ip route add default via 192.168.0.1
# ip route show
```

```
default via 192.168.0.1 dev eth0 proto static
↳metric 1024
192.168.0.0/24 dev eth0 proto kernel scope link
↳192.168.0.101
```

Все просто. Достаточно лишь указать команду и IP-адрес.

Предположим, что помимо Ethernet-карты в вашей машине появилась также карта беспроводной связи, соответствующая интерфейсу `ath0`. Вы хотите организовать через нее доступ к локальной сети `10.1.xxx.xxx`, но не собираетесь передавать через нее информацию в глобальную сеть. Для того чтобы добавить соответствующий маршрут, надо выполнить команду, приведенную в следующем примере:

```
# route
Kernel IP routing table
Destination  Gateway      Genmask       Flags Metric
↳Ref   Use   Iface
192.168.0.0  *            255.255.255.0 U     0
↳0       0    eth0
default      192.168.0.1  0.0.0.0       UG    0
↳0       0    eth0
# route add -net 10.1.0.0 netmask 255.255.0.0 dev ath0
# route
Kernel IP routing table
Destination  Gateway     Genmask       Flags Metric
↳Ref   Use   Iface
192.168.0.0  *           255.255.255.0 U     0
↳0       0    eth0
10.1.0.0     *           255.255.0.0   U     0
↳0       0    ath0
default      192.168.0.1 0.0.0.0       UG    0
0        0   eth0
```

Здесь мы указали, что карта соответствует устройству `ath0`, и задали маску 255.255.0.0. Теперь маршрутизация будет выполняться корректно. Если впоследствии вы захотите удалить этот маршрут, сделать это можно будет следующим образом:

```
# route
Kernel IP routing table
Destination  Gateway     Genmask       Flags Metric
↳Ref   Use  Iface
```

```
192.168.0.0 *           255.255.255.0 U     0
↳0     0    eth0
10.1.0.0    *           255.255.0.0   U     0
↳0     0    ath0
default     192.168.0.1 0.0.0.0       UG    0
↳0     0    eth0
# route del -net 10.1.0.0 netmask 255.255.0.0 dev eth0
# route
Kernel IP routing table
Destination Gateway     Genmask       Flags
  Metric Ref Use Iface
192.168.0.0 *           255.255.255.0 U
  0      0   0   eth0
default     192.168.0.1 0.0.0.0       UG
  0      0   0   eth0
```

Как видите, для этого используется та же команда, только вместо add вводится ключевое слово del.

А что насчет ip route? Для того чтобы добавить маршрут для карты Wi-Fi без выхода в Интернет, необходимо сделать следующее:

```
# ip route show
default via 192.168.0.1 dev eth0 proto static
↳metric 1024
192.168.0.0/24 dev eth0 proto kernel scope link
↳192.168.0.101
# ip route add 10.1.0.0/16 dev ath0
# ip route show
default via 192.168.0.1 dev eth0 proto static
.metric 1024
192.168.0.0/24 dev eth0 proto kernel scope link
↳192.168.0.101
10.1.0.0/16 dev ath0 scope link
```

Просто добавьте маршрут с IP-адресом в свою сеть вместе с его сетевой маской (255.255.0.0), а также имя устройства.

Удаление также выполняется очень просто.

```
# ip route show
default via 192.168.0.1 de v eth0 proto static
↳metric 1024
192.168.0.0/24 dev eth0 proto kernel scope link
```

```
.192.168.0.101
10.1.0.0/16 dev wlan0 scope link
# ip route del 10.1.0.0/16 dev ath0
# ip route show
default via 192.168.0.1 dev eth0 proto static
↳metric 1024
192.168.0.0/24 dev eth0 proto kernel scope link
↳192.168.0.101
```

Здесь мы всего лишь заменили команду `del` командой `add`.

Устранение проблем, связанных с сетевым взаимодействием

В настоящее время сетевые средства Linux разработаны столь тщательно, что пользователь вспоминает об их существовании только тогда, когда обмен по сети по каким-то причинам становится невозможным. Рассмотрим некоторые проблемы, которые могут возникнуть в процессе работы, и способы их устранения.

Если ваш сетевой интерфейс активизирован, но вы не можете взаимодействовать с другими узлами сети, попробуйте в первую очередь обратиться с помощью программы `ping` к собственному компьютеру, адрес которого 127.0.0.1. Если это не удается, значит, ваша система серьезно повреждена. Получив ответ с адреса 127.0.0.1, выполните следующий шаг: проверьте снова свой компьютер, но на этот раз установите в качестве параметра команды `ping` внешний адрес вашей машины. Если вы не получите ответа, убедитесь в том, что в вашей системе установлены сетевые средства. Если же отклик от компьютера получен, проверьте остальные машины в локальной сети. Отрицательный результат говорит о том, что что-то не в порядке с интерфейсом (предполагается, что маршрутизатор работает нормально). Убедитесь в том, что кабели подключены к соответствующим разъемам. Используйте `ifconfig` (или `iwconfig` в случае беспроводной связи) для проверки состояния интерфейсов, при необходимости активизируйте интерфейс с

помощью команды `ifup` и снова попытайтесь выполнить команду `ping`.

Если компьютеры локальной сети откликаются на запросы `ping`, проверьте с помощью той же команды маршрутизатор. Если вы можете обмениваться данными с другими машинами в сети, но не можете взаимодействовать с маршрутизатором, значит, надо проверить с помощью команды `route` таблицу маршрутизации. Если в таблице нет нужных записей, добавьте их. Как это сделать, описано выше в данной главе.

ЗАМЕЧАНИЕ

Диагностировать и устранять проблемы проще, если у вас есть образец конфигурационных файлов и других данных из работающей системы. Если вы знаете, что система функционирует правильно, выполните команду `route` и сохраните результаты. Вы сможете сверяться с ними, если таблица маршрутизации будет повреждена и вам придется восстанавливать ее.

Если маршрутизатор доступен, обратитесь с помощью программы `ping` к той машине из глобальной сети, о которой вы твердо знаете, что она в данный момент работает, например `www.google.com` или `www.apple.com`. Если вы не получите ответа, повторно вызовите команду `ping`, но на этот раз укажите IP-адрес компьютера. Конечно, это возможно только в том случае, если вы записали некоторые IP-адреса в записную книжку или в текстовый файл на компьютере. При необходимости вы можете использовать адреса, приведенные ниже. На данный момент они доступны, но не забывайте, что впоследствии они могут измениться.

Доменное имя	IP-адрес
`www.google.com`	72.125.141.106
`www.apple.com`	128.31.0.62
`www.ubuntu.com`	91.189.89.115

Доменное имя	IP-адрес
www.ibm.com	82.103.136.226
www.granneman.com	23.239.25.194

ЗАМЕЧАНИЕ

Как были получены эти адреса? Определить их просто. Надо лишь вызвать команду `ping`, указав ей в качестве параметра доменное имя интересующего вас компьютера. В результате `ping` отобразит его IP-адрес. Ту же информацию можно получить с помощью команды `traceroute`. Еще быстрее можно выяснить IP-адрес с помощью команды `host`, которая рассматривалась ранее в этой главе.

Если обращение по IP-адресу возможно, а по доменному имени — нет, то источником проблемы является система DNS. Если вы используете DHCP, выполните команду `dhclient`, чтобы обновить информацию о DNS, предоставляемую DHCP-сервером. Если DHCP не используется, проверьте установки в вашей системе, связанные с DNS (нужные данные можно получить у системного администратора или в службе поддержки провайдера), и при необходимости измените содержимое файла `/etc/resolv.conf`. В этом файле должна содержаться информация, подобная приведенной ниже.

```
nameserver 24.217.0.5
nameserver 24.217.0.55
```

После ключевого слова `nameserver` следует IP-адрес DNS-сервера, который вы собираетесь использовать в работе. Если ваш маршрутизатор может выполнять функции DNS-сервера, то в первую очередь надо указать его адрес (например, 192.168.0.1). Тогда запись в файле `/etc/resolv.conf` будет иметь такой вид:

```
nameserver 192.168.0.1
```

Выполните команду `ifdown`, а затем `ifup` и проверьте, происходит ли обмен данными. Здесь я должен упомянуть о проблеме, на которую обратил мое внимание читатель Брайан Грир (Brian Greer): "Если вручную добавить имя сервера в файл `/etc/resolv.conf`, то это окажется временным решением, поскольку команда `dhclient` перезаписывает этот файл при перезагрузке". Спасибо, Брайан! Вы абсолютно правы! Если вы хотите, чтобы система DNS хранила изменения постоянно, выполните следующие шаги.

Во-первых, на всякий случай сделайте копию текущего файла `resolv.conf`.

```
$ sudo cp /etc/resolv.conf /etc/resolv.conf.auto
```

Во-вторых, отредактируйте файл `dhclient.conf`, но сначала найдите его, поскольку его местоположение зависит от конкретного дистрибутивного пакета. (В системе Ubuntu 14.04 она находится в каталоге `/etc/dhcp/dhclient.conf`.) Если найти его оказалось трудно, выполните команду `find`, описанную в главе 11.

```
$ sudo find /etc -name dhclient.conf
```

Найдя файл `dhclient.conf`, отредактируйте его с помощью текстового редактора. Поскольку для редактирования этого файла необходимо иметь права пользователя `root`, редактор необходимо открыть с помощью программы `sudo` или пройти аутентификацию для доступа к файлу `dhclient.conf`, . Затем добавьте в документ следующую строку перед строкой `return subnet-mask` (обратите внимание на точку с запятой в конце!):

```
prepend domain-name-servers 208.67.222.222,
.208.67.220.220;
```

Сохраните файл `dhclient.conf` и закройте его. Отключите соединение, затем подключите его снова и проверьте, записано ли ваше имя DNS в файле `cat /etc/resolv.conf`.

ЗАМЕЧАНИЕ

Инструкции по работе с программами SUSE, Mint и Cinnamon можно найти на страницах `https://support.opendns.com/forums/21618384-Computer-Configuration`.

Если проблема не устранена, начните все сначала, а именно с аппаратного обеспечения. Все ли устройства настроены правильно? Все ли подключено? Убедившись, что все в порядке, переходите к программному обеспечению. В худшем случае у вас может не оказаться драйверов для вашего аппаратного обеспечения для работы в системе Linux. Такие события происходят редко, и со временем они происходят все реже и реже, но все-таки происходят.

И все же карты беспроводной связи могут оказаться трудно совместимыми с системой Linux из-за скрытности производителей, которые не желают помогать разработчикам Linux. Для предотвращения проблем при обдумывании покупки желательно проверить свою карту беспроводной карты на совместимость с системой Linux. Для этого могут оказаться полезными сайты Linux Wireless LAN Support (`http://linux-wless.passys.nl`) и Ubuntu Wiki's WirelessCardsSupported (`https://help.ubuntu.com/community/WifiDocs/WirelessCardsSupported`).

Мы завершаем обсуждение проблем, связанных с сетевой работой. Если вы успешно проверили свой IP-адрес и доменное имя с помощью команды `ping`, то задача решена — вы уже в сети! Желаю удачи!

Выводы

В данной главе мы рассмотрели самые разные инструменты для работы с сетевыми средствами. Многие из них, например `ifconfig`, `iwconfig` и `route`, могут решать разные задачи,

а именно: информировать вас о состоянии средств обмена по сети и изменять их конфигурацию. В некоторых случаях приходится прибегать к дополнительным средствам диагностирования проблем, например `ping`, `traceroute` и `host`. И наконец, ряд команд влияет на саму возможность взаимодействия по сети; это `dhclient`, `ifup` и `ifdown`. Если вы собираетесь серьезно работать в системе Linux, вам надо хорошо знать рассмотренные здесь инструменты. Отсутствие сетевого соединения очень неприятно, но во многих случаях, правильно выбрав программу, вы сможете быстро устранить проблему.

16

Работа в сети

Многие из команд, рассматриваемых в данной главе, предоставляют столь обширные возможности, что для подробного обсуждения их потребовалась бы отдельная книга. Мы не будем даже пытаться выяснить все, что позволяют сделать `ssh`, `rsync`, `wget` и `curl`. Материал, изложенный в данной главе, можно рассматривать лишь как общие сведения о работе с этими инструментами. Самое лучшее, что вы можете сделать, — это попытаться использовать эти команды самостоятельно, освоить их основные функции, а затем постепенно расширять свои знания о них. Это может занять достаточно много времени, но, освоив данные инструменты, вы наверняка придумаете самые различные применения для них.

Организация защищенного взаимодействия с другим компьютером

ssh

Поскольку разработчики Unix с самого начала позаботились о сетевом взаимодействии, неудивительно, что даже в самых ранних версиях системы имелись команды, позволяющие пользователям устанавливать соединения с другими машинами. Посредством этих соединений можно было запускать программы, просматривать файлы и обращаться к ресурсам. В течение длительного времени основным средством для регистрации на удаленном компьютере была система Telnet, но она имела существенный недостаток — в ней практически отсутствовала защита. Все данные, передаваемые с помощью

Telnet, — имя пользователя, пароль, любые команды и данные — пересылались в незашифрованном виде. Любой, кто имел возможность перехватить сетевой трафик, мог получить в свое распоряжение всю передаваемую информацию.

Для того чтобы справиться с этой проблемой, была разработана система ssh (secure shell). С ее помощью можно было делать все, что позволяла система Telnet, и даже много больше. Кроме того, весь трафик был закодирован, что делало систему ssh более полезной для пользователей, чем Telnet. Если вам необходимо установить взаимодействие с другим компьютером, независимо от того, находится ли он на противоположном конце земного шара или в соседней комнате, используйте для этой цели ssh.

Предположим, вы хотите обратиться средствами ssh с вашего портативного компьютера (он имеет имя pound и адрес 192.168.0.15) к настольной системе (с именем eliot и адресом 192.168.0.25) для просмотра файла. Пусть на портативном компьютере ваше пользовательское имя ezra, а на настольной системе для вас создана учетная запись tom. Чтобы организовать взаимодействие, вам надо выполнить следующую команду (если необходимо, при ее вызове можно также использовать доменное имя, например hoohah.granneman.com):

```
$ ssh tom@192.168.0.25
tom@192.168.0.25's password:
Last login: Mon Feb 6 22:40:31 2006
↳from 192.168.0.15
```

После установки соединения система запросит у вас пароль. Введите его (вы не увидите пароль на экране; это сделано для того, чтобы его не увидели и другие пользователи, которые могут в это время случайно оказаться рядом) и нажмите клавишу <Enter>. Если пароль задан правильно, вы увидите некоторую информацию о машине, к которой вы только что подключились, в частности, ее имя, версию системы и время предыдущей регистрации. Теперь вы можете вызывать на удаленной машине любые команды в пределах ваших полномочий,

причем они буду выполняться так, как будто вы работаете за ее терминалом. С точки зрения ssh и eliot ваше расположение несущественно — вы зарегистрированы и можете работать в системе.

Если вы обращаетесь к eliot впервые, то можете увидеть следующее сообщение:

```
$ ssh tom@192.168.0.25
The authenticity of host '192.168.0.25
↳(192.168.0.25)' can't be established.
RSA key fingerprint is 54:53:c3:1c:9a:07:22:0c:82:
↳7b:38:53:21:23:ce:53.
Are you sure you want to continue connecting
↳(yes/no)?
```

Система ssh сообщает, что она не может распознать вашу машину, и предлагает подтвердить ее подлинность. Введите yes, нажмите клавишу <Enter>, и вы получите другое сообщение с приглашением для ввода пароля.

```
Warning: Permanently added '192.168.0.25' (RSA)
↳to the list of known hosts.1
tom@192.168.0.25's password:
```

С этого момента взаимодействие будет происходить как обычно. Приведенное выше сообщение вы увидите лишь при первом соединении с eliot. В дальнейшем система ssh сохранит ключ RSA на компьютере pound в файле ~/.ssh/known_hosts. Откройте этот файл с помощью текстового редактора, и вы увидите, что в нем появилась новая строка. В зависимости от состояния опции HashKnownHosts в файле /etc/ssh/ssh_config эта строка будет записана в одном из двух форматов. Если значение опции равно no, эта строка будет выглядеть следующим образом:

```
192.168.0.25 ssh-rsa SkxPUQLYqXSzknsstN6Bh2MHK5AmC6E
↳pg4psdNL69R5pHbQi3kRWNNNNO3AmnP1lp2RNNNNOVjNN9mu5
↳FZe l6zK0iKfJBbLh/Mh9KOhBNtrX6prfcxO9vBEAHYITeLTMm
↳YZLQHB xSr6ehj/9xFxkCHDYLdKFmxaffgA6Ou2ZUX5NzP6Rct
↳4cfqAY69E 5cUoDv3xEJ/gj2zv0bh630zehrGc=
```

IP-адрес присутствует в ней в явном виде вместе с закодированными данными. Если же значение `HashKnownHosts` равно `yes`, то эта же строка будет иметь следующий вид:

```
NNNNO3AmnP1lp2RNNNNOVjNNNVRNgaJdxOt3GIrh00lPD6KBIU1k
↳aT6nQoJUMVTx2tWb5KiF/LLD4Zwbv2Z/j/0czCZIQNPwDUf6Y
↳iKUFFC6eagqpLDDB4T9qsOajOPLNinRZpcQoPlXflu6j1agfJ
↳zqUJUYE+Lwv8yzmPidCvOuCZ0LQH4qfkVNXEQxmyy6iz6b2wp=?
```

Теперь закодированы все данные, включая IP-адрес или доменное имя машины. С точки зрения безопасности такая запись предпочтительнее, однако если на компьютере `eliot` будет установлена другая операционная система, то возникнет проблема. Вновь обратившись после переустановки операционной системы к `eliot`, вы увидите следующее устрашающее сообщение:

```
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
@ WARNING: REMOTE HOST IDENTIFICATION HAS CHANGED! @
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
IT IS POSSIBLE THAT SOMEONE IS DOING SOMETHING NASTY
Someone could be eavesdropping on you right now
↳(man-in-the-middle attack)!
It is also possible that the RSA host key has just
.been changed.
The fingerprint for the RSA key sent by the remote
↳host is
19:85:59:5c:6a:24:85:53:07:7a:dc:34:37:c6:72:1b.
Please contact your system administrator.
Add correct host key in /home/pound/.ssh/
↳known_hosts to get rid of this message.
Offending key in /home/pound/.ssh/known_hosts:8
RSA host key for 192.168.0.125 has changed and you
↳have requested strict checking.
Host key verification failed.
```

Проблема состоит в том, что ключ `ssh` на компьютере `eliot` при установке операционной системы изменился. Средства поддержки `ssh` обнаруживают различие между ключом, хранящимся в файле `known_hosts` на компьютере `pound`, и новым ключом и предупреждают вас об этом. Вам следует удалить из

файла known_hosts строку, соответствующую eliot, сохранить файл и повторно установить соединение. Теперь с точки зрения ssh это будет ваше первое соединение, и вам будет предложено принять значение ключа. Согласитесь с этим предложением, и дальнейшее взаимодействие будет проходить, как и прежде.

ПОДСКАЗКА

В главе 12 мы рассмотрели функцию оболочки, которая позволяет легко удалить проблематичную строку из файла known_hosts.

Защищенная регистрация на другой машине без использования пароля

ssh

На первый взгляд может показаться, что в название данного раздела вкралась ошибка, однако это не так. Действительно, есть возможность зарегистрироваться средствами ssh, но не указывать при этом пароль. Если вы ежедневно работаете с одним и тем же компьютером (лично мне приходится регистрироваться на некоторых системах несколько раз в день), то подход, описанный в этом разделе, может оказаться очень полезным.

Предположим, вы хотите, чтобы при регистрации на компьютере eliot (пользовательское имя tom) с машины pound (пользовательское имя ezra) можно было не вводить пароль. Для этого прежде всего надо создать на компьютере pound аутентификационный ключ ssh, используя следующую команду:

```
$ ssh-keygen
Generating public/private rsa key pair.
Enter file in which to save the key
```

```
↳(/home/ezra/.ssh/id_rsa):
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in
↳/home/ezra/.ssh/id_rsa.
Your public key has been saved in
↳/home/ezra/.ssh/id_rsa.pub.
The key fingerprint is:
d0:c8:8c:a3:6e:c5:bd:d5:e8:0f:c8:45:6c:75:09:25
↳ezra@pound
The key's randomart image is:
+--[ RSA 2048]----+
|           Eoo.  |
|      + + . o.   |
|       o = =     |
|      o o + o    |
|    . o . S  .   |
|  . . . *        |
|   o   +  o      |
|  .        o     |
|             .   |
+-----------------+
```

ЗАМЕЧАНИЕ

В настоящее время существуют три варианта типа ключа, созданного с помощью опции `-t`: `dsa` (сокращение от *Digital Signature Algorithm*), `rsa` (аббревиатура, составленная из первых букв фамилий его изобретателей) и `ecdsa` (сокращение от *Elliptic CurveDigital Signature Algorithm*). Алгоритм DSA с августа 2015 года, когда появилась последняя версия оболочки OpenSSH, считается устаревшим из-за проблем с безопасностью (см. `http://lists.mindrot.org/pipermail/openssh-unixannounce/2015-August/000122.html`), поэтому я не рекомендую его использовать. Алгоритм ECDSA пока широко не поддерживается, и от него пока стоит воздержаться. Остается алгоритм RSA, который широко поддерживается и обеспечивает достаточно высокий уровень безопасности, при условии, что длина ключа

составляет не менее 2048 битов (число, указанное после опции -t). Если вы страдаете паранойей, то можете установить длину ключа равной 3072 и даже 4096 битов, но имейте в виду, что чем длиннее ключ, тем дольше осуществляется шифрование и расшифровка. Впрочем, быстродействие современных компьютеров делает этот процесс практически незаметным. Например, программа ssh-keygen создает ключи RSA по умолчанию, поэтому, если вы захотите сгенерировать ключ RSA длиной 4096 битов, то следует выполнить команду sshkeygen -b 4096.

Примите расположение ключа по умолчанию, нажав клавишу <Enter>, и оставьте поле passphrase пустым, дважды нажав в ответ на соответствующее приглашение клавишу <Enter>. Таким образом вы создадите закрытый ключ в файле ~/.ssh/id_dsa и открытый ключ в файле ~/.ssh/id_dsa.pub.

Теперь вам надо передать открытый ключ (но не закрытый!) с компьютера pound на компьютер eliot. Разработчики системы ssh позаботились о вас и создали программу, которая делает это быстро и без ошибок. Для того чтобы автоматически скопировать открытый ключ с компьютера pound на компьютер eliot, введите следующую команду:

```
$ ssh-copy-id -i ~/.ssh/id_dsa.pub tom@192.168.0.25
```

Теперь попытайтесь зарегистрироваться, используя ssh-идентификатор tom@192.168.0.25, и убедитесь, что в файле .ssh/autorized_keys не появились дополнительные ключи.

Все необходимые действия выполнены (если хотите последовать дополнительным рекомендациям, отображаемым при выполнении ssh-copy-id, можете сделать это). Выясним, что произойдет, если вы теперь обратитесь с компьютера pound на компьютер eliot средствами ssh.

```
$ ssh tom@192.168.0.25

Last login: Sun Sep 20 19:52:53 2015 from
↳192.168.0.15
```

Заметьте, что вам не предлагалось ввести пароль. Именно такого поведения системы вы и добивались.

Возможно, у вас возникнут сомнения в том, обеспечивает ли подобный подход реальную защиту, поскольку пароль отсутствует, а ключ передается свободно. Действительно, это так, но давайте подробнее разберемся в происходящем. Если кто-либо сможет проникнуть на компьютер `pound`, он также сможет без проблем соединиться с компьютером `eliot`, так как пароль не запрашивается. Но ведь вы приняли меры к защите системы `pound`. Если этот компьютер удастся взломать, вы будете иметь достаточно серьезные проблемы, независимо от того, поймет ли злоумышленник, что он может также проникнуть на `eliot`. С другой стороны, пароли постоянно передаются по сети. Если кто-то посторонний сумеет получить его, он получит доступ к компьютеру и сможет использовать хранящуюся на нем информацию в своих целях. Но ведь закрытый ключ так же важен, как и пароль, и его также надо надежно хранить. Немного поразмыслив, вы придете к выводу, что обмен ключами посредством `ssh` обеспечивает не меньший, а во многих случаях более высокий уровень защиты, чем использование паролей.

ПОДСКАЗКА

Если работа с ключом без пароля пугает вас, то попробуйте воспользоваться программой `ssh-agent`, управляющей ключами. Введите пароль для вашего ключа, и программа `ssh-agent` сохранит его в памяти. Это определенно безопаснее, чем использовать ключ без пароля, но при этом возникают некоторые проблемы, о которых следует знать. Одна из главных проблем — автоматический запуск этой программы; этот вопрос лучше всего освещен на странице `http://unix.stackexchange.com/questions/90853/how-can-i-run-ssh-add-automaticallywithout-password-prompt`.

Если вы все же предпочитаете пароли — вы вольны поступать так, как вам хочется. Одно из преимуществ открытых систем и, в частности, Linux — возможность свободного выбора.

Защищенная система FTP

```
sftp
```

Подобно тому, как ssh во многим превосходит Telnet, система SFTP предпочтительнее, чем обычный протокол FTP. Как и Telnet, FTP передает по сети пользовательское имя и пароль, не кодируя их, и любой, кто имеет возможность перехватывать пакеты, сможет узнать их. В отличие от FTP, SFTP использует SSH для шифрования всех данных: пользовательского имени, пароля и остального трафика. Если не учитывать тот факт, что SFTP обеспечивает в миллион раз более надежную защиту, чем FTP, то в остальном эти системы очень похожи друг на друга. В них используются одинаковые системы команд, в результате пользователь, который привык работать с FTP, без труда перейдет на SFTP.

ЗАМЕЧАНИЕ

Отличное, хотя местами упрощенное описание протокола FTP и причин, по которым его не следует использовать, см. на странице `http://vywiki.wooledge.org/FtpMustDie`.

Если вы можете получить доступ к машине средствами SSH, то вы также можете работать с ней с помощью SFTP. Для того чтобы вызвать на компьютере `pound` (192.168.0.15; пользовательское имя `ezra`) команду `sftp`, которая позволит взаимодействовать с машиной `eliot` (192.168.0.25; пользовательское имя `tom`), надо ввести в командной строке следующее выражение:

```
$ sftp tom@192.168.0.25
Connecting to 192.168.0.25...
tom@192.168.0.25's password:
sftp>
```

Если вы внимательно прочитали предыдущий раздел, то, возможно, спросите, почему система предлагает ввести пароль? Вы правы, в данный момент этого не должно быть. Дело в том, что этот пример был получен еще до того, как система была настроена для установления соединения без пароля. Если же мы будем работать с системой `ssh`, не запрашивающей пароль, то процедура регистрации будет выглядеть следующим образом:

```
$ sftp tom@192.168.0.25
Connecting to 192.168.0.25...
sftp>
```

После регистрации в системе SFTP можно использовать стандартные команды для работы с каталогами и передачи файлов. Некоторые из наиболее часто применяемых команд описаны в табл. 16.1. Полный список операций можно получить с помощью команды `man sftp`.

Таблица 16.1. Наиболее часто используемые команды SFTP

Команда	Описание
`cd`	Перейти в другой каталог
`exit`	Закрыть соединение с удаленным SSH-сервером
`get`	Скопировать указанный файл на локальную машину
`help`	Отобразить справочную информацию
`lcd`	Перейти в другой каталог на локальной машине
`lls`	Отобразить список файлов на локальной машине
`ls`	Отобразить список файлов в текущем каталоге на удаленном SSH-сервере
`put`	Скопировать указанный файл на удаленный SSH-сервер
`rm`	Удалить указанный файл с удаленного SSH-сервера

ПОДСКАЗКА

Приступая к настройке конфигурации серверов SSH, администраторы иногда переключают порт, используемый SSH по умолчанию, с 22 на другой, чтобы повысить безопасность. Это не фокус, но все же так можно повысить скорость выполнения некоторых сценариев. Если сервер SSH, с которым вы соединяетесь с помощью команды `sftp`, изменил порт, используемый по умолчанию, то с ним невозможно установить соединение. Для того чтобы использовать порт SSH, заданный не по умолчанию (или для того, чтобы учесть любые другие изменения, которые мог внести администратор сервера SSH), перед конкретной опцией SSH необходимо указать опцию `-o`. Например, если сервер SSH использует порт 2345, а не 22, с ним можно соединиться с помощью команды `sftp -oPort=2345` (обратите внимание на отсутствие пробела между опциями `-o` и SSH). Если вы используете дополнительные опции SSH, то просто повторите опцию `-o`, например, `sftp -oPort=2345 -oPasswordAuthentication=no` (полный список см. на справочной странице `man ssh_config`).

Защищенное копирование файлов между узлами сети

scp

Если у вас мало времени и вам надо скопировать файл с одной машины на другую, обеспечив при этом защиту, вам поможет команда `scp` (secure copy). Данная команда вызывается следующим образом:

```
scp пользователь@узел_1:файл_1 пользователь@
узел_2:файл_2
```

Как видите, синтаксис похож на известную вам команду cp, "расширенную" для работы в сети. Для того чтобы подробнее ознакомиться с работой данной команды, рассмотрим пример. Предположим, что вам надо скопировать с помощью scp файл backup.sh с компьютера pound (192.168.0.15; пользовательское имя ezra) в каталог /home/tom/bin на компьютере eliot (129.168.0.25; пользовательское имя tom).

```
$ pwd
/home/ezra
$ ls ~/bin
backup.sh
$ scp ~/bin/backup.sh tom@192.168.0.25/home/tom/bin
backup.sh 100% 8806 8.6KB/s 00:00
$
```

Пароль не запрашивается. Дело в том, что scp работает на основе системы ssh, которую мы в предыдущем разделе настроили для регистрации без указания пароля. Если в вашей системе такая настройка не была выполнена, то, перед тем, как продолжить работу, система предложит вам ввести пароль пользователя tom.

Допустим, у вас есть несколько JPEG-файлов, которые вы хотите скопировать с компьютера pound на компьютер eliot. Сделать это очень просто: надо лишь применить символ групповой операции.

```
$ ls -1 ~/covers
earth_wind_&_fire.jpg
handel_-_chamber_music.jpg
smiths_best_1.jpg
strokes_-_is_this_it.jpg
u2_pop.jpg
$ scp *.jpg tom@192.168.0.25:/home/tom/album_covers
earth_wind_&_fire.jpg       100%   44KB   43.8KB/s
handel_-_chamber_music.jpg  100%   12KB   12.3KB/s
smiths_best_1.jpg           100%   47KB   47.5KB/s
strokes_-_is_this_it.jpg    100%   38KB   38.3KB/s
u2_pop.jpg                  100%  9222    9.0KB/
sQ
```

Теперь предположим, что вам надо выполнить копирование в обратном направлении. Вы по-прежнему работаете с компьютером pound и хотите скопировать несколько изображений с eliot на компьютер pound в каталог, отличный от текущего.

```
$ scp tom@192.168.0.25:/home/tom/pictures/dog/libby*
↳~/pix/libby
libby_in_window_1.20020611.jpg  100%  172KB  172.4KB/s
libby_in_window_2.20020611.jpg  100%  181KB  180.8KB/s
libby_in_window_3.20020611.jpg  100%  197KB  196.7KB/s
libby_in_window_4.20020611.jpg  100%  188KB  187.9KB/s
```

Как видите, команда scp очень удобна, если вам надо выполнить защищенное копирование файлов между различными машинами. Если же вам надо скопировать большое количество файлов, работа с данной командой вскоре станет для вас утомительной. Возможно, в этом случае будет более целесообразно воспользоваться системой SFTP или организовать обмен с диском, смонтированным средствами Samba (система Samba будет рассмотрена в следующей главе).

Защищенная передача файлов и создание резервных копий

rsync

Команда rsync — великолепный, чрезвычайно полезный инструмент, который многие пользователи (и я в их числе) применяют ежедневно. Что делает эта команда? Она может решать неисчислимое множество задач (это одна из тех команд, о которых можно написать отдельную книгу), но здесь мы сосредоточим внимание на ее самой необходимой функции — способности эффективно создавать резервные копии, затрачивая на это минимальный объем сетевого трафика.

Предположим, вы хотите каждый вечер создавать копии файлов общим объемом 2 Гбайт, находящихся на машине

coleridge (пользовательское имя sam), и хранить их на другом компьютере, wordsworth (пользовательское имя will). Если бы в вашем распоряжении не было команды rsync, вам пришлось бы каждый вечер копировать 2 Гбайт информации. Это была бы значительная нагрузка даже на быстродействующие каналы связи. Благодаря команде rsync необходимую информацию удается передать почти мгновенно. Почему так происходит? Дело в том, что при формировании резервных копий команда rsync передает лишь информацию о различиях между файлами. Если за последние 24 часа изменение претерпели только несколько сотен килобайт, то именно такой объем информации и будет передан. Если же модифицированы были 100 Мбайт, команда rsync передаст этот объем данных. В любом случае это намного меньше, чем 2 Гбайт.

Ниже приведена команда, которая запускается на компьютере coleridge и передает содержимое каталога с документами на диск компьютера wordsworth, предназначенный для хранения резервных копий. Ознакомьтесь с командой и результатами ее выполнения и приготовьтесь к рассмотрению конкретных опций. Как видите, в листинге приведены два варианта одной и той же команды: в первом использованы "длинные" имена опций, а во втором некоторые опции представлены в виде одной буквы.

```
$ rsync --archive --verbose --compress --rsh=ssh
↳--progress --stats --delete /home/sam/documents/
↳will@wordsworth:/media/backup/documents
$ rsync -a -v -z -e ssh --progress --stats
↳--delete /home/sam/documents/
↳will@wordsworth:/media/backup/documents
```

Ту же команду можно записать и так:

```
$ rsync -vrtplze ssh --progress --stats
↳--delete /home/sam/documents/
↳will@wordsworth:/media/backup/documents
```

Независимо от способа вызова, вы получите результаты наподобие приведенных ниже.

```
building file list ...
107805 files to consider
deleting clientele/Linux_Magazine/do_it_yourself/13/
↳gantt_chart.txt~
deleting Security/diebold_voting/black_box_voting/
↳bbv_chapter-9.pdf
deleting Security/diebold_voting/black_box_voting/
↳bbv_chapter-8.pdf
deleting E-commerce/Books/20050827 ebay LIL ABNER
↳FRAZETTA SUNDAYS 2.txt
deleting E-commerce/Books/20050827 The State We're
↳In.txt
deleting E-commerce/Books/20050811 eBay LIL ABNER
↳DAILIES 6 1940.txt
Security/electronic_voting/diebold/black_box_voting/
↳bbv_chapter-8.pdf
Security/electronic_voting/diebold/black_box_voting/
↳bbv_chapter-9.pdf
legal_issues/free_speech/Timeline A history of free
↳speech.txt
E-commerce/2005/Books/20050811 eBay LIL ABNER
↳DAILIES 6 1940.txt
E-commerce/2005/Books/20050827 The State We're
↳In.txt
E-commerce/2005/Books/20050827 ebay LIL ABNER
↳FRAZETTA SUNDAYS 2.txt
connectivity/connectivity_info.txt
[Результаты сильно сокращены для экономии места]
Number of files: 107805
Number of files transferred: 120
Total file size: 6702042249 bytes
Total transferred file size: 15337159 bytes
Literal data: 0 bytes
Matched data: 0 bytes
File list size: 2344115
File list generation time: 86.041 seconds
File list transfer time: 0.000 seconds
Total bytes sent: 2345101
Total bytes received: 986

sent 2345101 bytes    received 986 bytes   7507.48
↳bytes/sec
total size is 6702042249   speedup is 2856.69
```

Сначала rsync формирует список файлов, предназначенных для обработки (в данном случае их количество равно 107805), затем удаляет с целевого носителя (wordsworth) те файлы, которых больше не существует на исходном носителе (coleridge). В данном примере удалены три файла: резервная копия статьи из Linux Magazine, PDF-файл с результатами электронного голосования и один текстовый файл.

После удаления файлов команда rsync заменяет файлы, подвергшиеся изменениям, новыми версиями. В данном случае копируются четыре файла. Несмотря на то что два PDF-файла были лишь перемещены в другой подкаталог, для команды rsync они являются новыми файлами, поэтому программа копирует их. Файл A history of free speech.txt — новый, поэтому он также копируется на wordsworth.

После вывода сообщений об изменениях команда rsync предоставляет общие сведения о переданных данных. В данном случае скопированы были 120 файлов, 15337159 байтов (около 14 Мбайт) из 6702042249 (приблизительно 6,4 Гбайт). Отображаются и другие данные, но эти — основные.

Теперь поговорим о том, какие действия вы предлагаете выполнить вашему компьютеру. Понять начало и окончание команды несложно. Вначале указывается имя rsync, затем опции, после них исходный каталог, который содержит файлы для копирования (/home/sam/documents/ на машине coleridge), после этого следует целевой каталог, в который должны быть скопированы файлы (/media/backup/documents на компьютере wordsworth). Перед тем как рассматривать опции, надо обратить внимание на особенности указания исходного и целевого каталогов; если не учитывать их, могут возникнуть проблемы.

В данном случае нам надо скопировать содержимое каталога documents, расположенного на машине coleridge, но не сам каталог, поэтому мы указываем documents/, а не documents. Последняя косая черта в выражении /home/sam/documents/ сообщает команде rsync о том, что необходимо копировать содержимое заданного каталога в каталог documents, расположенный на машине wordsworth. Если вместо documents/ мы

зададим documents, будет скопировано и содержимое, и сам каталог, в результате чего на компьютере wordsworth появится каталог /media/backup/documents/documents.

ЗАМЕЧАНИЕ

Косая черта важна только при указании исходного каталога. Задавая целевой каталог, включать ее не обязательно.

Существует опция, которая не была указана в приведенных выше примерах, но которая может быть очень полезна при использовании команды rsync, — это опция -n (или --dry-run). Если вы зададите эту опцию, команда rsync выполнится, но не будет реально удалять или копировать файлы. Такая "пробная попытка" особенно полезна, когда есть опасность удалить важные файлы. Прежде чем вызывать команду rsync, особенно если для нее указана опция --delete, испробуйте ее "на холостом ходу".

Теперь рассмотрим остальные опции. Опция -v (или --verbose) вместе с опцией --progress указывает команде rsync на то, что в процессе работы надо подробно информировать пользователя о выполненных действиях. Вы видели результаты выполнения команды ранее в этом разделе. В данном случае команда rsync сообщала об удаленных и скопированных файлах. Если вы включаете вызов команды rsync в состав сценария, то опция -v, вероятно, будет излишней, но при интерактивном выполнении данной команды полезно иметь сведения о выполняемых действиях.

Метаданные в конце информации, отображаемой rsync, т.е. сведения о числе переданных файлов и их общем размере, а также другие данные выводятся потому, что при вызове команды была указана опция --stats. Эта опция, как и -v, не нужна при выполнении команды rsync в составе сценария, но полезна при вызове команды вручную.

Опция `-r` (или `--recursive`) знакома вам по другим командам. Здесь она имеет то же назначение, что и обычно, — расширяет область действия команды на подкаталоги. В приведенном выше примере эта опция задана, поскольку нам надо скопировать все содержимое каталога `documents`.

Если в исходном каталоге имелась мягкая ссылка, то опция `-l` (или `--links`) создает такую же ссылку в целевом каталоге. Вместо копирования файла копируется ссылка на него. Таким образом, содержимое каталога воссоздается в том виде, в каком оно было на компьютере-источнике.

Права доступа обсуждались в главе 7. Теперь пришло время снова вспомнить о них. Опция `-p` (или `--perms`) указывает `rsync` на то, что для каждого файла, записанного в целевой каталог, надо установить те же права доступа, что и для исходного файла. Данная опция позволяет выполнять резервное копирование максимально точно, поэтому не следует пренебрегать ею.

Опция `-t` (или `--times`) указывает команде `rsync` на то, что вместе с файлами надо копировать информацию о времени их модификации. Если вы не включите эту опцию, команда `rsync` не сможет определить состояние файлов, скопированных ранее, поэтому при очередном вызове снова скопирует все файлы. Такое поведение вряд ли устроит вас, так как оно сводит на нет все преимущества команды `rsync`. Поэтому всегда указывайте при вызове данной команды опцию `-t`.

Даже при наличии быстродействующих соединений желательно использовать опцию `-z` (или `--compress`). Если она указана, то при передаче файлов `rsync` выполняет gzip-сжатие. Если линии связи имеют малую пропускную способность, эта опция обязательна. Для высокоскоростных линий также желательно задавать ее, так как она позволит сэкономить вам время.

Обеспечить безопасность позволяет опция `-e` (или `--rsh=ssh`). Она задает передачу данных средствами `ssh`. Как видите, команда `rsync` обеспечивает не только быстрое, но и безопасное копирование данных.

ЗАМЕЧАНИЕ

Если используется система SSH, то почему не задается пароль? Ранее в этой главе мы настроили средства SSH для работы без введения пароля.

И напоследок рассмотрим самую опасную опцию `--delete`. Если вы создаете зеркальное отражение файлов, то, вероятно, хотите, чтобы оно было как можно более точным. Это означает, что файлы, удаленные в исходном каталоге, должны быть удалены и в целевом. Но при этом можно случайно удалить нужные данные. Если вы собираетесь задать опцию `--delete` (а рано или поздно вам придется сделать это), сначала выполните команду с опцией `-n` (или `--dry-run`), которая рассматривалась выше.

ПРЕДУПРЕЖДЕНИЕ

Существует еще одна, ранее не рассмотренная, опция, которая позволяет прояснить структуру команды `rsync`: `-n` (или `--dry-run`). Если эта опция включена, то команда `rsync` начинает работать, но на самом деле она ничего не удаляет и не копирует. Это позволяет избежать неприятностей, если вы рискуете удалить важные файлы. Перед тем как выполнить команду `rsync`, особенно если она содержит опцию `--delete`, сначала выполните команду `rsync -n`!

В команде `rsync` предусмотрено много других опций; существует множество вариантов ее применения (в справочном руководстве описывается восемь таких вариантов). Материал этого раздела позволит вам начать работу с команды `rsync`. Вызовите на своем компьютере команду `man rsync` или выполните средствами Google поиск по ключевым словам `rsync tutorial`, и вы найдете огромное количество информации о команде `rsync`. Если вы случайно удалите файл, а потом

вспомните, что существует его резервная копия, созданная с помощью rsync, вы поймете, что не зря потратили время на изучение этой замечательной команды.

ПОДСКАЗКА

Если вы хотите, чтобы ваши данные действительно были в безопасности, организуйте регулярный запуск rsync средствами cron. Например, создайте файл backup.sh (его можно поместить в каталог ~/bin) и введите в нем следующую команду:

```
$ rsync --verbose --progress -stats --recursive
--times --perms -links --compress --rsh=ssh -delete
/home/sam/documents/ will@wordsworth:/media/backup/
documents
```

С помощью chmod установите признак исполняемого файла.

```
$ chmod 744 /home/scott/bin/backup.sh
```

Затем включите в файл cronfile (он также может храниться в каталоге ~/bin) следующие строки:

```
# backup documents every morning at 3:05 am
05 03 * * * /home/scott/bin/backup.sh
```

Первая строка представляет собой комментарии, в которых объясняется назначение задачи, а вторая строка сообщает cron о том, что каждую ночь в 3:05 надо автоматически выполнить инструкции, заданные в файле /home/scott/bin/backup.sh.

После создайте задачу для cron.

```
$ crontab /home/scott/bin/cronfile
```

Теперь вам не надо заботиться о создании резервных копий. Требуемые действия будут выполняться автоматически, вам надо лишь оставить компьютер включенным на ночь.

(Дополнительную информацию о cron можно получить, выполнив команду man cron либо обратившись

к документу *Newbie: Intro to cron*, который доступен по адресу `http://www.unixgeeks.org/security/newbie/unix/cron-1.html`.)

Копирование файлов из веб

`wget`

Глобальная сеть — неисчерпаемый источник данных. В частности, в ней есть огромное количество изображений, видео- и аудиофайлов, доступных для копирования. Однако, если вручную копировать, например, каждый mp3-файл из каталога, содержащего 200 таких файлов, эта затея быстро надоест. Команда `wget` позволяет организовать копирование файлов с веб-сайтов, но не предоставляет графического пользовательского интерфейса. Вы вводите ее в командной строке, и спустя некоторое время (возможно, даже через несколько часов) она предоставит вам результат.

Однако правильно вызвать команду `wget` не всегда просто. Она предоставляет огромные возможности, описание которых займет отдельную книгу, и в рамках одной главы невозможно рассказать, что она позволяет сделать. Поэтому мы сосредоточимся на решении двух задач с помощью команды `wget`: копирования множества отдельных файлов и копирования целых веб-сайтов.

В сети есть сайт The Old Time Radio Archives, на котором собраны записи радиопередач, представленных в формате mp3. Годовое собрание составляет 365 mp3-файла, по одному файлу на каждый день в году. Предположим, вы хотите скопировать эти файлы, но перспективу щелкать правой кнопкой мыши на каждой ссылке, выбирать в контекстном меню пункт меню Save Link As ..., а затем щелкать на кнопке OK вряд ли можно назвать привлекательной.

Ознакомившись с каталогом, вы заметите, что он организован следующим образом:

```
http://www.oldtimeradioarchives.com/mp3/
  season_10/
  season_11/
  ...
  season_20/
  ...
  season_3/
  season_4/
  ...
  season_9/
```

ЗАМЕЧАНИЕ

Каталоги отсортированы не по номерам, как это сделал бы, наверное, каждый человек, а в алфавитном порядке. Именно такой принцип сортировки по умолчанию реализует большинство компьютерных программ. В этом случае получается, что слово "ten" предшествует слову "three".

В каждом каталоге размещены mp3-файлы. Если вы щелкнете на ссылке на каталог, отобразится веб-страница, в которой перечислены файлы из этого каталога.

```
[BACK] Parent Directory   19-May-2002 01:03 -
[SND]  1944-12-24_532.mp3 06-Jul-2002 13:54 6.0M
[SND]  1944-12-31_533.mp3 06-Jul-2002 14:28 6.5M
[SND]  1945-01-07_534.mp3 06-Jul-2002 20:05 6.8M
[SND]  1945-01-14_535.mp3 06-Jul-2002 19:53 6.9M
```

Таким образом, задача сводится к следующему: как скопировать все mp3-файлы, имеющие различные имена и находящиеся в разных каталогах. Решить ее поможет команда `wget`.

Начните с создания на вашем компьютере каталога, в который вы поместите скопированные mp3-файлы.

```
$ mkdir radio_mp3s
```

Затем сделайте этот каталог текущим и вызовите команду `wget`.

```
$ cd radio_mp3s
$ wget -r -l2 -np -w 5 -A.mp3 -R.html,.gif
↳http://www.oldtimeradioarchives.com/mp3/`
```

Рассмотрим подробнее команду и ее опции.

В начале командной строки указано имя команды, а в конце — URL ресурса, с которым она должна работать: `http://www.oldtimeradioarchives.com/mp3`. Между именем команды и URL содержатся опции, которым необходимо уделить особое внимание.

Опция `-r` (или `--recursive`) задает следование по ссылкам и просмотр каталогов в поисках файлов. В данном случае рекурсивное выполнение `wget` означает, что команда просмотрит каталоги, соответствующие каждому сезону, и скопирует найденные в них mp3-файлы.

Опция `-l2` (или `--level=[#]`) очень важна. Она указывает команде `wget` глубину рекурсивного поиска файлов. Буква `l` означает `level`, или уровень, а следующее за ней число — глубину поиска. Если вы введете `-l1`, команда `wget` будет извлекать файлы только из каталога `/mp3`. В результате вы ничего не получите, так как в этом каталоге находятся лишь подкаталоги `season_10`, `season_11` и т.д., а уже в них размещаются mp3-файлы. Задав опцию `-l2`, вы сообщите команде `wget` о том, что надо в первую очередь проверить `/mp3` (этот каталог соответствует первому уровню), а затем по очереди обратиться к каждому каталогу `season_#` и извлечь файлы из него. Задавая уровень, будьте очень внимательны, так как вы можете заполнить весь жесткий диск ненужными данными.

Один из способов избежать копирования лишних файлов — использование опции `-np` (или `--no-parent`). Она не позволяет осуществлять рекурсивный поиск в родительском каталоге. Если вы вернетесь к предыдущему листингу, то заметите, что первая ссылка указывает именно на родительский каталог. Если вы находитесь в каталоге `season_10`, то родительским для него является `/mp3`. То же самое справедливо для `season_11`, `season_12` и т.д. Однако нам не нужно перемещение вверх по иерархии объектов, команда `wget` должна перемещаться только

вниз. И конечно же, не следует тратить время на возврат в уже обработанный каталог, в данном случае /mp3.

Следующая опция не обязательна, однако, если вы не хотите доставить лишние хлопоты администратору сайта, к которому обращаетесь, установите ее. Речь идет об опции -w (или --wait=[#]), которая задает небольшую задержку перед началом копирования каждого файла. Это предотвращает излишнюю нагрузку на сервер, возникающую вследствие постоянного извлечения файлов. По умолчанию считается, что значение указано в секундах, но при желании вы можете использовать задержку в минутах, если введете после числа букву m, в часах (буква h) и даже в днях (буква d).

Теперь начинается самое интересное. Опция -A (или --accept) сообщает команде wget о том, что вы хотите скопировать только файлы определенного типа. В данном случае A означает accept; после опции указываются суффиксы имен файлов, разделенные запятыми. Нам нужны файлы только одного типа, mp3, поэтому мы задаем -A.mp3.

Противоположные действия выполняет опция -R (или --reject). С ее помощью мы указываем файлы, которые нам не нужны, а именно HTML и GIF. Таким образом, мы, в частности, отказываемся от пиктограммы с изображением нот, которая в приведенном выше листинге представлялась с помощью символов [SND]. Суффиксы в списке разделяются запятыми, и опция имеет следующий вид: -R.html,.gif.

При выполнении команды wget с указанным выше набором опций на ваш компьютер будет скопировано 365 mp3-файлов. Не исключено, что передача прервется. Это может произойти по многим причинам: маршрутизатор выйдет из строя, или кто-то наступит на сетевой кабель и выдернет его из гнезда, или при монтажных работах будет поврежден волоконно-оптический кабель и т.д. В этом случае вам надо повторно вызвать команду, задав дополнительно опцию -c (или --continue), которая сообщает wget о том, что надо возобновить прерванное копирование. В этом случае уже полученные файлы не будут копироваться вновь.

ЗАМЕЧАНИЕ

Следующий пример больше не работает, но я сохранил его, чтобы объяснить необходимые концепции. Не пытайтесь его выполнить!

Рассмотрим еще один пример использования команды `wget` для копирования файлов. Предположим, мы узнали о появлении двух новых альбомов. Файлы mp3 с композициями из этих альбомов перечислены на странице `http://www.djbc.net/beastles/`. Приведенная ниже команда извлечет ссылки с веб-страницы, запишет их в файл, а затем инициирует копирование соответствующих ресурсов средствами `wget`.

```
$ lynx -dump http://www.djbc.net/beastles/ |
↳awk '/http/{print $2}' |
↳grep mp3 > beastles ; wget -i beastles
--12:58:12-- http://www.djbc.net/beastles/
↳webcontent/djbc-holdittogethernow.mp3
          => 'djbc-holdittogethernow.mp3'
Resolving www.djbc.net... 216.227.209.173
Connecting to www.djbc.net|216.227.209.173|:80...
↳connected.
HTTP request sent, awaiting response... 200 OK
Length: 4,533,083 (4.3M) [audio/mpeg]

100%[=========>] 4,533,083 203.20K/s ETA 00:00
12:58:39 (166.88 KB/s) - 'djbc-holdittogethernow.
↳mp3' saved [4533083/4533083]
[Результаты сокращены для экономии места]
```

Практически все, кто использует веб-браузер с графическим пользовательским интерфейсом, наподобие Firefox, Chrome или Safari, для повышения быстродействия, безопасности или возможности выполнения сценариев предпочитают использовать текстовые браузеры с командной строкой. Их немало — например, `links`, `elinks`, `m3m`, — но мне нравится `lynx`, который включен во многие дистрибутивные пакеты. Если вы еще не установили его, то всегда можете это сделать с помощью менеджера пакетов (см. главу 14).

Если вы вызовете программу lynx с опцией -dump и зададите в качестве параметра указатель URL, ссылки будут извлечены из указанного документа и выведены в STDOUT. Для фильтрации я использую программу awk (см. главу 7). В данном примере полученные ссылки передаются команде grep, которая фильтрует данные, оставляя строки, содержащие символы mp3, а затем выходная информация перенаправляется в текстовый файл beastles (о перенаправлении вывода см. в главе 5, а о команде grep в главе 10).

После точки с запятой вводится новая команда — wget (о разделении команд посредством точки с запятой см. в главе 4). Опция -i (или --input-file) сообщает команде wget о том, что URL ресурсов для копирования указываются не в стандартном входном потоке, а в файле. Если у вас есть много ссылок, поместите их в файл, а затем задайте при вызове wget опцию -i. В данном примере эта опция указывает на файл beastles, созданный в результате совместной работы команд dog и grep, и wget начинает копировать требуемые файлы один за другим.

И все эти действия были заданы в одной строке. Такие богатые возможности может предоставить только Linux!

ЗАМЕЧАНИЕ

Как вы уже знаете, команда wget загружает файлы. А если вы захотите *выгрузить* файлы в неинтерактивном режиме? Для этого используется команда wput! Более подробную информацию об этой команде вы найдете на странице http://wput/sourceforge.net. Она работает только на FTP-серверах, но все еще считается антиподом программы wget.

Копирование веб-сайтов

`wget`

Если вы хотите скопировать на свой компьютер веб-сайт или создать резервную копию своего сайта, вам поможет в этом команда `wget`. В предыдущем разделе мы использовали команду `wget` для получения отдельных файлов, но она позволяет также работать и с целыми сайтами.

ЗАМЕЧАНИЕ

Используйте команду `wget` разумно. Не пытайтесь скопировать сайты, содержащие слишком большой объем данных, и имейте в виду, что сайт, который вы собираетесь скопировать, кто-то создал и имеет права на него.

Предположим, вас интересует сайт `http://www.neato.com`, а сами вы в текущий момент работаете с документом `http://www.neato.com/articles/index.htm`. Вам нужно содержимое раздела `/articles` и ничего более. Приведенная ниже команда предоставит вам требуемые данные.

```
$ wget -E -r -k -p -w 5 -np
  http://www.neato.com/articles/index.htm
```

При желании вы можете объединить опции так, как показано ниже.

```
$ wget -Erkp -w 5 -np
↳http://www.neato.com/articles/index.htm
```

Подобно примеру из предыдущего раздела, данная строка начинается с имени `wget` и заканчивается интересующим нас URL. Вы уже знакомы с опциями `-w` (или `--wait=[#]`), `-np` (или `--no-parent`) и `-r` (или `--recursive`); здесь они выполняют те же функции, что и прежде. Рассмотрим подробнее опции, которые встретились вам впервые.

На сайте, предназначенном для копирования, не все файлы имеют суффиксы .htm или .html; встречаются также файлы .asp, .php, .cfm и т.д. Если вы попытаетесь просмотреть содержимое такого сайта, скопированного на свой компьютер, возникнут проблемы. Если на вашей машине установлен веб-сервер, все в порядке, но вероятнее всего, что его там нет. Однако и без сервера файлы .htm и .html можно открыть с помощью браузера. Если вы зададите опцию -E (или --html-extension), команда wget преобразует каждую страницу в html-файл, что позволит вам просматривать их с помощью браузера, не используя дополнительные программы.

При копировании сайта возникают и другие проблемы, с которыми можно успешно справиться, правильно выбрав набор опций команды wget. Не исключено, что, открыв веб-документ на своем компьютере, вы обнаружите, что ссылки не работают, и вы не можете переходить с одной страницы на другую. Задав опцию -k (или --convert-links), вы укажете команде wget на то, что надо преобразовать ссылки в составе веб-страниц. Эта опция затрагивает ссылки, указывающие не только на другие страницы, но и на изображения, каскадные таблицы стилей и файлы. Используйте ее, и вы будете довольны результатами.

Наличие каскадных таблиц стилей (Cascading Style Sheets — CSS) и изображений обусловливает необходимость установить опцию -p (или --page-requisites). Для того чтобы веб-страница отображалась корректно, веб-разработчик должен включать в нее ссылки на файлы, содержащие таблицы стилей, изображения и JavaScript-сценарии, которые будут использоваться с HTML-файлом. Опция -p указывает команде wget на то, что следует скопировать все файлы, необходимые для отображения веб-страницы. Если данная опция задана, то, открыв на своем компьютере скопированную веб-страницу, вы увидите, что она выглядит точно так же, как будто бы вы открыли ее, обращаясь к серверу. Без этой опции вы обнаружите, что некоторые файлы отсутствуют.

Страница справочного руководства, посвященная команде wget, содержит огромный объем информации. Освоив ее, вы смо-

жете использовать данную команду для решения еще более сложных задач. Если, прочитав данную главу, вы поняли, что команда wget будет полезна вам, принимайтесь за изучение руководства.

Указание последовательностей имен копируемых файлов

curl

На первый взгляд команды wget и curl выглядят одинаково: обе позволяют копировать файлы при запуске из командной строки. Однако между ними есть одно существенное различие и много мелких. Главное отличие состоит в том, что команда curl позволяет указывать последовательности и множества файлов, предназначенных для копирования, а команда which не позволяет сделать этого; с другой стороны, wget поддерживает рекурсивную обработку, чего не обеспечивает curl.

ЗАМЕЧАНИЕ

Эти программы имеют множество более мелких отличий. Полный список возможностей команды curl можно найти в документе *Features -- what can curl do* (http://curl.haxx.se/docs/features.html), а некоторые функции wget описаны в документе *Overview* (http://www.gnu.org/software/wget/manual/html_node/Overview.html#Overview). На сайте, посвященном программе curl, содержится диаграмма сравнения curl с другими продуктами аналогичного назначения. Она включена в документ *Compare cURL Features with Other FTP+HTTP Tools* (http://curl.haxx.se/docs/comparison-table.html). Судя только по диаграмме, предпочтение надо отдавать curl (это неудивительно, учитывая направленность материалов на сайте).

Рассмотрим пример, демонстрирующий способность команды curl поддерживать последовательности имен файлов, предназначенных для копирования. Public Radio предоставляет архивы записей своей передачи This American Life. Эти архивы доступны для копирования с веб-сайта в формате Real Audio (почему в данном случае выбран данный формат, а не другой, открытый, остается для меня загадкой). Если вы хотите скопировать десять файлов Real Audio, введите следующую команду:

```
$ curl -O http://www.wbez.org/ta/[1-10].rm
[1/10]: http://www.wbez.org/ta/1.rm --> 1.rm
--_curl_--http://www.wbez.org/ta/1.rm

[Результаты сокращены для экономии места]
```

Обратите внимание: для того, чтобы указать, что вы хотите скопировать файлы 1.rm, 2.rm, 3.rm и т.д., вы используете запись [1-10].rm. Если бы эти файлы имели имена one.rm, two.rm и three.rm, вам пришлось бы использовать вместо последовательности множество.

```
$ curl -O http://www.wbez.org/ta/{one,two,three}.rm
```

Опция -O (или --remote-name) необходима. Если вы не будете использовать ее, команда curl выведет результаты копирования в стандартный выходной поток, в результате чего окно вашего терминала будет заполнено непонятными символами. Опция -O указывает команде curl на то, что данные надо записывать в файл и что имя создаваемого файла на локальном компьютере должно совпадать с именем соответствующего файла на удаленной машине.

ЗАМЕЧАНИЕ

Предыдущий пример также устарел и больше не работает, но я сохранил его для иллюстрации полезности команды curl.

В данном разделе были приведены лишь самые общие сведения о команде curl. Страница справочного руководства, посвященная этой команде, имеет меньший объем, чем страница о команде wget, но также предоставляет полезную информацию, которая позволит вам в полной мере использовать возможности curl. Советую прочитать ее.

Выводы

Команды, описанные в этой главе, выбраны по одному принципу: лично я предпочитаю их остальным, так как они помогают мне решить трудные задачи, требующие выполнения рутинной работы либо представляющие опасность для информации на компьютере. Эти команды расширяют сферу применения данных, программ, да и вашего воображения на всю сеть. Предоставляя свободный доступ к таким мощным инструментам, как ssh, rsync, wget и curl, Linux открывает дорогу для использования новых технологий. В операционных системах, которые почему-то предпочитают большинство пользователей, это не всегда возможно. Если вы хотите устанавливать защищенные соединения с другими компьютерами, освойте средства ssh. Чтобы автоматически выполнять резервное копирование, обеспечивая при этом безопасность данных, изучите команду rsync. Программы wget и curl позволят вам наиболее эффективно копировать требуемые ресурсы. Эти инструменты не подведут вас!

Предметный указатель