

Дмитрий Колисниченко

Командная строка **Linux**

Работа в консоли

**Автоматизация на языках
оболочек bash и tcsh**

**Особенности файловой
системы Linux**

Управление процессами

**Команды системного
администратора**

**Команды для обработки текста
и для работы в сети**

Дмитрий Колисниченко

Командная строка Linux

Санкт-Петербург

«БХВ-Петербург»

2023

УДК 004.4

ББК 32.973.26-018.2

К60

Колисниченко Д. Н.

К60 Командная строка Linux. — СПб.: БХВ-Петербург, 2023. — 176 с.: ил. —
(Системный администратор)

ISBN 978-5-9775-1750-8

Рассмотрены задачи, которые выполняются из командной строки операционной системы Linux. Объясняется, как попасть в командную строку, работать в консоли, настраивать систему с помощью программ, обладающих только текстовым интерфейсом. Описаны особенности файловой системы Linux, наиболее полезные команды для работы с текстом, сетью и Интернетом, а также команды системного администратора. Особое внимание уделено написанию сценариев автоматизации рутинных задач на языках командных оболочек bash и tcsh. Рассмотрены способы направления ввода-вывода, маски и псевдонимы, различные варианты запуска программ, эффективные приемы использования клавиатуры, примеры сложных команд и другие вопросы.

*Для системных администраторов, программистов
и квалифицированных пользователей Linux*

УДК 004.4

ББК 32.973.26-018.2

Группа подготовки издания:

Руководитель проекта	<i>Евгений Рыбаков</i>
Зав. редакцией	<i>Людмила Гауль</i>
Редактор	<i>Григорий Добин</i>
Компьютерная верстка	<i>Ольги Сергиенко</i>
Дизайн обложки	<i>Зои Канторович</i>

Подписано в печать 09 01 23

Формат 70×100¹/₁₆. Печать офсетная Усл. печ л 14,19

Тираж 1000 экз Заказ № 5799

"БХВ-Петербург", 191036, Санкт-Петербург, Гончарная ул., 20

Отпечатано с готового оригинал-макета

ООО "Принт-М", 142300, МО, г Чехов, ул Полиграфистов, д 1

Больше книг в <https://t.me/portalToIT>

Оглавление

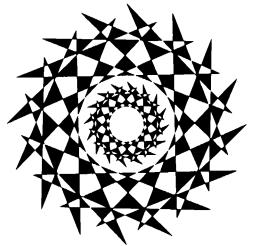
Глава 1. Как попасть в командную строку?	9
1.1. Переключаемся из графического режима в консоль	9
1.2. Графические терминалы	12
1.3. Удаленная консоль	12
1.4. Завершение работы из консоли. Команды <i>poweroff</i> , <i>halt</i> , <i>reboot</i> , <i>shutdown</i>	14
Глава 2. Как работать в консоли?	15
2.1. Ввод команд	15
2.2. Ввод/вывод и каналы.....	16
2.3. Шесть полезных команд	17
2.3.1. Команда <i>wc</i>	17
2.3.2. Команда <i>head</i>	18
2.3.3. Команда <i>cut</i>	19
2.3.4. Команда <i>grep</i>	19
2.3.5. Команда <i>sort</i>	20
2.3.6. Команда <i>uniq</i>	21
Глава 3. Оболочка bash	23
3.1. Маски имен файлов	23
3.2. Переменные окружения	24
3.3. Псевдонимы	25
3.4. Изменение приглашения командной строки	25
3.5. Еще раз о перенаправлении ввода/вывода	26
3.6. Пробелы в именах файлов	28
3.7. Инициализация переменных	29
3.8. Просмотр истории команд.....	29
3.9. Вызов предыдущих команд	30
3.10. Защита от случайного удаления файла	30
3.11. Использование последнего вывода команды	31
3.12. Редактирование командной строки.....	31

Глава 4. Файловая система. Команды для работы с файловой системой	33
4.1. Особенности файловой системы Linux	33
4.1.1. Имена файлов в Linux	33
4.1.2. Файлы и устройства	33
4.1.3. Корневая файловая система и монтирование	34
4.1.4. Стандартные каталоги Linux	35
4.2. Команды для работы с файлами и каталогами	36
4.2.1. Работа с файлами	36
4.2.2. Работа с каталогами	39
4.2.3. Удобная навигация по файловой системе	41
4.3. Команда <i>ln</i> : создание ссылок	44
4.4. Команды <i>chmod</i> , <i>chown</i> и <i>chattr</i>	45
4.4.1. Команда <i>chmod</i> : права доступа к файлам и каталогам	45
4.4.2. Команда <i>chown</i> : смена владельца файла	47
4.4.3. Специальные права доступа (SUID и SGID)	47
4.4.4. Команда <i>chattr</i> : атрибуты файла, запрет изменения файла	48
4.5. Монтирование файловых систем	49
4.5.1. Команды <i>mount</i> и <i>umount</i>	49
4.5.2. Файлы устройств и монтирование	50
Жесткие диски	50
Приводы оптических дисков	52
Дискеты	52
Флешки и внешние жесткие диски	53
4.5.3. Опции монтирования файловых систем	53
4.5.4. Монтирование разделов при загрузке	54
4.5.5. Подробно о UUID и файле <i>/etc/fstab</i>	56
4.5.6. Монтирование флешек	59
4.6. Настройка журнала файловой системы ext3	59
4.7. Файловая система ext4	60
4.7.1. Сравнение ext3 и ext4	61
4.7.2. Совместимость с ext3	62
4.8. Особые команды	62
4.8.1. Команда <i>mksfs</i> : создание файловой системы	62
4.8.2. Команда <i>fsck</i> : проверка и восстановление файловой системы	62
4.8.3. Команда <i>chroot</i> : смена корневой файловой системы	63
4.8.4. Установка скорости CD/DVD	63
4.8.5. Монтирование каталога к каталогу	64
4.8.6. Команды поиска файлов	64
4.9. Многофункциональная команда <i>dd</i>	65
4.9.1. Копирование файлов с помощью команды <i>dd</i>	66
4.9.2. Разделение файла на несколько частей	67
4.9.3. Создание резервной копии жесткого диска	67
4.9.4. Создание архива с резервной копией всего жесткого диска	67
4.9.5. Уничтожение всех данных раздела жесткого диска	68
4.10. Команда <i>du</i>	68

Глава 5. Процессы.....	70
5.1. Оболочки, команды и программы	70
5.2. Родительский и дочерний процессы	71
5.3. Команды <i>kill</i> , <i>killall</i> , <i>xkill</i> и <i>ps</i>	72
5.4. Программа <i>top</i> : кто больше всех расходует процессорное время?	74
5.5. Команды <i>nice</i> и <i>renice</i> : изменение приоритета процесса	76
5.6. Команда <i>fuser</i> : кто открыл ресурс?	77
Глава 6. Различные способы выполнения команд.....	78
6.1. Списки	78
6.1.1. Условные списки	78
6.1.2. Безусловные списки.....	79
6.2. Подстановка	80
6.2.1. Подстановка команды	80
6.2.2. Подстановка процесса	81
6.3. Команда как строка	82
6.3.1. Передача команды в виде аргумента	82
6.3.2. Перенаправление команды на стандартный ввод <i>bash</i>	82
6.4. Удаленное выполнение команды по SSH	83
6.5. Фоновое выполнение команд	83
Глава 7. Некоторые полезные команды.....	85
7.1. Команда <i>seq</i>	85
7.2. Фигурные скобки	86
7.3. Команда <i>find</i>	86
Глава 8. Команды для работы с текстом.....	88
8.1. Команда <i>sort</i> : сортировка файлов.....	88
8.2. Команда <i>diff</i> : сравнение файлов	88
8.3. Команда <i>diff3</i> : сравнение трех файлов	89
8.4. Команда <i>cmp</i> : сравнение двух файлов	90
8.5. Команда <i>comm</i> : еще одна команда для сравнения файлов	90
8.6. Команда <i>column</i> : разбивка текста на столбцы	91
8.7. Команда <i>egrep</i> : расширенный текстовый фильтр	91
8.8. Команда <i>expand</i> : замена символов табуляции пробелами	92
8.9. Команда <i>fmt</i>	93
8.10. Команда <i>fold</i>	93
8.11. Команда <i>grep</i> : текстовый фильтр	93
8.12. Команды <i>more</i> и <i>less</i> : постраничный вывод	94
8.13. Команды <i>head</i> и <i>tail</i> : вывод начала и хвоста файла	94
8.14. Команда <i>look</i>	94
8.15. Команда <i>split</i> : разбиение файлов на несколько частей	94
8.16. Команда <i>unexpand</i> : замена пробелов символами табуляции	95
8.17. Команды <i>vi</i> , <i>nano</i> , <i>ee</i> , <i>mcedit</i> , <i>pico</i> : текстовые редакторы	95
8.18. Команда <i>sed</i> : потоковый текстовый редактор	99

8.19. Команда <i>wc</i> : подсчет слов в файле	100
8.20. Некоторые команды преобразования символов и форматов	101
Глава 9. Эффективное использование клавиатуры	102
9.1. Работа с окнами	102
9.2. Доступ к веб-браузерам из командной строки	104
9.3. Работаем с буфером обмена	106
Глава 10. Команды для работы с сетью и Интернетом	107
10.1. Команда <i>ifconfig</i> : управление сетевыми интерфейсами	107
10.2. Маршрутизация.....	109
10.2.1. Команда <i>netstat</i> : просмотр таблицы маршрутизации и другой сетевой информации.....	109
10.2.2. Команда <i>route</i> : изменение таблицы маршрутизации	113
10.3. Команды получения информации об узле	116
10.3.1. Получение информации о доменном имени.....	116
10.3.2. Команды <i>host</i> и <i>dig</i>	116
10.3.3. Утилита DMitry	118
10.4. Текстовые браузеры	118
10.5. Команда <i>ftp</i> : FTP-клиент	118
10.6. Команда <i>wget</i> : загрузка файлов	120
10.7. Команды для диагностики сети	121
10.8. Команда <i>ssh</i>	125
10.9. Сетевой сканер <i>nmap</i>	127
10.9.1. Что такое <i>nmap</i> ?.....	127
10.9.2. Где мне взять <i>nmap</i> ?	128
10.9.3. Примеры использования <i>nmap</i>	128
Глава 11. Команды системного администратора	131
11.1. Программы разметки диска	131
11.1.1. Программа <i>fdisk</i>	131
11.1.2. Программа <i>parted</i>	134
11.1.3. Добавление диска на виртуальном сервере	138
11.1.4. Расширение существующего диска	141
11.1.5. Несколько слов о GPT. Утилиты для работы с GPT	144
11.2. Информация о системе и пользователях	145
11.2.1. Команда <i>uptime</i> : информация о работе системы.....	145
11.2.2. Команда <i>users</i> : информация о пользователях	145
11.2.3. Команды <i>w</i> , <i>who</i> , <i>ftwho</i> и <i>whoami</i> : информация о пользователях	146
11.2.4. Мониторинг работы системы	147
11.3. Планировщик <i>at</i>	149
11.3.1. Команда <i>at</i> : добавление задания.....	149
11.3.2. Команды <i>atq</i> и <i>atrm</i> : очередь заданий и удаление задания	149
11.4. Планировщик <i>cron</i>	150
11.5. Команда <i>date</i> : вывод и установка даты и времени	151
11.6. Команды <i>free</i> и <i>df</i> : информация о системных ресурсах.....	152

Глава 12. Автоматизация рутинных задач с помощью оболочки bash	153
12.1. Настройка bash.....	153
12.2. Автоматизация задач с помощью сценариев bash	154
12.3. Привет, мир!	154
12.4. Использование переменных в собственных сценариях.....	155
12.5. Передача параметров сценарию	156
12.6. Массивы и bash	157
12.7. Циклы	157
12.8. Условные операторы	158
12.9. Функции.....	160
12.10. Примеры сценариев.....	160
12.10.1. Сценарий мониторинга журнала	160
12.10.2. Переименование файлов	161
12.10.3. Преобразование систем счисления	161
12.10.4. Проверка прав пользователя.....	162
12.10.5. Генератор имени временного файла	162
12.10.6. Проверка свободного дискового пространства с уведомлением по электронной почте.....	163
Глава 13. Полезные примеры	164
13.1. Поиск дубликатов файлов.....	164
13.2. Сценарий scanner	165
13.3. Изменение прав доступа к файлам и каталогам.....	166
13.4. Аварийный перезапуск сервисов.....	166
13.4.1. Проверка работоспособности веб-сервера	166
13.4.2. Проверка работоспособности MySQL	167
13.4.3. Если «падают» процессы...	167
13.5. Поиск битых ссылок с помощью AWK	168
13.6. Считаем количество файлов в папке и подпапках	169
13.7. Резервное копирование базы данных.....	169
Предметный указатель	171



ГЛАВА 1

<https://t.me/portalToIT>

Как попасть в командную строку?

1.1. Переключаемся из графического режима в консоль

Настоящий линуксоид должен уметь работать в консоли. Ведь когда система Linux только появилась, существовала только консоль, о графическом интерфейсе не было и речи. Знаете, почему UNIX и Linux отталкивали обычных пользователей? Потому что не было хорошего графического интерфейса. Раньше в Linux работали одни профессионалы. Сейчас все изменилось — в Linux очень удобный графический интерфейс, который с удовольствием используют как новички, так и профессионалы (дождались наконец-то!), забывая о командной строке. Современные дистрибутивы Linux вообще ориентированы на работу в графическом режиме, а в официальных руководствах, которые можно найти в Интернете, о консоли вообще не упоминается. А ведь она есть! И в этой главе мы поговорим о том, как добраться до командной строки в Linux. Совсем необязательно работать полностью в текстовом режиме, вы можете использовать материал этой главы для эффективной работы с терминалом — эмулятором консоли или для удаленного управления сервером, подключившись к нему по SSH.

Обычные пользователи в консоль ни ногой — даже принципиально: мол, зачем возвращаться в DOS? Под DOS они имеют в виду командную строку Linux. Да, вид ее не очень дружелюбный, но это только на первый взгляд. Стоит вам поработать в консоли, и вы поймете все ее преимущества. Начнем с того, что командная строка Linux намного удобнее командной строки DOS, — об этом мы еще поговорим. В консоли можно выполнять те же операции, что и в графическом режиме, причем намного быстрее. Понятно, что, даже несмотря на наличие консольного браузера, вы не будете смотреть в консольном режиме веб-страницы. Сейчас консоль используется в основном для тонкой настройки и администрирования локальных и удаленных машин. Причем для администрирования удаленной машины вам даже не нужен Linux — вы можете работать в привычной для себя Windows, установив какой-либо SSH-клиент (например, PuTTY или Bitvise).

По умолчанию в современных дистрибутивах при входе в систему запускается графический менеджер регистрации (рис. 1.1). Однако из всех правил могут быть

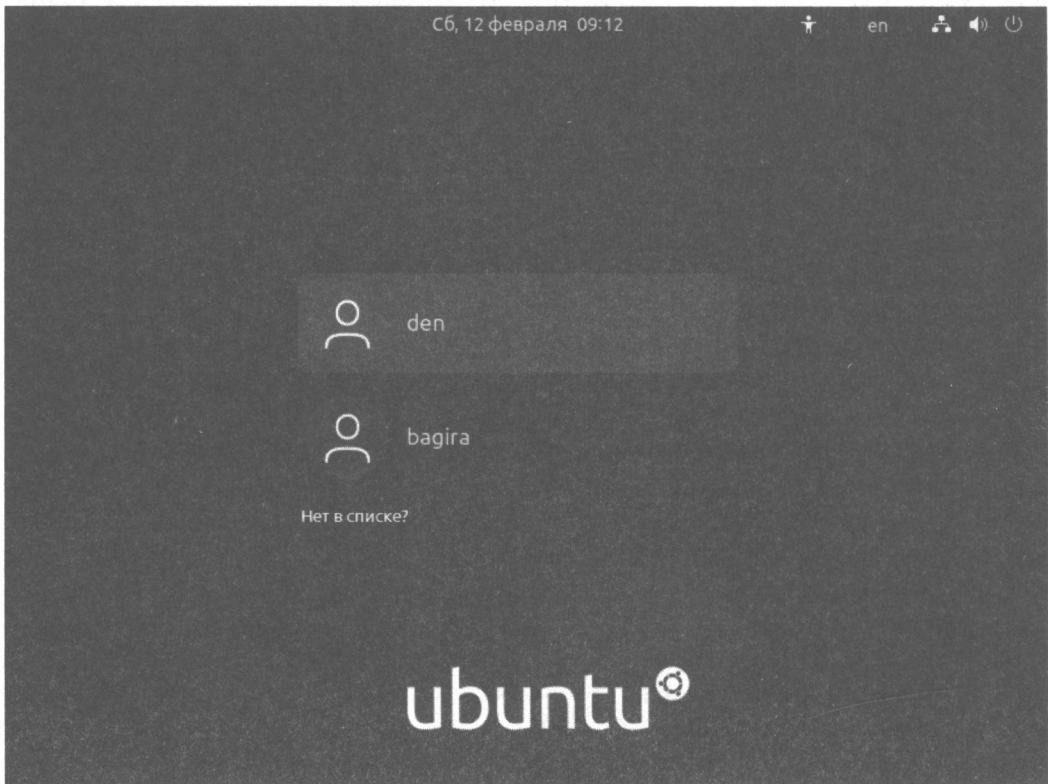


Рис. 1.1. Графический вход в систему (Ubuntu 21.10)

исключения. Пример тому дистрибутив Slackware — в нем сначала нужно выполнить вход в консоли (рис. 1.2), а потом для запуска графического интерфейса ввести команду startx. Тем не менее любой дистрибутив можно настроить на запуск исключительно в консольном режиме — чтобы графика не загружалась. Как правило, так настраиваются серверы — там графический интерфейс не нужен.

Для входа в систему вам потребуется указать имя пользователя и пароль. После этого загрузится графическая среда KDE или GNOME (в зависимости от того, какая графическая среда установлена в вашем дистрибутиве по умолчанию). Конечно, по желанию может быть загружена и какая-либо другая графическая среда, но обычно по умолчанию загружается KDE или GNOME. Для выбора графической среды нужно нажать кнопку **Тип сессии**, **Тип сеанса** или просто **Сеанс** (в Fedora и некоторых других дистрибутивах) — в некоторых дистрибутивах эта кнопка также может быть представлена графической пиктограммой. На рис. 1.3 показан экран входа дистрибутива Astra Linux — в нем команда **Консольный вход** почему-то помещена в меню, открывающееся по нажатии кнопки **Меню**, а не в меню **Тип сессии**.

Итак, сейчас вы находитесь в графическом режиме. Чтобы перейти из графического режима в консоль (см. рис. 1.2), нажмите клавиатурную комбинацию <Ctrl>+<Alt>+<Fn>, где *n* — номер консоли (от 1 до 6). То есть для перехода на первую

```

Polling for DHCP server on interface eth0:
dhpcd: MAC address = 00:0c:29:6f:40:83
Starting Internet super-server daemon: /usr/sbin/inetd
Starting OpenSSH SSH daemon: /usr/sbin/sshd
Starting ACPI daemon: /usr/sbin/acpid
Starting system message bus: /usr/bin/dbus-uuidgen --ensure ; /usr/bin/dbus-daemon --system
Starting HAL daemon: /usr/sbin/hald --daemon=yes
ALSA warning: No mixer settings found in /etc/asound.state.
Sound may be muted. Use 'alsamixer' to unmute your sound card,
and then 'alsactl store' to save the default ALSA Mixer settings
to be loaded at boot.
Loading OSS compatibility modules for ALSA.
Loading /usr/share/kbd/keymaps/i386/qwerty/us.map.gz
Starting gpm: /usr/sbin/gpm -m /dev/mouse -t ps2

Welcome to Linux 2.6.21.5-smp (tty1)

dhsilabs login: root _____ ИМЯ ПОЛЬЗОВАТЕЛЯ
Password: _____ пароль при вводе не отображается
Linux 2.6.21.5-smp.
Last login: Mon Mar  3 13:21:39 +0300 2008 on tty1.
You have mail.
root@dhsilabs:~# 
```

Рис. 1.2. Регистрация в консоли (Slackware)

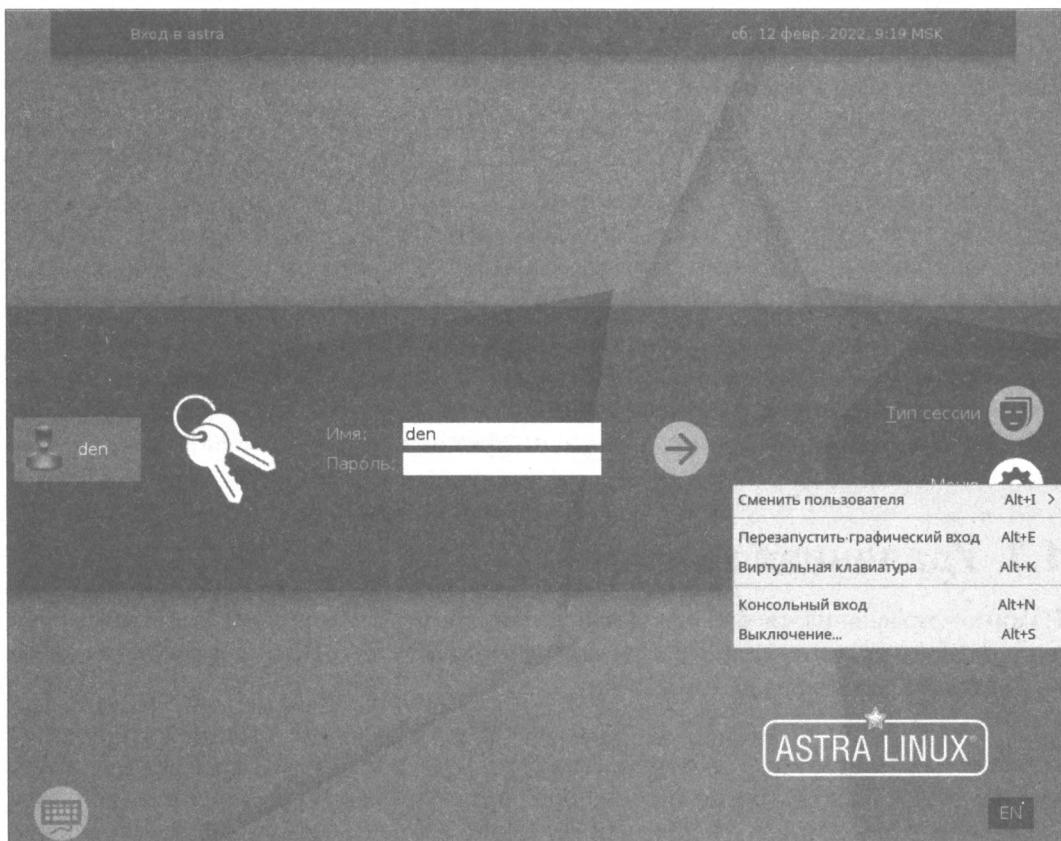


Рис. 1.3. Консольный вход Astra Linux

консоль нужно нажать комбинацию клавиш $<\text{Ctrl}>+<\text{Alt}>+<\text{F}1>$, на вторую — $<\text{Ctrl}>+<\text{Alt}>+<\text{F}2>$ и т. д. Обратите внимание, что так можно перейти в консоль только из графического режима. Если вы уже находитесь в консоли, то для переключения между консолями служат комбинации клавиш от $<\text{Alt}>+<\text{F}1>$ до $<\text{Alt}>+<\text{F}6>$, а также $<\text{Alt}>+<\text{F}7>$ — для перехода в графический режим. Чтобы вы лучше запомнили все эти комбинации, они сведены в табл. 1.1.

Таблица 1.1. Клавиши переключения между консолями и графическим режимом

Комбинация клавиш	Назначение
$<\text{Ctrl}>+<\text{Alt}>+<\text{Fn}>$ (n — от 1 до 6)	Переключение из графического режима в консоль с номером n
$<\text{Alt}>+<\text{Fn}>$ (n — от 1 до 6)	Переключение между консолями
$<\text{Alt}>+<\text{F}7>$	Переключение из консоли в графический режим

1.2. Графические терминалы

Большинство современных дистрибутивов Linux оснащены графическим интерфейсом, который к тому же запускается по умолчанию. Поэтому понятно, что основной контингент пользователей не будет жертвовать удобным и привычным интерфейсом ради консоли.

Однако вместо того, чтобы переключаться в консоль, можно использовать терминалы — эмуляторы консоли. *Терминал* — это графическая программа (рис. 1.4), в окне которой вы можете вводить команды и видеть результат их выполнения. Терминал легко запустить из графического интерфейса — например, в Ubuntu для этого надо щелкнуть правой кнопкой мыши на рабочем столе и выбрать команду **Открыть терминал**.

Работая в современных дистрибутивах Linux, как правило, нет необходимости переключаться в консоль, поскольку в них всегда можно запустить терминал, открыть консоль в окне терминала и работать с ней так же, как бы вы работали в консольном режиме. Использовать терминал или переключаться в консоль — дело вкусовых предпочтений.

1.3. Удаленная консоль

Помимо управления своей локальной машиной, вы можете подключиться к удаленной Linux-системе. При этом все вводимые вами команды будут выполняться не на локальной машине, а на удаленной.

Если вы работаете в Linux, то для подключения к удаленной машине используется команда `ssh` (подробно о ней рассказано в разд. 10.8), а вот если вы работаете в Windows и есть необходимость подключиться к Linux-серверу, тогда придется установить SSH-клиент — например, Bitvise SSH Client, очень удобную и бесплатную программу (рис. 1.5).

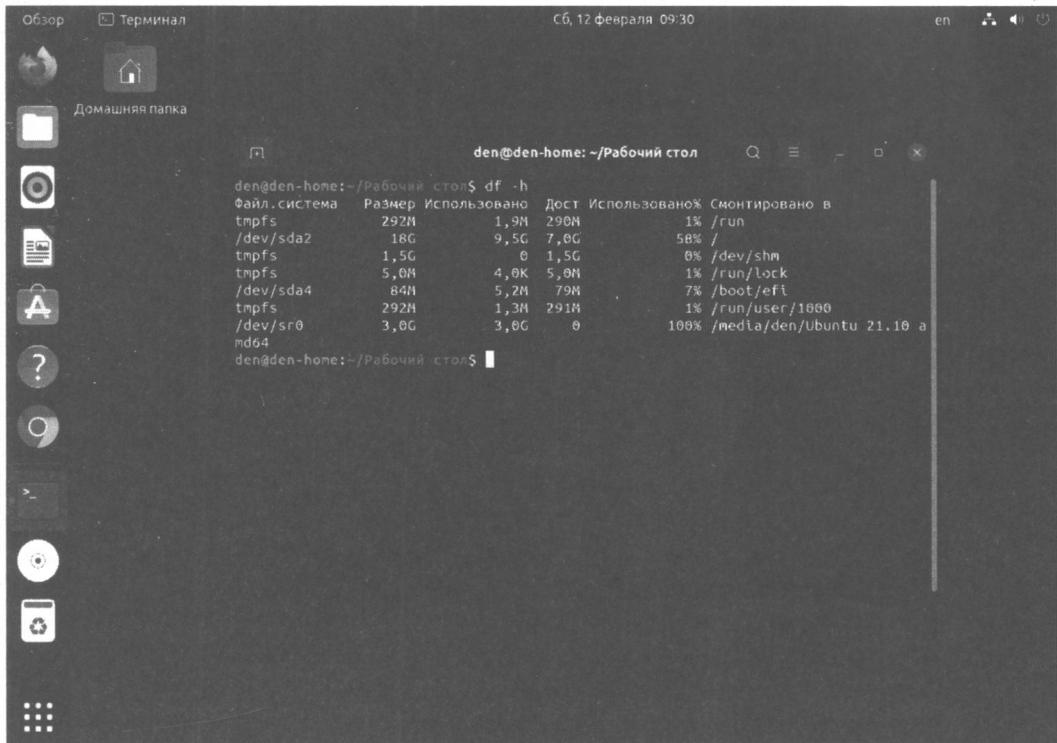


Рис. 1.4. Терминал

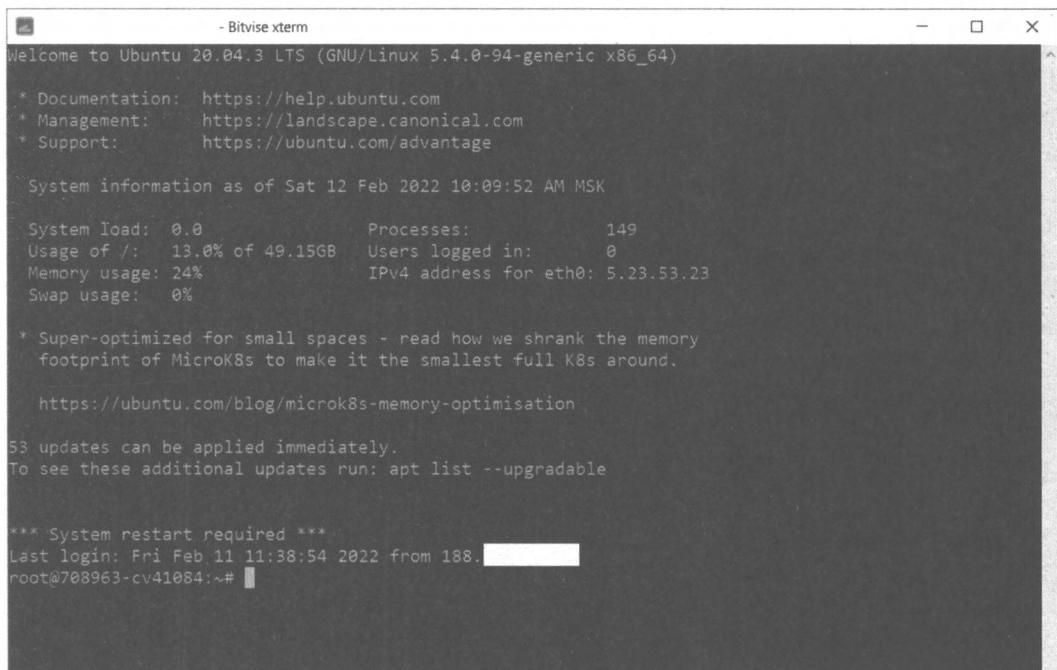


Рис. 1.5. Консоль удаленного сервера

Работая в Linux с удаленной системой, соблюдайте осторожность — очень легко перепутать окна локального и удаленного терминалов (они выглядят одинаково) и вместо того, чтобы вводить команды на удаленной системе, выполнять их на локальной (или наоборот). Поэтому лучше на время работы с удаленной системой закрывать окно локального терминала — так уж точно не перепутаешь.

1.4. Завершение работы из консоли. Команды *poweroff*, *halt*, *reboot*, *shutdown*

Для выхода из консоли (чтобы ею никто не воспользовался во время вашего отсутствия) следует выполнить команду `logout` или ее аналог `exit`.

Перезагрузка компьютера осуществляется командой `reboot`. Кроме нее вы можете использовать еще две команды: `halt` и `poweroff`:

- команда `halt` завершает работу системы, но не выключает питание. Вы увидите сообщение `System is halted`, свидетельствующее о возможности выключения питания. Эта команда предназначена для старых компьютеров, не поддерживающих расширенное управление питанием;
- команда `poweroff` завершает работу системы и выключает ее питание.

Самая «продвинутая» команда — `shutdown`. Она позволяет завершить работу и перезагрузить систему в назначенное время. Предположим, вы хотите уйти пораньше, но компьютер нужно выключить ровно в 19:30 (вдруг некоторые пользователи задержались на работе, а вы выключите сервер, — некрасиво получится). Вот тут-то вам и поможет команда `shutdown`:

```
# shutdown -h 19:30 [сообщение]
```

Пояснение

Здесь и далее решетка (#) означает, что команда должна быть выполнена от имени пользователя `root`. Если перед командой ничего не указано или же указан символ доллара (\$), команду можно выполнить от имени обычного пользователя.

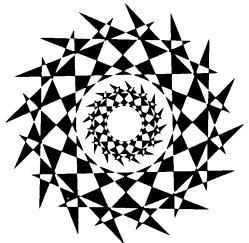
Сообщение [сообщение] можно и не указывать, все равно Windows-пользователи его не увидят.

Если нужно завершить работу системы прямо сейчас, вместо времени укажите `now`:

```
# shutdown -h now
```

Для перезагрузки системы служит опция `-r`:

```
# shutdown -r now
```



ГЛАВА 2

Как работать в консоли?

2.1. Ввод команд

Работа в консоли заключается во вводе нужной команды. Вы вводите команду (например, создания каталога, просмотра файла, вызова редактора и т. д.) и нажимаете клавишу <Enter>. Команда содержит как минимум имя запускаемой программы. Кроме того, команда может содержать параметры, которые будут переданы программе, а также символы перенаправления ввода/вывода (об этом чуть позже). Естественно, вам нужно знать имя программы, а также параметры, которые следует ей передать. Если вы помните название программы, а назначение параметров забыли, вспомнить поможет команда `man`. *Man* (от англ. *manual*) — это справочная система Linux. В ней содержится информация о каждой программе, которая установлена в вашей системе. Как система знает всё обо всех программах? Это очень просто. Разработчики программ под Linux договорились, что вместе с программой будет поставляться специальный *man*-файл — файл справочной системы. Понятно, недобросовестный разработчик может и не создать файл справочной системы, но это происходит очень редко. Чтобы получить справку по какой-либо программе, нужно ввести команду:

```
man имя_программы
```

Вы никак не можете запомнить, как пишется та или иная команда? Если вы помните хотя бы, на какую букву она начинается, то воспользуйтесь функцией *автодополнения* командной строки: введите первые буквы команды и нажмите клавишу <Tab>. При первом нажатии система попытается дополнить команду, если это возможно. Иногда дополнить команду невозможно. Например, вы ввели букву `a` и нажали клавишу <Tab>. Ясное дело, в системе есть несколько команд, которые начинаются на букву «`a`», и система не сможет дополнить командную строку. А если вы хотите просмотреть все команды на букву «`a`», тогда нажмите еще раз клавишу <Tab>.

Вам не с руки писать (даже с автодополнением) длинные команды? — всегда можно создать псевдонимы команд. Для этого в файл `.bashrc` добавьте строки вида:

```
alias псевдоним='команда'
```

Например:

```
alias aprest='systemctl restart apache2'
```

Тогда вместо ввода длинной команды `systemctl restart apache2` вам будет достаточно ввести команду `aprest`.

Чтобы произведенные изменения вступили в силу, выйдите из консоли (командой `logout`) и заново зарегистрируйтесь.

Пожалуй, для полноценной работы с консолью вам нужно знать еще одну команду — `clear`. Она очищает консоль (терминал). Очень полезная команда, особенно когда вы хотите все начать с «чистого листа».

2.2. Ввод/вывод и каналы

Большинство команд Linux читают данные с клавиатуры и записывают вывод на экран. Именно поэтому в Linux есть понятия ввода/вывода и даже предоставляются стандартные имена:

- `stdin` — стандартный ввод. Как правило, это клавиатура. Но при желании вы можете перенаправить стандартный вывод одной команды на ввод другой. Далее будет показано, как это сделать;
- `stdout` — стандартный вывод. Обычно это консоль, т. е. экран монитора (или экран терминала — консольные программы не знают, запущен ли графический интерфейс, и не подозревают, что они выполняются в окне графического терминала). Так что когда вы введете, например, команду `ls`, она произведет печать содержимого каталога на стандартный вывод.

А теперь самое интересное. Вы можете перенаправить стандартный вывод одной команды на стандартный ввод другой. При этом для самих программ ничего не изменяется. Первая программа будет «думать», что отправляет данные на консоль (экран монитора), а вторая — что читает данные с клавиатуры. Представим, что у нас есть текстовый файл `README.txt` с несколькими экранными страницами текста (т. е. все содержимое этого файла не может поместиться на одном экране). Если для его просмотра вы введете команду:

```
cat README.txt
```

то все содержимое файла мгновенно прокрутится и вы увидите лишь последнюю экранную его страницу.

Дабы исправить ситуацию, мы можем перенаправить вывод файла на ввод команды `less`:

```
cat README.txt | less
```

которая обеспечит удобный просмотр — вы сможете с помощью клавиш `<↑>` и `<↓>` пролистывать содержимое файла.

Вертикальная черта (`|`) между двумя простыми командами — это символ **канала** (**pipe**). Каналы используются для перенаправления ввода/вывода — здесь мы задей-

ствовали канал для перенаправления вывода команды `cat README.txt` на ввод команды `less`.

В рассматриваемом случае мы взяли две простые команды (`cat README.txt` и `less`) и сконструировали из них комбинированную команду, объединяющую две простые. Впрочем, никто не говорит: «введите комбинированную команду» — достаточно просто сказать: «введите команду», поэтому все команды — и простые, и комбинированные — мы в дальнейшем будем называть командами.

Однако возможности Linux выходят за рамки простого перенаправления вывода одной команды на ввод другой. Вы можете перенаправить вывод программы в файл. Для этого служат символы `>` и `>>`:

```
ls *.doc > docs.txt  
ls *.zip >> zips.txt
```

Первая команда (с символом `>`) отправляет список doc-файлов в файл `docs.txt`. Если он существует, то будет перезаписан. Если не существует — создан. Вторая команда (с символом `>>`) перенаправляет список zip-файлов в файл `zips.txt`. Если файл существует, то информация будет добавлена в конец файла.

Также есть возможность записывать в файл не весь вывод, а только ошибки — т. е. то, что программа отправляет на стандартный поток ошибок (`stderr`). Для этого нужно использовать конструкции вида `2>` и `2>>`. Работают они аналогично `>` и `>>`, но в результирующий файл попадет не вся информация, выводимая программой, а только то, что программа отправляет на стандартный поток ошибок (`stderr`).

2.3. Шесть полезных команд

По мере чтения этой книги вы уже ознакомились со следующими командами:

- `poweroff`, `halt`, `reboot`, `shutdown` — команды завершения работы (см. разд. 1.4);
- `man` — вызов справочной системы;
- `logout` и `exit` — завершение текущей сессии, выход из оболочки;
- `cat` — вывод содержимого файла;
- `ls` — вывод содержимого каталога;
- `less` — удобное чтение текстовой информации.

В этом разделе мы познакомимся еще с шестью полезными командами: `wc`, `head`, `cut`, `grep`, `sort` и `uniq`. Получить справку по каждой команде можно с помощью команды `man` — например:

```
man sort
```

2.3.1. Команда `wc`

Команда `wc` выводит число строк, слов и символов в файле. Например:

```
$ wc README.txt  
10 82 498 README.txt
```

Этот вывод говорит о том, что файл README.txt содержит 10 строк, 82 слова и 498 символов. С помощью параметров `-l`, `-w` и `-c` мы можем заставить программу выводить соответственно только количество строк, слов и символов. Например, если вам нужно подсчитать количество элементов в каталоге (количество в нем файлов и вложенных каталогов), вы можете ввести команду:

```
ls -1 | wc -l  
10
```

Параметр `-l` команды `ls` заставляет ее выводить содержимое каталога в одну колонку — т. е. так, чтобы одному элементу каталога соответствовала одна строка (по умолчанию программа экономит место на консоли и делает вывод в несколько колонок). Подсчитав количество строк, мы можем узнать количество элементов в каталоге.

Некоторые команды могут определять, куда они выводят данные — на `stdout` (консоль) или же куда-то еще. При этом в зависимости от назначения они могут менять свой вывод. Одна из таких команд — `ls`. Введите просто `ls`, и ее вывод будет разбит на 4–5 колонок (в зависимости от настроек вашего терминала). А вот если ввести команду `ls | cat` — т. е. перенаправить вывод `ls` на команду просмотра файла `cat`, то вывод будет организован в одну строку. Другими словами, для подсчета количества элементов в каталоге нам в этом случае при команде `ls` не понадобится параметр `-l`. Достаточно будет ввести команду `ls | wc -l`, и вы получите количество элементов в каталоге:

```
$ ls | wc -l  
143
```

2.3.2. Команда `head`

Команда `head` выводит первые строки файла. Нужное количество строк можно задать параметром `-n`. Например, для вывода первых пяти строк файла `README.txt` надо выполнить команду:

```
$ head -n5 README.txt
```

По умолчанию (если параметр `-n` не задан) команда выводит первые 10 строк.

Команду `head` можно комбинировать с любыми другими командами — например, вывести первые пять элементов каталога можно так:

```
$ ls | head -n5  
0103  
0207  
02-11-2020  
02-11-21 BoxLeaf  
09-01-nginx backup
```

Существует также команда `tail`, выводящая последние строки, отправленные на ее стандартный ввод (или последние строки файла, если он задан в командной строке). Использовать ее можно аналогично команде `head`.

2.3.3. Команда *cut*

Команда *cut* выводит одну или более колонок из файла. Чтобы указать команде, которую из колонок файла выводить, служит параметр *-f*, задающий номер колонки. Например, если у нас есть файл, внутри которого содержимое организовано в виде колонок, то вывести вторую колонку можно так:

```
$ cut -f2 file.txt
```

Скомбинировав эту команду с командой *head*, мы можем вывести первые пять строк второй колонки:

```
$ cut -f2 file.txt | head -n5
```

Если нужно вывести первую и третью колонки файла, то параметры задаются так:

```
$ cut -f1,3 file.txt | head -n5
```

Можно также указать диапазон выводимых колонок — например, с первой по третью колонки:

```
$ cut -f1-3 file.txt | head -n5
```

По умолчанию в качестве разделителя колонок программа использует символ табуляции. С помощью параметра *-d* можно указать другой разделитель. Например, в файлах формата CSV (Comma-Separated Values) в качестве разделителя применяется запятая. Работать с колонками файла в этом формате можно так:

```
$ cut -f1-3 -d, file.csv | head -n5
```

Здесь в качестве разделителя (параметр *-d*) мы указываем запятую (,).

Надо отметить, что с командой *cut* используется много разных параметров — введите команду *man cut*, чтобы познакомиться с остальными.

2.3.4. Команда *grep*

Команда *grep* — это очень мощная команда-фильтр, позволяющая отфильтровать лишнее. Очень часто команда *grep* используется в качестве второй части комбинированной команды: первая команда генерирует какой-то большой вывод, а команда *grep* выбирает из него то, что нужно нам.

Попробуйте ввести команду *dmesg* — она выводит сообщения, генерируемые ядром при загрузке Linux. Если ввести ее вместе с командой *wc -l*:

```
dmesg | wc -l
```

то вывод составит более 1500 строк. Представим, что из всей этой каши нам нужно получить информацию об устройствах компании Intel, установленных на нашем компьютере. Команда будет такой:

```
$ dmesg | grep Intel
[    0.227390] smtboot: CPU0: Intel(R) Xeon(R) CPU      X5670 @ 2.93GHz
                                         (fam: 06, model: 25, stepping: 01)
[    1.154108] agpgart-intel 0000:00:00.0: Intel 440BX Chipset
```

```
[ 1.490749] e1000: Intel(R) PRO/1000 Network Driver - version 7.3.21-k8-NAPI
[ 1.490751] e1000: Copyright (c) 1999-2006 Intel Corporation.
[ 1.836748] e1000 0000:02:01.0 eth0: Intel(R) PRO/1000 Network Connection
```

Здесь мы видим, что в компьютере установлены процессор Intel Xeon, чипсет Intel 440BX, а также сетевой адаптер Intel PRO/1000 (гигабитный).

Команду `grep` можно использовать не только в канале. Рассмотрим пример. Пусть в файле `cpus.txt` содержится перечень разных процессоров, а мы хотим получить список только процессоров Intel. Тогда следует выполнить команду:

```
$ grep Intel cpus.txt
```

Если, наоборот, нужно получить строки, не содержащие упоминания про Intel, включите в запрос параметр `-v`:

```
$ grep -v Intel cpus.txt
```

2.3.5. Команда `sort`

Команда `sort` служит для сортировки строк файла в алфавитном порядке:

```
$ sort lines.txt
```

Строки также можно отсортировать и в порядке, обратном алфавитному:

```
$ sort -r lines.txt
```

Если нужен не алфавитный, а числовой порядок, используется параметр `-n`. Представим, что в файле `cars.txt` у нас содержится список автомобилей, и вторая колонка этого списка определяет год выпуска. Выведем содержимое второй колонки, отсортированное с помощью команды `sort`:

```
$ cut -f2 cars.txt | sort -n
2007
2010
2014
2022
```

Давайте выведем дату выпуска самого нового автомобиля:

```
$ cut -f2 cars.txt | sort -nr | head -n1
```

Здесь мы выбираем из файла `cars.txt` вторую колонку, отправляем ее на команду `sort`, которая сортирует строки в обратном числовом порядке, а команда `head` выбирает первую строку. То есть в итоге мы должны получить одно значение — год выпуска самого нового автомобиля.

Сохранить отсортированный результат можно с помощью перенаправления полученного вывода в файл:

```
$ sort list.txt > sorted_list.txt
```

Рассмотрим еще один пример. В файле `/etc/passwd` хранится список пользователей компьютера, зарегистрированных в системе. Поля в этом файле разделяются двоеточием:

точием. Имя пользователя — первое поле. Попробуем вывести список пользователей в алфавитном порядке:

```
$ cat /etc/passwd | cut -d: -f1 | sort
lcdev
lctransfer
apache
bin
bitrix
daemon
dbus
developer
devusr2
exim
ftp
games
halt
lp
mail
memcached
munin
mysql
nfsnobody
nginx
nobody
ntp
operator
polkitd
postfix
root
rpc
rpcuser
saslauth
shutdown
sshd
sync
systemd-network
tcpdump
tss
zabbix
```

Здесь надо отметить, что полученный список пользователей взят с реального компьютера, поэтому у вас вывод этой команды будет сильно отличаться.

2.3.6. Команда *uniq*

Команда *uniq* удаляет повторяющиеся строки из файла. Пусть у нас в файле *letters.txt* содержится список букв:

```
$ cat letters.txt
```

```
A  
B  
B  
A  
C  
C  
D  
E  
D  
C  
A
```

Выведем только уникальные буквы:

```
$ uniq letter.txt
```

```
A  
B  
A  
C  
D  
E  
D  
C  
A
```

Однако команда `uniq` работает несколько не так, как мы представляем. Она удаляет лишь те повторяющиеся строки, которые идут непосредственно друг за другом. Именно поэтому были удалены только «лишние» буквы В и С. Параметр `-c` позволяет увидеть количество повторяющихся последовательностей:

```
$ uniq -c letters.txt
```

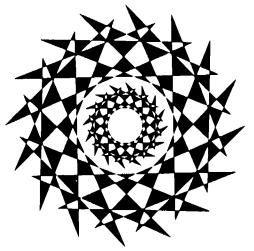
```
1 A  
2 B  
1 A  
2 C  
1 D  
1 E  
1 D  
1 C  
1 A
```

Как же заставить `uniq` работать так, как мы ожидаем? Правильно, с помощью сортировки:

```
$ cat letters.txt | sort | uniq
```

```
A  
B  
C  
D  
E
```

Команда `sort` заставляет одинаковые последовательности следовать друг за другом, а команда `uniq` «подчищает» результат.



ГЛАВА 3

Оболочка bash

Используя в предыдущих главах различные команды консоли, мы начали знакомиться с *командной оболочкой bash*, которая в Linux принимает ваши команды и передает их операционной системе. В этой главе мы продолжим это знакомство и рассмотрим некоторые полезные приемы.

3.1. Маски имен файлов

Как и в других операционных системах, в Linux мы можем также использовать общепринятые маски * и ?. Символ * заменяет любое количество любых символов, а символ ? — один какой-то символ. Например, команда:

```
$ ls *.doc
```

выводит список всех doc-файлов в текущем каталоге (файлов с именами *.doc). А следующая команда выводит список doc-файлов, имена которых состоят из трех символов (любых трех допустимых символов) до точки и расширения doc:

```
$ ls ????.doc
```

Рассмотрим еще одну команду:

```
$ grep Ubuntu ch01.txt ch02.txt ch03.txt .....
```

Эта команда ищет слово Ubuntu в файлах глав книги. Если файлов много и нужно произвести поиск по всем главам, тогда лучше использовать команду:

```
$ grep Ubuntu ch*
```

Оболочка сама развернет список файлов, и реальная команда будет выглядеть так (при условии, что глав 40):

```
$ grep Ubuntu ch01.txt ... ch40.txt
```

Обратите внимание, что такое «разворачивание» делает оболочка, а не команда grep. Команда grep получает уже готовый список файлов.

Идем дальше. Если нужно произвести поиск только в главах с первой по девятую (при условии, что файлы глав носят имена ch01.txt, ch02.txt и т. д.), можно использовать следующую команду:

```
$ grep Ubuntu ch0?.txt
```

А вот если файлы называются `ch1.txt`, `ch2.txt`, ..., `ch9.txt`, `ch10.txt` и т. д., то здесь такая команда не поможет, поскольку их номера не содержат предшествующего нуля. В этом случае поможет вот такая команда:

```
$ grep Ubuntu ch[1-9].txt
```

Если нужны конкретные главы — например, 1, 4, 7 и 9, то их номера можно указать в квадратных скобках:

```
$ grep Ubuntu ch[1479].txt
```

И вообще, если вы знакомы с регулярными выражениями, то заметили, что скобки здесь явно на них похожи. Скажу больше — так и есть. Но в нашем случае это не регулярные выражения, а *шаблоны*. Да, по сути, шаблоны являются ограниченной версией регулярных выражений, у которых гораздо больше возможностей.

3.2. Переменные окружения

Оболочка `bash` позволяет использовать *переменные*. Существуют стандартные переменные, они же *переменные окружения*, вроде `HOME` — содержит путь к домашнему каталогу пользователя и `USER` — имя пользователя (см. также разд. 12.4). Просмотреть содержимое этих переменных можно так:

```
$ printenv HOME  
/home/evg  
$ printenv USER  
evg
```

Когда вам нужно использовать значения переменных в выводе, то следует предварять имя переменной знаком `$`:

```
echo Пользователь $USER Домашний каталог - $HOME  
Пользователь evg Домашний каталог - /home/evg
```

Создать собственные переменные можно так:

```
$ projects=$HOME/Projects  
$ cd $projects  
$ pwd  
/home/evg/Projects
```

Обратите внимание, что между присваиваемым значением и знаком равенства не должно быть пробелов. Например, в этом случае:

```
$ projects = $HOME/Projects  
projects: command not found
```

оболочка воспримет `projects` как команду, а не как попытку присвоить значение переменной.

Еще один пример:

```
work=$HOME/Projects  
cd $work
```

```
pwd  
/home/evg/Projects
```

Здесь мы создали переменную `work`, которая использует значение переменной `$HOME`. Обратите внимание: когда мы определяем значение переменной, знак `$` нам не нужен. Но он обязательен, когда нам нужно использовать значение переменной.

Установить переменную окружения можно с помощью команды `export`, что будет показано в разд. 12.4.

3.3. Псевдонимы

Псевдонимы используются для сокращения команд. Вам не нужно писать сценарии оболочки, если требуется лишь сократить ввод одной команды. Для этого достаточно определить псевдоним с помощью команды `alias`:

```
alias h='fc -l'  
alias ll='ls -laFo'  
alias l='ls -l'  
alias g='egrep -i'  
alias cfiles='find . -type f | wc -l'
```

Последний в этом списке псевдоним определяет команду, подсчитывающую количество файлов в текущем каталоге и всех его подкаталогах.

Псевдонимы работают просто: при вводе команды `l` на самом деле будет выполнена команда `ls -l`.

Чтобы псевдонимы не исчезли после выхода из системы, их нужно определить в файле `.bashrc` — просто добавьте в него введенные ранее команды `alias`, и они станут доступными при повторном входе в систему.

3.4. Изменение приглашения командной строки

За внешний вид командной строки отвечает глобальная переменная `PS1`. По умолчанию командная строка имеет формат:

```
пользователь@компьютер:рабочий_каталог
```

Значение `PS1` при этом будет таким:

```
PS1='\u@\h:\w\$'
```

В табл. 3.1 приведены допустимые модификаторы командной строки.

Вот вариант альтернативного приглашения командной строки:

```
PS1='[\u@\h] $(date +%m/%d/%y) \$'
```

В квадратных скобках выводится имя пользователя и имя компьютера, затем используется конструкция `$()` для подстановки даты в нужном нам формате и символ приглашения, который изменяется в зависимости от идентификатора пользователя.

Вид приглашения при этом будет таким:

```
[ubuntu@host] 12/06/21 $
```

Таблица 3.1. Модификаторы командной строки

Модификатор	Описание
\a	ASCII-символ звонка (код 07). Не рекомендуется его использовать — очень скоро начнет раздражать
\d	Дата в формате "день недели, месяц, число"
\h	Имя компьютера до первой точки
\H	Полное имя компьютера
\j	Количество задач, запущенных в оболочке в текущее время
\l	Название терминала
\n	Символ новой строки
\r	Возврат каретки
\s	Название оболочки
\t	Время в 24-часовом формате (ЧЧ:ММ:СС)
\T	Время в 12-часовом формате (ЧЧ:ММ:СС)
\@	Время в 12-часовом формате (AM/PM)
\u	Имя пользователя
\v	Версия bash (сокращенный вариант)
\V	Версия bash (полная версия: номер релиза, номер патча)
\w	Текущий каталог (полный путь)
\W	Текущий каталог (только название каталога, без пути)
\!	Номер команды в истории
\#	Системный номер команды
\\$	Если UID пользователя равен 0, будет выведен символ #, иначе — символ \$
\`	Обратный слеш
\$ ()	Подстановка внешней команды

3.5. Еще раз о перенаправлении ввода/вывода

Перенаправление ввода/вывода — это очень важный механизм в Linux, поэтому рассмотрим еще некоторые его примеры. Ранее мы уже познакомились с символами >, >> и |.

Вернемся к символу >. Он служит для перенаправления стандартного вывода в файл. Например, если вы хотите найти в файле report.txt все строки, в которых есть слово Bug, можно воспользоваться командой:

```
grep Bug report.txt
```

Чтобы перенаправить этот вывод в файл bug_report.txt мы можем использовать перенаправление ввода/вывода:

```
grep Bug report.txt > bug_report.txt
```

Если файл bug_report.txt уже существует, он будет перезаписан. Не хотите потерять его содержимое? Тогда используйте символ >>:

```
grep Bug report.txt >> bug_report.txt
```

Вывод команды grep будет добавлен в конец файла.

С этим вроде бы все понятно. Но кроме символа > есть также и символ <. Многие команды могут принимать имена файлов, указанные в качестве аргументов, а также читать данные со стандартного ввода. Представим, что нам нужно подсчитать количество строк, слов и символов в файле report.txt:

```
wc report.txt  
10 61 485 report.txt
```

В этом случае команда wc принимает имя report.txt в качестве аргумента и сообщает, что в файле report.txt содержится 10 строк, 61 слово и 485 символов.

Рассмотрим другую команду:

```
wc < report.txt  
10 61 485
```

Здесь команда wc принимает содержимое файла report.txt со стандартного ввода. Обратите внимание на вывод команды — те же 10 строк, 61 слово и 485 символов, но при этом не выводится имя файла, поскольку команда wc получила данные со стандартного вывода.

Если бы мы не знали о существовании <, то эту команду можно было бы записать иначе:

```
cat report.txt | wc
```

Результат был бы таким же, но при этом сама команда сложнее, поскольку мы вовлекаем в нее еще и cat — для вывода содержимого файла. А этого делать не нужно, поскольку благодаря < за нас это сделает система.

Более того, мы можем комбинировать < и > в одной команде:

```
wc < report.txt > report.cnt  
cat report.cnt  
10 61 485
```

Первая команда вычисляет количество строк, слов и символов, полученных со стандартного ввода, и перенаправляет результат в файл report.cnt.

С помощью конструкции `>` мы можем перенаправить в файл возникшие ошибки. Например, мы вводим какую-то команду, в результате которой возникают ошибки. Все эти ошибки (весь вывод, который программа отправляет на `stderr`) мы можем записать в файл:

```
some_command > errors
```

Аналогично — в файл, если он существует, мы можем дописать ошибки, воспользовавшись конструкцией `>>`:

```
some_command >> errors
```

Если вам нужно перенаправить в какой-либо файл и обычный вывод (который программа генерирует на `stdout`), и ошибки, используйте эту конструкцию с амперсандом `&`:

```
some_command &>> errors
```

3.6. Пробелы в именах файлов

Представим, что у нас есть файл `Command Line.txt` и мы хотим просмотреть его содержимое:

```
cat Command Line.txt
```

Мы получим вывод:

```
cat: Command: No such file or directory
cat: Line.txt: No such file or directory
```

А все потому, что фрагменты имени файла `Command` и `Line.txt` команда `cat` считает двумя разными файлами.

Чтобы избежать подобной проблемы, заключайте имя файла в одинарные или двойные кавычки:

```
cat 'Command Line.txt'
cat "Command Line.txt"
```

Кроме этого, одинарные кавычки могут использоваться для запрещения разрешения имен переменной:

```
echo 'Мой каталог $HOME'
Мой каталог $HOME
echo "Мой каталог $HOME"
Мой каталог /home/evg
```

При использовании двойных кавычек значение переменной `$HOME` будет вычислено, а при использовании одинарных символы `$HOME` будут трактоваться как есть.

Если в двойных кавычках вам нужно запретить интерполирование некоторых переменных, тогда используйте символ `\`:

```
echo "Значение переменной \$HOME - $HOME"
Значение переменной $HOME - /home/evg
```

3.7. Инициализация переменных

Если некоторые переменные вам нужны на постоянной основе, тогда есть смысл определить их в файле `~/.bashrc`, чтобы они стали доступны при повторном входе в систему. Откройте этот файл и просто определите нужные вам переменные. Например:

```
EDITOR=/usr/bin/nano
```

Переменная `EDITOR` очень важна. Она задает текстовый редактор по умолчанию, который будет использоваться в различных системных программах вроде `visudo`.

Во многих дистрибутивах используется неудобный редактор `vi`, и все пользователи хотят сменить его на что-то более удобное. Сделать это можно с помощью переменной `EDITOR`, как показано в приведенном примере. Просто укажите в качестве ее значения путь к редактору, который вам нравится, — например, к `nano`. Если вы не знаете полный путь к нему, используйте команду `which`:

```
$ which nano
```

3.8. Просмотр истории команд

Оболочка `bash` ведет историю команд. Для перемещения по истории команд используйте клавиши со стрелками `<↑>` и `<↓>`. Если же вы хотите просмотреть сразу всю историю команд, введите команду `history`:

```
history
```

Журнал команд может содержать тысячи команд, поэтому вы можете задать количество последних команд, которые нужно показать:

```
$ history 3
$ find . -type f | wc -l
$ chmod 777 cache
$ chown +x scanner.sh
```

Вывод `history` можно перенаправить на любую другую команду, например:

```
$ history | less
$ history | grep find
```

Первая команда служит для удобного постраничного просмотра всей истории, а вторая — для вывода всех команд `find`, которые вводил пользователь.

Для очистки истории команд используется параметр `-c`:

```
$ history -c
```

Сколько команд может хранить история? Ответ на этот вопрос хранится в переменной `HISTSIZE`:

```
echo $HISTSIZE
1000
```

Установить значение этой переменной можно так:

```
HISTSIZE=100
```

Посредством переменной окружения HISTCONTROL можно запретить внесение в историю команд-дубликатов:

```
HISTCONTROL=ignoredups
```

Переменная HISTFILE позволяет узнать имя файла, в котором хранится история команд:

```
$ echo $HISTFILE  
/home/evg/.bash_history
```

3.9. Вызов предыдущих команд

Чтобы повторить одну из предыдущих команд, нажмите несколько раз стрелку <↑> — пока не увидите в командной строке нужную команду, а потом нажмите клавишу <Enter>. Вы также можете использовать команду !! — она повторит предыдущую команду:

```
$ wc report.txt  
10 62 532 report.txt  
$ !!  
wc report.txt  
10 62 532 report.txt
```

С помощью конструкции ! команда можно ввести последнюю команду. Например:

```
$ !wc
```

Эта команда запустит последнюю команду wc:

```
wc report.txt
```

Конструкция !номер позволяет запустить команду с указанным номером из истории. Например,

```
$ history | grep wc  
100 wc report.txt  
101 wc < report.txt > report.cnt  
$ !101  
wc < report.txt > report.cnt
```

Здесь команда !101 повторит команду с номером 101 из истории команд.

3.10. Защита от случайного удаления файла

Посредством механизма псевдонимов мы можем задать для команды rm параметр -i, что позволит нам защититься от случайного удаления файла, — команда rm станет запрашивать подтверждение удаления файла:

```
$ alias rm="rm -i"
$ rm *.txt
/bin/rm: remove regular file '1.txt'? y
/bin/rm: remove regular file '2.txt'? y
```

3.11. Использование последнего вывода команды

Представим, что вы ввели команду:

```
$ ls *.bk *.bak *.log
```

и получили список файлов:

```
scan.bk config.back scan.log
```

Вы его просмотрели и хотите теперь эти файлы удалить. Чтобы передать вывод команды `ls` команде `rm` даже не обязательно использовать перенаправление ввода/вывода. Мы можем передать последний вывод какой-либо команды на команду `rm` с помощью конструкции `!*`:

```
$ rm !*
```

3.12. Редактирование командной строки

Ранее было сказано, что перемещаться по истории команд удобно с помощью клавиш-стрелок `<↑>` и `<↓>`. А вот клавишами-стрелками `<←>` и `<→>` вы можете перемещаться по символам самой команды. Таким способом при необходимости можно исправить в команде ошибки и снова запустить ее на выполнение нажатием клавиши `<Enter>`.

Также вам нужно знать, что существуют два стиля редактирования командной строки: стиль `Emacs` и стиль `Vi`. Выбор режима осуществляется так:

```
set -o vi
set -o emacs
```

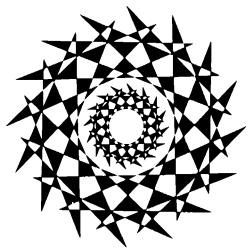
Данные, приведенные в табл. 3.2, описывают разницу между этими стилями, и вы можете выбрать тот, который вам больше нравится. Для новичков рекомендуется использовать стиль `Emacs` как более простой. Стиль `Vi` пригодится «динозаврам», которые используют `Linux` последние 30 лет.

Таблица 3.2. Сравнение стилей редактирования командной строки

Действие	Emacs	Vi
Переместить курсор вперед на один символ	<code><Ctrl>+<F></code>	<code><h></code>
Переместиться назад на один символ	<code>Ctrl+</code>	<code><I></code>
Переместиться вперед на одно слово	<code><Alt>+<F></code>	<code><W></code>

Таблица 3.2 (окончание)

Действие	Emacs	Vi
Переместиться назад на одно слово	<Alt>+	
Переместиться к началу строки	<Ctrl>+<A>	<0>
Переместиться к концу строки	<Ctrl>+<E>	<\$>
Поменять местами два символа	<Ctrl>+<T>	<Ctrl>+<R>
Поменять местами два слова	<Alt>+<T>	—
Капитализировать следующее слово (первая буква — в верхнем регистре)	<Alt>+<C>	—
Перевести в верхний регистр следующее слово	<Alt>+<U>	—
Перевести в нижний регистр следующее слово	<Alt>+<l>	—
Изменить регистр текущего символа	—	<~>
Удалить один символ спереди	<Ctrl>+<D>	<x>
Удалить один символ сзади	<Backspace> или <Ctrl>+<H>	<X>
Вырезать следующее слово	<Alt>+<D>	—
Вырезать предыдущее слово	<Alt>+<Backspace> или <Ctrl>+<W>	<Ctrl>+<W>
Вырезать текст от текущей позиции к началу строки	<Ctrl>+<U>	<Ctrl>+<U>
Вырезать текст от текущей позиции до конца строки	<Ctrl>+<k>	<D>
Удалить всю строку	<Ctrl>+<e> или <Ctrl>+<u>	<dd>
Вставить недавно удаленный текст	<Ctrl>+<y>	<p>
Отменить предыдущую операцию редактирования	<Ctrl>+<_>	<u>
Удалить все сделанные изменения	<Alt>+<r>	<U>
Очистить экран	<Ctrl>+<l>	<Ctrl>+<l>



ГЛАВА 4

<https://t.me/portalToIT>

Файловая система. Команды для работы с файловой системой

4.1. Особенности файловой системы Linux

4.1.1. Имена файлов в Linux

По сравнению с Windows в Linux несколько иные правила построения имен файлов, и вам придется с этим смириться. Начнем с того, что в Linux нет такого понятия, как *расширение имени файла*. В Windows, например, для файла Document1.doc именем файла является фрагмент Document1, а doc — это расширение. В Linux Document1.doc — это просто имя файла, и ни о каком расширении речи нет.

Максимальная длина имени файла в Linux — 254 символа. Имя может содержать любые символы (в том числе и кириллицу), кроме / \ ? < > * " |. Но кириллицу в именах файлов я бы не рекомендовал вообще. Впрочем, если вы уверены, что не будете эти файлы передавать Windows-пользователям (на флешке, по электронной почте), — используйте на здоровье. Но при обмене файлами по электронной почте (кодировка-то у всех разная, поэтому вместо русскоязычного имени пользователь увидит абракадабру) имя файла лучше писать латиницей.

Также вам придется привыкнуть к тому, что система Linux чувствительна к регистру в имени файла: FILE.txt и FiLe.Txt — это два разных файла.

Разделение элементов пути в Linux осуществляется символом / (прямой слеш), а не \ (обратный слеш), как в Windows.

4.1.2. Файлы и устройства

Пользователи Windows привыкли к тому, что файл — это именованная область данных на диске. Отчасти так оно и есть. Отчасти — потому, что приведенное определение файла было верно для DOS (Disk Operating System) и верно для Windows.

В Linux же понятие файла значительно шире. Сейчас Windows-пользователи будут очень удивлены: в Linux есть файлы устройств, позволяющие обращаться с устрой-

ством как с обычным файлом. Файлы устройств находятся в каталоге `/dev` (от англ. *devices*). Да, через файл устройства мы можем обратиться к устройству! Если вы работали в DOS, то, наверное, помните, что нечто подобное было и там — существовали зарезервированные имена файлов: PRN (принтер), CON (клавиатура при вводе, дисплей при выводе), LPT n (параллельный порт, n — номер порта), COM n (последовательный порт).

ПРИМЕЧАНИЕ

Кому-то может показаться, что разработчики Linux «украли» идею специальных файлов у Microsoft — ведь Linux появилась в начале 90-х годов, а DOS — в начале 80-х годов прошлого века. На самом деле это не так. Наоборот, Microsoft позаимствовала идею файлов устройств из операционной системы UNIX, которая была создана еще до возникновения DOS. Однако сейчас не время говорить об истории развития операционных систем, поэтому лучше вернемся к файлам устройств.

Вот самые распространенные примеры файлов устройств:

- `/dev/sdx` — файл жесткого диска или USB-накопителя;
- `/dev/sdXN` — файл устройства раздела на жестком диске, где N — это номер раздела;
- `/dev/mouse` — файл устройства мыши;
- `/dev/modem` — файл устройства модема (на самом деле является ссылкой на файл устройства `ttySn`);
- `/dev/ttySn` — файл последовательного порта, где n — номер порта (`ttyS0` соответствует COM1, `ttyS1` — COM2 и т. д.).

ПРИМЕЧАНИЕ

В современных дистрибутивах имена вида `/dev/hdx` уже не используются (см. далее).

В свою очередь, файлы устройств бывают двух типов: блочные и символьные. Обмен информации с блочными устройствами — например, с жестким диском, осуществляется блоками информации, а с символьными — отдельными символами. Пример символьного устройства — последовательный порт.

4.1.3. Корневая файловая система и монтирование

Наверняка на вашем компьютере установлена система Windows. Откройте Проводник (рис. 4.1).

Скорее всего, вы увидите пиктограммы разделов жесткого диска (пусть у вас там будут два раздела: C: и E:) и пиктограмму привода CD/DVD (D:). Буквы A: и B: вы увидите вряд ли — они зарезервированы для накопителей на гибких дисках, которые уже давно не используются.

Таким образом, с помощью буквенных обозначений C:, D: и т. д. в Windows обозначаются корневые каталоги разделов жесткого диска и сменных носителей.

В Linux существует понятие *корневой файловой системы*. Допустим, вы установили Linux в раздел с именем `/dev/sda3`. В этом разделе и будет развернута корневая

файловая система вашей Linux-системы. Корневой каталог обозначается прямым слешем (/), т. е. для перехода в корневой каталог в терминале (или консоли) нужно ввести команду `cd /`.

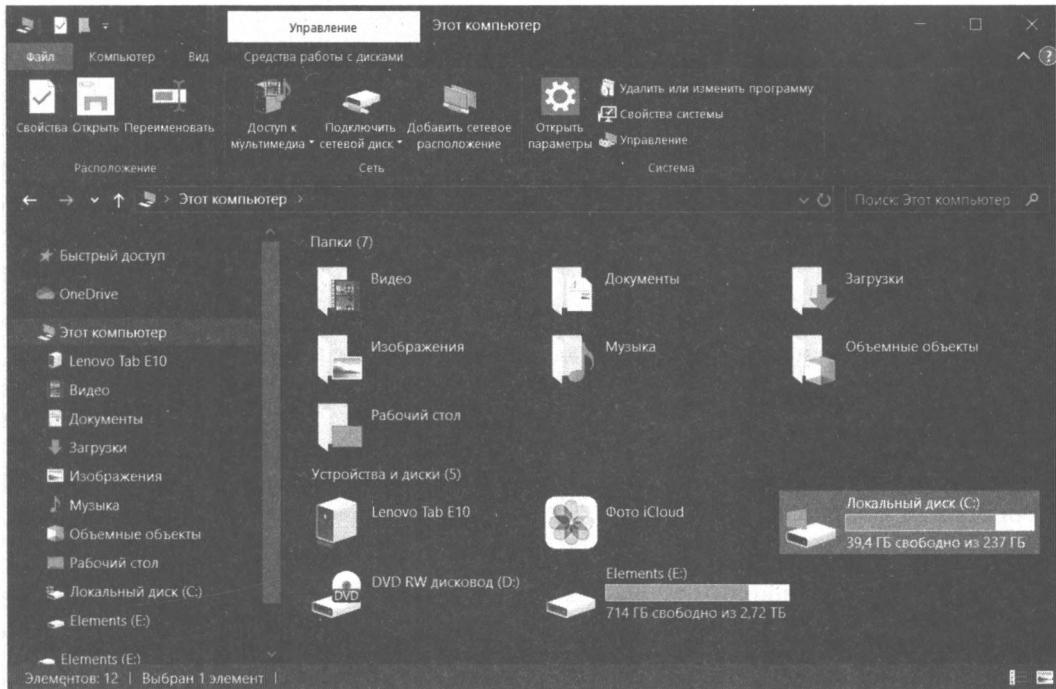


Рис. 4.1. Окно Компьютер

Понятно, что на вашем жестком диске есть еще разделы. Чтобы получить доступ к этим разделам, вам нужно *подмонтировать* их к корневой файловой системе. После монтирования вы можете обратиться к содержимому разделов через *точку монтирования* — назначенный вами при монтировании специальный каталог, например `/mnt/usb`. Монтированию файловых систем посвящен *разд. 4.6*, поэтому сейчас мы не будем говорить об этом процессе подробно.

4.1.4. Стандартные каталоги Linux

Файловая система любого дистрибутива Linux содержит следующие каталоги:

- / — корневой каталог;
- /bin — содержит стандартные программы (команды) Linux (`cat`, `cp`, `ls`, `login` и т. д.);
- /boot — каталог загрузчика, содержит образы ядра и `Initrd` (RAM-диска инициализации), может содержать конфигурационные и вспомогательные файлы загрузчика;
- /dev — содержит файлы устройств;
- /etc — содержит конфигурационные файлы системы;

- `/home` — содержит домашние каталоги пользователей;
- `/lib` — содержит библиотеки и модули;
- `/lost+found` — содержит восстановленные после некорректного размонтирования файловой системы файлы и каталоги;
- `/media` — в современных дистрибутивах содержит точки монтирования сменных носителей (CD-, DVD-, USB-накопителей);
- `/misc` — может содержать все что угодно, равно как и каталог `/opt`;
- `/mnt` — обычно содержит точки монтирования;
- `/proc` — каталог псевдофайловой системы `procfs`, предоставляющей информацию о процессах;
- `/root` — каталог суперпользователя `root`;
- `/sbin` — каталог системных утилит, выполнять которые имеет право пользователь `root`;
- `/tmp` — каталог для временных файлов;
- `/usr` — содержит пользовательские программы, документацию, исходные коды программ и ядра;
- `/var` — постоянно изменяющиеся данные системы. Например, очереди системы печати, почтовые ящики, протоколы, замки и т. д.

В зависимости от дистрибутива в корневом каталоге могут находиться дополнительные каталоги (либо, наоборот, отсутствовать некоторые каталоги, приведенные в списке). Например, в Debian и Ubuntu обязательно есть каталог `/opt` (ранее он служил для установки альтернативного программного обеспечения, сейчас его можно использовать для чего угодно), кроме того, в Ubuntu есть каталог `/cdrom` (который по непонятным мне причинам не используется).

4.2. Команды для работы с файлами и каталогами

4.2.1. Работа с файлами

Здесь мы рассмотрим основные команды для работы с файлами в Linux (табл. 4.1), а в последующих разделах этой главы — команды для работы с каталогами, ссылками и поговорим о правах доступа к файлам и каталогам.

Таблица 4.1. Основные команды Linux, предназначенные для работы с файлами

Команда	Назначение
<code>touch <файл></code>	Создание пустого файла
<code>cat <файл></code>	Просмотр текстового файла

Таблица 4.1 (окончание)

Команда	Назначение
tac <файл>	Вывод содержимого текстового файла в обратном порядке, т. е. сначала выводится последняя строка, потом предпоследняя и т. д.
cp <файл1> <файл2>	Копирование файла <файл1> в файл <файл2>. Если <файл2> существует, программа попросит разрешение на его перезапись
mv <файл1> <файл2>	Перемещение файла <файл1> в файл <файл2>. Этую же команду можно использовать и для переименования файла
rm <файл>	Удаление файла
locate <файл>	Быстрый поиск файла
which <программа>	Вывод каталога, в котором находится программа, если она вообще установлена. Поиск производится в каталогах, указанных в переменной окружения PATH (это путь поиска программ)
less <файл>	Используется для удобного просмотра файла с возможностью скроллинга (постстраничной прокрутки)

ПРИМЕЧАНИЕ

Все представленные команды предназначены для работы в консоли, т. е. в текстовом режиме. Понятно, что большинство современных дистрибутивов запускается в графическом режиме, поэтому некоторые пользователи Linux даже не подозревают о том, что существует консоль. Да, таково новое поколение Linux-пользователей, которым проще использовать графический файловый менеджер, чем вводить команды. Но если вы хотите стать квалифицированным пользователем Linux, то просто обязаны знать, как работать в консоли, иначе уподобитесь Windows-пользователям, которые при каждом сбое переустанавливают операционную систему... Если вы пропустили главы 1–3, в которых рассматривается работа с консолью, настоятельно рекомендую прочитать их!

Рассмотрим небольшую серию команд (протокол выполнения этих команд приведен на рис. 4.2):

```
touch file.txt
echo "some text" > file.txt
cat file.txt
cp file.txt file-copy.txt
cat file-copy.txt
rm file.txt
cat file.txt
mv file-copy.txt file.txt
cat file.txt
```

Первая команда (`touch`) создает в текущем каталоге файл `file.txt`. Вторая команда (`echo`) записывает строку `some text` в этот же файл. Обратите внимание на символ `>` — это символ перенаправления ввода/вывода, о котором мы поговорим чуть позже.

Третья команда (`cat`) выводит содержимое файла — в файле это записанная нами строка `some text`. Четвертая команда (`cp`) копирует файл `file.txt` в файл с именем `file-`

```
[root@localhost ~]# touch file.txt
[root@localhost ~]# echo "some text" > file.txt
[root@localhost ~]# cat file.txt
some text
[root@localhost ~]# cp file.txt file-copy.txt
[root@localhost ~]# cat file-copy.txt
some text
[root@localhost ~]# rm file.txt
rm: удалить обычный файл `file.txt'? y
[root@localhost ~]# cat file.txt
cat: file.txt: No such file or directory
[root@localhost ~]# mv file-copy.txt file.txt
[root@localhost ~]# cat file.txt
some text
[root@localhost ~]# █
```

Рис. 4.2. Операции с файлом

copy.txt. После этого мы опять используем команду cat, чтобы вывести содержимое файла file-copy.txt, — надо же убедиться, что файл действительно скопировался.

Шестая команда (`rm`) удаляет файл file.txt. При удалении система спрашивает, хотите ли вы удалить файл. Если хотите удалить, то нужно нажать клавишу `<Y>`, а если нет, то `<N>`. Точно ли файл удален? Убедимся в этом: введите команду `cat file.txt`. Система нам сообщает, что нет такого файла.

Восьмая команда (`mv`) переименовывает файл file-copy.txt в файл file.txt. Последняя команда выводит исходный файл file.txt.

Думаю, особых проблем с этими командами у вас не возникло, тем более что принцип их действия вам должен быть знаком по командам DOS, которые, как квалифицированный пользователь Windows, вы должны знать наизусть.

Вместо имени файла иногда очень удобно указать *маску имени файла*. Например, у нас есть много временных файлов, имена которых заканчиваются фрагментом tmp. Для их удаления нужно воспользоваться командой `rm *tmp`.

Если же требуется удалить все файлы в текущем каталоге, можно просто указать звездочку: `rm *`.

Аналогично можно использовать символ ?, который, в отличие от звездочки, заменяющей последовательность символов произвольной длины, заменяет всего один символ. Например, нам нужно удалить все файлы, имена которых состоят из трех букв и начинаются на s:

```
rm s??
```

Будут удалены файлы s14, sqm, sr6 и т. д., но не будут затронуты файлы, имена которых состоят более чем из трех букв и которые не начинаются на s.

Маски имен можно также использовать и при работе с каталогами.

4.2.2. Работа с каталогами

Основные команды для работы с каталогами приведены в табл. 4.2.

Таблица 4.2. Основные команды для работы с каталогами

Команда	Описание
<code>mkdir <каталог></code>	Создание каталога
<code>cd <каталог></code>	Изменение каталога
<code>ls <каталог></code>	Вывод содержимого каталога
<code>rmdir <каталог></code>	Удаление пустого каталога
<code>rm -r <каталог></code>	Рекурсивное удаление каталога

При указании имени каталога можно использовать следующие символы:

- . — означает текущий каталог. Если вы введете команду `cat ./file`, то она выведет файл `file`, который находится в текущем каталоге;
- .. — родительский каталог. Например, команда `cd ..` переведет вас на один уровень вверх по дереву файловой системы;
- ~ — домашний каталог пользователя (об этом мы поговорим позже).

Теперь рассмотрим пример работы с каталогами на практике. Выполните следующие команды:

```
mkdir directory
cd directory
touch file1.txt
touch file2.txt
ls
cd ..
ls directory
rm directory
rmdir directory
rm -r directory
```

Первая команда (`mkdir`) создает каталог `directory` в текущем каталоге. Вторая команда (`cd`) переводит (изменяет каталог) в только что созданный каталог. Следующие две команды `touch` создают в новом каталоге два файла: `file1.txt` и `file2.txt`.

Команда `ls` без указания каталога выводит содержимое текущего каталога. Команда `cd ..` переводит в родительский каталог. Как уже было отмечено, в Linux родительский каталог обозначается так: .. (две точки), а текущий так: . (одна точка). То есть, находясь в каталоге `directory`, мы можем обращаться к файлам `file1.txt` и `file2.txt` без указания каталога или же так: `./file1.txt` и `./file2.txt`.

ВНИМАНИЕ!

Еще раз напомню: в Linux, в отличие от Windows, для разделения элементов пути используется прямой слеш (/), а не обратный (\)!

Кроме обозначений `..` и `.` в Linux часто используется обозначение «тильда» (`~`) — это *домашний каталог*. Предположим, что наш домашний каталог — `/home/evg`. В нем мы создали подкаталог `dir` и поместили в него файл `file1.txt`. Полный путь к файлу можно записать так:

```
/home/evg/dir/file1.txt
```

или же так:

```
~/dir/file1.txt
```

Как видите, тильда (`~`) заменяет часть пути. Удобно? Конечно!

Поскольку мы находимся в родительском для каталога `directory` каталоге, чтобы вывести содержимое только что созданного каталога, в команде `ls` нам нужно четко указать имя каталога:

```
ls directory
```

```
[root@localhost ~]# mkdir directory
[root@localhost ~]# cd directory
[root@localhost directory]# touch file.txt
[root@localhost directory]# touch file2.txt
[root@localhost directory]# ls
file2.txt  file.txt
[root@localhost directory]# cd ..
[root@localhost ~]# ls directory
file2.txt  file.txt
[root@localhost ~]# rm directory
rm: невозможно удалить каталог `directory': Is a directory
[root@localhost ~]# rmdir directory
rmdir: `directory': Directory not empty
[root@localhost ~]# rm -r directory
rm: спуститься в каталог `directory'? y
rm: удалить пустой обычный файл `directory/file.txt'? y
rm: удалить пустой обычный файл `directory/file2.txt'? y
rm: удалить Каталог `directory'? y
[root@localhost ~]#
```

Рис. 4.3. Операции с каталогами

Команда `rm` служит для удаления каталога. Но что мы видим: система отказывается удалять каталог! Пробуем удалить его командой `rmdir`, но и тут отказ. Система сообщает нам, что каталог не пустой, т. е. содержит файлы. Для удаления каталога сначала нужно удалить все его файлы. Конечно, делать это не очень хочется, поэтому проще указать опцию `-r` команды `rm` для рекурсивного удаления каталога. В этом случае сначала будут удалены все подкаталоги (и все файлы в этих подкаталогах), а затем будет уничтожен и сам каталог (рис. 4.3).

Команды `cp` и `mv` работают аналогично: для копирования (перемещения/переименования) сначала указывается каталог-источник, а потом — каталог-назначение. Для каталогов желательно указывать параметр `-r`, чтобы копирование (перемещение) производилось рекурсивно.

4.2.3. Удобная навигация по файловой системе

Несмотря на то что по умолчанию Linux не предоставляет какой-либо удобной оболочки для навигации по файловой системе, сама навигация очень удобна и без всяких оболочек. Рассмотрим приглашение командной строки по умолчанию:

```
evgis@hosting-test-6:/var/www$
```

Здесь

- ❑ evgis — логин пользователя, под которым вы в текущий момент работаете;
- ❑ hosting-test6 — имя компьютера, к консоли которого вы подключены (без разницы, работаете вы удаленно или находитесь за монитором локального компьютера);
- ❑ : — разделитель полей;
- ❑ /var/www — текущий каталог. Если вместо названия каталога вы видите тильду (~), это означает, что вы находитесь в домашнем каталоге. Обычно это каталог /home/<имя пользователя>, т. е. в нашем случае /home/evgis;
- ❑ \$ или # — признак прав пользователя. Если вы работаете с правами root, то будете видеть решетку (#), а если как обычный пользователь — знак доллара (\$).

Получается, что оболочка сразу сообщает имя текущего каталога, в котором вы находитесь, и нет смысла в команде pwd, которая возвращает имя текущего каталога. В ней появляется смысл только при написании сценариев — когда есть необходимость вывести рабочий каталог.

Представим, что мы храним исходный код нашего текущего проекта в каталоге /home/evgis/Sources/Projects/Apps/Neutron/src, различную финансовую информацию — в каталоге /home/evgis/Work/Finances/Reports, а коллекцию фото за 2022 год — в каталоге /data/media/photos/2022. Разберемся, как быстро добраться в эти каталоги.

Если ввести команду cd без параметров, то вы перейдете в свой домашний каталог (об этом знают не все линуксоиды, даже со стажем):

```
$ pwd  
/var/www  
$ cd  
$ pwd  
/home/evgis
```

Также для перехода в домашний каталог можно было бы использовать переменную окружения \$HOME или тильду. Следующие три команды аналогичны:

```
cd  
cd $HOME  
cd ~
```

Все они производят переход в домашний каталог. Переменная \$HOME и тильда особенно вам пригодятся, когда вам нужно перейти не просто в домашний каталог, а в один из его подкаталогов, например:

```
cd $HOME/Sources  
cd ~/Sources
```

Тильда также позволяет быстро перейти в домашний каталог другого пользователя, указав его имя, например:

```
$ cd ~evg  
$ pwd  
/home/evg
```

При навигации по файловой системе можно использовать *автодополнение командной строки* (см. также разд. 2.2). Это очень удобная штука. Пусть, например, вы находитесь в домашнем каталоге:

```
$ pwd  
/home/evgis
```

и хотите перейти в каталог Sources/Projects/Apps/Neutron/src. Вам необязательно вводить весь путь — вводите только начальные символы названий каталогов. Например, введите So вместо Sources и нажмите клавишу <Tab> — оболочка сама дополнит имя каталога. Если в домашнем каталоге больше нет каталогов на букву S, и вы это точно знаете, то можно даже не вводить So — достаточно ввести s и нажать <Tab>. Поступая таким образом, для попадания в каталог Sources/Projects/Apps/Neutron/src вам будет достаточно ввести команду:

```
cd So[tab]/Pr[tab]/A[tab]/Ne[tab]/src
```

Автодополнение — это хорошо, но все равно рано или поздно вам надоест для попадания в свои каталоги постоянно вводить столь длинные команды. Если это разовое мероприятие, то еще не страшно, но когда изо дня в день вам приходится вводить одни и те же команды... На помощь приходят *псевдонимы команд* (см. также разд. 2.2 и 3.3). Откройте ваш файл `~/.bashrc` и добавьте в него следующие строки:

```
alias work="cd $HOME/Sources/Projects/Apps/Neutron/src"  
alias finance="cd $HOME/Work/Finances/Reports"  
alias photos="cd /data/media/photos/2022"
```

Выходите из консоли (командой `exit`) и зайдите в консоль снова. А если вы используете графический терминал, достаточно закрыть окно терминала и снова его открыть. После этого для быстрого перехода к нужному каталогу будет достаточно ввести команду:

```
$ work  
$ pwd  
/home/evgis/Sources/Projects/Apps/Neutron/src  
$ finance  
$ pwd  
/home/evgis/Work/Finances/Reports  
$ photos  
$ pwd  
/data/media/photos/2022
```

Если вы любите порядок и не хотите городить отдельные команды для каждого каталога, тогда можно в файле `.bashrc` определить следующий код:

```
qcd () {  
    case "$1" in  
        work)  
            cd $HOME/Sources/Projects/Apps/Neutron/src  
            ;;  
        finance)  
            cd $HOME/Work/Finances/Reports  
            ;;  
        photos)  
            cd /data/media/photos/2022  
            ;;  
        *)  
            echo "Неизвестный аргумент"  
            return 1  
            ;;  
    esac
```

Здесь мы определили функцию `qcd()`, которая принимает один аргумент — псевдоним каталога. В зависимости от переданного значения происходит переход в один из каталогов. Если псевдоним не опознан, то выводится соответствующее сообщение (впрочем, вы можете предусмотреть для такого случая переход в домашний каталог — как кому больше нравится).

Теперь, после определения функции `qcd()`, мы можем ввести команду `qcd photos` — и перейдем в каталог с фотографиями. Вариант достаточно неплохой, но есть вариант лучше. Недостаток нашей функции `qcd()` в том, что для каждой нужной локации вам придется модифицировать код функции, а это надоедает со временем.

Оболочка `bash` предоставляет более удобное средство — переменную окружения `CDPATH`. Когда вы просто вводите команды вроде `joe`, `nano`, `mc` и пр., оболочка ищет исполняемые файлы этих команд по пути, указанном в переменной окружения `PATH`. Переменная окружения `CDPATH` работает так же, но при смене каталога.

Представим, что у нас есть каталоги `/home/evgis/Family/2022/Photos` и `/home/evgis/Family/2022/Videos`. Пока вы знаете один способ сократить путь к каталогу — использование тильды:

```
cd ~/Family/2022/Photos  
cd ~/Family/2022/Videos
```

А что будет, если вы, находясь в домашнем каталоге, введете команду `cd Photos`? Если в домашнем каталоге нет каталога `Photos`, вы соответствующее сообщение и получите. Однако если вы установите переменную `CDPATH` так:

```
CDPATH = $HOME:$HOME/Projects:$HOME/Family/2022
```

то после ввода команды `cd Photos` поиск каталога `Photos` будет произведен в следующих локациях:

- текущий каталог;
- домашний каталог;
- ~/Projects;
- ~/Family/2022.

Если в текущем, домашнем или каталоге ~/Projects не будет каталога Photos, то в конечном итоге вы перейдете в каталог /home/evgis/Family/2022/Photos. Попробуйте использовать переменную CDPATH — это очень удобно. А чтобы изменения сохранились на постоянной основе, добавьте определение этой переменной в ваш файл .bashrc.

Существует еще один трюк, о котором знают далеко не все линуксоиды, даже со стажем. Представим, что вы работаете в каком-то «глубоком» каталоге вроде /home/evgis/Reports/Company/2022/IT/Finances/Approved. Затем вы перешли в другой каталог — допустим, в ваш домашний каталог:

```
cd /home/evgis/Reports/Company/2022/IT/Finances/Approved  
cd ~
```

А затем возникла необходимость опять вернуться в каталог Approved. Что делать? Снова вводить столь длинную команду или искать команду cd в истории команд, нажимая клавишу-стрелку <↑>? Но если вы после перехода в каталог ~ ввели с десяток команд, то вам придется столько же раз нажать на эту стрелку, прежде чем вы доберетесь до нужной команды. Оказывается есть способ проще:

```
cd -
```

В этом случае команда cd вернет вас в предыдущий каталог!

4.3. Команда ln: создание ссылок

В Linux допускается, чтобы один и тот же файл существовал в системе под разными именами. Для этого используются ссылки. Ссылки бывают двух типов: жесткие и символические. Жесткие ссылки жестко привязываются к файлу — вы не можете удалить файл, пока на него указывает хотя бы одна жесткая ссылка. А вот если на файл указывают *символические ссылки*, его удалению ничего не помешает.

Жесткие ссылки не могут указывать на файл, который находится за пределами файловой системы. Предположим, у вас два Linux-раздела: один — корневой, а второй используется для домашних файлов пользователей и монтируется к каталогу /home корневой файловой системы. Так вот, вы не можете создать в корневой файловой системе ссылку, которая ссылается на файл в файловой системе, подмонтированной к каталогу /home. Это очень важная особенность жестких ссылок. Если вам нужно создать ссылку на файл, который находится за пределами файловой системы, вам следует использовать символические ссылки.

Для создания ссылок предназначена команда ln:

```
ln file.txt link1  
ln -s file.txt link2
```

Первая команда создает жесткую ссылку `link1`, указывающую на текстовый файл `file.txt`. Вторая команда создает символическую ссылку `link2`, которая ссылается на этот же текстовый файл `file.txt`.

Модифицируя ссылку (все равно какую: `link1` или `link2`), вы автоматически изменяете исходный файл — `file.txt`.

Особого внимания заслуживает операция удаления. По идеи, если вы удаляете ссылку `link2`, файл `file.txt` также должен быть удален, но не тут-то было — вы не можете его удалить до тех пор, пока на него указывает хоть одна жесткая ссылка. При удалении ссылки `link2` просто будет удалена символическая ссылка, но жесткая ссылка и сам файл останутся. Если же вы удалите ссылку `link1`, будет удален и файл `file.txt`, поскольку на него больше не указывает ни одна жесткая ссылка.

4.4. Команды `chmod`, `chown` и `chattr`

4.4.1. Команда `chmod`: права доступа к файлам и каталогам

Для каждого каталога и файла можно задать *права доступа*. Точнее, права доступа автоматически задаются при создании каталога и файла, а вы при необходимости можете их изменить. Какова же эта необходимость? Например, вам нужно, чтобы к вашему файлу-отчету смогли получить доступ пользователи — члены вашей группы. Или вы создали обычный текстовый файл, содержащий инструкции командного интерпретатора. Чтобы этот файл стал сценарием, вам нужно установить право на выполнение для этого файла.

Существуют три права доступа: чтение (`r`), запись (`w`), выполнение (`x`). Для каталога право на выполнение означает право на просмотр содержимого каталога.

Вы можете установить разные права доступа для владельца (т. е. для себя), для группы владельца (т. е. для всех пользователей, входящих в одну с владельцем группу) и для прочих пользователей. Пользователь `root` может получить доступ к любому файлу или каталогу вне зависимости от прав, которые вы установили.

Чтобы просмотреть текущие права доступа, введите команду:

```
ls -l <имя файла/каталога>
```

Например,

```
ls -l video.txt
```

В ответ программа выведет следующую строку:

```
-r--r---- 1 evg group 300 Apr 11 11:11 video.txt
```

В этой строке фрагмент `-r--r----` описывает права доступа:

- первый символ — это признак каталога. Сейчас перед нами файл. Если бы это был каталог, то первый символ был бы символом `d` (от англ. `directory`);

- последующие три символа (`r--`) определяют *права доступа владельца файла или каталога*. Первый символ — это чтение, второй — запись, третий — выполнение. Как можно видеть, владельцу разрешено только чтение этого файла, запись и выполнение запрещены, поскольку в правах доступа режимы `w` и `x` не определены;
- следующие три символа (`r--`) задают *права доступа для членов группы владельца*. Права такие же, как и у владельца: можно читать файл, но нельзя изменять или запускать;
- последние три символа (`---`) задают *права доступа для прочих пользователей*. Прочие пользователи не имеют права ни читать, ни изменять, ни выполнять файл. При попытке получить доступ к файлу они увидят сообщение `Access denied`.

ПРИМЕЧАНИЕ

После прав доступа команда `ls` выводит имя владельца файла, имя группы владельца, размер файла, дату и время создания, а также имя файла.

Права доступа задаются командой `chmod`. Существуют два способа указания прав доступа: *символьный* (когда указываются символы, задающие право доступа, — `r, w, x`) и *абсолютный*.

Так уж заведено, что в мире UNIX чаще пользуются абсолютным методом. Разберемся, в чем он заключается. Рассмотрим следующий набор права доступа:

`rw-r-----`

Этот набор прав доступа разрешает владельцу чтение и модификацию файла (`rw-`), запускать файл владелец не может. Члены группы владельца могут только просматривать файл (`r--`), а все остальные пользователи не имеют вообще никакого доступа к файлу.

Возьмем отдельный набор прав, например для владельца: `rw-`.

Чтение разрешено — мысленно записываем 1, запись разрешена — запоминаем еще 1, а вот выполнение запрещено, поэтому запоминаем 0. Получается число 110. Если из двоичной системы перевести число 110 в восьмеричную, получится число 6. Для перевода можно воспользоваться табл. 4.3.

Таблица 4.3. Преобразование чисел из двоичной системы в восьмеричную

Двоичная система	Восьмеричная система	Двоичная система	Восьмеричная система
000	0	100	4
001	1	101	5
010	2	110	6
011	3	111	7

Аналогично произведем разбор прав для членов группы владельца. Получится двоичное 100, т. е. восьмеричное 4. С третьим набором (---) все вообще просто — это 000, т. е. 0.

Записываем полученные числа в восьмеричной системе в порядке «владелец-группа-остальные». Получится число 640 — это и есть права доступа. Для того чтобы установить эти права доступа, выполните команду:

```
chmod 640 <имя_файла>
```

Наиболее популярные права доступа:

- 644 — владельцу можно читать и изменять файл, остальным пользователям — только читать;
- 666 — читать и изменять файл можно всем пользователям;
- 777 — всем можно читать, изменять и выполнять файл.

ПРИМЕЧАНИЕ

Напомню, что для каталога право выполнения — это право просмотра оглавления каталога.

Иногда символьный метод оказывается проще. Например, у нас есть файл `script`, который нужно сделать исполняемым, — для этого можно применить команду:

```
chmod +x script
```

А чтобы снять право выполнения, указывается параметр `-x`:

```
chmod -x script
```

Подробнее о символьном методе вы сможете прочитать в руководстве по команде `chmod` (выполнив команду `man chmod`).

4.4.2. Команда `chown`: смена владельца файла

Если вы хотите «подарить» кому-то файл, т. е. сделать какого-то пользователя владельцем файла, то вам нужно применить команду `chown`:

```
chown пользователь файл
```

ПРИМЕЧАНИЕ

Возможно, что после изменения владельца файла вы сами не сможете получить к нему доступ, ведь владельцем будете уже не вы.

4.4.3. Специальные права доступа (SUID и SGID)

Мы рассмотрели обычные права доступа к файлам, но в Linux есть еще так называемые *специальные права доступа*: SUID (Set User ID root) и SGID (Set Group ID root).

Эти права доступа позволяют обычным пользователям запускать программы, требующие для своего запуска привилегий пользователя `root`. Например, демон `pppd`

требует привилегий root, но чтобы каждый раз при установке PPP-соединения (модемное, ADSL-соединение) не входить в систему под именем root, достаточно установить специальные права доступа для демона pppd. Делается это так:

```
chmod u+s /usr/sbin/pppd
```

Однако не нужно увлекаться такими решениями, поскольку каждая программа, для которой установлен бит SUID, является потенциальной «дырой» в безопасности вашей системы. Для выполнения программ, требующих прав root, намного рациональнее использовать команды sudo и su (описание которых можно получить по командам `man sudo` и `man su`).

4.4.4. Команда `chattr`: атрибуты файла, запрет изменения файла

С помощью команды `chattr` можно изменить атрибуты файла. Параметр (+) устанавливает атрибут, а параметр (-) атрибут снимает.

Например:

```
# chattr +i /boot/grub/menu.lst
```

Эта команда устанавливает атрибут `i`, запрещающий любое изменение, переименование и удаление файла. Установить этот атрибут, равно как и снять его, имеет право только суперпользователь или процесс с возможностью `CAP_LINUX_IMMUTABLE`. Чтобы изменить файл, нужно очистить атрибут с помощью команды:

```
# chattr -i /boot/grub/menu.lst
```

- Для файла с установленным атрибутом `j` его данные перед записью в файл сохраняются в журнале файловой системы ext3/ext4. Это происходит, только если файловая система смонтирована с опциями `data=ordered` или `data=writeback`. Если файловая система смонтирована с опцией `data=journal`, все данные уже и так журналируются, и атрибут `j` никак на это не влияет. Этот атрибут имеет право установить (или очистить) суперпользователь или процесс с возможностью `CAP_LINUX_IMMUTABLE`.
- Когда для файла установлен атрибут `a` (прописная буква), тогда не происходит обновление записи `atime` (в ней хранится время доступа к файлу). Это позволяет избежать лишних дисковых операций ввода/вывода, что полезно для медленных компьютеров.
- Если для файла установлен атрибут `a`, в файл можно только добавлять данные. Этот атрибут имеет право установить (или очистить) суперпользователь или процесс с возможностью `CAP_LINUX_IMMUTABLE`.
- Атрибут `c` заставляет систему упаковывать (сжимать) содержимое файла, что позволяет сэкономить место на диске. При чтении из этого файла возвращаются несжатые данные. При записи в файл информация автоматически сжимается и записывается на диск в уже сжатом виде.

- Когда изменяется каталог с установленным атрибутом `D`, изменения сразу же записываются на диск. Это эквивалентно применению опции монтирования `dirsync`.
- Если для файла установлен атрибут `d`, для него не будет выполнено резервное копирование программой `dump`.
- При изменении файла с установленным атрибутом `s` его данные синхронно записываются на диск. Это аналогично опции монтирования `sync` к подмножеству файлов.
- Когда удаляется файл с установленным атрибутом `s`, система выполняет обнуление его блоков и запись их обратно на диск.
- При удалении файла с атрибутом `u` его содержимое сохраняется на диске, что позволяет впоследствии легко восстановить этот файл.
- Атрибуты `x` и `z` используются экспериментальными заплатками сжатия для служебных целей.

Установить любой атрибут можно командой `chattr`, а просмотреть — командой `lsattr`. Об остальных атрибутах вы сможете прочитать в справочной системе:

```
man chattr
```

4.5. Монтирование файловых систем

4.5.1. Команды `mount` и `umount`

Чтобы работать с какой-либо файловой системой, необходимо *примонтировать* ее к корневой файловой системе. Например, вставив в соответствующий разъем флешку, нужно подмонтировать файловую систему флешки к корневой файловой системе — только так мы сможем получить доступ к файлам и каталогам, которые на этой флешке записаны. Аналогичная ситуация с жесткими, оптическими дисками и другими носителями данных.

Если вы хотите заменить сменный носитель данных (флешку, компакт-диск), вам нужно сначала размонтировать файловую систему, затем извлечь носитель данных, установить новый и заново смонтировать файловую систему. В случае с флешкой о размонтировании должны помнить вы сами, поскольку при этом выполняется синхронизация буферов ввода/вывода и файловой системы, т. е. данные физически записываются на диск, если это еще не было сделано. А компакт-диск система не разрешит вам извлечь, если он не размонтирован. В свою очередь, размонтировать файловую систему можно, только когда ни один процесс ее не использует.

При завершении работы системы (перезагрузке, выключении компьютера) размонтирование всех файловых систем выполняется автоматически.

Команда монтирования (ее нужно выполнять с привилегиями `root`) выглядит так:

```
# mount [опции] <устройство> <точка монтирования>
```

Точка монтирования — это каталог, через который будет осуществляться доступ к монтируемой файловой системе. Например, если вы подмонтировали компакт-диск к каталогу `/mnt/cdrom`, то получить доступ к файлам и каталогам, записанным на компакт-диске, можно будет через его точку монтирования (именно через каталог `/mnt/cdrom`). Точкой монтирования может быть любой каталог корневой файловой системы, хоть `/aaa-111`. Главное, чтобы этот каталог существовал на момент монтирования файловой системы.

В некоторых современных дистрибутивах запрещен вход в систему под именем суперпользователя `root`. Поэтому для выполнения команд с привилегиями `root` вам следует использовать команду `sudo`. Например, чтобы выполнить команду монтирования привода компакт-диска, вам надо ввести команду:

```
sudo mount /dev/hdc /mnt/cdrom
```

Перед выполнением команды `mount` команда `sudo` попросит вас ввести пароль `root`. Если введенный пароль правильный, то будет выполнена команда `mount`.

Для размонтирования файловой системы служит команда `umount`:

```
# umount <устройство или точка монтирования>
```

4.5.2. Файлы устройств и монтирование

В этой главе мы уже говорили о файлах устройств. Здесь мы вернемся к ним снова, но в контексте монтирования файловой системы.

Ранее было отмечено, что для Linux нет разницы между устройством и файлом. Все устройства системы представлены в корневой файловой системе как обычные файлы. Например, `/dev/fd0` — это ваш дисковод для гибких дисков (если на вашем компьютере такой дисковод еще имеется), а `/dev/sda` (ранее `/dev/hda`) — жесткий диск. Как можно видеть, файлы устройств хранятся в каталоге `/dev`.

Жесткие диски

С жесткими дисками сложнее всего, поскольку одно и то же устройство может в разных версиях одного и того же дистрибутива называться по-разному. Например, мой IDE-диск, подключенный как первичный мастер, в Fedora 5 все еще назывался `/dev/hda`, а в Fedora 8 он стал называться `/dev/sda`. Раньше накопители, подключающиеся к интерфейсу IDE (PATA), назывались `/dev/hdx`, а SCSI/SATA-накопители — `/dev/sdx` (где в обоих случаях `x` — буква устройства).

После принятия `udev` (см. далее) и глобального уникального идентификатора устройств (UUID) все дисковые устройства, вне зависимости от интерфейса подключения (PATA, SATA, SCSI), называются `/dev/sdx`, где `x` — буква устройства. `udev` и UUID поддерживают все современные дистрибутивы. Так что не удивляйтесь, если вдруг ваш старенький IDE-винчестер будет назван `/dev/sda`. С одной стороны, это вносит некоторую путаницу. С другой стороны, все современные компьютеры оснащены именно SATA-дисками (т. к. PATA-диски уже устарели, а SCSI — дорогие), а на современных материнских платах только один контроллер IDE (PATA), потому многие пользователи даже ничего не заметят.

Пояснение

`udev` — это менеджер устройств, используемый в ядрах Linux версии 2.6. Пришел на смену более громоздкой псевдофайловой системе `devfs`. Управляет всеми манипуляциями с файлами из каталога `/dev`.

Рассмотрим ситуацию с жесткими дисками чуть подробнее. Пусть у нас есть устройство `/dev/sda`. На жестком диске, понятное дело, может быть несколько разделов (логических дисков). Предположим, что в нашем случае на диске имеются три раздела, которые в Windows называются C:, D: и E:. Диск C: обычно является загрузочным (активным), поэтому он будет записан в самом начале диска. Нумерация разделов жесткого диска в Linux начинается с 1, поэтому в большинстве случаев диску C: будет соответствовать имя `/dev/sda1` — первый раздел на первом жестком диске.

Резонно предположить, что двум оставшимся разделам (D: и E:) были присвоены имена `/dev/sda2` и `/dev/sda3`. Это может быть так — и не так. Сейчас поясню. Раздел может быть первичным (primary partition), расширенным (extended partition) или логическим (logical partition). Всего на диске может быть или четыре первичных раздела, или три первичных и один расширенный.

Пусть на жестком диске имеются четыре первичных раздела, для которых зарезервированы номера 1, 2, 3 и 4. Если разделы D: и E: — первичные, то им будут присвоены имена `/dev/sda2` и `/dev/sda3`. Но в большинстве случаев эти разделы являются логическими, а логические разделы содержатся в расширенном разделе (там может быть максимум 11 логических разделов). При этом в Windows расширенному разделу не присваивается буква, потому что этот раздел не содержит данных пользователя, а только информацию о логических разделах. Логические разделы именуются, начиная с 5, т. е. если разделы D: и E: — логические, то им будут присвоены имена `/dev/sda5` и `/dev/sda6` соответственно.

Узнать номер раздела очень просто: достаточно запустить утилиту, работающую с таблицей разделов диска, — `fdisk` (будет рассмотрена в главе 11).

Чтобы узнать, используя эту утилиту, номера разделов первого жесткого диска (`/dev/sda`), введите команду:

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

и в ответ на приглашение утилиты введите `r` и нажмите клавишу `<Enter>`. Вы увидите таблицу разделов этого диска. После этого для выхода из программы введите `q` и нажмите клавишу `<Enter>`.

На рис. 4.4 показана таблица разделов моего первого жесткого диска. Первый раздел (это мой диск C:, где установлена система Windows) — первичный. Сразу после него расположен расширенный раздел (его номер — 2). Следующий за ним — логический раздел (номер 5). Разделы с номерами 3 и 4 пропущены, потому что их нет на моем жестком диске. Это те самые первичные разделы, которые я не создавал — они мне не нужны.

- 1) программами, запускаемыми при загрузке (напр., старые версии LILO)
 2) загрузкой и программами разметки из других ОС
 (напр., DOS FDISK, OS/2 FDISK)

Команда (**m** для справки): **parted**

```
Диск /dev/sda: 160.0 ГБ, 160041885696 байт
255 heads, 63 sectors/track, 19457 cylinders
Units = цилинды of 16065 * 512 = 8225280 bytes
Disk identifier: 0xe905e905
```

Устр-во	Загр	Начало	Конец	Блоки	Id	Система
/dev/sda1	*	1	543	4361616	b	W95 FAT32
/dev/sda2		544	19457	151926705	f	W95 расшир. (LBA)
/dev/sda5		544	1021	3839503+	83	Linux
/dev/sda6		1022	1759	5927953+	83	Linux
/dev/sda7		1760	1825	530113+	82	Linux swap / Solaris
/dev/sda8		1826	5963	33238453+	b	W95 FAT32
/dev/sda9		5964	10101	33238453+	b	W95 FAT32
/dev/sda10		10102	14268	33471396	b	W95 FAT32
/dev/sda11		14269	16949	21535101	b	W95 FAT32
/dev/sda12		16950	19457	20145478+	b	W95 FAT32

Команда (**m** для справки): **fdisk -l**

Рис. 4.4. Таблица разделов жесткого диска

Приводы оптических дисков

Любое из устройств **sdx** может быть приводом для чтения CD- или DVD-дисков. Если система видит, что устройство является приводом CD-ROM, то автоматически создается ссылка **/dev/cdrom**. А если ваш привод умеет также читать и DVD-диски, то в каталоге **/dev** появится еще одна ссылка — **/dev/dvd**. Например, мой DVD-RW подключен как первый подчиненный (**/dev/sdb**), и в каталоге **/dev** ему соответствуют три файла: **/dev/sdb**, **/dev/cdrom**, **/dev/dvd**. Понятно, что обратиться к приводу можно, используя любой из этих файлов.

Поэтому, чтобы подмонтировать привод для чтения оптических дисков, нужно ввести одну из трех команд:

```
# mount /dev/sdb /mnt/cdrom
# mount /dev/cdrom /mnt/cdrom
# mount /dev/dvd /mnt/cdrom
```

После этого обратиться к файлам, записанным на этом диске, можно будет через каталог **/mnt/cdrom**. Напомню, что каталог **/mnt/cdrom** должен существовать.

Дискеты

Аналогичная ситуация и с дискетами (опять же, если эта информация для вас актуальна, поскольку дискеты давным-давно вышли из употребления). В системе могут быть установлены два дисковода для дискет: первый (**/dev/fd0**) и второй (**/dev/fd1**).

Для их монтирования можно использовать команды:

```
# mount /dev/fd0 /mnt/floppy
# mount /dev/fd1 /mnt/floppy
```

Напомню, что в Windows-терминологии устройство `/dev/fd0` — это диск A:, а устройство `/dev/fd1` — диск B:.

Флешки и внешние жесткие диски

Флешка или USB-диск определяется системой как обычный жесткий диск. Предположим, что у вас установлен всего один жесткий диск, — тогда ему будет соответствовать имя устройства `/dev/sda`.

Когда вы подключите флешку или внешний жесткий диск, ему будет присвоено имя `/dev/sdb`. Обычно на флешке или USB-диске всего один раздел, поэтому подмонтировать такое устройство можно командой:

```
# mount /dev/sdb1 /mnt/usbdisk
```

Далее мы поговорим о монтировании флешек (и устройств, определяемых как флешки: цифровые фотоаппараты, видеокамеры, мобильные телефоны) более подробно (см. разд. 4.5.6). А пока отметим, что в современных дистрибутивах флешки, внешние жесткие диски и диски CD/DVD монтируются автоматически (правда, не к подкаталогу `/mnt` — чаще для этих целей используется каталог `/media/<ID накопителя>`, но все зависит от дистрибутива), и все команды их монтирования приведены в книге для вашего общего развития или на аварийный случай, когда вы загрузите систему в однопользовательском режиме, и вам придется монтировать носители вручную.

4.5.3. Опции монтирования файловых систем

Теперь, когда мы знаем номер раздела, можно подмонтировать его файловую систему. Делается это так:

```
# mount <раздел> <точка монтирования>
```

Например:

```
# mount /dev/sda5 /mnt/win_d
```

У команды `mount` довольно много опций, но на практике наиболее часто используются только некоторые из них: `-t`, `-r`, `-w`, `-a`.

□ Параметр `-t` позволяет задать тип файловой системы. Обычно программа сама определяет файловую систему, но иногда это у нее не получается. Тогда мы должны ей помочь. Формат использования этого параметра следующий:

```
# mount -t <файловая система> <устройство> <точка монтирования>
```

Например:

```
# mount -t iso9660 /dev/hdc /mnt/cdrom
```

Вот опции для указания наиболее популярных монтируемых файловых систем:

- `ext2` или `ext3` — файловая система Linux;
- `iso9660` — указывается при монтировании CD-ROM;
- `vfat` — FAT, FAT32 (поддерживается Windows 9x, ME, XP);

- ntfs — NT File System (поддерживается Windows NT, XP, 7, 8, 10, 11). Будет использована стандартная поддержка NTFS, при которой NTFS-раздел доступен только для чтения;
- ntfs-3g — будет использован модуль ntfs-3g, входящий в большинство современных дистрибутивов. Этот модуль позволяет производить запись информации на NTFS-разделы;

ПРИМЕЧАНИЕ

Если в вашем дистрибутиве нет модуля ntfs-3g, т. е. при попытке указания файловой системы с поддержкой NTFS вы увидели сообщение об ошибке, вы можете скачать этот модуль с сайта www.ntfs-3g.org. Там доступны как исходные коды, так и уже откомпилированные для разных дистрибутивов пакеты.

- Параметр `-r` монтирует указанную файловую систему в режиме «только чтение»;
- Параметр `-w` монтирует файловую систему в режиме «чтение/запись». Этот параметр используется по умолчанию для файловых систем, поддерживающих запись (например, NTFS по умолчанию запись не поддерживает, как и файловые системы CD/DVD-дисков);
- Параметр `-a` служит для монтирования всех файловых систем, указанных в файле `/etc/fstab` (кроме тех, для которых указано `noauto`, — такие файловые системы нужно монтировать вручную). При загрузке системы вызывается команда `mount` с параметром `-a`.

Если вы не можете смонтировать NTFS-раздел с помощью опции `ntfs-3g`, то, вероятнее всего, он был неправильно размонтирован (например, работа Windows не была завершена корректно). В этом случае для монтирования раздела нужно использовать опцию `-o force`, например:

```
sudo mount -t ntfs-3g /dev/sdb1 /media/usb -o force
```

4.5.4. Монтирование разделов при загрузке

Если вы не хотите при каждой загрузке монтировать постоянные файловые системы (например, ваши Windows-разделы), то вам нужно прописать их в файле `/etc/fstab`. Обратите внимание: в этом файле не следует прописывать файловые системы смешанных носителей (CD/DVD-привода, флеш-диска и т. п.). Надо отметить, что программы установки некоторых дистрибутивов — например, Mandriva, читают таблицу разделов и автоматически заполняют файл `/etc/fstab`. В результате все ваши Windows-разделы доступны сразу после установки системы. К сожалению, не все дистрибутивы могут похвастаться такой интеллектуальностью, поэтому вам нужно знать формат файла `fstab`:

устройство_точка_монтирования тип_ФС опции флаг_РК флаг_проверки

Здесь: *тип_ФС* — это тип файловой системы, а *флаг_РК* — флаг резервного копирования. Если он установлен (1), то программа `dump` заархивирует эту файловую систему при создании резервной копии. Если не установлен (0), то резервная копия этой

файловой системы создаваться не будет. Флаг проверки устанавливает, будет ли эта файловая система проверяться на наличие ошибок программой `fsck`. Проверка производится в двух случаях:

- если файловая система размонтирована некорректно;
- если достигнуто максимальное число операций монтирования для этой файловой системы.

ФАЙЛЫ *FSTAB* И *SYSTEMD*

С появлением системы инициализации `systemd` файл `/etc/fstab` стал далеко не единственным источником автоматического монтирования файловых систем. Он все еще поддерживается, и все, что здесь сказано об `fstab`, — верно, но `systemd` позволяет монтировать файловые системы другим способом.

Поле опций содержит важные параметры файловой системы. Некоторые из них представлены в табл. 4.4.

Таблица 4.4. Опции монтирования файловой системы в файле `/etc/fstab`

Опция	Описание
<code>auto</code>	Файловая система должна монтироваться автоматически при загрузке. Опция используется по умолчанию, поэтому ее указывать не обязательно
<code>noauto</code>	Файловая система не монтируется при загрузке системы (при выполнении команды <code>mount -a</code>), но ее можно смонтировать вручную с помощью все той же команды <code>mount</code>
<code>defaults</code>	Используется стандартный набор опций, установленных по умолчанию
<code>exec</code>	Разрешает запуск выполняемых файлов для этой файловой системы. Опция используется по умолчанию
<code>noexec</code>	Запрещает запуск выполняемых файлов для этой файловой системы
<code>ro</code>	Монтирование в режиме «только чтение»
<code>rw</code>	Монтирование в режиме «чтение/запись». Используется по умолчанию для файловых систем, поддерживающих запись
<code>user</code>	Эту файловую систему разрешается монтировать/размонтировать обычному пользователю (не <code>root</code>)
<code>nouser</code>	Файловую систему может монтировать только пользователь <code>root</code> . Используется по умолчанию
<code>umask</code>	Определяет маску прав доступа при создании файлов. Для не Linux-подобных файловых систем маску нужно установить так: <code>umask=0</code>
<code>utf8</code>	Применяется только на дистрибутивах, которые используют кодировку UTF-8 в качестве кодировки локали. В старых дистрибутивах (где используется KOI8-R) для корректного отображения русских имен файлов на Windows-разделах нужно задать параметры <code>iocharset=koi8-u,codepage=866</code>

ПРИМЕЧАНИЕ

Редактировать файл `/etc/fstab`, как и любой другой файл из каталога `/etc`, можно в любом текстовом редакторе (например, `gedit`, `kate`), но перед этим нужно получить права `root` (командой `su` или `sudo`).

Рассмотрим небольшой пример:

```
/dev/sdc /mnt/cdrom auto umask=0,user,noauto,ro,exec 0 0
/dev/sda1 /mnt/win_c vfat umask=0,utf8 0 0
```

Первая строка — это строка монтирования файловой системы компакт-диска, а вторая — строка монтирования диска С:.

- Начнем с первой строки. `/dev/sdc` — это имя устройства CD-ROM. Точка монтирования — `/mnt/cdrom`. Понятно, что этот каталог должен существовать. Обратите внимание: в качестве файловой системы не указывается жестко `iso9660`, поскольку компакт-диск может быть записан в другой файловой системе, поэтому в качестве типа файловой системы задано `auto`, т. е. автоматическое определение. Далее идет довольно длинный набор опций. Ясно, что `umask` установлен в ноль, поскольку файловая система компакт-диска не поддерживает права доступа Linux. Параметр `user` говорит о том, что эту файловую систему можно монтировать обычному пользователю. Параметр `noauto` запрещает автоматическое монтирование этой файловой системы, что правильно — ведь на момент монтирования в приводе может и не быть компакт-диска. Опция `ro` разрешает монтирование в режиме «только чтение», а `exec` разрешает запускать исполняемые файлы. Понятно, что компакт-диск не нуждается ни в проверке, ни в создании резервной копии, поэтому два последних флага равны нулю.
- Вторая строка проще. Первые два поля — это устройство и точка монтирования. Третье — тип файловой системы. Файловая система постоянна, поэтому можно явно указать тип файловой системы (`vfat`), а не `auto`. Опция `umask`, как и в предыдущем случае, равна нулю. Указание опции `utf8` позволяет корректно отображать русскоязычные имена файлов и каталогов.

4.5.5. Подробно о UUID и файле `/etc/fstab`

Продолжая разговор о формате файла `/etc/fstab`, рассмотрим стандарт идентификации UUID (Universally Unique Identifier) и *длинные имена* дисков. В некоторых дистрибутивах — например, в Ubuntu, вместо имени носителя (первое поле файла `fstab`) указывается его ID, поэтому содержимое `fstab` выглядит устрашающе — например, вот так:

```
# /dev/sda6
UUID=1f049af9-2bdd-43bf-a16c-ff5859a4116a / ext3 defaults 0 1
# /dev/sda1
UUID=45AE-84D9 /media/sda1 vfat defaults,utf8,umask=007 0 0
```

В SUSE идентификаторы устройств указываются немного иначе:

```
/dev/disk/by-id/scsi-SATA_WDC_WD1600JB-00_WD-WCANM7959048-part5 / ext3 acl,user_xattr 1 1
/dev/disk/by-id/scsi-SATA_WDC_WD1600JB-00_WD-WCANM7959048-part7 swap swap defaults 0 0
```

Понятно, что использовать короткие имена вроде `/dev/sda1` намного проще, чем идентификаторы в стиле `1f049af9-2bdd-43bf-a16c-ff5859a4116a`. Использование имен

дисков еще никто не отменял, поэтому вместо идентификатора носителя вы можете смело указывать его файл устройства — так вам будет значительно проще!

Но все же разбираться в соответствии длинных имен файлов коротким именам устройств весьма важно. Ведь система использует именно эти имена, и в файле `/etc/fstab` не всегда указывается, какой идентификатор какому короткому имени устройства принадлежит (или указывается, но не для всех разделов).

Узнать длинное имя устройства можно с помощью простой команды:

```
ls -l /dev/disk/by-uuid/
```

Результат выполнения этой команды приведен на рис. 4.5.

```
[den@localhost ~]$ ls -l /dev/disk/by-uuid/
итого 0
lrwxrwxrwx 1 root root 10 Фев 12 15:25 1c3b8bd3-c26f-449e-9eba-8f254fef814 -> ../../sda1
lrwxrwxrwx 1 root root 10 Фев 12 15:25 3106fa17-65ef-42e9-a1d4-2313daee96b5 -> ../../sda2
[den@localhost ~]$ ls -l /dev/disk/by-label
итого 0
lrwxrwxrwx 1 root root 10 Фев 12 15:25 SWAP-sda2 -> ../../sda2
lrwxrwxrwx 1 root root 10 Фев 12 15:25 \x2f -> ../../sda1
[den@localhost ~]$
```

Рис. 4.5. Соответствие длинных имен дисков коротким

Спрашивается, зачем были введены длинные имена, если короткие имена удобнее, во всяком случае для пользователей? Оказывается, разработчики Linux в первую очередь и заботились как раз о пользователях. Возьмем обычный IDE-диск. Как известно, этот диск можно подключить либо к первичному (primary), либо ко вторичному (secondary), если он есть, контроллеру. Кроме того, в зависимости от положения перемычки выбора режима винчестер может быть либо главным устройством (master), либо подчиненным (slave). Таким образом, диск может иметь следующие короткие имена: `sda` (primary master), `sdb` (primary slave), `sdc` (secondary master), `sdd` (secondary slave). То же самое происходит с SATA/SCSI-винчестерами — при изменении параметров подключения изменяется и короткое имя устройства.

А вот при использовании длинных имен идентификатор дискового устройства остается постоянным вне зависимости от типа подключения устройства к контроллеру. Именно поэтому длинные имена дисков часто также называются *постоянными именами* (*persistent name*). Получается, что раньше вы могли ошибочно подключить жесткий диск немного иначе, и разделы, которые назывались, скажем, `/dev/sdaN`, стали называться `/dev/sdbN`. Понятно, что загрузить Linux с такого диска не получится, поскольку везде указаны другие имена устройств. Если же используются длинные имена дисков, система загрузится в любом случае, как бы вы ни подключили жесткий диск. Удобно? Конечно.

Но это еще не все. Постоянные имена — это только первая причина. Вторая причина заключается в обновлении библиотеки `libata`. В новой версии `libata` все PATA-

устройства именуются не как раньше (`hdx`), а по-новому (`sdx`), что (как отмечалось в этой главе ранее) вносит некую путаницу. Длинные имена дисков от этого не изменяются, поэтому они избавляют пользователя от беспокойства по поводу того, что его старый IDE-диск вдруг превратился в диск SATA/SCSI.

При использовании UUID однозначно идентифицировать раздел диска можно несколькими способами:

- `UUID=45AE-84D9 /media/sda1 vfat defaults,utf8,umask=007, gid=46 0 0` — здесь с помощью параметра **UUID** указывается идентификатор диска;
- `/dev/disk/by-id/scsi-SATA_WDC_WD1600JB-00_WD-WCANM7959048-part7 swap swap defaults 0 0` — здесь указывается длинное имя устройства диска;
- `LABEL=/ / ext3 defaults 1 1` — самый компактный третий способ, позволяющий идентифицировать устройства по их метке.

ПРИМЕЧАНИЕ

Первый способ получения длинного имени в англоязычной литературе называется `by-uuid`, т. е. длинное имя составляется по UUID; второй способ называется `by-id`, т. е. по аппаратному идентификатору устройства. Третий способ называется `by-label` — по метке. Просмотреть соответствие длинных имен коротким можно с помощью команд:

```
ls -l /dev/disk/by-uuid
ls -l /dev/disk/by-id
ls -l /dev/disk/by-label
```

Но есть еще и четвертый способ, который называется `by-path`. В этом случае имя генерируется по `sysfs`. Этот способ является наименее используемым, поэтому вы вряд ли столкнетесь с ним.

Узнать метки разделов можно с помощью команды:

```
ls -lF /dev/disk/by-label
```

а установить метку — посредством команд, приведенных в табл. 4.5.

Таблица 4.5. Команды для установки меток разделов

Файловая система	Команда
ext2/ext3/ext4	<code># e2label /dev/XXX <метка></code>
ReiserFS	<code># reiserfstune -l <метка> /dev/XXX</code>
JFS	<code># jfs_tune -L <метка> /dev/XXX</code>
XFS	<code># xfs_admin -L <метка> /dev/XXX</code>
FAT/FAT32	Только средствами Windows
NTFS	<code># ntfslabel /dev/XXX <метка></code>

Итак, в файле `/etc/fstab` вы можете использовать длинные имена в любом формате. Можно указывать имена устройств в виде: `/dev/disk/by-uuid/*`, `/dev/disk/by-id/*` или `/dev/disk/by-label/*`, можно использовать параметры `UUID=идентификатор` или `LABEL=метка`. Применяйте тот способ, который вам больше нравится.

4.5.6. Монтирование флешек

Принцип использования флешки очень прост — достаточно подключить ее к шине USB, и через несколько секунд система определит диск. После этого с ним можно будет работать как с обычным диском. Да, Flash-диски не очень шустры, но молниеносной реакции от них никто и не ожидает — во всяком случае, они выглядят настоящими спринтерами на фоне обычных дискет. Хотя, если ваш компьютер поддерживает USB 3.0 и вы обзавелись соответствующей флешкой (тоже с USB 3.0), то недостатка в скорости вы испытывать не будете.

Ранее для монтирования флешки нужно было выполнить настоящий танец с бубном, сейчас же флешка автоматически монтируется при подключении. Как правило, монтирование происходит в каталоге `/media/<ID флешки>`, через который и можно обратиться к записанным на нее файлам. Современные дистрибутивы Linux нормально работают с флешками, отформатированными в Windows, поэтому даже при копировании информации на флешку (т. е. при операции записи) у вас не должно быть никаких проблем.

Смонтировать флешку вручную так же просто. USB-диск — это обычный накопитель, и его можно увидеть в каталоге `/dev/disk/by-id`. Напомню, что способ `by-id` подразумевает получение длинного имени по аппаратному идентификатору устройства, а поэтому с помощью каталога `/dev/disk/by-id` проще всего найти длинное имя USB-диска среди имен других накопителей — в его начале вы увидите префикс `usb_`. Введите команду:

```
ls -l /dev/disk/by-id | grep usb
```

Результат выполнения этой команды представлен на рис. 4.6. Как можно видеть, для монтирования Flash-диска нужно выполнить команду:

```
# mount /dev/sdb1 /mnt/flash
```

Впрочем, у вас может ничего не получиться, если система уже автоматически смонтировала диск.

```
[root@localhost 001]# ls -l /dev/disk/by-id | grep usb
lrwxrwxrwx 1 root root 9 Фев 12 17:41 usb-AIT Card Reader_0 DISK01-0:0 -> ../../sdb
lrwxrwxrwx 1 root root 10 Фев 12 17:41 usb-AIT_Card_Reader_0_DISK01-0:0-part1 -> ../../sdb1
[root@localhost 001]#
```

Рис. 4.6. USB-диск найден

4.6. Настройка журнала файловой системы ext3

Напомню, что файловая система ext3 — это та же ext2, но с небольшой надстройкой в виде журнала операций. Долгое время она оставалась стандартной файловой системой Linux, пока ее окончательно не вытеснила ext4. Сейчас с ext3 вам придется

ся сталкиваться или на весьма древних дистрибутивах, или же если вы сами по какой-то причине отформатировали диск в эту файловую систему.

Журналируемая файловая система имеет три режима работы: `journal`, `ordered` и `writeback`. Первый режим является самым медленным, но он позволяет минимизировать потери ваших данных в случае сбоя системы (или отключения питания). В этом режиме в системный журнал записывается все, что только можно, — это позволяет максимально восстановить файловую систему после сбоя.

В последовательном режиме (`ordered`) в журнал заносится информация только об изменении метаданных (служебных данных файловой системы). Этот режим используется по умолчанию и является компромиссным вариантом между производительностью и отказоустойчивостью.

Самым быстрым является режим обратной записи (`writeback`). Но использовать его я вам не рекомендую, поскольку особого толку от него не будет. Проще тогда уже при установке Linux выбрать файловую систему `ext2` вместо `ext3/ext4`.

Если отказоустойчивость для вас на первом месте — выбирайте режим `journal`, во всех остальных случаях лучше выбрать `ordered`. Выбор режима осуществляется редактированием файла `/etc/fstab`. Например:

```
# Режим ordered используется по умолчанию,  
# поэтому ничего указывать не нужно  
/dev/sda1 / ext3 defaults 1 0  
# На этом разделе важные данные – используем режим journal  
/dev/sda2 /var ext3 data=journal 1 0  
# Здесь ничего важного нет – режим writeback  
/dev/sda3 /opt ext3 data=writeback 0 0
```

После изменения этого файла выполните команду:

```
# mount -a
```

Эта команда заново смонтирует все файловые системы, чтобы изменения вступили в силу.

4.7. Файловая система ext4

Файловая система `ext4` заслуживает отдельного разговора. Все, что было сказано о файловых системах ранее, справедливо и для `ext4`, но у новой файловой системы есть ряд особенностей, о которых мы сейчас и поговорим.

Поддержка `ext4` как стабильной файловой системы появилась в ядре Linux версии 2.6.28. Если сравнивать эту файловую систему с `ext3`, то производительность и надежность новой файловой системы существенно увеличена, а максимальный размер раздела доведен до 1024 Пбайт (петабайт, 1024 Пбайт = 1 Эбайт, эксабайт). Максимальный размер файла — более 2 Тбайт. Ресурс Phoronix (www.phoronix.com) произвел тестирование новой файловой системы на SSD-накопителе (такие накопители устанавливаются на современные ноутбуки) — результат, как говорится, налицо: `ext4` почти в два раза превзошла файловые системы `ext3`, `XFS`, `JFS` и `ReiserFS`.

4.7.1. Сравнение ext3 и ext4

Описание особенностей файловой системы ext4 и ее преимуществ по сравнению с ext3 сведены в табл. 4.6.

Таблица 4.6. Особенности ext4

Особенность	Комментарий
Увеличенный размер файла и файловой системы	Для ext3 максимальный размер файловой системы составляет 32 Тбайт, а файла — 2 Тбайт, но на практике ограничения были более жесткими. Так, в зависимости от архитектуры, максимальный размер тома составлял до 2 Тбайт, а максимальный размер файла — до 16 Гбайт. В случае с ext4 максимальный размер тома составляет 1 эксабайт (EiB) — это 2^{60} байт. Максимальный размер файла — 16 Тбайт. Такие объемы информации пока не нужны обычным пользователям, однако весьма пригодятся на серверах, работающих с большими дисковыми массивами
Экстенты	Основной недостаток ext3 — ее метод выделения места на диске. Дисковые ресурсы выделялись с помощью битовых карт свободного места, а такой способ не отличается ни скоростью, ни масштабируемостью. Получилось, что ext3 более эффективна для небольших файлов, но совсем не подходит для хранения больших файлов. Для улучшения выделения ресурсов и более эффективной организации данных в ext4 были введены экстенты. Экстент — это способ представления непрерывной последовательности блоков памяти. Благодаря использованию экстентов сокращается количество метаданных (служебных данных файловой системы), поскольку вместо информации о том, где находится каждый блок памяти, экстент содержит информацию о том, где находится большой список непрерывных блоков памяти. Для эффективного представления маленьких файлов в экстентах применяется уровневый подход, а для больших файлов используются деревья экстентов. Например, один индексный дескриптор может ссылаться на четыре экстента, каждый из которых может ссылаться на другие индексные дескрипторы и т. д. Такая структура является мощным механизмом представления больших файлов, а также более защищена и устойчива к сбоям
Отложенное выделение пространства	Файловая система ext4 может отложить выделение дискового пространства до последнего момента, что увеличивает производительность системы
Контрольные суммы журналов	Контрольные суммы журналов повышают надежность файловой системы
Большее количество каталогов	В ext3 могло быть максимум 32 000 каталогов, в ext4 количество каталогов не ограничивается
Дефрагментация «на лету»	Файловая система ext3 не особо склонна к фрагментации, но все же такое неприятное явление имеется. В ext4 производится дефрагментация «на лету», что позволяет повысить производительность системы в целом

Таблица 4.6 (окончание)

Особенность	Комментарий
Наносекундные временные метки	В большинстве файловых систем временные метки (timestamp) устанавливаются с точностью до секунды, в ext4 точность повышенна до наносекунды. Кроме того, ext4 поддерживает временные метки до 25 апреля 2514 года, в отличие от ext3 (18 января 2038 года)

4.7.2. Совместимость с ext3

Файловая система ext4 является прямо и обратно совместимой с ext3, однако все же существуют некоторые ограничения. Предположим, что у нас на диске имеется файловая система ext4. Ее можно смонтировать и как ext3, и как ext4 (это и есть прямая совместимость) — и тут ограничений никаких нет. А вот с обратной совместимостью не все так безоблачно — если файловую систему ext4 смонтировать как ext3, то она будет работать без экстентов, что снизит ее производительность.

4.8. Особые команды

4.8.1. Команда *mkfs*: создание файловой системы

С помощью команды *mkfs* мы можем создать файловую систему на разделе жесткого диска — например, так: *mkfs.ext2 /dev/hda1*.

Вообще, создать файловую систему нужного типа (если эта файловая система поддерживается ядром вашей системы) можно с помощью команды *mkfs.<имя_файловой_системы>*, например:

```
mkfs.ext3
mkfs.vfat
mkfs.reiserfs
```

Подробнее прочитать об этом можно, введя команду *man mkfs.<имя_файловой_системы>*.

4.8.2. Команда *fsck*: проверка и восстановление файловой системы

Для проверки файловой системы служит команда *fsck*. Использовать ее нужно так:

```
fsck <раздел>
```

Например:

```
fsck /dev/sda5
```

Перед использованием этой команды следует размонтировать проверяемую файловую систему. Если надо проверить корневую файловую систему, то необходимо загрузиться с LiveCD и запустить *fsck* для проверки необходимого раздела.

Если же жесткий диск «посыпался», т. е. на нем появились плохие блоки, нужно, не дожидаясь полной потери данных, выполнить следующие действия:

1. Выполнить команду `fsck -c <раздел>` (эта команда пометит плохие блоки).
2. Сделать резервную копию всех важных данных.
3. Отправиться в магазин за новым жестким диском и перенести данные со старого жесткого диска на новый. Проверить жесткий диск на наличие плохих секторов можно программой `badblocks`.

ПРИМЕЧАНИЕ

Программа `fsck` может проверять не только файловые системы `ext2/ext3`. Для проверки, например, `vfat` можно использовать команду `fsck.vfat <раздел>`.

Для восстановления «упавшей» таблицы разделов можно использовать программу `gpart`. Только применяйте ее осторожно и внимательно читайте все сообщения, выводимые этой программой.

4.8.3. Команда *chroot*: смена корневой файловой системы

Предположим, мы установили Windows после установки Linux, и программа установки Windows перезаписала начальный загрузчик. Теперь Windows загружается, а Linux — нет. Что делать? Нужно загрузиться с LiveCD и выполнить команду:

```
# chroot <раздел, содержащий корневую файловую систему>
```

Например, если Linux была установлена в раздел `/dev/sda5`, то надо ввести команду:

```
# chroot /dev/sda5
```

Эта команда сменит корневую файловую систему (т. е. вы загрузите ядро Linux с LiveCD), а затем сделает подмену корневой файловой системы. Вам останется только ввести команду записи загрузчика (например, `lilo`) для восстановления начального загрузчика.

4.8.4. Установка скорости CD/DVD

Команда `hdparm` позволяет ограничить скорость оптического привода (CDROM/DVDROM). Иногда нужно ограничить эту скорость, чтобы информация была считана без ошибок (как правило, если поверхность носителя информации немного повреждена). Рассмотрим команду ограничения скорости:

```
# hdparm -q -E<множитель> <устройство>
```

Множитель — это и есть скорость. Например, 1 соответствует скорости 150 Кбайт/с для CD, 1385 Кбайт/с для DVD. Чтобы установить вторую (300 Кбайт/с) скорость чтения для CD, используется команда:

```
# hdparm -q -E2 /dev/cdrom
```

Для ограничения скорости DVD можно выполнить следующую команду:

```
# hdparm -q -E1 /dev/dvd
```

4.8.5. Монтирование каталога к каталогу

В Linux можно подмонтировать каталог к каталогу, а не только каталог к устройству. Делается это с помощью все той же команды `mount`, запущенной с параметром `--bind`:

```
# mount --bind исходный_каталог каталог_назначения
```

4.8.6. Команды поиска файлов

Для поиска файлов в Linux используется команда `find`. Это весьма мощная команда со сложным синтаксисом, и далеко не всегда она нужна обычному пользователю. Ему намного проще будет установить файловый менеджер `mc` и использовать встроенную функцию поиска.

Но команду `find` мы все же рассмотрим, по крайней мере ее основы. Синтаксис команды следующий:

```
find список_поиска выражение
```

Мощность команды `find` заключается во множестве самых разных параметров поиска, которые не так легко запомнить, — их просто много. К тому же `find` может выполнять команды для найденных файлов. Например, вы можете найти временные файлы и сразу удалить их.

Подробно опции команды `find` мы рассматривать не будем — это вы можете сделать самостоятельно с помощью команды `man find`. Зато мы покажем несколько примеров использования этой команды.

Найти файлы с именем, например, `a.out` (точнее, в имени которых содержится фрагмент `a.out`), начав поиск с корневого каталога (/):

```
find / -name a.out
```

Найти файлы по маске `*.txt`:

```
find / -name '*.txt'
```

Найти файлы нулевого размера, начав поиск с текущего каталога (.):

```
find . -size 0c
```

Впрочем, для поиска пустых файлов намного проще использовать параметр `-empty`:

```
find . -empty
```

Найти файлы, размер которых от 100 до 150 Мбайт, поиск производить в домашнем каталоге и всех его подкаталогах:

```
find ~ -size +100M -size -150M
```

Найти все временные файлы и удалить их (для каждого найденного файла будет запущена команда `rm`):

```
# find / -name *.tmp -ok rm {} \;
```

Вместо параметра `-ok` можно использовать параметр `-exec`, который также запускает указанную после него команду, но не запрашивает подтверждение выполнения этой команды для каждого файла.

Кроме команды `find` можно использовать команды `which` и `locate`. Первая выводит полный путь к программе или к сценарию, если программа или сценарий находится в списке каталогов, заданном в переменной окружения `PATH`:

```
which sendmail
```

Команда `locate` ищет в базе данных демона `located` файлы, соответствующие заданному образцу. Недостаток этой команды в том, что `located` имеется далеко не во всех дистрибутивах, поэтому демона `locate` у вас может и не быть. Зато если `located` имеется и запущен, поиск файлов будет осуществляться быстрее, чем с помощью `find`.

4.9. Многофункциональная команда `dd`

Команда `dd` в Linux особенная. Ее синтаксис не похож на синтаксис других команд, а благодаря ее широкой функциональности она заслуживает отдельного рассмотрения, что мы здесь и сделаем.

Команда `dd` хотя и весьма старая, но тем не менее очень полезная. Ее история начинается вместе с историей UNIX — 1 января 1970 года. Тогда эта команда использовалась для работы с ленточными накопителями. Почему ее назвали именно так — `dd` — теперь уже никто не вспомнит. Встречаются различные варианты расшифровки этого названия (вроде `data definition` — определение данных), но так это или нет, точно не известно.

О команде `dd` можно было бы написать целую главу, в которой подробно рассмотреть ее синтаксис, историю и даже придумать несколько десятков вариантов названия. Но я этого делать не стану — синтаксис `dd` описан в `man` (обязательно ознакомьтесь со страницей руководства, даже если вы полностью прочитаете этот раздел!), поэтому имеет смысл сразу перейти к практическим примерам ее использования.

Но прежде всего хочу предупредить вас, что команда очень опасна, и ее неправильное применение может полностью уничтожить данные на жестком диске вашего компьютера — и не только на разделе Linux! Поэтому я снимаю с себя за это всякую ответственность, а вы должны принять, что используете команду `dd` на свой страх и риск. Если вас это не устраивает, тогда забудьте о ней и перейдите к чтению следующего раздела.

Да, повторяю, команда `dd` весьма опасна. Но если ее применять осторожно и осмысленно, можно извлечь из нее много пользы. Далее мы рассмотрим типичные примеры использования команды `dd`.

4.9.1. Копирование файлов с помощью команды *dd*

Обычно никто не копирует файлы командой *dd* — для этого существует команда *cp*. Но при желании можно выполнить копирование и командой *dd*. Здесь мы рассмотрим копирование файлов с ее помощью, чтобы немного разобраться с синтаксисом команды, который проще изучить на безобидных примерах, которые не повредят ваши данные. Сразу отмечу, что для выполнения команды *dd* нужны права *root*, поэтому здесь и далее приглашение командной строки будет иметь вид *#*. В тех дистрибутивах, где вы не можете зарегистрироваться как *root*, придется выполнять команду *dd* или через команду *sudo*, или через команду *su*.

Чтобы скопировать файл */home/evg/example.txt* в файл */home/evg/example.bak*, выполните следующую команду:

```
# dd if=/home/evg/example.txt of=/home/evg/example.bak
```

Параметр *if* (*input file*) задает входной файл, параметр *of* (*output file*) — выходной файл. Как видите, синтаксис для Linux несколько необычен и имеет вид «*опция=значение*». Была бы *dd* обычной командой Linux, ее параметры выглядели бы примерно так:

```
-if /home/evg/example.txt -of /home/evg/example.bak
```

Ничего интересного, в общем-то, мы не сделали — просто скопировали один файл в другой, что можно было бы достичь и средствами команды *cp*. Но интересное только начинается. Команде *dd* все равно, что будет входным, а что выходным файлом, она работает с необработанными данными (*raw*) на самом низком уровне — на уровне секторов жесткого диска. Именно поэтому *dd* так опасна. Ведь она может запросто взять секторы одного диска и заменить ими секторы другого диска. В результате вместо данных на диске останется нечитаемая «каша».

Параметры *if* и *of* правильнее называть *входным* и *выходным буферами*. Величина буфера задается параметром *bs* (*block size*). По умолчанию размер блока равен 512 байтам, что подходит для большинства задач.

Параметр *count* задает число блоков, которое должно быть считано. Давайте сейчас попробуем считать в файл главную загрузочную запись (MBR) жесткого диска вашего компьютера:

```
# dd if=/dev/sda of=mbr.bak bs=512 count=1
```

Эта команда скопировала всего 512 байтов первого жесткого диска (*/dev/sda*) вашего компьютера и поместила их в файл *mbr.bak*. Теперь скопируйте этот файл на флешку и храните ее в безопасном месте. Если что-либо повредит MBR этого жесткого диска, вы всегда сможете восстановить его такой командой:

```
# dd if=/dev/sdb1/mbr.bak of=/dev/sda bs=512 count=1
```

где */dev/sdb1* — это имя флешки.

Вот вам и первый практический и полезный пример использования команды *dd*!

4.9.2. Разделение файла на несколько частей

Бывает так, что большой файл не полностью помещается на носитель, и его нужно разбить на несколько частей. Следующие две команды разделят файл размером 1000 Мбайт на два файла по 500 Мбайт (part1 и part2):

```
dd if=file1000 of=part1 bs=1M count=500  
dd if=file1000 of=part2 bs=1M skip=500
```

Параметр `skip` задает смещение — т. е. вторая команда должна пропустить первые 500 Мбайт файла, т. к. они уже вошли в первую часть (part1).

В качестве домашнего задания предлагаю вам соединить файлы part1 и part2 в один файл, равный исходному.

4.9.3. Создание резервной копии жесткого диска

Представим, что у нас есть два одинаковых жестких диска: `/dev/sda` и `/dev/sdb` — и вам нужно создать резервную копию жесткого диска `/dev/sda` на диске `/dev/sdb`. Это делается следующей командой:

```
# dd if=/dev/sda of=/dev/sdb conv=noerror, sync bs=4k
```

Обратите внимание — мы установили размер блока в `4k` (4096 байт), чтобы повысить производительность копирования. Если размер блока не задать, то будет использовано значение по умолчанию (512 байтов), в итоге число операций чтения/записи будет увеличено в 8 раз, что негативно отразится на производительности.

4.9.4. Создание архива с резервной копией всего жесткого диска

Как правило, двух одинаковых жестких дисков у нас нет, но необходимость создания резервной копии никуда не делась. Сейчас мы создадим архив с резервной копией жесткого диска `/dev/sda`. Сама команда `dd` не умеет создавать архивы, но мы можем перенаправить ее вывод программе `gzip`, которая и сделает основную работу.

Итак, для резервного копирования выполните следующую команду:

```
# dd if=/dev/sda | gzip > /mnt/sdb1/backups/sda.img.gz
```

Созданный архив будет записан в файл `/mnt/sdb1/backups/sda.img.gz`. В случае необходимости архив можно развернуть обратно на диск `/dev/sda` с помощью следующей команды (только выполнять ее нужно, предварительно загрузившись с LiveCD):

```
# gzip -dc /mnt/sdb1/sda.img.gz | dd of=/dev/sda
```

4.9.5. Уничтожение всех данных раздела жесткого диска

Командой dd легко нечаянно уничтожить данные на всем жестком диске. Но сейчас мы разберемся, как уничтожить данные намеренно и целенаправленно, т. е. на определенном разделе жесткого диска. Ведь не секрет, что ни простое удаление файлов, ни даже форматирование диска не дает гарантии безопасности — стертые таким образом данные удается восстановить. А с помощью команды dd можно перезаписать все секторы жесткого диска нулями или случайными числами, и после такой процедуры уже никто и никогда старые данные не восстановит, поскольку они будут безвозвратно затерты. Итак, две команды:

```
# dd if=/dev/zero of=/dev/sdb2 bs=1M  
# dd if=/dev/urandom of=/dev/sdb2 bs=1M
```

Первая перезаписывает все секторы раздела sdb2 нулями, а вторая — случайными числами. На момент ввода любой из этих команд устройство /dev/sdb2 должно быть размонтировано. После ввода команды подмонтировать устройство будет невозможно. И чтобы снова использовать этот раздел, нужно будет его отформатировать, т. е. заново создать файловую систему (как это сделать командой mkfs, было показано ранее), после чего раздел можно будет снова подмонтировать.

4.10. Команда du

Команда du позволяет узнать, сколько места на диске занимает тот или иной файл или каталог. С файлами всегда проще, поскольку можно ввести команду ls -l, и вы увидите листинг каталога и размеры всех находящихся в нем файлов. Узнать, сколько же занимает весь каталог, можно с помощью команды du.

Синтаксис команды du:

```
du [OPTION]... [FILE] [directory]
```

По умолчанию команда du отображает размер каталога в байтах. Конечно, нам удобнее использовать мегабайты, поэтому не забываем добавлять параметр -m. Но еще лучше использовать параметры -ah. В этом случае программа будет выводить размеры файлов (a) в удобном для человека виде (h):

```
du -ah /home/evgis
```

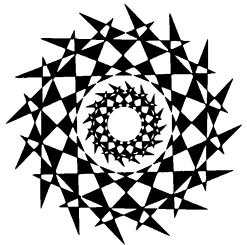
Вывод будет примерно таким:

```
...  
4.0K  /home/evgis/test.xml  
4.0K  /home/evgis/date.php  
8.0K  /home/evgis/change_collection_id.php  
324K  /home/evgis/Good taras/taras_styles.css  
328K  /home/evgis/Good taras
```

```
4.0K    /home/evgis/order.php
4.1G    /home/evgis
```

То есть выводится размер каждого файла и каталога, а затем — общий размер заданного каталога. Если в файле много каталогов и подкаталогов, вы получите вывод-простыню. Если вам такого вывода не нужно, а надо просто узнать, сколько места занимает тот или иной каталог, используйте параметры `-csh`:

```
evgis@hosting-test-6:~$ du -csh /home/evgis
4.1G    /home/evgis
4.1G    total
```



ГЛАВА 5

Процессы

5.1. Оболочки, команды и программы

По умолчанию в большинстве дистрибутивов используется оболочка `bash`. В настоящее время — это стандарт. Но, кроме нее, существуют и другие оболочки — например, `zsh` или `csh` (оболочка с С-подобным синтаксисом скриптов). Дополнительные оболочки могут быть установлены посредством менеджера пакетов (`apt` в `Debian/Ubuntu` или `dnf` в `Fedor/CentOS`). Список установленных оболочек хранится в файле `/etc/shells`.

При работе с оболочкой вы вводите различные команды. Например, `cat` — для просмотра файла, `ls` — для просмотра каталога и т. д.:

```
/bin$ ls -l bash cat ls grep
-rwxr-xr-x 1 root root 1113504 Jun  7  2021 bash
-rwxr-xr-x 1 root root   35064 Jan 18  2021 cat
-rwxr-xr-x 1 root root  219456 Sep 18  2021 grep
-rwxr-xr-x 1 root root  133792 Jan 18  2021 ls
```

В большинстве своем каждая команда представлена в виде отдельного исполняемого файла. Напомню, *исполнимым* является файл для которого установлен флаг выполнения (`x`) в наборе прав доступа. Как правило, исполнимым имеет смысл делать откомпилированную программу или же сценарий оболочки — иначе система просто не сможет выполнить этот файл. Для сценариев оболочки в первых строках сценария указывается, какая программа должна обработать тот или иной файл:

```
#!/bin/bash
```

Обратите внимание, что между `#`, `!` и путем к оболочке не должно быть пробелов. Также пробелов не должно быть и перед символом `#`. Приведенная запись говорит о том, что дальнейший контент скрипта должен быть обработан программой `/bin/bash`. Если же вы пишете свой сценарий в оболочке `csh`, то конструкция будет такой: `#!/bin/csh`. Также в качестве оболочки может выступать любая другая программа, способная обработать ваш скрипт. Система увидит запись `#!` и передаст этот файл на выполнение программе, указанной после `!`.

Сделать файл исполняемым можно так:

```
chmod +x test.sh
```

Подробно об этом мы поговорим в главе 12, когда будем рассматривать сценарии командной оболочки. А пока вернемся к оболочкам.

Как уже было отмечено, список установленных оболочек хранится в файле `/etc/shells`:

```
$ cat /etc/shells
# /etc/shells: valid login shells
/bin/sh
/bin/bash
/bin/rbash
/bin/dash
/usr/bin/tmux
/usr/bin/screen
```

Но, как показывает практика, не все программы из приведенного списка являются командными оболочками в прямом смысле этого слова. Некоторые программы указываются здесь, чтобы была возможность использовать их при назначении в виде оболочки пользователя в файле `/etc/passwd` — в некоторых «узких» сценариях это нужно. Например, самый простой способ запретить вход пользователя в консоль — это установить для него в файле `/etc/passwd` программу `/bin/false` в качестве оболочки. Но тогда ее нужно не забыть указать в файле `/etc/shells`.

Узнать, в какой оболочке работаете вы сейчас, можно с помощью переменной окружения `$SHELL` (в главе 12 вы познакомитесь с другими переменными окружения):

```
~$ echo $SHELL
/bin/bash
```

5.2. Родительский и дочерний процессы

В Linux используется иерархическая структура процессов — у каждого процесса есть родитель. Когда вы запускаете какую-то команду в `bash`, то автоматически эта команда становится дочерним по отношению к оболочке `bash` процессом. Да, каждый раз, когда вы вводите команду, вы создаете дочерний процесс.

Сама оболочка также является дочерним процессом. Например, если вы заходите по SSH на свой сервер, то именно процесс `sshd` запускает оболочку, поэтому для оболочки родительским процессом будет `ssh`:

```
└─sshd,1293 -D
  └─sshd,38681
    └─sshd,38803
      ├─bash,38805
      └─pstree,42271 -a -p
```

Посмотрим на фрагмент вывода команды `pstree -a -p` (рис. 5.1). Программа `pstree` отображает дерево процессов в Linux. Параметр `-a` заставляет ее выводить все запущенные процессы, а параметр `-p` — выводить их идентификаторы (PID).

```

denis@hosting-test-6:~$ pstree -a -p
systemd,1 maybe-ubiquity
├─accounts-daemon,982
│ ├─{accounts-daemon},1005
│ ├─{accounts-daemon},1187
├─agetty,1378 -o -p -- \\u --noclear tty1 linux
├─atd,980 -f
├─atop,54044 -R -w /var/log/atop/atop_20220220 600
├─atopacctd,1038
├─cron,1192 -f
│ ├─cron,1933 -f
│ │ └─sh,1943 -c...
│ │   └─grep,1948 -v Ran jobs by schedule
│ ├─cron,37898 -f
│ │ └─sh,37900 -c...
│ │   └─grep,37905 -v Ran jobs by schedule
│ ├─cron,42219 -f
│ │ └─sh,42221 -c...
│ │   └─grep,42226 -v Ran jobs by schedule
│   └─php7.2,42224 /var/www/www/bin/magento cron:run
dbus-daemon,1162 --system --address=systemd: --nofork --nopidfile --systemd-activation--syslog-
irqbalance,1064 --foreground
└─{irqbalance},1105
lvmtd,548 -f
lxcfs,1096 /var/lib/lxcfs/
├─{lxcfs},1151
├─{lxcfs},1152
└─{lxcfs},48160
mysqld,1387 --daemonize --pid-file=/run/mysqld/mysqld.pid
├─{mysqld},1435
├─{mysqld},1622
├─{mysqld},1623
├─{mysqld},1624
├─{mysqld},1625
└─{mysqld},1626

```

Рис. 5.1. Команда `pstree -a -p`

Праородитель всех процессов — на вершине иерархии — система с PID = 1. PID планировщика задач — 1192, а его дочерних процессов — 1933, 37898 и 42219. Но подробно о идентификаторах процессов — в следующем разделе.

5.3. Команды *kill*, *killall*, *xkill* и *ps*

Каждому процессу в Linux присваивается уникальный номер — идентификатор процесса (PID, Process ID). Зная ID процесса, вы можете управлять процессом, а именно — завершить процесс или изменить его приоритет. Принудительное завершение процесса необходимо, если процесс завис и его нельзя завершить обычным образом. А изменение приоритета может понадобиться, если вы хотите, чтобы процесс доделал свою работу быстрее.

Предположим, у вас зависла какая-то программа — например, файловый менеджер mc. Хотя это и маловероятно (не помню, чтобы он когда-нибудь зависал), но для примера пусть будет так. Принудительно завершить («убить») процесс можно с помощью команды `kill`. Формат ее вызова следующий:

`kill [параметры] PID`

PID (Process ID) — это идентификатор процесса, который присваивается процессу системой, и он уникален для каждого процесса. Но мы знаем только имя процесса (имя команды), но не знаем идентификатор процесса. Узнать идентификатор процесса позволяет программа ps. Предположим, что mc находится на первой консоли. Поскольку он завис, вы не можете больше использовать эту консоль, и вам нужно переключиться на вторую консоль (клавиатурной комбинацией <Alt>+<F2>). Зарегистрировавшись на второй консоли, введите команду ps. Она выведет список процессов, запущенных на второй консоли, — это будет bash и сам ps (рис. 5.2).

Чтобы добраться до нужного нам процесса (mc), который запущен на первой консоли, введите команду ps -a или ps -U root. В первом случае вы получите список процессов, запущенных вами, а во втором — список процессов, запущенных от вашего имени (я предполагаю, что вы работаете под именем root). Обратите внимание: вы сами запустили процессы mc и ps (рис. 5.3), а от вашего имени (root) система запустила множество процессов. Следует заметить, что программа ps выводит также имя терминала (tty1), на котором запущен процесс. Это очень важно — если на разных консолях у вас запущены одинаковые процессы, можно легко ошибиться и завершить не тот процесс.

```
host login: root
Password:
Last login: Fri Aug  4 01:29:58 on tty1
[root@host ~]# ps
  PID TTY      TIME CMD
 2448 tty2    00:00:00 bash
 2521 tty2    00:00:00 ps
[root@host ~]# -

```

Рис. 5.2. Список процессов на текущей консоли

```
[root@host ~]# ps
  PID TTY      TIME CMD
 2448 tty2    00:00:00 bash
 2521 tty2    00:00:00 ps
[root@host ~]# ps -a
  PID TTY      TIME CMD
 2404 tty1    00:00:00 mc
 2581 tty2    00:00:00 ps
[root@host ~]# -

```

Рис. 5.3. Определение PID программы mc

Теперь, когда мы знаем PID нашего процесса, мы можем его «убить»:

```
# kill 2484
```

Перейдите на первую консоль после выполнения этой команды — mc на ней уже не будет. Если выполнить команду ps -a, то в списке процессов mc тоже не окажется.

Проще всего вычислить PID процесса с помощью следующей команды:

```
# ps -ax | grep <имя>
```

Например: # ps -ax | grep firefox.

Вообще-то все эти действия, связанные с вычислением PID процесса, мы рассмотрели только для того, чтобы познакомиться с командой ps. Так что, если вы знаете только имя процесса, гораздо удобнее использовать команду:

```
# killall <имя процесса>
```

Но имейте в виду, что она завершит все экземпляры заданного процесса. А вполне может быть, что у нас на одной консоли находится mc, который нужно «убить», а на другой — нормально работающий mc. Команда killall «убьет» оба процесса.

При выполнении команд `kill` и `killall` нужно помнить, что если вы работаете от имени обычного пользователя, они могут завершить только те процессы, которые принадлежат вам. А если вы работаете от имени пользователя `root`, то можете завершить любой процесс в системе.

«Убить» процесс можно по-разному. «Убийство» процесса на самом деле сопровождается посылкой процессу определенного системного сигнала. Вот наиболее распространенные сигналы, используемые для завершения процесса:

- HUP (1) — полезен для программ-демонов. Получив этот сигнал, демон (служба) обязан перезапуститься (или перечитать файл конфигурации). Обычная программа, получив такой сигнал, завершается;
- INT (2) — прерывание. Именно этот сигнал посыпается, когда вы нажимаете комбинацию клавиш `<Ctrl>+<C>`;
- KILL (9) — аварийное завершение процесса. Программа не может игнорировать этот сигнал или как-то его обработать;
- TERM (15) — «мягкое» завершение программы (именно этот сигнал посыпается программой `kill` по умолчанию). Программа, получив этот сигнал, должна корректно завершиться — например, освободить все занятые ресурсы и сохранить все данные. Обычно такой сигнал равносителен нормальному выходу из программы. В случае с «сильно» зависшими процессами толку от этого сигнала мало, нужно использовать KILL;
- STOP (19) — приостанавливает выполнение процесса. Этот сигнал нельзя обрабатывать или игнорировать;
- CONT (18) — возобновляет выполнение процесса.

Чтобы полностью «убить» процесс, нужно передать ему сигнал 9 (KILL):

```
# kill -9 <PID>
```

Кроме команды `kill`, пользователи, предпочитающие графический интерфейс, могут применять программу `xkill`, позволяющую «убить» графическую программу. Введите команду `xkill` — указатель мыши изменится на череп, который нужно навести на окно зависшей программы и щелкнуть мышью (иначе, если окно не зависло, зачем его «убивать»?), — процесс, относящийся к выбранному окну, будет немедленно завершен.

5.4. Программа `top`: кто больше всех расходует процессорное время?

Иногда бывает, что система ужасно тормозит — весь день работала нормально, а вдруг начала притормаживать. Если вы даже не догадываетесь, из-за чего это случилось, вам нужно использовать программу `top` (рис. 5.4) — она выводит список процессов с сортировкой по процессорному времени. То есть на вершине списка

будет процесс, который занимает больше процессорного времени, чем сама система. Вероятно, из-за него и происходит эффект торможения.

На рис. 5.4 показано, что больше всего процессорного времени (0.3%) занимает программа top. Конечно, в реальных условиях все будет иначе. Выйти из программы top можно, нажав клавишу <Q>. Кроме <Q> действуют следующие клавиши:

- <U> — показывает только пользовательские процессы (т. е. те процессы, которые запустил пользователь, под именем которого вы работаете в системе);
- <D> — изменяет интервал обновления;
- <F> — изменяет столбец, по которому сортируются задачи. По умолчанию задачи сортируются по столбцу %CPU, т. е. по процессорному времени, занимающему процессом;
- <H> — получить справку по остальным командам программы top.

PID	USER	PR	NI	VIRT	RES	SHR	%CPU	TIME	COMMAND
2599	root	16	0	1996	1012	804	R	0.3	0.5
1	root	16	0	1564	548	472	S	0.0	0.3
2	root	RT	0	0	0	0	S	0.0	0.0
3	root	34	19	0	0	0	S	0.0	0.0
4	root	10	-5	0	0	0	S	0.0	0.0
5	root	16	-5	0	0	0	S	0.0	0.0
6	root	11	-5	0	0	0	S	0.0	0.0
8	root	20	-5	0	0	0	S	0.0	0.0
61	root	10	-5	0	0	0	S	0.0	0.0
93	root	28	0	0	0	0	S	0.0	0.0
94	root	15	0	0	0	0	S	0.0	0.0
96	root	16	-5	0	0	0	S	0.0	0.0
95	root	25	0	0	0	0	S	0.0	0.0
684	root	16	0	0	0	0	S	0.0	0.0
766	root	13	-5	0	0	0	S	0.0	0.0
775	root	18	0	0	0	0	S	0.0	0.0
784	root	16	0	0	0	0	S	0.0	0.0
924	root	15	-4	1564	496	420	S	0.0	0.3

Рис. 5.4. Программа top

Назначение столбцов программы top указано в табл. 5.1.

Таблица 5.1. Назначение столбцов программы top

Столбец	Описание
PID	Идентификатор процесса
USER	Имя пользователя, запустившего процесс
PR	Приоритет процесса
NI	Значение nice (см. разд. 5.5)

Таблица 5.1 (окончание)

Столбец	Описание
VIRT	Виртуальная память, использованная процессом (в килобайтах)
RES	Размер процесса, не перемещенный в область подкачки (в килобайтах). Этот размер равен размерам сегментов кода и данных, т. е. RES = CODE + DATA
S	Состояние процесса: <ul style="list-style-type: none"> • R — выполняется; • S — «спит» (режим ожидания), в этом состоянии процесс выгружен из оперативной памяти в область подкачки; • D — «непрерываемый сон» (uninterruptible sleep), из такого состояния процесс можно вывести только прямым сигналом от оборудования; • T — процесс в состоянии трассировки или остановлен; • Z («зомби») — специальное состояние процесса, когда сам процесс уже завершен, но его структура еще осталась в памяти
%CPU	Занимаемое процессом процессорное время
%MEM	Использование памяти процессом
TIME+	Процессорное время, израсходованное с момента запуска процесса
COMMAND	Команда, которая использовалась для запуска процесса (обычно имя исполняемого файла процесса)

5.5. Команды *nice* и *renice*: изменение приоритета процесса

Предположим, что вы работаете с видео, и вам нужно перекодировать файл из одного видеоформата в другой. Конвертирование видео занимает много процессорного времени, а хотелось бы все сделать как можно быстрее и уйти домой пораньше. Тогда вам поможет программа *nice* — она позволяет запустить любую программу с указанным приоритетом. Ясно — чем выше приоритет, тем быстрее будет выполняться программа. Формат вызова программы следующий:

```
sudo nice -n <приоритет> команда аргументы
```

Максимальный приоритет задается числом -20, а минимальный — числом 19. Приоритет по умолчанию равен 10.

Если процесс уже запущен, тогда для изменения его приоритета можно использовать команду *renice*:

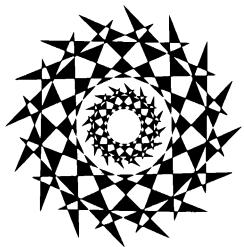
```
sudo renice -n <приоритет> -p PID
```

5.6. Команда *fuser*: кто открыл ресурс?

Иногда очень полезно знать, какая программа использует тот или иной порт или файл (каталог). В Windows для этого нужно устанавливать дополнительные программы сторонних разработчиков, в Linux же применяется штатная команда *fuser*. Рассмотрим примеры ее использования:

```
$ fuser -va 22/tcp  
$ fuser -va /home/evg/report.doc  
$ sudo fuser -k -TERM -m /home/den/report.doc
```

Первая команда показывает PID процесса, использующего порт 22. Вторая демонстрирует, какой процесс имеет доступ к файлу */home/evg/report.doc*. А третья команда завершает все процессы (отправляет им сигнал TERM), которые используют файл */home/den/report.doc*.



ГЛАВА 6

Различные способы выполнения команд

Теперь, когда в вашем «командном» запасе уже есть приличный список команд, настало время научиться... их выполнять. Позвольте, а разве мы до этого их не выполняли? Да, выполняли, но в очень простой форме, указывая или просто команду с параметрами, например:

```
ls -l
```

или перенаправление ввода/вывода, например:

```
cut -f1 report.txt | sort | uniq -c | sort -nr
```

По сути, это тоже простые команды, которые просто заключены в конвейер. В этой главе вы познакомитесь с несколькими техниками выполнения команд. У каждой из них есть свои достоинства и недостатки, которые и будут рассмотрены далее.

6.1. Списки

Список — это последовательность команд в одной командной строке. Мы уже сталкивались с одним из списков — конвейер (перенаправление ввода/вывода посредством символа `|`). Списки бывают условными и безусловными. В первом случае каждая следующая команда зависит от результата выполнения предыдущей. Во втором случае команды просто выполняются одна за другой.

6.1.1. Условные списки

Представим, что вы хотите создать файл `report.txt` в каталоге `dir`. Это можно сделать с помощью двух команд:

```
$ cd dir  
$ touch report.txt
```

Обратите внимание, что вторая команда зависит от успеха выполнения первой команды. Если каталог `dir` не существует, то нет смысла выполнять вторую команду. И если вы вводите команды вручную, то не будете вводить вторую команду, если первая не была успешно выполнена. Не так ли?

Оболочка `bash` позволяет вам выполнить вторую команду в случае успешного выполнения первой команды. Для этого используются символы `&&`:

```
$ cd dir && touch report.txt
```

Вторая команда будет выполнена, только если выполнение первой команды завершится успешно.

Символы `&&` можно использовать с произвольным количеством команд, а не только с двумя. Представим, что нам нужно перед редактированием файла — например, `.bashrc`, сделать его резервную копию, которую мы хотим удалить сразу после выхода из редактора. Для этого обычно выполняются следующие команды:

```
$ cp .bashrc .bashrc.bk  
$ nano .bashrc  
$ rm .bashrc.bk
```

Вместо последовательного ввода этих команд можно выполнить одну такую:

```
$ cp .bashrc .bashrc.bk && nano .bashrc && rm .bashrc.bk
```

В этом случае система попытается создать резервную копию файла `.bashrc`. Если она будет создана удачно, запустится редактор `nano` для редактирования `.bashrc`. Если что-то пошло не так (мало ли: место на диске закончилось, или нет такого файла), редактор не будет запущен, и выполнение списка будет прервано. При успешном завершении работы редактора — например, когда вы отредактировали файл и сохранили результат, будет выполнена последняя команда — удаление резервной копии.

Такой пример мало когда используется. А вот пример с `Git` реализуется гораздо чаще. Рассмотрим команду:

```
git add . && git commit -m "исправлена ошибка" && git push
```

Здесь мы добавляем текущий каталог, затем делаем коммит, а после отправляем в репозиторий исправленный код (`git push`).

Что считается успешным выполнением? Каждая программа в `Linux` в случае успешного выполнения возвращает так называемый *код возврата*. По договоренности, если этот код равен 0, то выполнение успешное. Любой ненулевой код возврата — возникла ошибка.

6.1.2. Безусловные списки

Безусловные списки, или просто списки, используются для последовательного выполнения команд, независимо от результата выполнения предыдущей команды. Команды просто выполняются друг за другом, а результат завершения предыдущей команды ни на что не влияет.

Пример:

```
$ sleep 3600; cp -a /home/user /mnt/backup
```

Здесь мы ждем 3600 секунд (1 час), а затем просто делаем бэкап всего каталога `/home/user` в каталог `/mnt/backup`. Зачем ждать час? Вполне вероятно, что запущен

какой-то процесс, который завершится плюс/минус через час (лучше время в sleep указывать с каким-то запасом), а вам нужно отойти, и вы не хотите ждать. Если это действие будет производиться на регулярной основе — например, в конце дня, то лучше команду копирования добавить в расписание планировщика cron. Но если действие надо выполнить разово, но с определенной задержкой выполнения, тогда сгодится такой вариант, — и при этом вам не нужно будет дважды редактировать таблицу расписания cron (один раз — чтобы добавить команду в расписание, и второй — чтобы ее оттуда удалить).

6.2. Подстановка

6.2.1. Подстановка команды

Представим, что у вас есть тысячи файлов с информацией о музыкальных композициях: название композиции, исполнитель, альбом и текст песни. Пример каждого такого файла:

Название: Title01

Исполнитель: Artist01

Альбом: Album

Текст песни

Найти все файлы с композициями Artist01 (чтобы не делать никому рекламы, назовем исполнителя именно так) можно командой:

```
$ grep -l "Исполнитель: Artist01" *.txt
song1011.txt
song2084.txt
song5401.txt
```

Затем вы все эти файлы хотите поместить в отдельный каталог artist01:

```
mkdir artist01
mv song1011.txt artist01
mv song2084.txt artist01
mv song5401.txt artist01
```

Если файлов немного, то в этом наборе команд нет ничего страшного. А вот если файлов много, то перемещать каждый отдельно взятый файл (еще и когда нет возможности воспользоваться масками * и ?) — дело хлопотное. К счастью, есть возможность выполнить так называемую *подстановку* с помощью конструкции:

```
$(любая команда)
```

Команда, которая делает то, что нам нужно, выглядит так:

```
$ mv $(grep -l "Исполнитель: Artist01" *.txt) artist01
```

Эта команда будет преобразована в следующую:

```
$ mv song1011.txt song2084.txt song5401.txt artist01
```

Вывод команды `grep` будет подставлен в команду `mv`, которая и выполнит перемещение файлов в нужный каталог.

6.2.2. Подстановка процесса

Подстановка команды, как было показано ранее, заменяет команду ее выводом. Подстановка процесса также заменяет команду ее выводом, но работает с выводом так, как будто бы он сохранен в файле. Это основное различие. Попробуем разобраться, что к чему.

Представим, что у нас есть каталог с документами формата PDF, названными от `1.pdf` до `1000.pdf`, но некоторые файлы таинственным образом исчезли, и вы хотите выяснить, какие именно. Для примера сгенерируем эти файлы:

```
$ mkdir /tmp/pdf && cd /tmp/pdf  
$ touch {1..1000}.pdf  
$ rm 3.pdf 200.pdf 900.pdf
```

Теперь попытаемся определить, какие файлы пропали. Самый простой способ — отсортировать содержимое каталога в алфавитном порядке и посмотреть, чего не хватает:

```
$ ls -1 | sort -n | less  
1.pdf  
2.pdf  
4.pdf  
5.pdf  
...
```

Такой вариант подойдет, если файлов не очень много. В случае с несколькими даже сотнями файлов вы можете легко упустить и не заметить отсутствующий файл. Второй вариант — сравнить вывод этой команды с полным списком `1.pdf ... 1000.pdf` и затем найти разницу командой `diff`. Для этого получим список файлов:

```
$ ls *.pdf | sort -n > /tmp/list
```

Затем сгенерируем полный список:

```
$ seq 1 1000 | sed 's/$/.pdf/' > /tmp/full
```

Теперь осталось сравнить эти два списка:

```
$ diff /tmp/list /tmp/full  
...  
> 3.pdf  
...  
> 200.pdf  
...  
> 300.pdf
```

Этот способ работает, но приходится выполнять слишком много шагов. Как сравнить два списка, не записывая их во временные файлы? Проблема еще в том, что команда `diff` не может сравнивать два списка со стандартного ввода — ей нужны

файлы как аргументы. Подстановка процесса решает проблему. Оба списка будут представлены как файлы — что и нужно diff. Синтаксис такой:

```
<(любая команда)
```

и результирующая команда:

```
$ diff <(ls *.pdf | sort -n) <(seq 1 1000 | sed 's/$/.pdf/') | grep '>' | cut -c3-  
3.pdf  
200.pdf  
300.pdf
```

Как работает подстановка процесса? Когда операционная система Linux открывает файл на диске, она представляет этот файл целым числом, называемым *файловым дескриптором*. Подстановка процесса имитирует файл, запуская команду и связывая ее вывод с дескриптором файла, поэтому вывод выглядит как дисковый файл с точки зрения программ, которые обращаются к нему.

6.3. Команда как строка

Существует еще одна техника запуска команд — представление команды в виде строки. Команду можно передать интерпретатору bash в качестве аргумента, перенаправить команды на stdin, отправить команды другому хосту по SSH и запустить последовательность команд с помощью команды xargs. Рассмотрим все эти способы.

6.3.1. Передача команды в виде аргумента

Команду можно передать в виде аргумента. Делается это с помощью параметра -c:

```
$ bash -c "ls -l"
```

Понятно, что вручную никто такую команду вводить не станет, поскольку проще сразу ввести ls -l. Как правило, такой трюк можно применять в сценариях, когда команда получена извне и хранится в какой-то переменной. Использовать или нет такой способ выполнения команды — решать только вам. Но знать о нем нужно — вдруг пригодится.

6.3.2. Перенаправление команды на стандартный ввод bash

Другая возможность заключается в перенаправлении команды на стандартный ввод bash — команда будет выполнена:

```
$ echo "ls -l" | bash
```

Этот метод отлично подходит, когда вам нужно запустить много одинаковых команд подряд. Например, вы можете в цикле формировать команды, которые будут отправлены на выполнение в bash.

6.4. Удаленное выполнение команды по SSH

Обычно подключение по SSH к удаленной машине выглядит так:

```
$ ssh example.com
```

После чего у пользователя запрашиваются логин и пароль или же происходит проверка его ключа. В случае успешной авторизации открывается приглашение командной строки удаленной машины, и у пользователя появляется возможность ввести команду, которая будет выполнена на «той стороне». В завершение работы нужно ввести команду `exit`, чтобы закрыть SSH-сесанс.

Все это слишком хлопотно, особенно, если нужно выполнить всего одну команду. Поэтому проще указывать эту команду в качестве параметра команды `ssh`. Например, команда:

```
$ ssh example.com ls
```

выведет содержимое каталога на удаленной машине. Но здесь нужно быть внимательным. Рассмотрим еще два примера:

```
$ ssh example.com ls > output  
$ ssh example.com "ls > output"
```

В первом случае на локальной машине будет создан файл `output`, в который будет помещен вывод команды `ls` на удаленной машине. Во втором случае файл `output` будет создан на удаленной машине, и в него будет помещен вывод `ls`. Другими словами, если ваша команда состоит из более чем одного слова, заключайте ее в кавычки.

6.5. Фоновое выполнение команд

Есть некоторые команды, которые можно успешно переместить в фоновый режим, дабы они не блокировали терминал. При этом команда продолжит выполняться в фоновом режиме (пусть более медленно), а вы сможете продолжить работу в текущем терминале, не открывая новое окно терминала (или не создавая новый сеанс, хотя при открытии нового окна также создается новый сеанс).

Для перевода команды в фоновый режим используйте амперсанд `&`:

```
$ wc -l очень_большой_файл &  
[1] 9248
```

Вывод команды говорит о том, что она перемещена в фоновый режим, а `1` — это номер фонового задания.

Когда фоновое задание будет выполнено, его состояние станет таким:

```
[1]+ Done      wc -l очень_большой_файл &
```

Если фоновое задание завершится с ошибкой, то вместо `Done` будет выведен код ошибки:

```
[1]+ Exit 1      wc -l очень_большой_файл &
```

Вот еще примеры:

```
$ command1 & command2 & command 3 &
$ command1 & command2 & command 3
```

В первом случае все три команды будут запущены в фоновом режиме, во втором — только первые две.

Управлять фоновыми заданиями можно с помощью команды `jobs`:

```
$ sleep 20 &
[1] 10778
$ jobs
[1]+  Running                  sleep 20 &
$ jobs
[1]+  Done                      sleep 20
```

Здесь мы выполнили команду `sleep 20` (она ничего не делает, просто ждет 20 секунд) в фоновом режиме. Затем просмотрели ее статус командой `jobs`, по истечении еще 20 секунд снова запросили статус фоновых заданий и увидели, что команда уже выполнена.

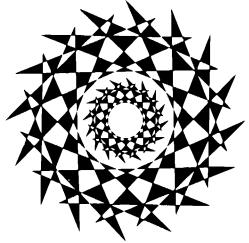
Перевести задание в обычный режим (`foreground`) можно командой `fg %n`, где `%n` — это номер задания, который вы видите в выводе команды `jobs`. Если фоновое задание всего одно, просто введите `fg`, и оно будет продолжено в обычном режиме.

Нужно также помнить о комбинации клавиш `<Ctrl>+<Z>`:

```
$ sleep 20
^Z
[1]+  Stopped                  sleep 20
$ jobs
[1]+  Stopped                  sleep 20
$ fg 1
sleep 20
$
```

Что произошло? Мы запустили команду `sleep 20` в обычном режиме. Затем нажали комбинацию `<Ctrl>+<Z>`, и команда перешла в фоновый режим, но ее выполнение было приостановлено — состояние `Stopped`. Далее мы перевели ее в обычный режим и через 20 секунд получили приглашение командной строки — выполнение команды завершено.

Используя механизм фонового выполнения, можно выполнять в фоновом режиме длительные процессы — например, архивирование информации, создание резервной копии, создание дампа базы данных и т. д.



ГЛАВА 7

Некоторые полезные команды

7.1. Команда seq

Команда `seq` выводит последовательность чисел из заданного диапазона. Например, выведем числа от 1 до 5:

```
seq 1 5
1
2
3
4
5
```

Если команда `seq` принимает три параметра, то второй задает инкремент, а первый и третий — границы диапазона. Пример:

```
seq 1 2 10
1
3
5
7
9
```

Здесь мы используем инкремент 2. Число 10 не будет напечатано, поскольку $9 + 2$ означает выход за рамки диапазона.

Инкремент может быть отрицательным. Пример:

```
seq 1 -1 3
3
2
1
0
```

По умолчанию для разделения выводимых значений используется символ новой строки. Но с помощью параметра `-s` можно задать альтернативный разделитель:

```
seq -s. 1 5
1.2.3.4.5 .
```

7.2. Фигурные скобки

Сгенерировать последовательность чисел можно и с помощью конструкции {} в самой bash:

```
echo {1..10}
1 2 3 4 5 6 7 8 9 10
echo {10..1}
10 9 8 7 6 5 4 3 2 1
echo {01..10}
01 02 03 04 05 06 07 08 09 10
```

А вот и более сложный пример: три параметра в скобках {x..y..z} генерируют значения от x до y с инкрементом z:

```
echo {1..1000..100}
1 101 201 301 401 501 601 701 801 901
```

Фигурные скобки могут генерировать последовательности букв, чего не может делать seq:

```
echo {A..D}
A B C D
```

Если вам не нужны пробелы между символами, их можно удалить:

```
echo {A..D} | tr -d ' '
ABCD
```

7.3. Команда *find*

Команда *find* очень полезная. Ее можно использовать не только для поиска файлов, но и для выполнения операций над найденными файлами.

С помощью *find* можно также выводить список файлов в каталоге — например:

```
find ~ -print
/home/user/projects/
/home/user/projects/main.c
/home/user/projects/library.c
...
```

Зачем все это? Такой вывод можно легко перенаправить в файл для последующей обработки или — потоком — на стандартный ввод другой команды, поскольку вывод *ls* подходит далеко не для всех команд.

Первые версии команды *find* — на древних UNIX-системах — не выводили найденные файлы на консоль. Поскольку *find* часто используется во всевозможных скриптах, считалось, что они не должны продюсировать вывод, и предполагалось, что если вам нужно увидеть список найденных файлов, следует задействовать параметр *-print*. Сейчас этот параметр считается устаревшим и указывать его не нужно — по умолчанию команда *find* сама выводит список найденных файлов.

Параметр `-type` позволяет указать, что именно вы хотите найти:

```
find . -type f  
find . -type d
```

В первом случае нужно найти только файлы, во втором — каталоги.

Параметр `-name` позволяет указать маску имени (или точное имя):

```
find . -type f -name "*.jpg"
```

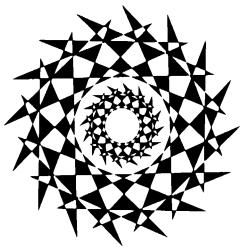
Здесь мы ищем в текущем каталоге файлы в формате JPEG — точнее, файлы, имя которых заканчивается на `.jpg`.

Параметр `-exec` позволяет выполнить указанную команду над каждым из найденных файлов. Удалим каждый найденный файл:

```
find . -type f -name "*.bk" -exec rm {} ";"
```

Обратите внимание: с параметром `-exec` всегда приводится конструкция `";"` — она указывает, что настал конец команды, которую должен выполнить параметр `-exec`. А конструкция `{}` заменяется на имя найденного файла.

Дополнительные примеры использования команды `find` вы найдете в главе 13.



ГЛАВА 8

Команды для работы с текстом

8.1. Команда *sort*: сортировка файлов

Команда *sort* сортирует все указанные файлы, результат сортировки этих файлов отправляется на стандартный вывод.

Синтаксис вызова команды:

```
sort [параметр] ... [файлы]
```

Параметры команды *sort* приведены в табл. 8.1.

Таблица 8.1. Параметры команды *sort*

Параметр	Описание
-b	Пробелы в начале сортируемых полей или в начале ключей будут игнорироваться
-d	При сортировке будут игнорироваться все символы, кроме букв, цифр и пробельных символов
-f	Игнорировать регистр букв
-r	Сортировка в обратном порядке
-o файл	Вывод результатов сортировки в файл
-t символ	Использование указанного символа в качестве разделителя полей

8.2. Команда *diff*: сравнение файлов

Команда *diff* используется для сравнения двух файлов. Формат вызова программы *diff*:

```
diff [параметры] файл1 файл2
```

В выводе программы различающиеся строки помечаются символами > и <:

- строка из первого файла помечается символом <;
- строка из второго файла — символом >.

Самые полезные параметры программы `diff` приведены в табл. 8.2.

Таблица 8.2. Некоторые параметры программы `diff`

Параметр	Описание
<code>-a</code>	Сравнение всех файлов, в том числе бинарных
<code>-b</code>	Программа будет игнорировать пробельные символы в конце строки
<code>-B</code>	Игнорирует пустые строки
<code>-e</code>	Применяется для создания сценария для редактора <code>ed</code> , который будет использоваться для превращения первого файла во второй
<code>-w</code>	Игнорирует пробельные символы
<code>-y</code>	Вывод в два столбца
<code>-r</code>	Используется для сравнения файлов в подкаталогах. Вместо первого файла указывается первый каталог, вместо второго файла — соответственно второй каталог

8.3. Команда `diff3`: сравнение трех файлов

Команда `diff3` похожа на команду `diff`, только применяется для сравнения трех файлов. Формат вызова команды таков:

`diff3 [параметры] файл1 файл2 файл3`

Программа выводит следующую информацию:

- — все три файла разные;
- =1** — первый файл отличается от второго и третьего;
- =2** — второй файл отличается от первого и третьего;
- =3** — третий файл отличается от первого и второго.

Параметры программы `diff3` представлены в табл. 8.3.

Таблица 8.3. Параметры программы `diff3`

Параметр	Описание
<code>-a</code>	Сравнивать файлы как текстовые, даже если они являются бинарными
<code>-A</code>	Создание сценария для редактора <code>ed</code> , который показывает в квадратных скобках все различия между файлами
<code>-e</code>	Создает сценарий для <code>ed</code> , который помещает все различия между файлами <code>файл2</code> и <code>файл3</code> в файл <code>файл1</code> (будьте осторожны!)
<code>-l</code>	Добавить команды <code>w</code> (сохранить файл) и <code>q</code> (выйти) в конец сценария <code>ed</code>
<code>-x</code>	Создание сценария редактора <code>ed</code> , который помещает различия между файлами в файл <code>файл1</code>

Таблица 8.3 (окончание)

Параметр	Описание
-x	То же, что и -x, но различия выделяются
-3	Создает сценарий ed, который помещает все различия между файлами <i>файл1</i> и <i>файл3</i> в <i>файл1</i>

8.4. Команда *cstr*: сравнение двух файлов

Команда *cstr* также служит для сравнения двух файлов. Если файлы идентичны, то *cstr* вообще никак не реагирует. А вот если файлы различаются, то *cstr* выводит номер строки и номер символа в строке, откуда начинается различие.

Команда *cstr* более универсальна, чем команда *diff* и ее аналоги, поскольку она может использоваться как для сравнения текстовых, так и двоичных файлов. А вот команда *diff* и ее аналоги умеют сравнивать только текстовые файлы.

Формат вызова команды следующий:

cstr [параметры] *файл1* *файл2*

Параметры команды *cstr* указаны в табл. 8.4.

Таблица 8.4. Параметры команды *cstr*

Параметр	Описание
-с	Вывод различающихся символов
-i <i>п</i>	Игнорировать первые <i>п</i> символов
-l	Вывод позиций всех различий, а не только первого
-s	Не выводить информацию на экран, при этом код возврата будет следующим: <ul style="list-style-type: none"> • 0 — файлы одинаковые; • 1 — файлы различаются; • 2 — ошибка при открытии одного из файлов

8.5. Команда *comm*: еще одна команда для сравнения файлов

Команда *comm* сравнивает содержимое двух файлов, которые были перед этим отсортированы командой *sort*. Вывод программы располагается в три столбца. В первом выводятся строки из *файла1*, во втором — из *файла2*, а в третьем — строки, которые имеются в обоих файлах.

Формат вызова программы следующий:

comm [параметры] *файл1* *файл2*

Параметры команды `comm` приведены в табл. 8.5.

Таблица 8.5. Параметры команды `comm`

Параметр	Описание
-1	Не выводить первый столбец
-2	Не выводить второй столбец
-3	Не выводить третий столбец
-12	Будет выведен только третий столбец
-13	Будет выведен только второй столбец
-23	Вывод только первого столбца

8.6. Команда `column`: разбивка текста на столбцы

Команда `column` используется для разбивки текста на несколько столбцов. Текст может быть прочитан как из файла, так и со стандартного ввода, если файл не указан.

Формат вызова команды:

`column [параметры] [файл]`

Параметры команды `column` приведены в табл. 8.6.

Таблица 8.6. Параметры команды `column`

Параметр	Описание
-c <i>н</i>	Задает количество столбцов (число <i>н</i>)
-s <i>символ</i>	Указанный символ будет использоваться в качестве разделителя столбцов
-t	Текст будет форматироваться как таблицы. По умолчанию разделителем полей считается пробел, но с помощью параметра <code>-s</code> можно задать другой разделитель
-x	Сначала будут заполняться столбцы, а потом строки

8.7. Команда `egrep`: расширенный текстовый фильтр

Команда `egrep` производит поиск строки в одном или нескольких файлах. Команда `egrep` похожа на команду `grep` (которая будет описана позже), но считается более быстрой и более функциональной. Если файлы не заданы, то программа читает текст из стандартного ввода.

Формат вызова программы:

`egrep [параметры] строка файлы`

Параметры команды `egrep` приведены в табл. 8.7.

Таблица 8.7. Параметры команды `egrep`

Параметр	Описание
<code>-A <i>л</i></code>	Вывод <i>л</i> строк после строки, в которой есть искомая строка
<code>-B <i>л</i></code>	Вывод <i>л</i> строк перед строкой, содержащей искомую строку
<code>-b</code>	Выводит для каждой строки файла, где есть искомая строка, ее положение в файле
<code>-c</code>	Выводит количество совпадений, но не выводит сами совпадения
<code>-C</code>	Выводит две строки до и две строки после строки, которая содержит искомую строку
<code>-e <i>строка</i></code>	Используйте этот параметр, если искомая строка начинается с символа «-»
<code>-f <i>файл</i></code>	Производит поиск искомых строк, которые имеются в указанном файле
<code>-h</code>	Выводит строки, содержащие искомую строку, но не выводит имена содержащих их файлов
<code>-i</code>	Игнорировать регистр букв
<code>-n</code>	Выводит номера строк (и сами строки), содержащих искомую строку
<code>-s</code>	Не выводить сообщения об ошибке, если некоторые файлы не могут быть открыты
<code>-w</code>	Поиск совпадения целого слова с искомой строкой
<code>-x</code>	Поиск совпадения целой строки с искомой строкой

Пример использования:

`egrep "ppp [1]" *`

Эта команда ищет строку, заключенную в кавычки во всех файлах в текущем каталоге.

8.8. Команда `expand`: замена символов табуляции пробелами

Команда `expand` заменяет в указанных файлах символы табуляции на соответствующее количество пробелов. Команде можно передать лишь один параметр `-1`, означающий, что замена должна быть только в начале строки.

Формат вызова команды:

`expand [-i] файлы`

8.9. Команда *fmt*

Команда *fmt* форматирует текст, выравнивает его по правой границе и удаляет символы новой строки. Синтаксис вызова команды:

```
fmt [параметры] файлы
```

Параметры команды *fmt* приведены в табл. 8.8.

Таблица 8.8. Параметры команды fmt

Параметр	Описание
-c	Не форматировать первые две строки
-р префикс	Форматировать только строки, начинающиеся с указанного префикса
-s	Не объединять строки
-t	Начинать параграф с красной строки
-w л	Задает максимальную длину строки в л символов (по умолчанию 72)

8.10. Команда *fold*

Команда *fold* выравнивает текст по правому краю и производит разрыв строк, если это необходимо. Максимальная длина строки — 80 символов. Установить другую длину строки можно с помощью параметра *-w*, как и в случае с командой *fmt*. Кроме того, можно указать параметр *-s*, разрешающий разрыв строки только на пробеле.

Формат вызова команды:

```
fold [параметры] файлы
```

8.11. Команда *grep*: текстовый фильтр

Команда *grep* похожа на команду *egrep*, которая была описана ранее. Поэтому сейчас, вместо описания параметров *grep*, мы рассмотрим использование этой команды.

Предположим, что у нас есть файл протокола */var/log/messages*, и мы хотим вывести все сообщения, связанные с демоном *pppd*. Понятно, что вручную выделить все нужные сообщения будет довольно трудно. Но с помощью *grep* можно автоматизировать эту задачу:

```
cat /var/log/messages | grep ppp
```

Команда *cat /var/log/messages* передаст содержимое файла */var/log/messages* на стандартный ввод команды *grep*, которая, в свою очередь, выделит строки, содержащие строку *ppp*.

СОВЕТ

Вообще-то просматривать журналы удобнее с помощью команды `tac`, которая выводит строки файла в обратном порядке — ведь сообщения дописываются в конец журнала, следовательно, если выводить строки в обратном порядке, то сначала получим самые новые сообщения, а потом уже все остальные:

```
tac /var/log/messages | grep ppp
```

8.12. Команды *more* и *less*: постстраничный вывод

Большой текстовый файл намного удобнее просматривать с помощью команд `less` или `more`. Программа `less` удобнее, чем `more`, если она есть в вашей системе:

```
tac /var/log/messages | grep ppp | less
```

8.13. Команды *head* и *tail*: вывод начала и хвоста файла

Команда `head` выводит первые десять строк файла, а `tail` — последние десять. Количество строк может регулироваться с помощью параметра `-n`.

Пример использования:

```
head -n 10 /var/log/messages  
tail -n 15 /var/log/messages
```

8.14. Команда *look*

Команда `look` производит поиск строк, начинающихся с указанной строки. Если файл не указан, то поиск осуществляется в файле `/usr/share/dict/words`. Можно задать параметр `-a` — тогда поиск будет произведен в файле `/usr/share/dict/web2`.

8.15. Команда *split*: разбиение файлов на несколько частей

Команда `split` служит для разделения файлов на части. По умолчанию создаются части размером в 1000 строк. Изменить размер можно, указав количество строк, например:

```
split -500 файл1
```

В этом случае файл будет разбит на части по 500 строк в каждой (кроме, возможно, последней части, где может быть строк меньше).

Команду можно также использовать для разделения файлов на части по размеру информации, а не по количеству строк. Например с помощью параметра `-b` можно указать количество символов в каждой части. Примеры вызова команды:

```
split -b100b файл
split -b100k файл
split -b100m файл
```

Первая команда разделит файл на части по 100 байтов каждая, вторая — на части по 100 Кбайт каждая, третья — по 100 Мбайт каждая.

8.16. Команда *unexpand*: замена пробелов символами табуляции

Команда *unexpand* заменяет последовательные пробелы символами табуляции. По умолчанию одним символом табуляции заменяются 8 пробелов. Количество пробелов можно задать с помощью параметра *-t n* (где *n* — количество пробелов).

Синтаксис вызова:

```
unexpand [параметры] файл
```

8.17. Команды *vi*, *nano*, *ee*, *mcedit*, *pico*: текстовые редакторы

Из первых версий UNIX в современные системы перекочевал текстовый редактор *vi*. То, что ему больше тридцати лет, — видно сразу. Более неудобного редактора я не встречал! Согласен, что тогда это был прорыв, но сегодня редактор смотрится уж очень архаично.

Некоторые гурманы (я бы их назвал мазохистами) говорят, что к нему нужно привыкнуть. Может, и так, но сначала следует изучить длинный *man* и выучить наизусть команды редактора. Как такого интерфейса пользователя у редактора *vi* практически нет, можно сказать, что вообще нет — то, что есть, сложно назвать интерфейсом. Однако в этой книге мы рассмотрим *vi* хотя бы вкратце. Тому есть две причины. Первая — ее будущие критики. Мол, как это в книге, посвященной командной строке, не будет «классики». Вторая — существуют системы, где по неизвестным мне причинам до сих пор используется по умолчанию *vi*, а другие редакторы недоступны. Да, можно изменить переменную окружения *EDITOR*, но нет никакой гарантии, что в системе будет установлен какой-нибудь другой редактор.

Итак, приступим к рассмотрению редактора *vi*. Он может работать в трех режимах:

- основной (визуальный) режим — в нем и осуществляется редактирование текста;
- командный режим — в нем выполняется ввод специальных команд для работы с текстом (если сравнивать *vi* с нормальным редактором, то этот режим ассоциируется с меню редактора, где есть команды вроде «сохранить», «выйти» и т. п.);
- режим просмотра — предназначен только для просмотра файла (если надумаете использовать этот режим, вспомните про команду *less*).

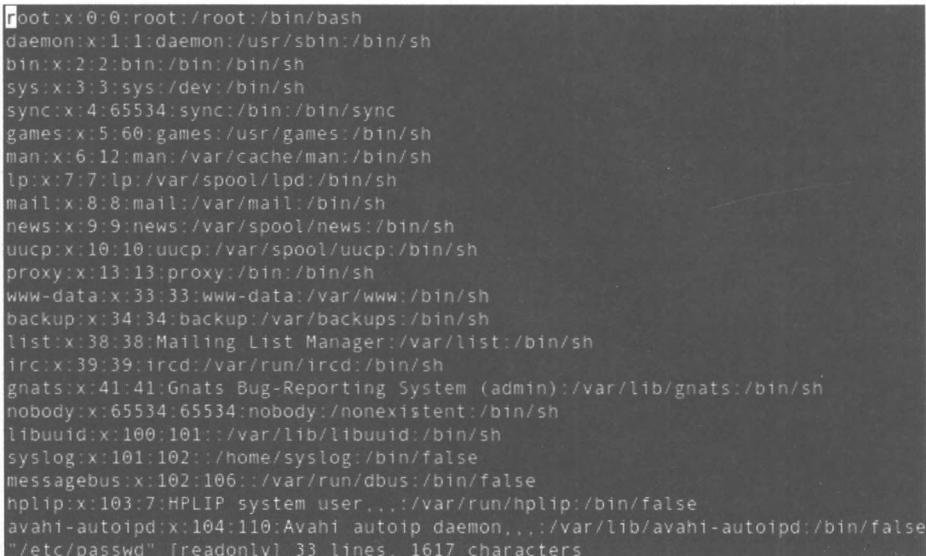
После запуска редактора вы можете переключать режимы (как — будет сказано позже), но выбрать режим можно и при запуске редактора:

```
vi файл
vi -e файл
vi -R файл
```

Первая команда запускает vi и загружает файл. Вторая команда запускает vi в командном режиме и загружает файл. Третья команда — это режим просмотра файла. Если указанный файл не существует, то он будет создан. По умолчанию активируется именно командный режим, поэтому в ключе -e нет смысла.

После запуска vi главное знать, как из него выйти. Ведь в нем не будет знакомой строки меню, редактор не будет реагировать и на привычные комбинации клавиш вроде <Alt>+<X>, <Ctrl>+<C>. На рис. 8.1 представлен редактор vi, в который загружен файл /etc/passwd.

В табл. 8.9 приведены основные команды редактора vi.



```
root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/bin/sh
bin:x:2:2:bin:/bin:/bin/sh
sys:x:3:3:sys:/dev:/bin/sh
sync:x:4:65534:sync:/bin:/sync
games:x:5:60:games:/usr/games:/bin/sh
man:x:6:12:man:/var/cache/man:/bin/sh
lp:x:7:7:lp:/var/spool/lpd:/bin/sh
mail:x:8:8:mail:/var/mail:/bin/sh
news:x:9:9:news:/var/spool/news:/bin/sh
uucp:x:10:10:uucp:/var/spool/uucp:/bin/sh
proxy:x:13:13:proxy:/bin:/bin/sh
www-data:x:33:33:www-data:/var/www:/bin/sh
backup:x:34:34:backup:/var/backups:/bin/sh
list:x:38:38:Mailing List Manager:/var/list:/bin/sh
irc:x:39:39:ircd:/var/run/ircd:/bin/sh
gnats:x:41:41:Gnats Bug-Reporting System (admin):/var/lib/gnats:/bin/sh
nobody:x:65534:65534:nobody:/nonexistent:/bin/sh
libuuid:x:100:101::/var/lib/libuuid:/bin/sh
syslog:x:101:102::/home/syslog:/bin/false
messagebus:x:102:106::/var/run/dbus:/bin/false
hplip:x:103:7:HPLIP system user...:/var/run/hplip:/bin/false
avahi-autoipd:x:104:110:Avahi autoip daemon...:/var/lib/avahi-autoipd:/bin/false
"/etc/passwd" [readonly] 33 lines, 1617 characters
```

Рис. 8.1. Редактор vi

Таблица 8.9. Основные команды редактора vi

Команда	Описание
:q!	Выход без сохранения
:w	Сохранить изменения
:w <файл>	Сохранить изменения под именем <файл>
:wq	Сохранить и выйти
:q .	Выйти, если нет изменений

Таблица 8.9 (окончание)

Команда	Описание
i	Перейти в режим вставки символов в позицию курсора
a	Перейти в режим вставки символов в позицию после курсора
o	Вставить строку после текущей
O	Вставить строку над текущей
x	Удалить символ в позицию курсора
dd	Удалить текущую строку
u	Отменить последнее действие

Команды, которые начинаются с двоеточия, будут отображены в нижней строке, остальные просто выполняются, но не отображаются. Как уже было отмечено, у редактора vi есть два основных режима (режим просмотра не считается): режим команд и режим редактирования (визуальный). Переключение в режим команд осуществляется нажатием клавиши <Esc>. Нажатие клавиш <i>, <a> и других переключает редактор в режим вставки, когда набираемые символы трактуются именно как символы, а не как команды. Для переключения обратно в командный режим используется клавиша <Esc>. В некоторых случаях (например, когда вы пытаетесь передвинуть курсор левее первого символа в строке) переход в командный режим осуществляется автоматически.

Теперь немного практики — введите команду:

```
$ vi file.txt
```

Далее нажмите клавишу <i>, чтобы переключиться в режим вставки. Наберите любой текст, но постарайтесь не ошибаться, поскольку исправление ошибок в vi — дело, требующее отдельного разговора.

Затем нажмите клавишу <Esc> и введите :wq. После выхода из редактора введите команду:

```
cat file.txt
```

Так вы убедитесь, что файл создан и в нем сохранен введенный вами текст.

Продолжим изучать редактор. Если ввести не команду i, а команду a, то вы тоже перейдете в режим вставки, но с одним различием. Введенный текст будет вставляться не перед символом, в котором находится курсор, а после него. Также в режим вставки можно перейти командами o и O. В первом случае будет добавлена пустая строка после текущей строки, а во втором — перед текущей строкой, а весь дальнейший ввод будет восприниматься именно как ввод текста, а не команд.

Чтобы удалить символ, нужно перейти в режим команд и над удаляемым символом нажать <x>. Да, клавиши <Backspace> и <Delete> тут не работают. Точнее, <Backspace> работает, но для удаления последней непрерывно введенной последо-

вательности символов. Например, у нас есть текст: vi – текстовый редактор. Вы перейдете в режим вставки и измените текст так: vi – неудобный текстовый редактор. Нажатие <Backspace> удалит слово неудобный, но не сможет удалить тире и другие символы.

Чтобы удалить строку, в которой находится курсор, нужно использовать команду dd. Помните, что vi считает строкой не то, что вы видите на экране, а последовательность символов до первого символа новой строки (\n). Если строка длиннее 80 символов, то она переносится на две экранные строки и визуально выглядит как две строки, а не как одна.

Чтобы перейти в конец строки (клавиши <Home> и <End> тоже не работают, как вы успели заметить, если уже запускали vi), нужно ввести команду \$. При навигации курсор перемещается не по экранным линиям, а как раз по строкам текста.

Для отмены последней операции служит команда u. Вот только истории изменений нет, да и по команде u отменяется вся предыдущая команда целиком. Например, вы создали файл, перешли в режим вставки (команда i) и ввели весь текст Большой медицинской энциклопедии. Если вы введете команду u, то она отменит всю предыдущую команду — т. е. удалит весь введенный вами текст. Так что будьте осторожны.

Что ж, азы vi я вам преподнес. Но не думаю, что вы будете им пользоваться. Если есть желание продолжить знакомство, введите команду:

```
man vi
```

А мы тем временем познакомимся с другими текстовыми редакторами. Самый удобный из известных мне текстовых редакторов — редактор nano (раньше он назывался pico и входил в состав почтового клиента pine). Редактор nano показан на рис. 8.2.

Внизу (под текстом) есть подсказка по комбинациям клавиш для управления редактором. Символ ^ означает <Ctrl>. То есть для выхода из редактора нужно нажать комбинацию клавиш <Ctrl>+<X>, а для сохранения текста — <Ctrl>+<O>.

В некоторых системах (например, в FreeBSD) вместо nano используется редактор ee. Он похож на nano, однако подсказки выводятся до текста (вверху экрана), а не после него. Также весьма удобен редактор joe.

В пакет mc (файловый менеджер) входит достаточно удобный редактор mcedit, который запускается по нажатии клавиши <F4> в mc (рис. 8.3). Но вы можете запустить этот редактор и отдельно:

```
mcedit <имя файла>
```

Кстати, редакторы joe, nano и ee запускаются аналогично:

```
joe <имя файла>
nano <имя файла>
ee <имя файла>
```

GNU nano 2.0.9 Файл: /etc/passwd

```
root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/bin/sh
bin:x:2:2:bin:/bin:/bin/sh
sys:x:3:3:sys:/dev:/bin/sh
sync:x:4:65534:sync:/bin:/bin/sync
games:x:5:60:games:/usr/games:/bin/sh
man:x:6:12:man:/var/cache/man:/bin/sh
lp:x:7:7:lp:/var/spool/lpd:/bin/sh
mail:x:8:8:mail:/var/mail:/bin/sh
news:x:9:9:news:/var/spool/news:/bin/sh
uucp:x:10:10:uucp:/var/spool/uucp:/bin/sh
proxy:x:13:13:proxy:/bin:/bin/sh
www-data:x:33:33:www-data:/var/www:/bin/sh
backup:x:34:34:backup:/var/backups:/bin/sh
list:x:38:38:Mailing List Manager:/var/list:/bin/sh
irc:x:39:39:ircd:/var/run/ircd:/bin/sh
gnats:x:41:41:Gnats Bug-Reporting System (admin):/var/lib/gnats:/bin/sh
nobody:x:65534:65534:nobody:/nonexistent:/bin/sh
libuuid:x:100:101::/var/lib/libuuid:/bin/sh
```

[ПРОЧИТАНО 33 СТРОКИ]

^G Помощь ^O Записать ^R ЧитФайл ^Y ПредСтр ^K Вырезать ^C ТекПозиц
^X Выход ^J Выровнять ^W Поиск ^V СледСтр ^U ОтмВырезк ^T Словарь

Рис. 8.2. Редактор папок

```
/etc/passwd: [root] 0:1:| 1+ 0 1/ 34] *(0 /1617b)= r 114 0x72  
root:x:0:0:root:/root:/bin/bash  
daemon:x:1:1:daemon:/usr/sbin:/bin/sh  
bin:x:2:2:bin:/bin:/bin/sh  
sys:x:3:3:sys:/dev:/bin/sh  
sync:x:4:65534:sync:/bin:/sync  
games:x:5:60:games:/usr/games:/bin/sh  
man:x:6:12:man:/var/cache/man:/bin/sh  
lp:x:7:7:lp:/var/spool/lpd:/bin/sh  
mail:x:8:8:mail:/var/mail:/bin/sh  
news:x:9:9:news:/var/spool/news:/bin/sh  
uucp:x:10:10:uucp:/var/spool/uucp:/bin/sh  
proxy:x:13:13:proxy:/bin:/bin/sh  
www-data:x:33:33:www-data:/var/www:/bin/sh  
backup:x:34:34:backup:/var/backups:/bin/sh  
list:x:38:38:Mailing List Manager:/var/list:/bin/sh  
irc:x:39:39:ircd:/var/run/ircd:/bin/sh  
gnats:x:41:41:Gnats Bug-Reporting System (admin):/var/lib/gnats:/bin/sh  
nobody:x:65534:65534:nobody:/nonexistent:/bin/sh  
libuuid:x:100:101::/var/lib/libuuid:/bin/sh  
syslog:x:101:102::/home/syslog:/bin/false  
messagebus:x:102:106::/var/run/dbus:/bin/false  
hplip:x:103:7:HPLIP system user...:/var/run/hplip:/bin/false
```

Рис. 8.3. Редактор mcedit

8.18. Команда sed: потоковый текстовый редактор

Команда `sed` — мощный потоковый редактор, и его описанию можно было бы посвятить целую главу. Впрочем, сейчас особой необходимости в этом нет, поскольку современные дистрибутивы снабжены соответствующей документацией на русском языке.

ском языке. Главное, знать, что такая программа имеется. А чтобы вас заинтересовать, предлагаю рассмотреть несколько примеров ее использования:

- заменить строку `string1` на `string2` в файле `example.txt`, результат вывести на стандартное устройство вывода:

```
sed 's/string1/string2/g' example.txt
```

- вывести пятую строку файла `example.txt`:

```
sed -n '5p;5q' example.txt
```

- удалить пустые строки из этого файла:

```
sed '/^$/d' example.txt
```

- удалить строку `string1` из текста, не изменяя всего остального:

```
sed -e 's/string1//g' example.txt
```

- удалить пустые символы в конце каждой строки:

```
sed -e 's/ *$//'" example.txt
```

- удалить из файла пустые строки и комментарии:

```
sed '/ *#/d; /^$/d' example.txt
```

- преобразовать символы из нижнего регистра в верхний:

```
echo 'test' | tr '[:lower:]' '[:upper:]'
```

- удалить из файла первую строку:

```
sed -e '1d' example.txt
```

8.19. Команда `wc`: подсчет слов в файле

Команда `wc` используется:

- для подсчета слов в текстовом файле:

```
wc /var/log/messages
```

- для подсчета количества строк (если задан параметр `-l`):

```
wc -l /var/log/messages
```

- для подсчета количества символов (параметр `-c`):

```
wc -c /var/log/messages.
```

8.20. Некоторые команды преобразования символов и форматов

Между текстовыми файлами, созданными в Windows (DOS) и UNIX, имеются различия в том, как заканчивается строка. В UNIX строка заканчивается просто символом `\n`, а в DOS, кроме этого символа, добавляется еще символ возврата каретки. Для преобразования текстового файла DOS-формата в формат UNIX служит команда `dos2unix`, например:

```
$ dos2unix file_dos.txt file_unix.txt
```

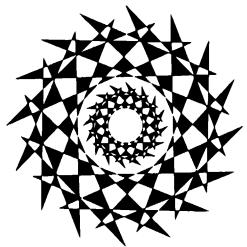
Для обратного преобразования используется команда `unix2dos`:

```
$ unix2dos file_unix.txt file_dos.txt
```

Весьма полезна и команда `recode`, позволяющая преобразовывать файлы из одного формата в другой — например, из HTML в обычный текст. Вывести список доступных форматов можно так:

```
recode -l | more
```

Дополнительную информацию об этих командах вы можете получить в `man`.



ГЛАВА 9

Эффективное использование клавиатуры

9.1. Работа с окнами

В этой небольшой главе мы продолжим разговор об эффективном использовании командной строки. Современная Linux оснащена довольно-таки удобным графическим интерфейсом, в котором, кроме окна терминала, пользователь может открыть множество других окон — например, окно браузера, текстового редактора, среду разработки, виртуальную машину и т. д.

Но, несмотря на наличие графического интерфейса, сама концепция Linux предполагает использование командной строки в большей степени, чем в других операционных системах: Windows и macOS. Советы, приведенные в этой главе, более нацелены на использование клавиатуры, нежели мыши, — вы узнаете дополнительные команды, а также комбинации клавиш, позволяющие быстро выполнить те или иные действия.

Большинство графических интерфейсов (GNOME, KDE, Unity, Citramon) позволяют управлять ими с помощью «горячих клавиш» (hotkeys) — специальных клавиатурных комбинаций, вызывающих программы или выполняющих другие действия.

Чтобы вам было удобнее работать, нужно определить как минимум две специальные комбинации: первая будет открывать новое окно терминала, вторая — новое окно браузера. Как только вы это сделаете, работать станет удобнее.

Как правило, команда, открывающая окно терминала, называется `gnome-terminal`, `konsole` или `xterm`. А следующие команды запускают браузер: `firefox`, `google-chrome`, `opera`.

Как именно определить клавиатурную комбинацию, зависит от используемого графического окружения. Инструкции меняются от версии к версии, поэтому обратитесь к документации или просто исследуйте параметры своей графической среды.

Для открытия терминала я назначил комбинацию клавиш `<Ctrl>+<Windows>+<T>`, а для браузера Chrome — `<Ctrl>+<Windows>+<C>`.

При работе с самим браузером вы можете использовать комбинации клавиш из табл. 9.1. Они работают для всех популярных браузеров: Firefox, Chrome и Opera.

Таблица 9.1. Клавиатурные комбинации для Firefox, Chrome, Opera

Комбинация клавиш	Действие
<Ctrl>+<N>	Открыть новое окно
<Ctrl>+<Shift>+<P> (Firefox) <Ctrl>+<Shift>+<N> (Chrome/Opera)	Новое окно в режиме инкогнито
<Ctrl>+<T>	Открыть новую вкладку
<Ctrl>+<Tab> <Ctrl>+<Shift>+<Tab> (в обратном порядке)	Циклическое переключение между вкладками
<Ctrl>+<L>	Перейти к адресной строке
<Ctrl>+<F>	Поиск на странице
<Ctrl>+<H>	Показать историю посещенных страниц

Для переключения между открытыми окнами приложений служит комбинация клавиш `<Alt>+<Tab>` — она работает в любой графической среде. По нажатии `<Alt>+<Tab>` происходит циклическое переключение между открытыми окнами. Комбинация `<Alt>+<Shift>+<Tab>` производит переключение в обратном порядке.

А вот комбинация `<Alt>+<'>` (`<Alt>+клавиша над <Tab>`) служит для переключения между открытыми вкладками в Firefox. Для обратного переключения используется комбинация клавиш `<Alt>+<Shift>+<'>`.

С переключением окон разобрались. Теперь нужно поговорить о переключении рабочих столов. Любая графическая среда — будь то GNOME или KDE — позволяет создать несколько рабочих столов и переключаться между ними. Рабочие столы предоставляют возможность более эффективно организовать работу с окнами, особенно когда этих окон много. Концепцию рабочих столов по достоинству оценият фрилансеры, которые работают с несколькими заказчиками, — тогда на разные рабочие столы можно поместить окна, относящиеся к разным заказчикам: редактор кода, SSH-клиент или окно терминала, электронную таблицу и пр. Довольно удобно — чтобы в конкретный момент времени ничего не мешало.

По умолчанию Linux не определяет комбинацию клавиш для переключения между рабочими столами, но вы можете это сделать вручную в настройках графической среды. Например, использовать комбинацию клавиш `<Win>+<n>`, где *n* — это номер рабочего стола. Попробуйте — вам понравится.

Напомню, как из графического режима переключиться в настоящую консоль: `<Ctrl>+<Alt>+<Fn>`, где *n* — номер консоли от 1 до 6. А уже в графическом (консольном) режиме переключение между консолями осуществляется с использованием комбинации клавиш `<Alt>+<Fn>`. Для возврата обратно в графический режим служит комбинация `<Alt>+<F7>`.

Вы вряд ли будете в современном дистрибутиве работать в настоящем графическом режиме, но, возможно, вам придется администрировать физический (невиртуальный) сервер — там графического интерфейса, как правило, нет, и тогда вам пригодятся комбинации клавиш `<Alt>+<Fn>`. Ведь пока на одной консоли запущен ка-

кой-нибудь процесс, на второй можно заняться чем-то полезным. Такой вариант в современном мире также встречается все реже и реже, разве что в дата-центрах. Да, сейчас многие предпочитают виртуальные серверы, но ведь эти виртуальные машины запущены на обычном физическом «железе». Следовательно, кому-то ведь надо это железо администрировать. Также «железный» сервер может позволить себе крупное предприятие, которое не желает, чтобы его данные хранились у третьего лица (у облачного провайдера).

9.2. Доступ к веб-браузерам из командной строки

Доступ к веб-браузерам можно получить непосредственно из командной строки консоли или терминала. Запускать из терминала браузеры вы можете с помощью следующих команд:

```
firefox &  
google-chrome &  
opera &
```

Чтобы выполнение команды не блокировало терминал, здесь используется амперсанд (напомню, он обеспечивает в терминале переход в фоновый режим работы), — иначе вы не сможете ввести следующую команду, пока не закроете окно браузера.

Более того, вы можете сразу указать страничку, которую хотите посетить:

```
firefox https://www.fbi.gov &  
google-chrome https://www.fbi.gov &  
opera https://www.fbi.gov &
```

Аналогичным образом можно открыть приватное окно, указав соответствующий параметр, название которого зависит от браузера:

```
firefox --private-window https://www.fbi.gov &  
google-chrome --incognito https://www.fbi.gov &  
opera --private https://www.fbi.gov &
```

Кстати, существуют и собственно консольные браузеры: lynx и links (рис. 9.1). Установив их, вы сможете в текстовом режиме просматривать различные сайты. Разумеется, изображения они не выводят, но текст прочитать удастся. В современном мире вероятность их использования низка, но вы должны о них знать.

Также весьма полезными являются команды curl и wget, с помощью которых можно получить какой-либо файл, — например, так:

```
curl https://example.com/hello.html  
wget https://example.com/hello.html
```

Возможности автоматизации открывают нам новые горизонты. Представьте, что существует ряд изображений, которые нужно скачать, — например:

```

Link: canonical
Link: home: Front page
Link: contents
Link: manifest
Link: search: Search this site
An official website of the United States government. Here's how you know

Official websites use .gov

A .gov website belongs to an official government organization in the United States.

Secure .gov websites use HTTPS

A lock () or https:// means you've safely connected to the .gov website. Share sensitive
information only on official, secure websites.

Search
FBI
More
* Most Wanted
* News
* What We Investigate
* Services
* Resources
* Submit a Tip
* About
* Contact Us
* * * * * Search FBI _____
Federal Bureau of Investigation Logo
FBIFederal Bureau of Investigation

Welcome to FBI.gov

```

<https://www.fbi.gov/>

Рис. 9.1. Диковинка: браузер links

```

https://example.com/images/1.jpg
https://example.com/images/2.jpg
https://example.com/images/3.jpg
...

```

Если таких изображений относительно немного, можно несколько раз ввести команду wget (не забываем, что если после выполнения какой-либо команды нажать клавишу-стрелку <↑>, то командная строка отобразит выполненную команду, где вы сможете изменить номер запрашиваемого файла и повторить ее выполнение, но уже для нового изображения). Но когда таких изображений сотня, то вам это быстро надоест. К тому же если эти картинки большие и загрузка каждой из них занимает некоторое время, то заниматься всем этим придется долго. К счастью, вы можете использовать следующую команду:

```
seq 1 100 | xargs -I@ wget https://example.com/images/@.jpg
```

Посмотрим, что здесь происходит. Команда seq генерирует числа от 1 до 100. Каждое сгенерированное число отправляется на стандартный ввод команды xargs, которая принимает число в качестве параметра при @ и подставляет его в нужном месте команды, которая будет выполнена, — в нашем случае это wget. Все довольно просто!

Это не единственное верное решение такой задачи — можно, конечно, задействовать скриптовый язык awk, но использование xargs — самое простое. Вам не нужно вникать в синтаксис awk, а всего лишь запомнить одну команду. Подробности — в man xargs.

9.3. Работаем с буфером обмена

Все мы знаем комбинации клавиш $<\text{Ctrl}>+<\text{C}>$, $<\text{Ctrl}>+<\text{X}>$ и $<\text{Ctrl}>+<\text{V}>$, обеспечивающие взаимодействие с буфером обмена. Они работают в большинстве графических приложений: в текстовом и графическом редакторах, в браузере, в электронной таблице и т. д.

А как быть с терминалом? Вполне вероятно, что вам захочется скопировать вывод какой-то программы, чтобы его кому-то отправить или вставить в документ. Как именно работать с буфером обмена, зависит от используемого терминала. В большинстве случаев вам придется работать или с терминалом среды GNOME — gnome-terminal или же с терминалом среды KDE — konsole.

Для выделения текста в обоих терминалах используется следующая техника: нужно нажать левую кнопку мыши и, не отпуская ее, перетаскивать мышь, выделяя текст. Двойной щелчок выделяет слово, на котором вы щелкнули. Тройной щелчок — текущую строку.

Чтобы скопировать выделение, нужно щелкнуть на нем правой кнопкой мыши и выбрать команду **Copy** из контекстного меню. При использовании клавиатуры надо нажать комбинацию $<\text{Ctrl}>+<\text{Shift}>+<\text{C}>$ — учтите при этом, что привычная комбинация $<\text{Ctrl}>+<\text{C}>$ в терминале трактуется как аварийное завершение работы выполняемой задачи. Например, если вы запустили какую-то программу и не хотите дожидаться ее завершения, нажмите $<\text{Ctrl}>+<\text{C}>$.

Для вставки текста в терминале нажмите среднюю клавишу (или колесико) мыши. Можно также использовать комбинации клавиш $<\text{Ctrl}>+<\text{Shift}>+<\text{V}>$ (в GNOME) или $<\text{Shift}>+<\text{Insert}>$ (в KDE).

Для работы с буфером обмена в терминале вы можете использовать команду xclip. Чтобы скопировать текст в буфер обмена, выполните команду:

```
echo "Hello" | xclip
```

Можно поместить в буфер обмена и содержимое целого файла:

```
xclip < file.txt
```

Просмотреть содержимое буфера обмена можно так:

```
xclip -o
```

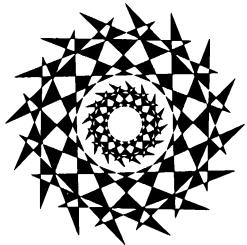
```
Hello
```

Можно вставить содержимое буфера обмена в файл:

```
xclip -o > file.txt
```

Аналогичным образом можно перенаправить содержимое буфера обмена на любую другую команду (например, для подсчета количества строк):

```
xclip -o | wc -l
```



ГЛАВА 10

Команды для работы с сетью и Интернетом

10.1. Команда *ifconfig*: управление сетевыми интерфейсами

Команда *ifconfig* служит для получения информации о сетевых интерфейсах и для установки параметров сетевых интерфейсов. Обычно эта команда вызывается при запуске системы сценариями инициализации системы, и вам не придется ее использовать для настройки интерфейсов вручную (разве что на самых древних дистрибутивах).

Формат вызова команды следующий:

```
ifconfig -a [параметры | семейство_протоколов]
ifconfig интерфейс [параметры | семейство_протоколов]
```

Чтобы просто просмотреть информацию о сетевых интерфейсах, введите команду:

```
ifconfig
```

Посмотрите на рис. 10.1. «Поднят» только интерфейс *lo* — это интерфейс локальной петли, используемой для тестирования сети. Если есть только интерфейс *lo*, значит, остальные сетевые интерфейсы не настроены. В современных дистрибутивах настройка интерфейсов производится автоматически при запуске системы — ведь практически всегда в сети есть DHCP-сервер, который и настраивает сетевые интерфейсы.

Настроить интерфейс можно и с помощью *ifconfig*. Вот пример настройки интерфейса *eth0* (первая сетевая плата), когда ему присваивается IP-адрес 192.168.1.7 и он «поднимается» (*up*):

```
# /sbin/ifconfig eth0 192.168.1.7 up
```

Для полной настройки сетевого интерфейса нужно использовать команду:

```
# /sbin/ifconfig eth0 адрес broadcast широковещательный_адрес netmask маска
```

После имени интерфейса задается IP-адрес, затем широковещательный адрес (*широковещательный_адрес*), потом сетевая маска. На рис. 10.2 показан уже настроенный интерфейс *eth0*.

```
[root@localhost /]# 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
            RX packets:62 errors:0 dropped:0 overruns:0 frame:0
            TX packets:62 errors:0 dropped:0 overruns:0 carrier:0
            collisions:0 txqueuelen:0
              RX bytes:3914 (3.8 Kb)   TX bytes:3914 (3.8 Kb)

[root@localhost /]#
```

Рис. 10.1. Настроен только интерфейс lo

```
[root@localhost /]# ifconfig
eth0      Link encap:Ethernet  HWaddr 00:60:97:88:BC:96
          inet addr:192.168.1.7  Bcast:192.168.1.255  Mask:255.255.255.0
            inet6 addr: fe80::20d:87ff:fe88:bc96/64 Scope:Link
              UP BROADCAST MULTICAST  MTU:1500  Metric:1
              RX packets:0 errors:0 dropped:0 overruns:0 frame:0
              TX packets:12 errors:0 dropped:0 overruns:0 carrier:0
              collisions:0 txqueuelen:1000
              RX bytes:0 (0.0 b)   TX bytes:672 (672.0 b)
              Interrupt:11 Base address:0xe800

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
            RX packets:83 errors:0 dropped:0 overruns:0 frame:0
            TX packets:83 errors:0 dropped:0 overruns:0 carrier:0
            collisions:0 txqueuelen:0
              RX bytes:5270 (5.1 Kb)   TX bytes:5270 (5.1 Kb)

[root@localhost /]#
```

Рис. 10.2. Интерфейсы lo и eth0

В современных дистрибутивах сетевые интерфейсы могут называться иначе — например, ens33 вместо eth0. Если с eth0 все понятно: интерфейс Ethernet, адаптер 0, то с другими интерфейсами многое иначе. Сетевые адAPTERы, встроенные в материнскую плату (`ID_NET_NAME_ONBOARD`), носят теперь название типа eno1. Если же сетевая карта подключена к PCI Express (`ID_NET_NAME_SLOT`), то у нее будет название типа ens33. Имена устройств, содержащие физическое/географическое расположение коннектора (`ID_NET_NAME_PATH`), будут выглядеть так: enp2s0. А самые сложные и «страшные» имена у сетевых адAPTERов, определяемых через MAC-адрес, — enx66e7bles34dd.

Если вы хотите вернуть привычную схему именования сетевых адAPTERов, передайте ядру параметр `net.ifnames=0`.

Для активации и деактивации интерфейса можно также использовать команды `ifup` и `ifdown` соответственно. Эти команды более удобны в использовании, чем

`ifconfig`, — с ними ручного ввода получается немного меньше. Например, для активации и деактивации интерфейса `eth0` можно использовать следующие команды:

```
# ifup eth0  
# ifdown eth0
```

Дополнительную информацию всегда можно получить в справочной системе (`man`).

Вообще, перед тем как вводить команду `ifconfig`, неплохо было бы добавить модуль сетевой платы, без него `ifconfig` работать не будет. Вот пример добавления модуля для сетевой платы Realtek 8139:

```
# insmod rtl8139.o
```

Пояснение

Наверняка многие из вас знают, как работает Ethernet-сеть. А для тех, кто не вполне себе это представляет, кратко поясню. Представим, что у нас в сети есть компьютеры А, Б и В. Компьютер А отправляет пакет компьютеру В. Но концентратор — довольно глупое устройство, и не знает, к какому из портов подключен тот или иной компьютер, поэтому он повторяет пакеты на все порты. Компьютер Б, получив «чужой» пакет (а это он понимает, сравнивая адрес назначения со своим адресом), его отбрасывает. Компьютер В видя, что пакет адресован ему, его принимает. Но если сетевой адаптер компьютера Б перевести в режим `promiscuous`, компьютер Б будет принимать («прослушивать») «чужие» пакеты, — т. е., учитывая специфику работы Ethernet, он будет принимать все пакеты, пересылаемые по сети. Такое явление называется *снiffингом* (sniffing). Перевести адаптер в этот заветный режим можно командой `ifconfig eth0 promisc` (где `eth0` — имя интерфейса), а вернуть в состояние «все как было» — командой `ifconfig eth0 -promisc`. Конечно, в коммутируемых сетях такой режим немного усложнен, но тоже возможен. Если вы заинтересовались, прочитайте статью: http://www.opennet.ru/base/sec/arp_snif.txt.html.

10.2. Маршрутизация

10.2.1. Команда `netstat`: просмотр таблицы маршрутизации и другой сетевой информации

Команду `netstat` можно использовать для получения различной информации о сети. Например, `netstat -at` выводит информацию обо всех соединениях. Выводится протокол, количество пакетов в очередях получения и отправки, локальный адрес, удаленный адрес, состояние соединения. Вывод можно отфильтровать с помощью команды `grep`. Например, так можно просмотреть только установленные соединения:

```
netstat -at | grep ESTABLISHED  
tcp      0      0 localhost.localdo:50090 localhost.localdo:mysql ESTABLISHED  
tcp      0      0 localhost.localdo:60112 localhost.localdo:mysql ESTABLISHED  
tcp      0      36 hosting-test-6:ssh     188-188-88-88.adr:18938 ESTABLISHED  
tcp      0      0 localhost.localdo:mysql localhost.localdo:60084 ESTABLISHED  
tcp      0      0 localhost.localdo:60066 localhost.localdo:mysql ESTABLISHED  
tcp      0      0 localhost.localdo:mysql localhost.localdo:60172 ESTABLISHED  
tcp      0      0 hosting-test-6:https   static.221.170.13:43874 ESTABLISHED
```

```
tcp      0      0 hosting-test-6:https static.208.239.13:57654 ESTABLISHED
tcp      0      0 localhost.localdo:mysql localhost.localdo:50092 ESTABLISHED
```

Для просмотра таблицы маршрутизации применяются команды `netstat -r` и `netstat -rn`. Можно также по старинке воспользоваться командой `route` без параметров. Разница между командами `netstat -r` и `netstat -rn` заключается в том, что параметр `-rn` запрещает поиск доменных имен в DNS, поэтому все адреса будут представлены в числовом виде (подобно команде `route` без параметров). А вот разница между выводом `netstat` и `route` заключается в представлении маршрута по умолчанию (`netstat` выводит адрес 0.0.0.0, а `route` — метку default) и в названии полей самой таблицы маршрутизации.

ДОПОЛНИТЕЛЬНЫЕ СРЕДСТВА СТАТИСТИКИ

Команда `netstat`, помимо всего прочего, используется для вывода статистической информации о сети — поэтому, собственно, она так и называется. Но это не единственное средство статистики. Так, например, команда `ethtool` отображает статистику по Ethernet-соединению. А команда `mii-tool`, кроме переключения режимов интерфейса, умеет отображать состояние и тип соединения. Команда `mii-tool` будет рассмотрена далее в этой главе, дополнительную же информацию о командах `ethtool` и `mii-tool` вы найдете в `man`.

Какую команду применять — решать вам. Раньше я использовал `route` и для просмотра, и для редактирования таблицы маршрутизации. Теперь для просмотра таблицы я применяю команду `netstat -rn`, а для ее изменения — команду `route`.

Вывод команды `netstat -rn` будет примерно такой:

Таблица маршрутизации ядра протокола IP

Destination	Gateway	Genmask	Flags	MSS	Window	irtt	Iface
192.168.181.0	0.0.0.0	255.255.255.0	U	0	0	0	eth0
169.254.0.0	0.0.0.0	255.255.255.0	U	0	0	0	eth0
0.0.0.0	192.168.181.2	0.0.0.0	UG	0	0	0	eth0

Сравним этот вывод с выводом команды `route`:

Destination	Gateway	Genmask	Flags	Metric	Ref	Use	Iface
192.168.181.0	*	255.255.255.0	U	1	0	0	eth0
link-local	*	255.255.0.0	U	1000	0	0	eth0
default	*	192.168.181.2	UG	0	0	0	eth0

В нашем случае есть две сети: 192.168.181.0 и 169.254.0.0 — обе на интерфейсе `eth0`. Такая ситуация сложилась из-за особенностей NAT/DHCP виртуальной машины VMware, в которой у меня была запущена ОС Linux. В реальных условиях обычно мы увидим по одной подсети на одном интерфейсе. С другой стороны, этот вывод демонстрирует поддержку VLAN, когда один интерфейс может использоваться двумя подсетями. Шлюз по умолчанию — компьютер с адресом 192.168.181.2, о чем свидетельствует таблица маршрутизации.

Поля таблицы маршрутизации объясняются в табл. 10.1.

Таблица 10.1. Поля таблицы маршрутизации

Поле	Описание
Destination	Адрес сети назначения
Gateway	Шлюз по умолчанию
Genmask	Маска сети назначения
Flags	Поле Flags содержит флаги маршрута: <ul style="list-style-type: none"> • U — маршрут активен; • H — маршрут относится не к сети, а к хосту; • G — эта машина является шлюзом, поэтому при обращении к ней нужно заменить MAC-адрес машины получателя на MAC-адрес шлюза (если MAC-адрес получателя почему-то известен); • D — динамический маршрут, установлен демоном маршрутизации; • M — маршрут, модифицированный демоном маршрутизации; • C — запись кеширована; • ! — запрещенный маршрут
Metric	Метрика маршрута, т. е. расстояние к цели в хопах (переходах). Один хоп (переход) означает один маршрутизатор
Ref	Количество ссылок на маршрут. Не учитывается ядром Linux, но в других операционных системах — например, в FreeBSD, вы можете столкнуться с этим полем
Use	Содержит количество пакетов, прошедших по этому маршруту
Iface	Используемый интерфейс
MSS	Максимальный размер сегмента (Maximum Segment Size) для TCP-соединений по этому маршруту
Window	Размер окна по умолчанию для TCP-соединений по этому маршруту
irtt	Протокол TCP гарантирует надежную доставку данных между компьютерами. Для такой гарантии используется повторная отправка пакетов, если они были потеряны. При этом ведется счет времени: сколько нужно ждать, пока пакет дойдет до назначения и придет подтверждение о получении пакета. Если время вышло, а подтверждение так и не было получено, то пакет отправляется еще раз. Это время и называется round-trip time (время «путешествия туда-обратно»). Параметр irtt — это начальное время rtt . В большинстве случаев подходит значение по умолчанию, но для некоторых медленных сетей — например, для сетей пакетного радио, значение по умолчанию слишком короткое, что вызывает ненужные повторы. Параметр irtt можно увеличить командой route . По умолчанию его значение 0

Добавить маршрут в таблицу маршрутизации можно статически (с помощью команды **route**), динамически или комбинированно (например, статические маршруты добавляются при запуске системы, а динамические — по мере работы системы). Статические маршруты добавляются, как правило, командой **route**, запущенной из

сценария инициализации системы. Например, следующая команда задает шлюз по умолчанию для интерфейса eth0:

```
# route add default gw 192.168.181.2 eth0
```

ПРИМЕЧАНИЕ

Чтобы превратить свой компьютер в маршрутизатор (шлюз), нужно первым делом разрешить пересылку (перенаправление) пакетов. Это можно сделать с помощью простой команды: echo "1" > /proc/sys/net/ipv4/ip_forward. Затем следует настроить правила брандмауэра, определяющие, как и какие пакеты нужно пересыпать. Прочитать о настройке брандмауэра можно в моей книге «Серверное применение Linux»¹.

Но после перезагрузки системы добавленная нами запись исчезнет из таблицы маршрутизации. Можно добавить эту команду в сценарии инициализации системы, но это будет некорректно. Есть более корректный способ установки шлюза по умолчанию. В Fedora, Red Hat и других совместимых с ними дистрибутивах (CentOS, ASPLinux) нужно отредактировать файл /etc/sysconfig/network. Переменная GATEWAY содержит IP-адрес шлюза по умолчанию. Пример этого файла приведен в листинге 10.1.

Листинг 10.1. Файл /etc/sysconfig/network: основные сетевые параметры в Fedora

```
NETWORKING=yes
FORWARD_IPV4=yes
HOSTNAME=server.example.com
GATEWAY=0.0.0.0
```

Параметр NETWORKING определяет, будет ли включена поддержка сети (yes — поддержка сети включена, no — выключена). Параметр FORWARD_IPV4 определяет, будет ли включено перенаправление пакетов. На компьютере, являющемся шлюзом, этот параметр должен быть включен (значение yes), на остальных компьютерах сети — выключен (значение no).

Параметр HOSTNAME задает имя узла, GATEWAY — шлюз по умолчанию. Если компьютер является шлюзом, то обычно для этого параметра устанавливается IP-адрес 0.0.0.0.

ИМЕНА И ИХ РАЗРЕШЕНИЕ

Для отображения имени компьютера служит команда hostname. Преобразовать имя компьютера в IP-адрес можно командой host — например: host www.example.com. Эта же команда применяется и для разрешения IP-адреса в символьное имя — например: host 127.0.0.1.

В SUSE для задания шлюза по умолчанию надо отредактировать файл /etc/sysconfig/network/routes (или /etc/route.conf — в старых версиях openSUSE). В него нужно добавить строку вида:

```
default      адрес      [маска]      [интерфейс]
```

¹ <http://www.dkws.org.ua/show/77>, <https://bhv.ru/product/servernoe-primenenie-linux-3-e-izd/>, а также <http://www.dkws.org.ua/show/77>. <http://www.bhv.ru/books/book.php?id=184941>.

Например:

```
default      192.168.181.2
```

Маску и интерфейс указывать необязательно. В этом же файле можно задать все остальные маршруты, т. е., по сути, он хранит таблицу маршрутизации. Маршрут по умолчанию, как правило, указывается последним. Пример файла конфигурации /etc/sysconfig/network/routes (/etc/route.conf) приведен в листинге 10.2.

Листинг 10.2. Файл /etc/sysconfig/network/routes

```
#  
# /etc/sysconfig/network/routes (/etc/route.conf)  
#  
# Этот файл содержит описание статических маршрутов  
#  
# Назначение Шлюз          Маска          Устройство  
#  
192.168.0.0    0.0.0.0        255.255.255.128  eth0  
default        192.168.0.1
```

Кроме файла route.conf, в SUSE вы можете редактировать файл /etc/rc.config, в котором хранится вся информация об имеющихся сетевых интерфейсах. Здесь важно отметить, что речь идет о старых версиях SUSE. Далее мы рассмотрим конфигурационные файлы современных версий openSUSE.

В Debian и Ubuntu вам нужно редактировать файл /etc/network/interfaces. Шлюз по умолчанию задается в нем параметром gateway. В листинге 10.3 приведен пример этого файла. Но позвольте себе несколько комментариев. Как видно из листинга 10.3, производится конфигурация интерфейса eth0, IP-адрес задается статически (static), присваивается IP-адрес 192.168.1.11, маска 255.255.255.0. Шлюз по умолчанию — это компьютер с IP-адресом 192.168.1.1.

Листинг 10.3. Файл /etc/network/interfaces

```
iface eth0 inet static  
address 192.168.1.11  
netmask 255.255.255.0  
gateway 192.168.1.1
```

10.2.2. Команда route: изменение таблицы маршрутизации

Мы уже знакомы с командой route, но мы ее использовали для просмотра таблицы маршрутизации. Сейчас мы научимся ее применять для изменения этой таблицы.

Маршрутизация осуществляется на сетевом уровне модели OSI. Когда маршрутизатор получает пакет, предназначенный для другого узла, его IP-адрес получателя

сравнивается с записями в таблице маршрутизации. Если есть хотя бы частичное совпадение с каким-то маршрутом из таблицы, пакет отправляется по IP-адресу шлюза, связанного с этим маршрутом.

Если совпадений не найдено (т. е. вообще нет маршрута, по которому можно было бы отправить пакет), пакет отправляется на шлюз по умолчанию, если такой задан в таблице маршрутизации. Если шлюза по умолчанию нет, отправителю пакета посыпается ICMP-сообщение «сеть недоступна» (network unreachable).

Команда `route` за один вызов может добавить или удалить только один маршрут. Другими словами, вы не можете сразу добавить или удалить несколько маршрутов. Формат вызова команды `route` следующий:

```
# route [операция] [тип] адресат gw шлюз [метрика] [dev интерфейс]
```

ПРИМЕЧАНИЕ

Команды добавления/удаления маршрута надо вводить от имени пользователя `root`. В современных системах под этой учетной записью входить необязательно — для получения `root`-доступа нужно использовать либо команду `sudo`, либо команду `su`.

Параметр `операция` может принимать два значения: `add` (добавить маршрут) и `del` (удалить маршрут). Параметр `тип` необязательный, он задает тип маршрута: `-net` (маршрут к сети), `-host` (маршрут к узлу) или `default` (маршрут по умолчанию). Параметр `адресат` содержит адрес сети (если задается маршрут к сети), адрес узла (при добавлении маршрута к сети) или вообще не указывается, если задается маршрут по умолчанию.

Параметр `шлюз` задает IP-адрес (или доменное имя) шлюза. Последние два параметра: `метрика` и `dev` — необязательны. Параметр `метрика` задает максимальное число переходов (через маршрутизаторы) на пути к адресату. В Linux он необязательный, в отличие от других ОС. Последний параметр имеет смысл указывать, если в системе установлено несколько сетевых интерфейсов и нужно задать, через какой именно сетевой интерфейс следует отправить пакеты по указанному маршруту.

Команда удаления маршрута выглядит так:

```
# route del адрес
```

В других UNIX-системах есть параметр `-f`, удаляющий все маршруты (`route -f`), но в Linux такого параметра нет. Следовательно, для очистки всей таблицы маршрутизации вам надо будет ввести серию команд `route del`. Изменять таблицу маршрутизации можно, только зарегистрировавшись на компьютере локально. При удаленной регистрации (например, по SSH) легко ошибочно удалить маршрут, по которому вы «вошли в систему». О последствиях такого действия, думаю, говорить не нужно.

Рассмотрим примеры использования команды `route`:

`route add -net 192.76.16.0 netmask 255.255.255.0 dev eth0`

Эта команда добавляет маршрут к сети 192.76.16.0 (сеть класса C, о чем свидетельствует сетевая маска, заданная параметром `netmask`) через устройство `eth0`. Шлюз не указан, просто все пакеты, адресованные сети 192.76.16.0, будут отправлены на интерфейс `eth0`.

❑ route add -net 192.16.16.0 netmask 255.255.255.0 gw 192.76.16.1

Эта команда добавляет маршрут к сети 192.16.16.0 через маршрутизатор 192.76.16.1. Сетевой интерфейс задавать не обязательно, но можно и указать при особом желании.

❑ route add default gw gate1

Эта команда добавляет маршрут по умолчанию. Все пакеты будут отправлены компьютеру с именем gate1. Обратите внимание: мы указываем доменное имя узла вместо IP-адреса.

❑ route add -net 10.1.0.0 netmask 255.0.0.0 reject

Эта команда добавляет запрещающий маршрут. Отправка пакетов по этому маршруту (в сеть 10.1.0.0) запрещена.

Итак, мы добавили необходимые маршруты, пропинговали удаленные узлы, все работает. Теперь нужно сохранить установленные маршруты, чтобы они были доступны при следующей загрузке системы. Для этого в openSUSE нужно отредактировать файл /etc/sysconfig/network/routes (/etc/route.conf — в старых версиях). Мы уже рассматривали этот файл (см. листинг 10.2), поэтому переходим сразу к другому дистрибутиву.

В Fedora/CentOS/ASPLinux (и других RedHat-совместимых дистрибутивах) статические маршруты хранятся в файле /etc/sysconfig/static-routes. Строки в этом файле имеют вид:

```
any net адрес_сети netmask маска gw адрес_шлюза
```

Здесь any означает любой интерфейс. Можно указать конкретный интерфейс, например:

```
eth0 net 192.168.2.0 netmask 255.255.255.0 gw 192.168.1.1
```

Файл /etc/sysconfig/static-routes по умолчанию отсутствует, и при необходимости его нужно создать самостоятельно.

В старых версиях Debian/Ubuntu статические маршруты прописываются вместе с конфигурацией сетевого интерфейса в файле /etc/network/interfaces. С помощью параметров up и down этого файла можно задать команды, которые будут выполняться при «поднятии» (up) и «закрытии» (down) интерфейса. После параметров up и down может следовать любая Linux-команда. Обычно это команда route. Например, при запуске интерфейса eth0 будет добавлен статический маршрут к сети 192.168.3.0 через шлюз 192.168.1.2:

```
up route add -net 192.168.3.0 netmask 255.255.255.0 gw 192.168.1.2
```

Можно также добавить маршрут по умолчанию:

```
up route add default gw 192.168.1.2
```

При «закрытии» интерфейса нужно удалить маршруты, которые использовали этот интерфейс, для этого применяется параметр down:

```
down route del default gw 192.168.1.2
```

```
down route del -net 192.168.3.0
```

В новых версиях Debian/Ubuntu, а также в других дистрибутивах, где используется для настройки сети сервис NetworkManager, конфигурация сети хранится в каталоге /etc/NetworkManager/system-connections/.

10.3. Команды получения информации об узле

10.3.1. Получение информации о доменном имени

Для получения информации о доменном имени используется команда whois:

```
$ whois google.com
Domain Name: GOOGLE.COM
Registry Domain ID: 2138514_DOMAIN_COM-VRSN
Registrar WHOIS Server: whois.markmonitor.com
Registrar URL: http://www.markmonitor.com
Updated Date: 2019-09-09T15:39:04Z
Creation Date: 1997-09-15T04:00:00Z
Registry Expiry Date: 2028-09-14T04:00:00Z
Registrar: MarkMonitor Inc.
Registrar IANA ID: 292
Registrar Abuse Contact Email: abusecomplaints@markmonitor.com
Registrar Abuse Contact Phone: +1.2083895740
...
...
```

Вывод довольно большой, поэтому здесь он сокращен. Команда предоставляет информацию о регистраторе домена, о владельце домена и контактных данных владельца. Информация о контактах может быть скрыта регистратором. В этом случае просмотреть ее нельзя без запроса к регистратору.

10.3.2. Команды *host* и *dig*

Команда host позволяет получить всевозможную информацию о хосте. Если запустить ее без дополнительных опций и указать только имя узла, то вы получите IPv4/IPv6-адреса узла, а также информацию о почтовом сервере домена. Если же запустить ее с опцией -a, то команда сообщит всю информацию о домене:

```
host -a google.com
Trying "google.com"
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 9703
;; flags: qr rd ra; QUERY: 1, ANSWER: 22, AUTHORITY: 0, ADDITIONAL: 0

;; QUESTION SECTION:
;google.com.      IN  ANY

;; ANSWER SECTION:
google.com. 300  IN  A    142.250.186.46
google.com. 300  IN  AAAA 2a00:1450:4001:827::200e
```

```

google.com. 86400 IN CAA 0 issue "pki.goog"
google.com. 86400 IN NS ns2.google.com.
google.com. 86400 IN NS ns4.google.com.
google.com. 86400 IN NS ns1.google.com.
google.com. 86400 IN NS ns3.google.com.
google.com. 3600 IN TXT "MS=E4A68B9AB2BB9670BCE15412F62916164C0B20BB"
google.com. 3600 IN TXT "apple-domain-verification=30afIBcvSuDV2PLX"
google.com. 3600 IN TXT "google-site-verification=wD8N7i1JTNTkezJ49swvWW48f8_
9xveREV4oB-0Hf5o"
google.com. 3600 IN TXT "docusign=05958488-4752-4ef2-95eb-aa7ba8a3bd0e"
google.com. 3600 IN TXT "google-site-verification=TV9-DBe4R80X4v0M4U_bd_
J9cpOJM0nikft0jAgjmsQ"
google.com. 3600 IN TXT "facebook-domain-verification=22rm551cu4k0ab0bxsw
536tlds4h95"
google.com. 3600 IN TXT "docusign=1b0a6754-49b1-4db5-8540-d2c12664b289"
google.com. 3600 IN TXT "v=spf1 include:_spf.google.com ~all"
google.com. 3600 IN TXT "globalsign-smime-dv=CDYX+XFHUw2wml6/Gb8+59BsH31KzUr
6c112BPvqKX8="
google.com. 600 IN MX 10 aspmx.l.google.com.
google.com. 600 IN MX 20 alt1.aspmx.l.google.com.
google.com. 600 IN MX 30 alt2.aspmx.l.google.com.
google.com. 600 IN MX 50 alt4.aspmx.l.google.com.
google.com. 600 IN MX 40 alt3.aspmx.l.google.com.
google.com. 60 IN SOA ns1.google.com. dns-admin.google.com. 431389227 900 900
1800 60

```

Received 911 bytes from 127.0.0.53#53 in 17 ms

Важно понимать, что вы получите в ответ на ввод команды `host -a`. Она выводит DNS-записи прямо из DNS-зоны:

- SOA — начало записи полномочий;
- NS — здесь содержится имя сервера имен;
- A — здесь хранится IP-адрес;
- MX — запись обмена почтой (здесь хранится инфа о почтовом сервере для домена);
- PTR — запись указателей;
- AAAA — запись IPv6-адреса;
- CNAME — аббревиатура канонического имени.

Команда `dig` предоставляет практически ту же информацию, но при этом ее вывод более лаконичен.

10.3.3. Утилита DMitry

Утилита DMitry (Deepmagic Information Gathering Tool) является инструментом по принципу «все в одном». Ее можно использовать для сбора следующей информации:

- записи протокола Whois (получение регистрационных данных о владельцах доменных имен) с применением IP-адреса или доменного имени;
- сведений о хосте от <https://www.netcraft.com/>;
- данных о поддоменах в целевом домене;
- адресов электронной почты целевого домена.

Посредством сканирования портов вы также можете получить списки открытых, фильтрованных и закрытых портов целевого компьютера. Конечно, всю эту информацию можно получить и с помощью других инструментов, но иногда удобнее получить все и сразу.

Пример:

```
dmitry -iwnse example.com  
dmitry -p example.com -f -b
```

Первая команда выводит много полезной информации о файле example.com, в том числе осуществляет в Google поиск e-mail адресов с этого домена, а вторая — выполняет сканирование портов, чтобы можно было понять, какие порты открыты.

10.4. Текстовые браузеры

Если графический режим недоступен (например, на сервере), а по сети побродить хочется, можно использовать текстовый браузер lynx. В некоторых дистрибутивах вместо lynx используются браузеры links и elinks, но суть остается той же — просмотр страниц Интернета в текстовом режиме.

В современных дистрибутивах текстовые браузеры не устанавливаются по умолчанию, поэтому их нужно установить отдельно.

10.5. Команда *ftp*: FTP-клиент

Для открытия соединения с любым FTP-сервером введите команду:

```
ftp <имя или адрес FTP-сервера>
```

Можно просто ввести команду *ftp*, а в ответ на приглашение:

```
ftp>
```

ввести команду:

```
open <имя или адрес FTP-сервера>
```

Лично мне больше нравится первый вариант, поскольку он позволяет сэкономить время. При подключении к серверу вы сможете ввести имя пользователя и пароль:

```
[den@dhsilabs ~]$ ftp
ftp> open ftp.server.com
Connected to ftp. server.com.
220 ftp.server.com (Libra FTP daemon 0.17 20050906)
500 Unrecognized command AUTH
Name (ftp.narod.ru:den): den
331 Password required
Password:
230 Logged in, proceed
Remote system type is UNIX.
ftp>
```

Подключившись к серверу, можно ввести команду `help`, чтобы просмотреть список доступных команд. Для получения справки по той или иной команде введите `help <имя_команды>`. Наиболее популярные команды приведены в табл. 10.2.

Таблица 10.2. Некоторые команды FTP-клиента

Команда	Описание
<code>ls</code>	Вывести содержимое каталога
<code>get</code>	Загрузить файл с сервера
<code>put</code>	Загрузить файл на сервер
<code>mget</code>	Получить несколько файлов с сервера. Допускается использование масок файлов — например: <code>*.grp</code>
<code>mput</code>	Загрузить несколько файлов на сервер
<code>cd</code>	Изменить каталог
<code>mkdir</code>	Создать каталог
<code>rmdir</code>	Удалить пустой каталог
<code>delete</code>	Удалить файл

Кроме `ftp` в Linux есть и другие текстовые FTP-клиенты — например: `NcFTP` (<http://www.ncftp.com>), `lukemftp` (<ftp://ftp.netbsd.org/pub/NetBSD/misc/lukemftp/>), `lftp` (<http://ftp.yars.free.net/projects/lftp/>) и др. Все эти FTP-клиенты не входят в состав дистрибутива, и их нужно устанавливать самостоятельно. Но стоит ли это делать — решать вам. Ведь все они подобны стандартному клиенту `ftp` и обладают двумя-тремя дополнительными функциями, которые, возможно, вам и не понадобятся. Например, `NcFTP` умеет докачивать файлы, а `lftp` — загружать одновременно несколько файлов. В любом случае, вы можете изучить документацию по тому или иному FTP-клиенту (ее легко найти в Интернете), а потом решить, стоит его использовать или нет.

10.6. Команда wget: загрузка файлов

Программа wget — это лучший текстовый менеджер закачки файлов. Программа поддерживает протоколы HTTP, HTTPS и FTP. Использовать ее нужно так:

```
wget [параметры] URL
```

Параметров у wget очень-очень много, и с ними вы ознакомитесь на странице `man wget`. Самые полезные параметры собраны в табл. 10.3.

Таблица 10.3. Некоторые параметры wget

Параметр	Описание
<code>--background</code>	Перейти в фоновый режим после запуска
<code>--quiet</code>	Тихий режим, сообщения wget не выводятся
<code>--input-file=file</code>	Считать URL из файла <i>file</i> , файл не обязательно должен быть в формате HTML. Если вы указали URL в файле и в командной строке, то сначала будут загружены URL из командной строки, а потом из файла
<code>--force-html</code>	Обязательно считать файл, указанный в предыдущем параметре, HTML-файлом
<code>--tries=number</code>	Устанавливает количество попыток загрузки URL
<code>--no-clobber</code>	Если при загрузке файла оборвалось соединение, то этот параметр позволит продолжить загрузку с места обрыва
<code>--continue</code>	Возобновление загрузки файла, если, например, прервалась связь. Этот параметр нужно использовать, если вы забыли указать параметр <code>--no-clobber</code> , а связь прервалась, и вам надо докачать файл, а не начинать его загрузку заново
<code>--wait=seconds</code>	Задает паузу в секундах между загрузками и повторами, что позволяет снизить нагрузку на сервер
<code>--quota=quota</code>	Задает максимальный размер загружаемых файлов в байтах, килобайтах (после числа указывается k) и мегабайтах (после числа указывается m). Квота не работает при загрузке одного файла, поскольку даже если квота превышена, то текущий файл загружается до конца (если есть физически место на диске)
<code>--http-user=user --http-passwd=pass</code>	Задают имя пользователя и пароль при HTTP-аутентификации, тип аутентификации устанавливается автоматически программой
<code>--proxy-user=user --proxy-passwd=pass</code>	Задают имя пользователя и пароль прокси-сервера
<code>--passive-ftp</code>	Пассивный режим FTP, обычно используется при наличии брандмауэра
<code>--recursive</code>	Включить рекурсивную загрузку, которая используется для рекурсивной загрузки сайтов
<code>--level=depth</code>	Максимальная длина рекурсивной загрузки (по умолчанию 5 уровней)

Примеры использования:

```
wget --recursive http://example.com
wget http://example.com/somefile.zip
```

Первая команда создаст пятиуровневую копию сайта **http://example.com**, а вторая просто загрузит файл somefile.zip с **http://example.com**.

10.7. Команды для диагностики сети

Причины отказа сети могут быть физическими и программными. Физические связаны с неработающим сетевым оборудованием или повреждением среды передачи данных. Программные причины возникают из-за неправильной настройки сетевого интерфейса. Как правило, избавиться от программных проблем помогает конфигуратор сети — вы еще раз его запускаете и настраиваете сетевые интерфейсы, только правильно. Если сомневаетесь в ваших действиях, обратитесь за помощью к более опытному коллеге.

Для диагностики работы сети мы будем использовать стандартные сетевые утилиты, которые входят в состав любого дистрибутива Linux. Предположим, что у нас не работает PPPoE/DSL-соединение. Проверить, «поднят» ли сетевой интерфейс, можно с помощью команды `ifconfig`. На рис. 10.3 показано, что сначала я предп

```
user@user-desktop:~$ sudo pon dsl-provider
Password:
Plugin rp-pppoe.so loaded.
user@user-desktop:~$ ifconfig
eth0      Link encap:Ethernet Hwaddr 00:0D:87:88:BC:96
          inet6 addr: fe80::20d:87ff:fe88:bc96/64 Диапазон:Link
          UP BROADCAST RUNNING MULTICAST MTU:1500 Metric:1
          RX packets:629 errors:0 dropped:0 overruns:0 frame:0
          TX packets:121 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:104484 (102.0 KiB) TX bytes:11682 (11.4 KiB)
          Interrupt:11 Base address:0xe800

lo       Link encap:Local Loopback
          inet addr:127.0.0.1 Mask:255.0.0.0
          inet6 addr: ::1/128 Диапазон:Host
          UP LOOPBACK RUNNING MTU:16436 Metric:1
          RX packets:25 errors:0 dropped:0 overruns:0 frame:0
          TX packets:25 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:1744 (1.7 KiB) TX bytes:1744 (1.7 KiB)

ppp0     Link encap:Point-to-Point Protocol
          inet addr:193.254.218.243 P-t-P:193.254.218.129 Mask:255.255.255.255
          UP POINTOPOINT RUNNING NOARP MULTICAST MTU:1488 Metric:1
          RX packets:107 errors:0 dropped:0 overruns:0 frame:0
          TX packets:95 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:3
          RX bytes:32174 (31.4 KiB) TX bytes:6001 (5.8 KiB)

user@user-desktop:~$
```

Рис. 10.3. Программа `rfconfig`

нял попытку установить соединение (ввел команду `sudo pon dsl-provider`), а затем вызвал `ifconfig`, чтобы убедиться, установлено ли соединение. В случае если соединение не было бы установлено, интерфейса `ppp0` в списке не было бы. Интерфейс `eth0` относится к первой сетевой плате (вторая называется `eth1`, третья — `eth2` и т. д.), а интерфейс `lo` — это интерфейс обратной петли, который используется для тестирования программного обеспечения (у вас он всегда будет «поднят»).

Если же интерфейс не «поднят», нужно просмотреть файл `/var/log/messages` сразу после попытки установки сообщения:

```
tail -n 10 /var/log/messages
```

Эта команда просматривает «хвост» файла протокола (выводит последние 10 сообщений). В случае удачной установки соединения сообщения в файле протокола будут примерно следующими:

```
Feb  6 14:28:33 user-desktop pppd[5176]: Plugin rp-pppoe.so loaded.
Feb  6 14:28:33 user-desktop kernel: [17179852.932000] CSLIP: code copyright 1989 Regents
                                         of the University of California
Feb  6 14:28:33 user-desktop kernel: [17179852.944000] PPP generic driver version 2.4.2
Feb  6 14:28:33 user-desktop pppd[5183]: pppd 2.4.4b1 started by root, uid 0
Feb  6 14:28:33 user-desktop pppd[5183]: PPP session is 2838
Feb  6 14:28:33 user-desktop kernel: [17179852.984000] NET: Registered protocol family 24
Feb  6 14:28:33 user-desktop pppd[5183]: Using interface ppp0
Feb  6 14:28:33 user-desktop pppd[5183]: Connect: ppp0 <--> eth0
Feb  6 14:28:33 user-desktop pppd[5183]: Remote message: Login ok
Feb  6 14:28:33 user-desktop pppd[5183]: PAP authentication succeeded
Feb  6 14:28:33 user-desktop pppd[5183]: peer from calling number 00:15:F2:60:28 :97
                                         authorized
Feb  6 14:28:33 user-desktop pppd[5183]: local IP address 193.254.218.243
Feb  6 14:28:33 user-desktop pppd[5183]: remote IP address 193.254.218.129
Feb  6 14:28:33 user-desktop pppd[5183]: primary DNS address 193.254.218.1
Feb  6 14:28:33 user-desktop pppd[5183]: secondary DNS address 193.254.218.27
```

Первая строка — сообщение о том, что загружен модуль поддержки PPPoE. Следующие два сообщения информируют нас о поддержке нашим компьютером протоколов CSLIP и PPP. Затем сообщается, что демон `pppd` запущен, указывается, от чьего имени он запущен (`root`) и версия самого `pppd`. Далее сообщаются имена используемого (`ppp0`) и вспомогательного интерфейсов (помните, что протокол PPPoE подразумевает передачу кадров PPP по Ethernet) — `eth0`. Следующие два сообщения свидетельствуют об удачной регистрации:

```
Feb  6 14:28:33 user-desktop pppd[5183]: Remote message: Login ok
Feb  6 14:28:33 user-desktop pppd[5183]: PAP authentication succeeded
```

Затем система сообщает нам наш IP-адрес, адрес удаленного компьютера, который произвел аутентификацию, а также IP-адреса серверов DNS.

А вот пример неудачной попытки соединения:

```
Feb  6 09:23:48 user-desktop pppd[6667]: PPP session is 2336
Feb  6 09:23:48 user-desktop pppd[6667]: Using interface ppp1
```

```
Feb 6 09:23:48 user-desktop pppd[6667]: Connect: ppp1 <-> eth0
Feb 6 09:23:48 user-desktop pppd[6667]: Remote message: Login incorrect
Feb 6 09:23:48 user-desktop pppd[6667]: Connection terminated.
```

Причина неудачи понятна: имя пользователя или пароль неправильные, о чём красноречиво свидетельствует сообщение `Login incorrect`. Для того чтобы изменить имя пользователя или пароль, следует запустить конфигуратор `pppoeconf`. Но не спешите это делать: если в предыдущий раз соединение было установлено (а настройки соединения вы не изменяли), возможно, нужно обратиться к провайдеру — это явный признак неправильной работы оборудования на стороне провайдера.

Вот еще один пример, характерный для PPPoE:

```
Feb 6 09:23:48 user-desktop pppd[6667]: PPP session is 2336
Feb 6 09:23:48 user-desktop pppd[6667]: Using interface ppp1
Feb 6 09:23:48 user-desktop pppd[6667]: Connect: ppp1 <-> eth0
Feb 6 09:23:48 user-desktop pppd[6667]: Connection terminated.
```

Он свидетельствует о неправильной работе оборудования провайдера. Возможно, нужно перезагрузить точку доступа (`access point`) — просто выключите и включите ее снова. Если это не помогает, обращайтесь к провайдеру.

Наиболее простая ситуация, когда сеть вообще не работает. В этом случае очень легко обнаружить причину неисправности. Если работает устройство, значит, повреждена среда передачи данных (сетевой кабель). В случае с модемной линией нужно проверить, нет ли ее обрыва. В случае с витой парой обрыв маловероятен (хотя возможен), поэтому нужно проверить, правильно ли обжат кабель (возможно, нужно обжать витую пару заново).

Намного сложнее ситуация, когда сеть то работает, то нет. Например, вы не можете получить доступ к какому-нибудь узлу, хотя пять минут назад все работало отлично. Если исключить неправильную работу удаленного узла, к которому вы подключаетесь, следует поискать решение в маршруте, по которому пакеты добираются от вашего компьютера до удаленного узла. Сначала *пропингуем* удаленный узел. Для этого используется команда `ping` (прервать выполнение команды `ping` можно с помощью нажатия комбинации клавиш `<Ctrl>+<C>`):

```
ping example.com.ua
```

```
PING example.com.ua (188.186.114.75) 56(84) bytes of data.
64 bytes from wdt.org.ru (188.186.114.75): icmp_seq=1 ttl=58 time=30.7 ms
64 bytes from wdt.org.ru (188.186.114.75): icmp_seq=2 ttl=58 time=24.8 ms
64 bytes from wdt.org.ru (188.186.114.75): icmp_seq=5 ttl=58 time=12.2 ms
64 bytes from wdt.org.ru (188.186.114.75): icmp_seq=6 ttl=58 time=159 ms
64 bytes from wdt.org.ru (188.186.114.75): icmp_seq=7 ttl=58 time=19.3 ms
64 bytes from wdt.org.ru (188.186.114.75): icmp_seq=9 ttl=58 time=29.0 ms
...
```

В этом случае все нормально. Но иногда ответы от удаленного сервера то приходят, то не приходят. Чтобы узнать, в чём причина (где именно теряются пакеты), нужно выполнить трассировку узла:

```
tracepath example.com.ua
```

В некоторых дистрибутивах вместо команды `tracepath` используется команда `traceroute`, а в Windows — `tracert`. На рис. 10.4 показано выполнение команды `tracepath`. Сразу видно, что есть определенные проблемы с прохождением пакетов до удаленного узла.

```
user@user-desktop:~$ tracepath [REDACTED]
 1: ip-193-254-218-243.romb.net (193.254.218.243)          0.320ms pmtu 1488
 1: ip-193-254-218-129.romb.net (193.254.218.129)          94.739ms
 2: sat-router.romb.net (193.254.218.2)                   16.841ms
 3: border.romb.net (80.91.172.97)                         48.562ms
 4: L9-KTU.rtr.newline.net.ua (80.91.178.81)                109.070ms
 5: utel-gw.ix.net.ua (195.35.65.89)                      asymm 6 54.850ms
 6: dc-m7i-1-ge.interfaces.dc.utel.ua (213.186.112.129)   asymm 7 29.092ms
 7: no reply
 8: no reply
 9: no reply
10: no reply
11: no reply
12: no reply
13: no reply
14: no reply
15: no reply
16: no reply
17: no reply
18: no reply
19: no reply
20: no reply
21: no reply
22: no reply
23: no reply
24: no reply
25: no reply
26: no reply
27: no reply
28: no reply
29: no reply
30: no reply
31: no reply
      Too many hops: pmtu 1488
```

Рис. 10.4. Проблема с прохождением пакетов

Понятно, что по пути пакеты теряются. Для того чтобы выяснить причину, вам нужно обратиться к администратору того маршрутизатора, который не пропускает дальше пакеты. Причина именно в нем. В нашем случае, как видно из рис. 10.4, пакеты доходят до маршрутизатора `dc-m7i-1-ge.interfaces.dc.utel.ua`, а после него движение пакетов прекращается.

Если соединение установлено (о чем свидетельствует наличие «поднятого» интерфейса в выводе `ifconfig`), а веб-страницы не открываются, попробуйте пропинговать любой удаленный узел по IP-адресу. Если не знаете, какой узел пинговать (т. е. не помните ни одного IP-адреса), пропингуйте узел `213.186.114.75`. Если вы получите ответ, а странички по-прежнему не открываются, когда вы вводите символьное имя, значит, у вас проблемы с DNS: сервер провайдера почему-то не передал вашему компьютеру IP-адреса DNS-серверов. Позвоните провайдеру, выясните причину этого, а еще лучше уточните IP-адреса серверов DNS и укажите их в файле `/etc/resolv.conf`.

Формат этого файла прост:

```
nameserver IP-адрес
```

Например:

```
nameserver 193.254.218.1  
nameserver 193.254.218.27
```

Всего можно указать до четырех серверов DNS.

Если же не открывается какая-то конкретная страничка, а все остальные работают нормально, тогда понятно, что причина в самом удаленном сервере, а не в ваших настройках.

В более сложных случаях бывает полезно просмотреть трафик, проходящий по определенному порту. Для этого нужно использовать команду `tcpdump` — например: `tcpdump tcp port 80`. Эта команда отобразит весь трафик, поступающий на TCP-порт 80.

10.8. Команда ssh

Раньше для организации удаленного доступа к консоли сервера использовался протокол Telnet.

Но технологии не стоят на месте, и протокол Telnet устарел. Сейчас им практически никто не пользуется. На смену ему пришел протокол SSH (Secure Shell). Этот протокол, как следует из его названия, представляет собой безопасную оболочку. Главное отличие SSH от Telnet состоит в том, что все данные (включая пароли доступа к удаленному компьютеру и передаваемые по SSH файлы) передаются в зашифрованном виде. Во времена Telnet участились случаи перехвата паролей и другой важной информации, что и стало причиной создания SSH.

SSH использует следующие алгоритмы для шифрования передаваемых данных: BlowFish, 3DES (Data Encryption Standard), IDEA (International Data Encryption Algorithm) и RSA (Rivest-Shamir-Adelman algorithm). Самыми надежными являются алгоритмы IDEA и RSA. Поэтому, если вы передаете действительно конфиденциальные данные, лучше использовать один из этих алгоритмов.

В состав любого дистрибутива Linux входят: SSH-сервер (программа, которая и обеспечивает удаленный доступ к компьютеру, на котором она установлена) и SSH-клиент (программа, позволяющая подключаться к SSH-серверу). Для установки SSH-сервера нужно установить пакет `openssh` (это разновидность SSH-сервера), а для установки SSH-клиента — пакет `openssh-clients`.

Команда `ssh` используется для удаленного подключения к другой Linux-машине, на которой запущен SSH-сервер. При аутентификации по паролю ничего сложного в подключении к другой машине нет:

```
ssh имя_пользователя@имя_машины
```

После установления соединения с сервером `имя_машины` вам будет предложено ввести пароль для `имя_пользователя`.

В качестве адреса можно указать как IP-адрес, так и доменное имя компьютера. Часто используемые опции программы ssh приведены в табл. 10.4.

Таблица 10.4. Опции программы ssh

Опция	Описание
-c blowfish 3des des	Применяется для выбора алгоритма шифрования при условии, что используется первая версия протокола SSH (об этом позже). Можно указать blowfish, des или 3des
-c <i>шифр</i>	Задает список шифров, разделенных запятыми в порядке предпочтения. Опция используется для второй версии SSH. Можно указать blowfish, twofish, arcfour, cast, des и 3des
-f	Переводит ssh в фоновый режим после аутентификации пользователя. Рекомендуется использовать для запуска программы X11. Например: ssh -f server xterm
-l <i>имя_пользователя</i>	Указывает имя пользователя, с правами которого нужно зарегистрироваться на удаленном компьютере. Опцию использовать не обязательно, поскольку удаленный компьютер и так запросит имя пользователя и пароль
-p <i>порт</i>	Определяет порт SSH-сервера (по умолчанию используется порт 22)
-q	«Тихий режим» — станут отображаться только сообщения о фатальных ошибках. Все прочие предупреждающие сообщения в стандартный выходной поток выводиться не будут
-x	Отключает перенаправление X11
-X	Задействовать перенаправление X11. Полезна при запуске X11-программ
-1	Использовать только первую версию протокола SSH
-2	Использовать только вторую версию протокола SSH. Вторая версия протокола более безопасна, поэтому при настройке SSH-сервера нужно применять именно ее

Существует практика *аутентификации по ключу* (Public Key Authentication), а не по паролю. Преимущества и недостатки этого подхода мы анализировать не будем, т. к. здесь одни преимущества: пароль никто не подберет, т. к. его нет. Это самый надежный способ аутентификации. Жаль, что в реальных условиях (из личного опыта) используется он реже обычной аутентификации по паролю.

Принцип аутентификации по ключу следующий: создается пара ключей: открытый и закрытый. Закрытый ключ хранится на стороне клиента и в идеале он недоступен ни для кого. Открытый загружается на сервер, к которому нужно получить доступ.

В идеальном мире ключи должны генерировать сами пользователи. Сгенерировать ключи можно с помощью команды ssh-keygen, а также используя функционал SSH-клиента (например, возможность создания ключевой пары есть в Bitvise SSH Client и во многих других клиентах). Если у вас Linux, ничего устанавливать не нужно. Введите команду:

```
$ mkdir ~/.ssh; ssh-keygen -t rsa -b 2048 -f ~/.ssh/rsa
```

Здесь мы генерируем RSA-ключи длиной 2048 битов. Когда ключи будут сгенерированы, публичный ключ будет доступен в файле с именем `~/.ssh/rsa.pub`. Этот файл нужно передать админу на сервер. Как это сделать — выбирайте сами: можно отправить по e-mail, можно загрузить командой `scp` прямо на сервер:

```
$ scp ~/.ssh/rsa.pub user@example.com:~
```

Эта команда загрузит открытый ключ в домашний каталог пользователя `user` — но только в том случае, если администратор еще не успел выключить аутентификацию по паролю. Скорее всего, администратор уже запретил входить по SSH посредством паролей, поэтому просто отправьте ему ключ на электронку.

Дальше ключом должен заниматься администратор. Ему нужно поместить содержимое полученного файла ключа в файл `~/.ssh/authorized_keys` того пользователя, вход по которому настраивается, т. е. в нашем случае тильда (~) будет означать `/home/<имя>`.

Если вы сам себе администратор, то прямо сейчас запретите вход по паролю и перезапустите SSH. Для этого откройте в любом текстовом редакторе файл `/etc/ssh/sshd_config` и установите в значение `no` директиву `PasswordAuthentication`:

```
PasswordAuthentication no
```

Также нужно убедиться, что две следующие директивы установлены так:

```
PubkeyAuthentication yes  
AuthorizedKeysFile %h/.ssh/authorized_keys
```

Первая включает аутентификацию по публичному ключу, а вторая указывает имя файла, в котором должны храниться ключи.

Осталось перезагрузить ssh:

```
systemctl ssh restart
```

На стороне клиента войти по ключу на SSH-сервер можно так:

```
ssh -i ~/.ssh/rsa.pub user@example.com
```

Здесь мы указываем имя файла ключа, имя пользователя и имя сервера.

10.9. Сетевой сканер `nmap`

10.9.1. Что такое `nmap`?

Команда `nmap` предназначена для сканирования как отдельных узлов, так и целых сетей с любым количеством узлов, определения состояния узлов сканируемой сети, а также открытых портов на этих узлах. Сетевой сканер `nmap` использует множество самых разных методов сканирования: UDP, TCP connect, TCP SYN, ICMP (ping) и пр. Подробно о различиях в методах сканирования можно прочитать в справочной системе (команда `man nmap`) или по адресу:

http://cherepovets-city.ru/insecure/runmap/nmap_manpage-ru.htm.

Сразу хочу отметить, что мы рассмотрим только практическое применение nmap, а с теорией вы и сами сможете ознакомиться в статье на русском языке, размещенной по указанному адресу (оригинальная страница руководства man на английском). Просто не вижу смысла переписывать сюда страницу руководства, которую можно и так свободно найти в Интернете.

Сканер nmap поддерживает множество дополнительных возможностей: определение операционной системы узла, «невидимое» сканирование, вычисление времени задержки, параллельное сканирование, определение неактивных узлов с помощью параллельного ping-опроса, определение наличия брандмауэров, прямое RPC-сканирование, произвольное указание IP-адресов и номеров портов сканируемых сетей.

Результат работы сканера: список отсканированных портов удаленного узла с указанием номера и состояния порта, используемого протокола, а также названия службы, использующей этот порт. У порта может быть три состояния: открыт, фильтруемый, нефильтруемый. Первое состояние означает, что удаленная машина прослушивает этот порт. Второе состояние сигнализирует, что брандмауэр блокирует доступ к этому порту и nmap не может определить его состояние. Третье состояние означает, что порт просто закрыт.

Сканер nmap очень популярен. Причем до такой степени, что его можно увидеть даже в фильме «Матрица». В большинстве случаев смотришь фильмы и видишь операционные системы какого-то непонятного происхождения. Создается впечатление, что все эти ОС создавались специально для фильма, чтобы не делать рекламы Microsoft или Apple. А тут старый добный знакомый — nmap. Не верите? Значит, вы невнимательно смотрели «Матрицу»...

10.9.2. Где мне взять nmap?

Сканер nmap — это не нечто мистическое. Это вполне реальная программа, причем абсолютно бесплатная и входящая в состав многих дистрибутивов. Например, в Ubuntu, чтобы установить nmap, нужно ввести команду:

```
sudo apt-get install nmap
```

Если в вашем дистрибутиве не оказалось nmap, его можно скачать с официального сайта (там же вы найдете и Windows-версию): <http://nmap.org/download.html>.

10.9.3. Примеры использования nmap

Теперь рассмотрим основы использования сканера nmap. Синтаксис его вызова такой (выполнять команду nmap нужно с привилегиями root):

```
# nmap параметры цель
```

Цель — это узел или список узлов для сканирования. Все опции мы изучать не станем — для этого есть страница руководства (см. ранее). Рассмотрим лишь самые интересные.

Предположим, что мы хотим знать, какая операционная система запущена на удаленном узле. Для этого нужно запустить nmap с опцией -O:

```
# nmap -O узел
```

Вот результат сканирования узла с запущенной ОС Ubuntu 10.04:

```
Starting Nmap 7.60 ( https://nmap.org ) at 2022-03-01 09:17 EET
Nmap scan report for 192.168.1.1
Host is up (0.0040s latency).
Not shown: 999 closed ports
PORT      STATE SERVICE
22/tcp    open  ssh
MAC Address: XX:XX:XX:XX:XX:XX
Device type: general purpose
Running: Linux 2.6.X
OS details: Linux 2.6.19 - 2.6.32
Network Distance: 1 hop

OS detection performed. Please report any incorrect results at nmap.org/submit/ .
Nmap done: 1 IP address (1 host up) scanned in 3.20 seconds
```

Как видно из вывода, узел работает под управлением Linux с ядром 2.6.19–2.6.32. Как раз в Ubuntu 10.04 ядро 2.6.32. Также видно, что открыт один порт — 22 (ssh), все остальные порты закрыты.

Если nmap запущен для сканирования узла локальной сети, то он также сообщает и MAC-адрес узла (свой MAC-адрес я в этом выводе скрыл).

Относительно любого сканирования нужно отметить, что администраторы не любят, когда несанкционированно сканируют их узлы. Во-первых, это не этично. Во-вторых, у вас могут быть проблемы — все зависит от организации, которую вы сканируете. Само сканирование не противозаконно, но все мы понимаем, что просто так никто ничего не сканирует. Если вам хочется поэкспериментировать, можно использовать тестовый сервер nmap: scanme.nmap.org.

Сканер позволяет просканировать сразу несколько узлов. Для этого их адреса нужно указать так:

```
nmap 192.168.1.1-100
```

В приведенном примере будут просканированы узлы от 192.168.1.1 до 192.168.1.100. Можно также указать имена узлов через пробел — например, так:

```
nmap host1 host2
```

Чтобы просканировать узел на предмет открытых портов, укажите просто его адрес, никаких опций указывать не нужно — например:

```
nmap 192.168.1.2
```

Вывод будет примерно таким:

```
Interesting ports on den (192.168.1.2):
Not shown: 1712 closed ports
```

```
PORT STATE SERVICE
22/tcp open ssh
80/tcp open http
```

Nmap done: 1 IP address (1 host up) scanned in 0.240 seconds

Как уже отмечалось ранее, nmap позволяет также узнать список запущенных служб, для этого нужно использовать опцию -sV:

```
nmap -sV 192.168.1.2
```

Вот вывод этой команды:

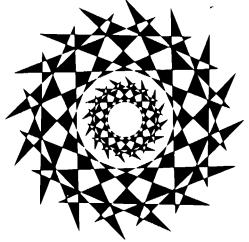
```
Starting Nmap 5.21 ( nmap.org ) at 2010-08-23 10:45 EST
Nmap scan report for den (192.168.1.2)
Host is up (0.099s latency).
Not shown: 962 closed ports, 32 filtered ports
PORT STATE SERVICE VERSION
22/tcp open ssh OpenSSH 4.7p1 Debian 8ubuntu1.2 (protocol 2.0)
80/tcp open http Apache httpd 2.2.8 ((Ubuntu) PHP/5.2.4-2ubuntu5.10 with Suhosin-Patch)
Service Info: OS: Linux
```

Service detection performed. Please report any incorrect results at nmap.org/submit/ .
Nmap done: 1 IP address (1 host up) scanned in 10.51 seconds

Еще один интересный тип сканирования — кто онлайн? Сканер позволяет просканировать сеть и определить доступные узлы. Для этого служит опция -sP:

```
nmap 192.168.1.1-254
```

На этом все. Теперь, когда вы знаете, что собой представляет сканер nmap, для вас не составит особого труда ознакомиться со страницей его руководства.



ГЛАВА 11

Команды системного администратора

11.1. Программы разметки диска

В этой главе мы рассмотрим две программы для разметки диска: классическую программу `fdisk` и более продвинутую — `parted`. Реально для разметки диска (если вам придется это делать) вы будете использовать `parted`, поскольку она умеет изменять размеры разделов, что пригодится при переразметке диска. А вот `fdisk` можно использовать разве что для разметки новых жестких дисков. Изменить размер раздела без потери данных `fdisk` не может: вам придется удалить один из разделов, а вместо него создать несколько разделов меньшего размера — только так и не иначе.

Зато `fdisk` установлена по умолчанию во всех дистрибутивах, и ее не нужно доуставливать самостоятельно.

11.1.1. Программа `fdisk`

Введите команду (можно использовать короткие имена):

```
# fdisk <имя_устройства>
```

Например, если вы подключили винчестер как вторичный мастер, то команда будет следующей:

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

Чтобы убедиться, что диск не размечен, введите команду `r`. Программа выведет пустую таблицу разделов (рис. 11.1).

Самое время создать раздел. Для этого служит команда `n` (рис. 11.2). Кстати, для справки можете ввести команду `m`, которая выведет список доступных команд `fdisk` (рис. 11.3).

После ввода команды `n` программа попросит вас уточнить, какого типа должен быть раздел. Можно выбрать первичный или расширенный раздел. В нашем случае больше подойдет первичный, поэтому вводим букву `p`. Затем нужно ввести номер раздела. Поскольку это первый раздел, то вводим `1`. После чего `fdisk` попросит вве-

сти номер первого цилиндра. Это первый раздел, поэтому вводим номер 1. После ввода первого цилиндра нужно ввести номер последнего цилиндра. Чтобы не вычисывать на калькуляторе номер цилиндра, намного проще ввести размер раздела. Делается это так: +<размер>M. После числа должна идти именно буква M, иначе размер будет воспринят в байтах, а этого нам не нужно. Например, если вы хотите создать раздел размером 10 Гбайт, то введите +10240M.

```
Command (m for help): p
Disk /dev/sda: 1825 MB, 1825360896 bytes
64 heads, 63 sectors/track, 884 cylinders
Units = cylinders of 4032 * 512 = 2064384 bytes

Device Boot      Start        End      Blocks   Id  System
Command (m for help): _
```

Рис. 11.1. Таблица разделов пуста

```
Command (m for help): n
Command action
  e   extended
  p   primary partition (1-4)
p
Partition number (1-4): 1
First cylinder (1-884, default 1): 1
Last cylinder or +size or +sizeM or +sizeK (1-884, default 884): +700M
```

Рис. 11.2. Создание нового раздела

```
64 heads, 63 sectors/track, 884 cylinders
Units = cylinders of 4032 * 512 = 2064384 bytes

Device Boot      Start        End      Blocks   Id  System
Command (m for help): m
'Command action
  a   toggle a bootable flag
  b   edit bsd disklabel
  c   toggle the dos compatibility flag
  d   delete a partition
  l   list known partition types
  m   print this menu
  n   add a new partition
  o   create a new empty DOS partition table
  p   print the partition table
  q   quit without saving changes
  s   create a new empty Sun disklabel
  t   change a partition's system id
  u   change display/entry units
  v   verify the partition table
  w   write table to disk and exit
  x   extra functionality (experts only)

Command (m for help): _
```

Рис. 11.3. Список команд программы fdisk

Для создания второго раздела опять введите команду `n`. Программа вновь попросит тип раздела, номер первого цилиндра (это будет номер последнего цилиндра первого раздела плюс 1) и размер раздела. Если вы хотите создать раздел до конца диска, то просто введите номер последнего цилиндра.

Теперь посмотрим на таблицу разделов. Для этого опять введите команду `p` (рис. 11.4).

```
Command (m for help): p
Disk /dev/sda: 1825 MB, 1825360896 bytes
64 heads, 63 sectors/track, 884 cylinders
Units = cylinders of 4032 * 512 = 2064384 bytes

   Device Boot      Start        End      Blocks   Id  System
/dev/sda1            1       340     685408+   83  Linux
/dev/sda2       341       884    1096704   83  Linux

Command (m for help): _
```

Рис. 11.4. Создание второго раздела: вывод таблицы разделов

По умолчанию программа fdisk создает Linux-разделы. Если вы собираетесь работать только в Linux, можно оставить и так, но ведь не у всех есть Linux. Если вы снимете этот винчестер, чтобы, например, переписать у товарища большие файлы, то вряд ли сможете комфортно с ним работать. Прочитать данные (например, с помощью Total Commander) вам удастся, а что-либо записать — уже нет. Поэтому давайте изменим тип разделов. Для этого служит команда `t`. Введите ее. Программа запросит у вас номер раздела и тип файловой системы. С номером раздела все ясно, а вот с кодом файловой системы сложнее. Введите `L`, чтобы просмотреть доступные файловые системы (рис. 11.5).

0	Empty	1e	Hidden W95 FAT1	80	Old Minix	be	Solaris boot
1	FAT12	24	NEC DOS	81	Minix / old Lin bf	Solaris	
2	XENIX root	39	Plan 9	82	Linux swap / So c1	DRDOS/sec (FAT-	
3	XENIX usr	3c	PartitionMagic	83	Linux	c4	DRDOS/sec (FAT-
4	FAT16 <32M	40	Venix 80286	84	OS/2 hidden C:	c6	DRDOS/sec (FAT-
5	Extended	41	PPC PReP Boot	85	Linux extended	c7	Syrinx
6	FAT16	42	SFS	86	NTFS volume set da	Non-FS data	
7	HPFS/NTFS	4d	QNX4.x	87	NTFS volume set db	CP/M / CTOS / .	
8	AIX	4e	QNX4.x 2nd part	88	Linux plaintext de	Dell Utility	
9	AIX bootable	4f	QNX4.x 3rd part	8e	Linux LVM	df	BootIt
a	OS/2 Boot Manag	50	OnTrack DM	93	Amoeba	e1	DOS access
b	W95 FAT32	51	OnTrack DM6 Aux	94	Amoeba BBT	e3	DOS R/O
c	W95 FAT32 (LBA)	52	CP/M	9f	BSD/OS	e4	SpeedStor
e	W95 FAT16 (LBA)	53	OnTrack DM6 Aux	a0	IBM Thinkpad hi eb	BeOS fs	
f	W95 Ext'd (LBA)	54	OnTrackDM6	a5	FreeBSD	ee	EFI GPT
10	OPUS	55	EZ-Drive	a6	OpenBSD	ef	EFI (FAT-12/16/
11	Hidden FAT12	56	Golden Bow	a7	NeXTSTEP	f0	Linux/PA-RISC b
12	Compaq diagnost	5c	Priam Edisk	a8	Darwin UFS	f1	SpeedStor
14	Hidden FAT16 <3	61	SpeedStor	a9	NetBSD	f4	SpeedStor
16	Hidden FAT16	63	GNU HURD or Sys	ab	Darwin boot	f2	DOS secondary
17	Hidden HPFS/NTF	64	Novell Netware	b7	BSDI fs	fd	Linux raid auto
18	AST SmartSleep	65	Novell Netware	b8	BSDI swap	fe	LANstep
1b	Hidden W95 FAT3	70	DiskSecure Mult	bb	Boot Wizard hid ff	BBT	
1c	Hidden W95 FAT3	75	PC/IX				
			Hex code (type L to list codes): _				

Рис. 11.5. Коды файловых систем

Код FAT32 — **b**. Введите его, и вы увидите сообщение программы, что тип файловой системы изменен (рис. 11.6).

Еще раз введите команду **p**, чтобы убедиться, что все нормально. Для сохранения таблицы разделов введите **w**, а для выхода без сохранения изменений — **q**.

Процедура добавления нового диска на виртуальном сервере описана в разд. 11.1.3.

```
Command (m for help): t
Partition number (1-4): 2
Hex code (type L to list codes): b
Changed system type of partition 2 to b (W95 FAT32)

Command (m for help): _
```

Рис. 11.6. Тип файловой системы изменен

11.1.2. Программа parted

Утилита **parted** (название — сокращение от PARTition EDitor) представляет собой консольную программу, которая используется для создания, удаления, копирования, изменения размера и размещения разделов диска.

Программа поддерживает следующие таблицы разделов: BSD, MAC, MSDOS, PC98, SUN, GPT, а также обеспечивает прямой доступ (raw access) к диску, что полезно при работе с логическими томами (LVM) и RAID-массивами.

Несмотря на то что **parted** поддерживает множество файловых систем, не для всех файловых систем доступны все выполняемые программой действия. В табл. 9.1 представлена информация о действиях, которые можно выполнить над той или иной файловой системой.

Таблица 11.1. Поддерживаемые действия

Файловая система	Обнаружение	Создание	Изменение размера	Копирование	Проверка
ext3	+		+	+	+
ext2	+	+	+	+	+
fat32	+	+	+	+	+
fat16	+	+	+	+	+
ntfs	+	+	+	+	+
linux-swap	+	+	+	+	+
ReiserFS	+	+	+	+	+
JFS	+	—	—	—	—
XFS	+	—	—	—	—
UFS	+	—	—	—	—

Программа не умеет создавать разделы ext3 и ext4, но она способна создать раздел ext2, который можно без особых проблем преобразовать в ext3 или ext4 (см. главу 4). Разделы типов JFS, XFS и UFS только обнаруживаются программой, но она не может выполнять над ними никаких действий.

ПРИМЕЧАНИЕ

Для работы с NTFS-разделами обязательна установка пакета linux-ntfs.

Запустим parted:

```
# parted <имя устройства>
```

Например:

```
# parted /dev/sda
```

СОВЕТ

Не забывайте, что программа должна быть выполнена от имени root! Получить права root можно с помощью команды sudo — например: sudo parted /dev/sda (рис. 11.7).

```
denix@denis-desktop:~$ sudo parted /dev/sda
GNU Parted 1.8.8.1 159-le0e
Использование /dev/sda
Добро пожаловать в GNU Parted! Наберите 'help' для получения списка команд.
(parted)
```

Рис. 11.7. Программа parted

Введите команду print для просмотра имеющихся разделов:

```
(parted) print
Disk geometry for /dev/sda: 0.000-9990.109 megabytes
Disk label type: msdos
Minor Start End Type Filesystem Flags
1 0.031 512.000 primary linux-swap
2 512.000 9990.109 primary ext2 boot
```

Первая колонка — это номер раздела, вторая и третья — смещение (в мегабайтах) от начала диска. Следующая колонка — тип раздела, далее — тип файловой системы. Последняя колонка — флаги. Например: boot — загрузочный раздел.

Введите команду help, чтобы увидеть список команд parted (рис. 11.8). Команды parted приведены в табл. 11.2.

ПРИМЕЧАНИЕ

Помнить формат каждой команды необязательно. Если вы введете команду, но не укажете ее параметры, то они будут запрошены.

Как видите, программа parted намного эффективнее fdisk. Любителям графического интерфейса можно порекомендовать графическую версию этой программы — GParted (рис. 11.9).

```

GNU Parted 1.8.8.1.159-1e0
Использование /dev/sda
Добро пожаловать в GNU Parted! Наберите 'help' для получения списка команд.
(parted) help
check НОМЕР           производит простую проверку файловой системы
cp [ИЗ_УСТРОЙСТВА] ИЗ_НОМ В_НОМ скопировать файловую систему на другой раздел
help [КОМАНДА]         распечатать общую справку или справку по
                       КОМАНДЕ
mklabel, mktable ТИП_МЕТКИ   создать новую метку диска (таблицу раздела)
mkfs НОМЕР ТИП_ФС        создать файловую систему ТИП_ФС на разделе
                           НОМЕР
mkpart ТИП_РАЗД [ТИП_ФС] НАЧ КОН создать раздел
mkpartfs ТИП_РАЗД ТИП_ФС НАЧ КОН создать раздел с файловой системой
move НОМЕР НАЧ КОН       переместить файловую систему НОМЕР
name НОМЕР ИМЯ          назначает имя разделу НОМЕР на ИМЯ
print [devices|free|list,all|НОМЕР]   отображает таблицу разделов, доступные
                           устройства, свободное место, все найденные разделы или определённый
                           раздел
quit                     выйти из программы
rescue НАЧАЛО КОНЦЕЦ   восстановить потерянный раздел в промежутке
                           от НАЧАЛА до КОНЦА
resize НОМЕР НАЧ КОН   изменить размер файловой системы на разделе
                           НОМЕР
rm НОМЕР                 удалить раздел НОМЕР
select УСТРОЙСТВО      выбор устройства для редактирования
set НОМЕР ФЛАГ СОСТОЯНИЯ  изменить ФЛАГ на разделе НОМЕР
toggle [НОМЕР [ФЛАГ]]    переключает состояния ФЛАГА на разделе НОМЕР
unit УСТРОЙСТВО        установить устройство по умолчанию на
                           УСТРОЙСТВО
version                  отображает текущую версию GNU Parted
                           и информацию о лицензии
(parted)

```

Рис. 11.8. Команда help

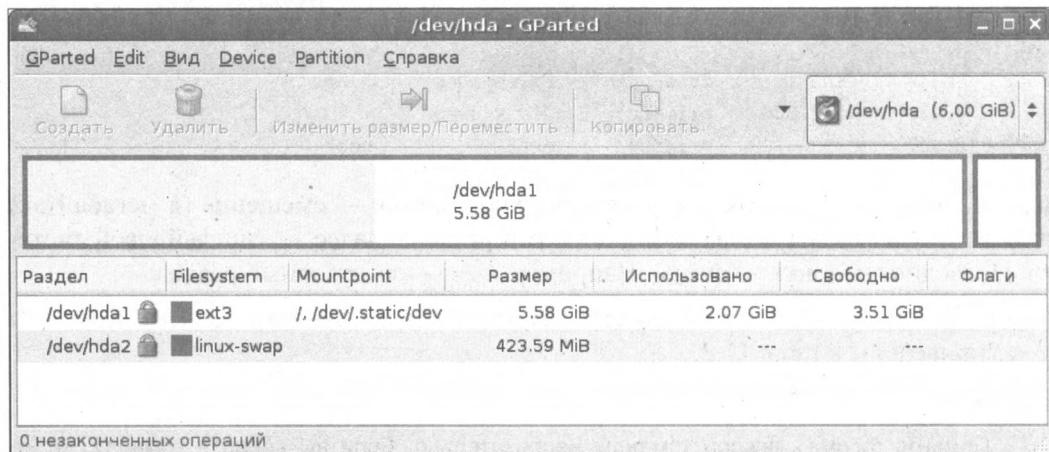


Рис. 11.9. Программа GParted

Таблица 11.2. Основные команды *parted*

Команда	Описание
check <i>n</i>	Проверить раздел с номером <i>n</i>
cp [устройство] <i>n m</i>	Копировать файловую систему из раздела <i>n</i> в раздел <i>m</i> , устройство — это номер устройства, где находится раздел <i>n</i> . Если устройство не задано, то считается, что используется текущее устройство
mklabel <i>тип</i>	Создает новую метку диска
mktable <i>тип</i>	Создает новую таблицу разделов
mkfs <i>n тип_фс</i>	Создает файловую систему заданного типа на разделе <i>n</i>
mkpart <i>тип [фс] нач кон</i>	Создать раздел указанного типа, [фс] — необязательный параметр, задающий тип файловой системы. Параметры нач и кон задают начало и конец раздела
move <i>n нач кон</i>	Переместить раздел с номером <i>n</i> , нач и кон — конечные «координаты» раздела, его начало и конец, заданные как смещение от начала диска в мегабайтах
print [devices free all <i>n</i>]	Отображает таблицу разделов (если параметры не заданы), список устройств (devices), свободное место (free), все найденные разделы (all) или информацию о разделе с номером <i>n</i>
quit	Выход из программы
rescue <i>нач кон</i>	Восстанавливает потерянный раздел в промежутке, заданном параметрами нач и кон
resize <i>n нач кон</i>	Изменяет размер раздела <i>n</i>
rm <i>n</i>	Удаляет раздел с номером <i>n</i>
select <i>устройство</i>	Выбирает устройство для редактирования, вам нет необходимости выходить из программы для изменения устройства
set <i>n флаг состояние</i>	Изменяет состояние флага для раздела <i>n</i> . Доступные флаги описаны в табл. 9.3
unit <i>устройство</i>	Устанавливает устройство по умолчанию
version	Выводит версию parted

Таблица 11.3. Флаги разделов

Флаг	Тип раздела	Описание
boot	Mac, msdos, pc98	Флаг загрузочного раздела. Нужен для некоторых операционных систем
lba	msdos	Нужен для MS-DOS и MS Windows старше 9x, чтобы эти ОС использовали для раздела режим линейной адресации (LBA)

Таблица 11.3 (окончание)

Флаг	Тип раздела	Описание
swap	Mac	Устанавливается, если раздел является разделом подкачки Linux
root	Mac	Устанавливается, если раздел является корневым разделом Linux
raid	msdos	Раздел используется в RAID-массиве
lvm	msdos	Раздел используется как физический том в LVM
hidden	msdos, pc98	Скрытый раздел, устанавливается, если нужно скрыть его от операционных систем семейства Microsoft

11.1.3. Добавление диска на виртуальном сервере

Добавление еще одного диска на виртуальном сервере выполняется в панели управления этим сервером. Процедура добавления зависит от самой панели. Но как правило, нужно выбрать сервер, перейти к списку его дисков и добавить еще один. Перечень требуемых для этого конкретных действий можно найти в справочной системе по панели управления или, в крайнем случае, обратиться в техническую поддержку — там точно подскажут. Обычно для добавления нового устройства сервер придется выключить. Панель управления предложит это сделать самостоятельно перед добавлением диска или же перезагрузит сервер в процессе операции добавления — все как на настоящем сервере. «Горячая замена» на виртуальных серверах работает не всегда.

После добавления диска нужно подключиться к серверу. Посмотрим, какие диски у нас теперь есть. Для этого введите команду:

```
fdisk -l
```

Вывод будет примерно таким (рис. 11.10) — он зависит от размеров созданных дисков. Из вывода видно, что в системе имеются два диска: /dev/sda и /dev/sdb. Первый является загрузочным, а второй — с размером 10 GiB — это и есть диск, который мы только что добавили.

Сначала на втором диске нужно создать разметку. Выполните для этого команду:

```
fdisk /dev/sdb
```

Дальнейшая работа с диском осуществляется путем ввода команд. Основные команды fdisk (если кто забыл):

- n — создать новый раздел;
- d — удалить раздел;
- p — вывести список разделов;
- m — справка;

```
root@ubuntu1804:~# fdisk -l
[ 220.571005] print_req_error: I/O error, dev fd0, sector 0
[ 220.594977] print_req_error: I/O error, dev fd0, sector 0
Disk /dev/sda: 30 GiB, 32212254720 bytes, 62914560 sectors
Units: sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes
Disklabel type: dos
Disk identifier: 0x4b30d60f

Device      Boot   Start     End   Sectors  Size Id Type
/dev/sda1    *       2048  1953791  1951744  953M 83 Linux
/dev/sda2        1953792 62890625 60936834 29.1G 8e Linux LVM

Disk /dev/sdb: 10 GiB, 10737418240 bytes, 20971520 sectors
Units: sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes
```

Рис. 11.10. Список дисков и разделов

- q — выход без сохранения;
- w — сохранить и выйти.

Введите команду n, затем укажите тип раздела (p — основной), номер раздела (1 — первый), номер первого сектора (просто нажмите клавишу <Enter>), номер последнего сектора (просто нажмите клавишу <Enter>). Так мы создали один первичный раздел размером во весь диск (рис. 11.11). Выполните в завершение команду w для сохранения изменений и выхода из программы.

```
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes
root@ubuntu1804:~# fdisk /dev/sdb

Welcome to fdisk (util-linux 2.31.1).
Changes will remain in memory only, until you decide to write them.
Be careful before using the write command.

Device does not contain a recognized partition table.
Created a new DOS disklabel with disk identifier 0xb245b7df.

Command (m for help): n
Partition type
  p  primary (0 primary, 0 extended, 4 free)
  e  extended (container for logical partitions)
Select (default p): p
Partition number (1-4, default 1): 1
First sector (2048-20971519, default 2048):
Last sector, +sectors or +size{K,M,G,T,P} (2048-20971519, default 20971519):

Created a new partition 1 of type 'Linux' and of size 10 GiB.

Command (m for help):
```

Рис. 11.11. Создан раздел диска

Итак, раздел создан. Осталось создать файловую систему и подмонтировать его. Введите команду:

```
mkfs.ext4 /dev/sdb1
```

Эта команда создаст на разделе `/dev/sdb1` файловую систему `ext4` (рис. 11.12).

```
Command (m for help): m
Command (m for help): w
The partition table has been altered.
Calling ioctl() to re-read partition table.
Syncing disks.

root@ubuntu1804:~# mkfs.ext4 /dev/sdb1
mke2fs 1.44.1 (24-Mar-2018)
Creating filesystem with 2621184 4k blocks and 655360 inodes
Filesystem UUID: e5e029af-7253-40d1-b8b6-3091eb7c27f4
Superblock backups stored on blocks:
            32768, 98304, 163840, 229376, 294912, 819200, 884736, 1605632

Allocating group tables: done
Writing inode tables: done
Creating journal (16384 blocks): done
Writing superblocks and filesystem accounting information: done

root@ubuntu1804:~#
```

Рис. 11.12. Создана файловая система

Теперь нужно подмонтировать диск к какому-нибудь каталогу:

```
mkdir /mnt/sdb1
mount /dev/sdb1 /mnt/sdb1
```

Каталог `/mnt/sdb1` — это точка монтирования (рис. 11.13), он может называться иначе, но должен существовать на момент монтирования диска. Обратиться к файлам нового диска можно будет через этот каталог.

Осталось только обеспечить автоматическое монтирование этого диска при загрузке сервера. Для этого введите команду:

```
echo '/dev/sdb1 /mnt/sdb1 ext4 defaults 0 0' >> /etc/fstab
```

Эта команда добавит нужную строку в файл `/etc/fstab`.

На этом всё: мы создали раздел на диске, создали файловую систему (отформатировали — в терминологии Windows) и подмонтировали созданный раздел к каталогу корневой файловой системы.

```
Command (m for help): w
The partition table has been altered.
Calling ioctl() to re-read partition table.
Syncing disks.

root@ubuntu1804:~# mkfs.ext4 /dev/sdb1
mke2fs 1.44.1 (24-Mar-2018)
Creating filesystem with 2621184 4k blocks and 655360 inodes
Filesystem UUID: e5e029af-7253-40d1-b8b6-3091eb7c27f4
Superblock backups stored on blocks:
    32768, 98304, 163840, 229376, 294912, 819200, 884736, 1605632

Allocating group tables: done
Writing inode tables: done
Creating journal (16384 blocks): done
Writing superblocks and filesystem accounting information: done

root@ubuntu1804:~# mkdir /mnt/sdb1
root@ubuntu1804:~# mount /dev/sdb1 /mnt/sdb1
root@ubuntu1804:~# ls /mnt/sdb1
lost+found
root@ubuntu1804:~# _
```

Рис. 11.13. Диск подмонтирован

11.1.4. Расширение существующего диска

Как и в случае добавления нового диска, эту операцию достаточно просто выполнить в панели управления сервером, однако операционная система не увидит изменения, пока ей явно не указать на них. Для этого мы воспользуемся утилитой parted, обладающей необходимым функционалом, и продемонстрируем процесс расширения группы томов LVM в OC Linux Ubuntu. Дело в том, что сейчас не найти сервер без LVM — разве что самому становить Linux без LVM и перенести ее в облако.

Прежде всего определим, сколько сейчас дискового пространства нам доступно:

```
df -h
```

Общий размер группы томов /dev/mapper/vgroup1-root составляет 19 Гбайт (рис. 11.14). Наша задача — расширить размер этой группы томов до полного размера диска.

Пересканируем текущую аппаратную конфигурацию, чтобы система увидела новый объем накопителя:

```
echo 1 > /sys/block/sda/device/rescan
```

и запустим утилиту parted (именно она используется для работы с разделами диска):

```
parted
```

Введите команду `r` для просмотра имеющихся разделов (рис. 11.15). Запомните номер раздела, который мы будем расширять (**2**), и новый размер диска (**42.9GB**).

Запустите команду изменения раздела:

```
resizepart
```

```
- Bitvise xterm - root@ubuntu1804: ~
Last login: Fri Jun 29 07:56:02 2018 from 5.101.73.10
root@ubuntu1804:~# df -h
Filesystem      Size   Used  Avail Use% Mounted on
udev            1.5G     0    1.5G   0% /dev
tmpfs           301M   4.5M  296M   2% /run
/dev/mapper/vgroup1-root  19G  2.1G  17G  12% /
tmpfs           1.5G     0    1.5G   0% /dev/shm
tmpfs           5.0M     0    5.0M   0% /run/lock
tmpfs           1.5G     0    1.5G   0% /sys/fs/cgroup
/dev/sda1        922M  140M  719M  17% /boot
tmpfs           301M     0   301M   0% /run/user/0
root@ubuntu1804:~#
```

Рис. 11.14. Доступно 19 Гбайт

```
- Bitvise xterm - root@ubuntu1804: ~
Last login: Fri Jun 29 06:08:28 2018 from 188.163.24.170
root@ubuntu1804:~# echo 1 > /sys/block/sda/device/rescan
root@ubuntu1804:~# parted
GNU Parted 3.2
Using /dev/sda
Welcome to GNU Parted! Type 'help' to view a list of commands.
(parted) p
Model: VMware Virtual disk (scsi)
Disk /dev/sda: 42.9GB ←
Sector size (logical/physical): 512B/512B
Partition Table: msdos
Disk Flags:
Number  Start   End    Size   Type      File system  Flags
 1      1049kB  1000MB  999MB  primary   ext4        boot
 2      1000MB  21.5GB  20.5GB  primary   lvm          ←
(parted)
```

Рис. 11.15. Текущая таблица разделов

Укажите номер раздела:

Partition number? 2

А затем — конец раздела. Нужно указать как раз то самое значение 42.9GB — именно так, без пробелов (рис. 11.16). Введите команду `quit` для выхода из parted.

```
- Bitvise xterm - root@ubuntu1804: ~
Last login: Fri Jun 29 08:09:14 2018 from 188.163.24.170
root@ubuntu1804:~# echo 1 > /sys/block/sda/device/rescan
root@ubuntu1804:~# parted
GNU Parted 3.2
Using /dev/sda
Welcome to GNU Parted! Type 'help' to view a list of commands.
(parted) p
Model: VMware Virtual disk (scsi)
Disk /dev/sda: 42.9GB
Sector size (logical/physical): 512B/512B
Partition Table: msdos
Disk Flags:

Number  Start   End     Size    Type      File system  Flags
 1      1049kB  1000MB  999MB   primary   ext4        boot
 2      1000MB   21.5GB  20.5GB  primary            lvm

(parted) resizepart
Partition number? 2
End? [21.5GB]? 42.9GB
(parted) quit
Information: You may need to update /etc/fstab.

root@ubuntu1804:~#
```

Рис. 11.16. Изменение размера раздела

Сообщим ядру об изменениях размера:

```
pvresize /dev/sda2
Physical volume "/dev/sda2" changed
 1 physical volume(s) resized / 0 physical volume(s) not resized
```

Изменим логический том:

```
lvextend -r -l +100%FREE /dev/mapper/vgroup1-root
```

По окончании работ введем команду `df -h`, чтобы убедиться, что дисковое пространство расширилось.

Посмотрите на рис. 11.17. Здесь показан результат выполнения команд `pvresize`, `lvextend` и `df -h`. Помеченная строка вывода сообщает нам, что размер группы томов `vgroup1-root` теперь составляет 41 Гбайт. Поставленная задача выполнена.

```

- Bitvise xterm - root@ubuntu1804: ~

(parted) resizepart
Partition number? 2
End? [21.5GB]? 42.9GB
(parted) quit
Information: You may need to update /etc/fstab.

root@ubuntu1804:~# pvresize /dev/sda2
  Physical volume "/dev/sda2" changed
  1 physical volume(s) resized / 0 physical volume(s) not resized
root@ubuntu1804:~# lvextend -r -l +100%FREE /dev/mapper/vgroup1-root
  Size of logical volume vgroup1/root changed from 18.11 GiB (4637 extents) to <38.07 GiB (9745 exten-
  nts).
  Logical volume vgroup1/root successfully resized.
meta-data=/dev/mapper/vgroup1-root isize=512    agcount=12,  agsize=400640 blks
          =                     sectsz=512  attr=2, projid32bit=1
          =                     crc=1    finobt=1 spinodes=0 rmapbt=0
data      =                     bsize=4096   blocks=4748288, imaxpct=25
          =                     sunit=0    swidth=0 blks
naming    =version 2           bsize=4096   ascii-ci=0 ftype=1
log       =internal            bsize=4096   blocks=2560, version=2
          =                     sectsz=512  sunit=0 blks, lazy-count=1
realtime  =none                extsz=4096   blocks=0, rtextents=0
data blocks changed from 4748288 to 9978880
root@ubuntu1804:~# df -H
Filesystem      Size  Used Avail Use% Mounted on
udev            1.6G   0    1.6G  0% /dev
tmpfs           315M  4.8M  311M  2% /run
/dev/mapper/vgroup1-root  41G  2.3G  39G  6% /
tmpfs           1.6G   0    1.6G  0% /dev/shm
tmpfs           5.3M   0    5.3M  0% /run/lock
tmpfs           1.6G   0    1.6G  0% /sys/fs/cgroup
/dev/sda1        967M 147M  754M 17% /boot
tmpfs           315M   0   315M  0% /run/user/0
root@ubuntu1804:~#

```

Рис. 11.17. Процедура расширения тома

11.1.5. Несколько слов о GPT. Утилиты для работы с GPT

GPT (GUID Partition Table) — стандартный формат размещения таблиц разделов на физическом жестком диске. GPT является частью EFI (Extensible Firmware Interface, расширяемый микропрограммный интерфейс) — стандарта, который был предложен компанией Intel на замену BIOS. В EFI таблица GPT задействуется там, где в BIOS используется MBR (Master Boot Record, главная загрузочная запись).

В отличие от MBR, начинающейся с исполняемой двоичной программы-загрузчика, которая должна идентифицировать и загрузить активный раздел, GPT использует для осуществления этих процессов EFI. Но MBR все же присутствует в самом начале диска для обратной совместимости и для защиты, GPT же начинается с оглавления таблицы разделов.

В GPT используется современная система адресации логических блоков (LBA) вместо устаревшей системы CHS (цилиндр-головка-сектор), которая применялась в MBR. Как и MBR, таблица GPT обеспечивает дублирование — оглавление и таблица разделов записаны как в начале, так и в конце диска.

С помощью GPT можно создавать разделы размером до 9,4 Збайт¹ (т. е. $9,4 \times 10^{21}$ байт), в MBR же максимальный размер диска — 2,2 Тбайт² (т. е. $2,2 \times 10^{12}$ байт).

Для работы с разделами GPT нужно использовать утилиту gdisk, т. к. при просмотре содержимого диска программой fdisk картина будет примерно такой:

```
WARNING: GPT (GUID Partition Table) detected on '/dev/sdb'! The util fdisk doesn't
support GPT. Use GNU Parted.
```

```
Disk /dev/sdb: 1000.2 GB, 1000204886016 bytes
255 heads, 63 sectors/track, 121601 cylinders, total 1953525168 sectors
Units = sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes
Disk identifier: 0xcd29a27d
```

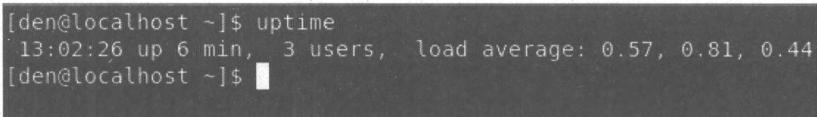
Device	Boot	Start	End	Blocks	Id	System
/dev/sdb1		1	1953525167	976762583+	ee	GPT

Для работы с GPT можно также использовать программу gpart.

11.2. Информация о системе и пользователях

11.2.1. Команда *uptime*: информация о работе системы

Команда *uptime* (рис. 11.18) выводит статистическую информацию о работе системы: сколько времени прошло с момента последней перезагрузки (собственно, это и есть время «*uptime*»), сколько пользователей в текущий момент подключено к системе и среднюю загрузку системы за последние 1, 5 и 15 минут.



```
[den@localhost ~]$ uptime
13:02:26 up 6 min, 3 users, load average: 0.57, 0.81, 0.44
[den@localhost ~]$
```

Рис. 11.18. Команда *uptime*

11.2.2. Команда *users*: информация о пользователях

Команда *users* выводит информацию о пользователях, подключенных к системе в текущий момент. На рис. 11.19 видно, что пользователь den подключился к системе двумя способами: вошел в консоли и в графическом режиме (или по FTP, ssh, telnet — способы подключения к системе могут быть разные).

¹ Зеттабайт.

² Терабайт.

```
[den@localhost ~]$ users
den den
[den@localhost ~]$ █
```

Рис. 11.19. Команда users

11.2.3. Команды *w*, *who*, *ftpwho* и *whoami*: информация о пользователях

Три родственные команды: *w*, *who* и *whoami* — выводят следующую информацию (рис. 11.20):

- команда *w* — список пользователей, подключенных к системе, виртуальный терминал, с которого работает пользователь, время входа в систему для каждого пользователя, статистику использования системы (IDLE — время простоя, JCPU — использование процессора), выполняемые каждым пользователем задачи;

```
den@den-home:~/Рабочий стол$ w
16:57:16 up 5 min, 2 users, load average: 0,47, 1,05, 0,60
USER   TTY      FROM      LOGIN@  IDLE   JCPU   PCPU WHAT
den    tty2     16:52  5:23  0.04s  0.03s /usr/libexec/gnome-session-binary --systemd --sess
bagira  tty3     -       16:55  2:12  0.02s  0.01s -bash
den@den-home:~/Рабочий стол$ who
den    tty2     2022-02-12 16:52 (tty2)
bagira  tty3     2022-02-12 16:55
den@den-home:~/Рабочий стол$ whoami
den
den@den-home:~/Рабочий стол$
```

Рис. 11.20. Команды *w*, *who* и *whoami*

- команда *who* — список пользователей, подключенных к системе, время и дату входа каждого пользователя;
- команда *whoami* — имя пользователя, который ввел команду.

Команда *ftpwho* похожа на команду *who*, но выводит только пользователей FTP-сервера, подключенных в текущий момент к серверу.

11.2.4. Мониторинг работы системы

Вам пригодятся также следующие программы:

- **htop** (рис. 11.21) — расширенная версия программы **top** (см. разд. 5.4). Она не устанавливается по умолчанию — вам придется установить пакет **htop** или скачать программу с <http://htop.sourceforge.net>;

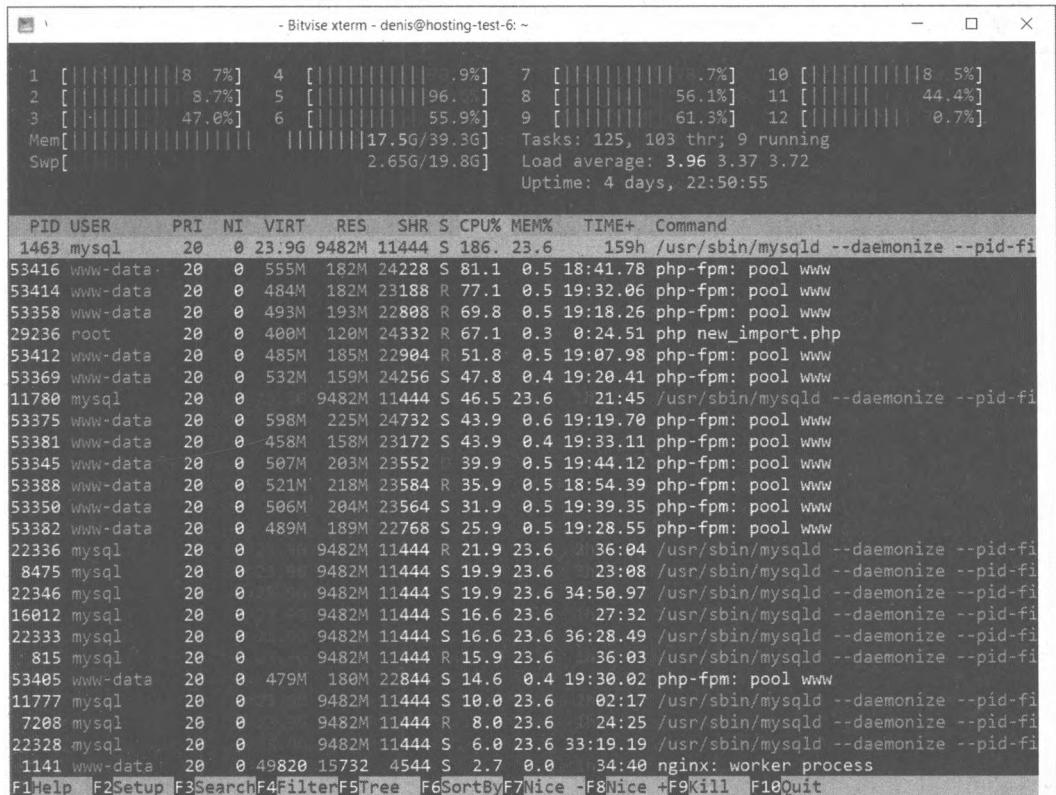


Рис. 11.21. Программа htop

- **mpstat** (рис. 11.22) — представляет расширенную статистику использования ресурсов системы в процентах. Программа также не устанавливается по умолчанию, и вам нужно будет установить пакет **sysstat**. Прервать выполнение программы можно нажатием комбинации клавиш **<Ctrl>+<C>**;
- **vmstat** (рис. 11.23) — представляет расширенную статистику использования виртуальной памяти. Прервать выполнение программы можно нажатием комбинации клавиш **<Ctrl>+<C>**;
- **iostat** (рис. 11.24) — представляет статистику использования прерываний.

```
- Bitvise xterm - root@hosting-test-6: ~
root@hosting-test-6:~# mpstat 1
Linux 4.15.0-167-generic (hosting-test-6)        02/12/2022      _x86_64_      (12 CPU)

04:02:27 PM CPU %usr %nice %sys %iowait %irq %soft %steal %guest %gnice %idle
04:02:28 PM all 29.86 0.00 4.67 0.92 0.00 0.17 0.00 0.00 0.00 64.39
04:02:29 PM all 17.79 0.00 2.09 0.75 0.00 0.17 0.00 0.00 0.00 79.20
04:02:30 PM all 19.18 0.00 2.00 1.00 0.00 0.00 0.00 0.00 0.00 77.81
04:02:31 PM all 24.27 0.00 2.50 1.08 0.00 0.17 0.00 0.00 0.00 71.98
04:02:32 PM all 27.76 0.00 3.34 0.59 0.00 0.25 0.00 0.00 0.00 68.06
04:02:33 PM all 18.85 0.00 1.58 1.08 0.00 0.08 0.00 0.00 0.00 78.40
04:02:34 PM all 15.89 0.00 1.59 1.34 0.00 0.08 0.00 0.00 0.00 81.10
04:02:35 PM all 19.52 0.00 1.25 2.25 0.00 0.08 0.00 0.00 0.00 76.90
04:02:36 PM all 15.60 0.00 1.17 0.33 0.00 0.00 0.00 0.00 0.00 82.90
04:02:37 PM all 20.58 0.00 1.58 0.83 0.00 0.08 0.00 0.00 0.00 76.92
04:02:38 PM all 34.45 0.00 3.51 0.33 0.00 0.08 0.00 0.00 0.00 61.62
04:02:39 PM all 36.68 0.00 3.76 0.08 0.00 0.25 0.00 0.00 0.00 59.23
04:02:40 PM all 31.35 0.00 3.44 0.42 0.00 0.42 0.00 0.00 0.00 64.38
04:02:41 PM all 29.97 0.00 4.84 0.17 0.00 0.08 0.00 0.00 0.00 64.94
04:02:42 PM all 32.11 0.00 4.60 0.25 0.00 0.50 0.00 0.00 0.00 62.54
04:02:43 PM all 28.99 0.00 3.68 0.42 0.00 0.17 0.00 0.00 0.00 66.75
04:02:44 PM all 30.22 0.00 6.18 0.33 0.00 0.33 0.00 0.00 0.00 62.94
04:02:45 PM all 34.39 0.00 3.84 0.08 0.00 0.50 0.00 0.00 0.00 61.19
04:02:46 PM all 27.44 0.00 1.67 0.17 0.00 0.17 0.00 0.00 0.00 70.56
04:02:47 PM all 19.23 0.00 2.42 0.42 0.00 0.17 0.00 0.00 0.00 77.76
04:02:48 PM all 17.86 0.00 2.50 0.25 0.00 0.00 0.00 0.00 0.00 79.38
04:02:49 PM all 26.88 0.00 4.01 0.08 0.00 0.50 0.00 0.00 0.00 68.53
04:02:50 PM all 43.95 0.00 4.17 0.00 0.00 0.33 0.00 0.00 0.00 51.54
04:02:51 PM all 37.66 0.00 3.77 0.00 0.00 0.84 0.00 0.00 0.00 57.74
04:02:52 PM all 38.32 0.00 3.61 0.00 0.00 1.43 0.00 0.00 0.00 56.64
04:02:53 PM all 39.77 0.00 4.09 0.08 0.00 0.75 0.00 0.00 0.00 55.30
04:02:54 PM all 21.79 0.00 1.67 0.42 0.00 0.33 0.00 0.00 0.00 75.79
04:02:55 PM all 20.38 0.00 1.09 0.00 0.00 0.25 0.00 0.00 0.00 78.28
```

Рис. 11.22. Программа mpstat

```
- Bitvise xterm - denis@zarina.ua:2206 - Bitvise xterm - root@hosting-test-6: ~
root@hosting-test-6:~# vmstat 1
procs --memory-- --swap-- --io--- -system-- --cpu--
r b swpd free buff cache si so bi bo in cs us sy id wa st
8 0 2779412 7753116 2618204 12348820 4 4 18 871 2 5 25 3 71 1 0
6 0 2779412 7792692 2618204 12352848 0 0 0 684488 3131 7815 48 7 45 0 0
7 0 2779412 7760380 2618204 12353784 0 0 0 22508 3043 7008 46 4 49 0 0
8 0 2779412 7771676 2618204 12353840 0 0 0 47248 2458 5561 55 5 40 0 0
7 0 2779412 7762696 2618204 12354900 0 0 0 11452 2915 6931 55 5 40 0 0
6 0 2779412 7721936 2618204 12355192 0 0 0 31124 3715 8106 52 6 42 0 0
5 0 2779412 7718636 2618204 12358616 0 0 0 63988 3730 7765 43 5 52 0 0
6 0 2779412 7735212 2618204 12359640 0 0 0 18524 2026 4626 47 4 49 0 0
5 0 2779412 776028 2618204 12359700 0 0 0 11616 3725 7718 43 4 53 0 0
4 0 2779412 7708944 2618204 12361668 0 0 0 4800 2953 6431 35 2 63 0 0
4 0 2779412 7717248 2618204 123611464 0 0 0 180 2014 4424 37 3 61 0 0
4 0 2779412 7741216 2618204 12361716 0 0 0 4968 2670 5709 37 3 61 0 0
5 0 2779412 7728744 2618204 12363640 0 0 0 1956 2054 4459 29 1 70 0 0
3 0 2779412 7731392 2618204 12364044 0 0 0 1128 2098 4515 33 2 65 0 0
3 0 2779412 7741152 2618204 12364224 0 0 0 1248 2242 4732 28 2 70 0 0
4 0 2779412 7715496 2618204 12362028 0 0 0 2476 2496 5481 29 2 69 0 0
11 0 2779412 7720200 2618204 12364176 0 0 0 35984 2071 4830 49 7 45 0 0
9 0 2779412 7789484 2618204 12364020 0 0 0 39516 2312 6821 74 8 18 0 0
5 0 2779412 7678828 2618204 12364640 0 0 0 62776 4488 10833 53 7 39 0 0
6 0 2779412 7724324 2618204 12365412 0 0 0 90628 5851 12418 42 7 51 0 0
8 0 2779412 7567564 2618204 12364660 0 0 0 83688 2913 7953 62 14 24 0 0
15 0 2779412 7199220 2618204 12369384 0 0 0 13272 2533 6529 60 12 28 0 0
10 0 2779412 6990096 2618204 12369564 0 0 0 1044 765 4165 86 12 2 0 0
4 0 2779412 7615772 2618204 12369448 0 0 0 704 2433 5951 56 8 36 0 0
4 0 2779412 7643612 2618204 12369448 0 0 0 3240 3162 6906 41 4 55 0 0
4 0 2779412 7673424 2618204 12369464 0 0 0 548 2924 6538 34 3 63 0 0
```

Рис. 11.23. Программа vmstat

```

MainZarina.tlp - denis@zarina.ua:2206 - Bitvise xterm - root@hosting-test-6: ~
root@hosting-test-6:~# iostat 1
Linux 4.15.0-167-generic (hosting-test-6)        02/12/2022      _x86_64_      (12 CPU)

avg-cpu: %user   %nice %system %iowait  %steal   %idle
          25.45     0.00    3.10    0.71     0.00   70.74

Device      tps   kB_read/s   kB_wrtn/s   kB_read   kB_wrtn
loop0       0.00     0.01     0.00     2651       0
loop1       0.31     0.31     0.00   133280       0
loop2       0.00     0.00     0.00       5       0
fd0        0.00     0.00     0.00      20       0
sda       157.01   209.50   10427.06  89703917 4464614056

avg-cpu: %user   %nice %system %iowait  %steal   %idle
          25.50     0.00    3.93    0.00     0.00   70.57

Device      tps   kB_read/s   kB_wrtn/s   kB_read   kB_wrtn
loop0       0.00     0.00     0.00       0       0
loop1       0.00     0.00     0.00       0       0
loop2       0.00     0.00     0.00       0       0
fd0        0.00     0.00     0.00       0       0
sda       205.00     0.00   62684.00       0   62684

avg-cpu: %user   %nice %system %iowait  %steal   %idle
          41.20     0.00    4.92    0.00     0.00   53.88

Device      tps   kB_read/s   kB_wrtn/s   kB_read   kB_wrtn
loop0       0.00     0.00     0.00       0       0
loop1       0.00     0.00     0.00       0       0
loop2       0.00     0.00     0.00       0       0
fd0        0.00     0.00     0.00       0       0
sda       180.00     0.00   51420.00       0   51420

```

Рис. 11.24. Программа iostat

11.3. Планировщик at

11.3.1. Команда at: добавление задания

Иногда нужно просто выполнить конкретные команды в определенное время (однократно), поэтому редактировать для этого таблицу crontab не совсем уместно. Такую задачу можно решить более рационально. Убедитесь, что у вас установлен и запущен демон atd. После этого введите команду:

```
at <время> [дата]
```

Затем просто вводите команды, которые вы хотите выполнить в указанное время. Для завершения ввода нажмите комбинацию клавиш **<Ctrl>+<D>**. Время указывается в АМ/PM-формате — например, если вам нужно выполнить команды в 14:00, то вы должны ввести команду: `at 2pm`.

11.3.2. Команды atq и atrm: очередь заданий и удаление задания

Просмотреть очередь заданий можно командой `atq`, а удалить какое-либо задание — командой `atrm`.

На рис. 11.25 показано добавление команды в очередь atq, просмотр очереди, удаление задачи и повторный просмотр очереди.

В целях повышения безопасности в файл /etc/at.deny можно добавить команды, которые запрещены для выполнения планировщиком atq.

```
[den@localhost ~]$ at 2pm
warning: commands will be executed using (in order) a) $SHELL b) login shell c)
/bin/sh
at> mc
at> <EOT>
job 1 at 2007-07-17 14:00
[den@localhost ~]$ atq
1      2007-07-17 14:00 a den
[den@localhost ~]$ atrm 1
[den@localhost ~]$ atq
[den@localhost ~]$ █
```

Рис. 11.25. Использование команды atq

11.4. Планировщик crond

В Linux есть специальный демон crond, позволяющий выполнять программы по расписанию. Откройте конфигурационный файл демона crond — /etc/crontab (листинг 11.1).

Листинг 11.1. Пример файла /etc/crontab

```
SHELL=/bin/bash
PATH=/sbin:/bin:/usr/sbin:/usr/bin
MAILTO=root
HOME=/

# run-parts
01 * * * * root nice -n 19 run-parts --report /etc/cron.hourly
02 4 * * * root nice -n 19 run-parts --report /etc/cron.daily
22 4 * * 0 root nice -n 19 run-parts --report /etc/cron.weekly
42 4 1 * * root nice -n 19 run-parts --report /etc/cron.monthly
```

Параметр SHELL задает имя программы-оболочки, параметр PATH — путь поиска программ, MAILTO — имя пользователя, которому будет отправлен отчет о выполнении расписания, а HOME — домашний каталог crond.

Но самое главное — не эти параметры, а сама таблица расписаний, занимающая в нашем случае последние четыре строки листинга. Согласно этой таблице каждый час будут выполняться программы из каталога /etc/cron.hourly, каждый день — из каталога /etc/cron.daily, каждую неделю — из каталога /etc/cron.weekly и раз в месяц — из каталога /etc/cron.monthly.

Предположим, вам нужно каждый день выполнять команду `update_av` в каталоге `/ftp://server.ru/bases/`. В каталоге `/etc/cron.daily` создайте файл `update_av` следующего содержания:

```
#!/bin/bash
update_av ftp://server.ru/bases/
```

Этот файл представляет собой небольшой bash-сценарий (сценарий командного интерпретатора). Теперь сделаем его исполняемым:

```
# chmod +x update_av
```

Правда, удобно?

Но иногда нужно создать более гибкое расписание. Например, мы хотим, чтобы одна программа выполнялась в 7:00, а другая в 7:20. Тут простым добавлением сценария в каталог `/etc/cron.daily` уже не отделаешься. Чтобы создать такое расписание, вам придется изучить формат записей таблицы расписаний:

минуты (0-59) часы (0-23) день (1-31) месяц (1-12) день_недели (0-6, 0 – Вс) команда

Чтобы реализовать наше расписание, следует добавить в файл `/etc/crontab` такие строки:

```
0    7      *      *      *      /usr/bin/command1 arguments
20   7      *      *      *      /usr/bin/command2 arguments
```

Первая команда будет запускаться каждый день в 7 часов утра, а вторая — тоже каждый день, но в 7:20.

Зная формат файла `crontab`, мы можем отредактировать стандартную таблицу расписаний (см. листинг 11.1). Обратите внимание: команды, выполняемые ежедневно, будут запускаться в 4 часа утра. Это, конечно, удобно, но они не будут выполнены, если вы выключаете сервер на ночь. Поэтому давайте установим другое время — например, 8 часов утра:

```
02  8  *  *  *  root nice -n 19 run-parts --report /etc/cron.daily
```

Аналогичная ситуация и с еженедельным запуском. Программы будут запущены не только в 4:22 утра, но еще и в воскресенье. Однако на выходные вы точно выключаете свой сервер (впрочем, это зависит от политики организации — в некоторых организациях и на выходные все компьютеры не выключают). Поэтому целесообразно назначить запуск на понедельник в 8 часов 22 минуты:

```
22  8  *  *  1  root nice -n 19 run-parts --report /etc/cron.weekly
```

С ежемесячным запуском вроде бы все нормально — программы будут выполнятьсь в 4:42 первого числа каждого месяца. Хотя время лучше изменить на 8:42:

```
42  8  1  *  *  root nice -n 19 run-parts --report /etc/cron.monthly
```

11.5. Команда `date`: вывод и установка даты и времени

Команда `date` служит для вывода текущей даты. Эта команда может применяться также для установки даты, если она запущена от имени администратора.

Пример использования:

```
$ date
# date 1609171722
```

Первая команда выводит дату, а вторая — устанавливает дату (при условии, что команда запущена от имени root) 16 сентября (1609) 2022 года (22) и время 17:17. Как видите, установка даты осуществляется в формате `mmddhhmmYY` (mm — месяц, dd — число, hh — часы, mm — минуты, YY — год).

Команда `date` может вывести дату в указанном вами формате. Для доступа к аппаратным часам служит команда `hwclock`. Параметр `-r` позволяет прочитать системные дату/время:

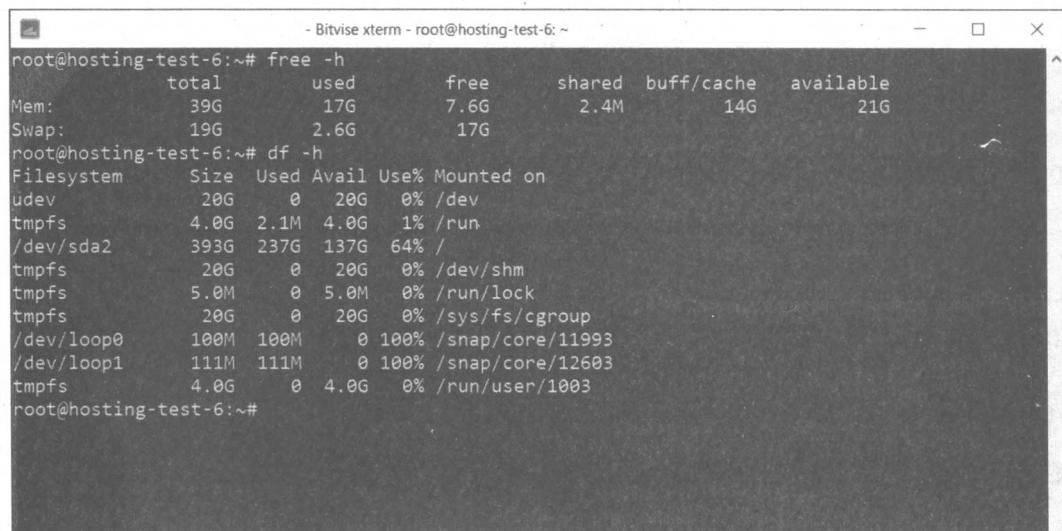
```
# hwclock -r
```

Для сохранения системной даты используется параметр `-w`:

```
# hwclock -w
```

11.6. Команды *free* и *df*: информация о системных ресурсах

Команда `free` выводит информацию об использовании оперативной и виртуальной памяти, а `df` — об использовании дискового пространства. На рис. 11.26 видно, что в системе установлено всего 39 Гбайт ОЗУ (это виртуальная машина, поэтому объем ОЗУ не равен степени двойки), из них 17 Гбайт занято и 7.6 Гбайт — свободно. В общей сложности доступен (если не считать пространства для буферов и кеша) 21 Гбайт. Параметр `-h` у обоих команд задает вывод в удобном для человека формате.

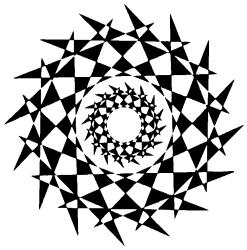


```
- Bitvise xterm - root@hosting-test-6: ~

root@hosting-test-6:~# free -h
              total        used        free      shared  buff/cache   available
Mem:           39G         17G        7.6G        2.4M       14G        21G
Swap:          19G         2.6G        17G

root@hosting-test-6:~# df -h
Filesystem      Size  Used Avail Use% Mounted on
udev            20G    0    20G   0% /dev
tmpfs           4.0G  2.1M  4.0G   1% /run
/dev/sda2        393G  237G  137G  64% /
tmpfs           20G    0    20G   0% /dev/shm
tmpfs           5.0M    0    5.0M   0% /run/lock
tmpfs           20G    0    20G   0% /sys/fs/cgroup
/dev/loop0       100M   100M    0  100% /snap/core/11993
/dev/loop1      111M  111M    0  100% /snap/core/12603
tmpfs           4.0G    0    4.0G   0% /run/user/1003
root@hosting-test-6:~#
```

Рис. 11.26. Команды `free` и `df`



ГЛАВА 12

Автоматизация рутинных задач с помощью оболочки bash

12.1. Настройка bash

bash — это самая популярная командная оболочка (командный интерпретатор) Linux. Основное предназначение bash — выполнение команд, введенных пользователем. Пользователь вводит команду, bash ищет программу, соответствующую команде, в каталогах, указанных в переменной окружения PATH. Если такая программа найдена, то bash запускает ее и передает ей введенные пользователем параметры. В противном случае выводится сообщение о невозможности выполнения команды.

Глобальные настройки bash содержатся в файле /etc/profile — он влияет на всю систему, на каждую запущенную оболочку. Обычно /etc/profile не нуждается в изменении, а при необходимости изменить параметры bash редактируют один из следующих файлов:

- `~/.bash_profile` — обрабатывается при каждом входе в систему;
- `~/.bashrc` — обрабатывается при каждом запуске дочерней оболочки;
- `~/.bash_logout` — обрабатывается при выходе из системы.

Файл `~/.bash_profile` часто не существует, а если и существует, то в нем есть всего одна строка:

```
source ~/.bashrc
```

Эта строка означает, что нужно прочитать файл `.bashrc`. Поэтому будем считать основным конфигурационным файлом bash файл `.bashrc`. Но помните, что он влияет на оболочку текущего пользователя (такой файл находится в домашнем каталоге каждого пользователя — не забываем: символ `~` означает домашний каталог). Если же вдруг понадобится задать параметры, которые повлияют на всех пользователей, то придется редактировать файл `/etc/profile`.

В файле `.bash_history` (тоже находится в домашнем каталоге) хранится история команд, введенных пользователем. Так что вы можете просмотреть свои же команды, которые накануне вводили.

Какие настройки могут быть в `.bashrc`? Как правило, в этом файле устанавливаются значения переменных окружения (см. разд. 3.2), задаются псевдонимы команд (см. разд. 3.3), определяется внешний вид приглашения командной строки (см. разд. 3.4).

12.2. Автоматизация задач с помощью сценариев bash

Представим, что нам нужно выполнить резервное копирование всех важных файлов, для чего создать архивы каталогов `/etc`, `/home` и `/usr`. Понятно, что понадобятся три команды вида:

```
tar -cvjf имя_архива.tar.bz2 каталог
```

Далее эти архивы нужно будет записать на удаленный сервер с помощью команды `scp`. Возможно, придется удалить созданные ранее резервные копии и выполнить другие операции — например, создать дамп базы данных, и это всё добавит еще команд.

Если выполнять эту процедуру раз в месяц (или хотя бы раз в неделю), то ничего страшного. Но представьте, что вам нужно делать это каждый день или даже несколько раз в день? Думаю, такая рутинная работа вам быстро надоест. А ведь можно написать *сценарий*, который сам будет создавать резервные копии и записывать их куда нужно.

Можно пойти и иным путем: написать сценарий, который будет делать резервные копии системных каталогов и записывать их на другой раздел жесткого диска. Ведь не секрет, что резервные копии делаются не только на случай сбоя системы, но и для защиты от некорректного изменения данных пользователем. Помню, я как-то удалил важную тему форума и попросил своего хостинг-провайдера сделать откат. Я был приятно удивлен, когда мне предоставили на выбор три резервные копии — осталось лишь выбрать наиболее подходящую. Не думаете же вы, что администраторы провайдера только и занимались тем, что три раза в день копировали домашние каталоги пользователей? Поэтому автоматизация — штука полезная, и любому администратору нужно знать, как автоматизировать свою рутинную работу.

12.3. Привет, мир!

По традиции напишем первый сценарий, выводящий всем известную фразу: «Привет, мир!» (`Hello, world!`). Для редактирования сценариев вы можете использовать любимый текстовый редактор, например `nano` или `ee` (листинг 12.1).

Листинг 12.1. Первый сценарий

```
#!/bin/bash
echo "Привет, мир!"
```

Первая строка нашего сценария — это указание, что он должен быть обработан программой `/bin/bash`. Обратите внимание: если между `#` и `!` окажется пробел, то эта директива не сработает, поскольку будет воспринята как обычный комментарий, поскольку комментарии начинаются, как вы уже догадались, с решетки:

```
# Комментарий
```

Вторая строка — это оператор `echo`, выводящий нашу строку. Сохраните сценарий под именем `hello` и введите команду:

```
$ chmod +x hello
```

Для запуска сценария введите команду:

```
./hello
```

На экране вы увидите строку:

```
Привет, мир!
```

Чтобы вводить для запуска сценария просто `hello` (без `./`), сценарий нужно скопировать в каталог `/usr/bin` (точнее, в любой каталог из переменной окружения `PATH`):

```
# cp ./hello /usr/bin
```

12.4. Использование переменных в собственных сценариях

В любом серьезном сценарии вы не обойдетесь без использования *переменных*. Переменные можно объявлять в любом месте сценария, но до места их первого применения. Рекомендуется объявлять переменные в самом начале сценария, чтобы потом не искать, где вы объявили ту или иную переменную.

Для объявления переменной используется следующая конструкция:

```
переменная=значение
```

Пример объявления переменной:

```
ADDRESS=www.dkws.org.ua
echo $ADDRESS
```

Обратите внимание на следующие моменты:

- при объявлении переменной знак доллара не ставится, но он обязателен при использовании переменной;
- при объявлении переменной не должно быть пробелов до и после знака `=`.

Значение для переменной указывать вручную не обязательно — его можно прочитать с клавиатуры:

```
read ADDRESS
```

или со стандартного вывода программы:

```
ADDRESS='hostname'
```

Чтение значения переменной с клавиатуры осуществляется с помощью инструкции `read`. При этом указывать символ доллара не нужно. Вторая команда устанавливает в качестве значения переменной `ADDRESS` вывод команды `hostname`.

В Linux часто используются *переменные окружения*. Это специальные переменные, содержащие служебные данные. Вот примеры некоторых часто используемых переменных окружения:

- `BASH` — полный путь до исполняемого файла командной оболочки `bash`;
- `BASH_VERSION` — версия `bash`;
- `HOME` — домашний каталог пользователя, который запустил сценарий;
- `HOSTNAME` — имя компьютера;
- `RANDOM` — случайное число в диапазоне от 0 до 32 767;
- `OSTYPE` — тип операционной системы;
- `PWD` — текущий каталог;
- `PS1` — строка приглашения;
- `UID` — ID пользователя, который запустил сценарий;
- `USER` — имя пользователя.

Для установки собственной переменной окружения используется команда `export`:

```
# присваиваем переменной значение
$ADDRESS=bhv.ru
# экспортируем переменную — делаем ее переменной окружения
# после этого переменная ADDRESS будет доступна в других сценариях
export $ADDRESS
```

12.5. Передача параметров сценарию

Очень часто сценариям нужно передавать различные параметры — например, режим работы или имя файла/каталога. Для передачи параметров используются следующие специальные переменные:

- `$0` — содержит имя сценария;
- `$n` — содержит значение параметра (`n` — номер параметра);
- `$#` — позволяет узнать количество параметров, которые были переданы.

Рассмотрим небольшой пример обработки параметров сценария. Я понимаю, что конструкцию `case-esac` мы еще не рассматривали, но общий принцип должен быть понятен (листинг 12.2).

Листинг 12.2. Пример обработки параметров сценария

```
# Сценарий должен вызываться так:
# имя_сценария параметр
```

```
# Анализируем первый параметр
case "$1" in
    start)
        # Действия при получении параметра start
        echo "Запускаем сетевой сервис"
        ;;
    stop)
        # Действия при получении параметра stop
        echo "Останавливаем сетевой сервис"
        ;;
*)
    # Действия в остальных случаях
    # Выводим подсказку о том, как нужно использовать сценарий,
    # и завершаем работу сценария
    echo "Usage: $0 {start|stop}"
    exit 1
    ;;
esac
```

Думаю, приведенных комментариев здесь достаточно, поэтому подробно рассматривать работу сценария из листинга 12.2 не станем.

12.6. Массивы и bash

Интерпретатор bash позволяет использовать *массивы*. Массивы объявляются подобно переменным.

Вот пример объявления массива:

```
ARRAY[0]=1
ARRAY[1]=2

echo ${ARRAY[0]}
```

12.7. Циклы

Как и в любом языке программирования, в bash можно использовать *циклы*. Мы рассмотрим циклы *for* и *while*, хотя в bash доступны также циклы *until* и *select*, но они применяются довольно редко.

Синтаксис цикла *for* выглядит так:

```
for переменная in список
do
    команды
done
```

В цикле при каждой итерации переменной будет присвоен очередной элемент списка, над которым будут выполнены указанные команды. Чтобы стало понятнее, рассмотрим небольшой пример:

```
for n in 1 2 3;
do
    echo $n;
done
```

Обратите внимание: список значений и список команд должны заканчиваться точкой с запятой.

Как и следовало ожидать, наш сценарий выведет на экран следующее:

```
1
2
3
```

Синтаксис цикла `while` выглядит немного иначе:

```
while условие
do
    команда
done
```

Цикл `while` выполняется до тех пор, пока истинно заданное условие. Подробно об условиях мы поговорим в следующем разделе, а сейчас напишем аналог предыдущего цикла — т. е. вывести 1, 2 и 3, но с помощью `while`, а не `for`:

```
n=1
while [ $n -lt 4 ]
do
    echo "$n "
    n=$(( $n+1 ))
done
```

12.8. Условные операторы

В `bash` доступны два *условных оператора*: `if` и `case`. Синтаксис оператора `if` следующий:

```
if условие_1 then
    команда_1
elif условие_2 then
    команда_2
...
elif условие_N then
    команда_N
else
    команда_N+1
fi
```

Оператор `if` в `bash` работает аналогично оператору `if` в других языках программирования. Если истинно первое условие, то выполняется первый список команд,

иначе — проверяется второе условие и т. д. Количество блоков `elif`, понятно, не ограничено.

Самая ответственная задача — это правильно составить условие. Условия записываются в квадратных скобках. Вот пример записи условий:

```
# Переменная N = 10
```

```
[ N==10 ]
```

```
# Переменная N не равна 10
```

```
[ N!=10 ]
```

Операции сравнения указываются не с помощью привычных знаков `>` или `<`, а с помощью следующих выражений:

- `-lt` — меньше;
- `-gt` — больше;
- `-le` — меньше или равно;
- `-ge` — больше или равно;
- `-eq` — равно (используется вместо `==`).

Применять эти выражения нужно следующим образом:

```
[ переменная выражение значение|переменная ]
```

Например:

```
# N меньше 10
```

```
[ $N -lt 10 ]
```

```
# N меньше A
```

```
[ $N -lt $A ]
```

В квадратных скобках вы также можете задать выражения для проверки существования файла и каталога:

- `-e файл` — условие истинно, если файл существует;
- `-d каталог` — условие истинно, если каталог существует;
- `-x файл` — условие истинно, если файл является исполняемым.

С оператором `case` мы уже немного знакомы, но сейчас рассмотрим его синтаксис подробнее:

```
case переменная in
    значение_1) команда_1 ;;
    ...
    значение_N) команда_N ;;
    *) команда_по_умолчанию;;
esac
```

Значение указанной переменной по очереди сравнивается с приведенными значениями (`значение_1`, ..., `значение_N`). Если есть совпадение, то будут выполнены коман-

ды, соответствующие значению. Если совпадений нет, то будут выполнены команды по умолчанию. Пример использования `case` был приведен в листинге 12.2.

12.9. Функции

В bash можно использовать функции. Синтаксис объявления функции следующий:

```
имя() { список; }
```

Вот пример объявления и использования функции:

```
list_txt()  
{  
echo "Выводим текстовые файлы"  
ls *.txt  
}
```

12.10. Примеры сценариев

12.10.1. Сценарий мониторинга журнала

Начнем с простого сценария мониторинга журнала (листинг 12.3). Системные журналы постоянно обновляются, и наш сценарий будет каждые 3 секунды выводить последние 15 строк выбранного вами журнала. Сценарий окажется полезен при настройке системы, когда нужно постоянно просматривать журналы, чтобы понять, как система реагирует на новые настройки. Для прекращения работы сценария надо нажать комбинацию клавиш `<Ctrl>+<C>`.

Листинг 12.3. Мониторинг системного журнала

```
#!/bin/bash  
# Интервал обновления в секундах  
INT=3  
  
while [ true ]  
do  
# Выводим последние 15 строк журнала  
tail -n 15 $1  
# Ждем  
sleep $INT  
# Две пустые строки  
echo; echo  
done
```

Формат вызова следующий:

```
./сценарий файл_ журнала
```

Например:

```
./script /var/log/messages
```

12.10.2. Переименование файлов

Следующий сценарий сложнее: он ищет в текущем каталоге файлы, в именах которых есть пробелы, и заменяет пробелы символами подчеркивания (листинг 12.4).

Листинг 12.4. Сценарий rename_blanks

```
#!/bin/bash
#
num=0          # Сколько файлов мы переименовали
for filename in *          # Перебираем все файлы в текущем каталоге
do
# Передаем имя файла фильтру grep
# Если имя файла содержит пробел, то код завершения
# последней операции равен 0
    echo "$filename" | grep -q " "
    if [ $? -eq 0 ]
    then
# Если код завершения равен 0, переименовываем файл
        fname=$filename
        n=`echo $fname | sed -e "s/ /_/g"`
        mv "$filename" "$n"
        let "num += 1"
    fi
done
echo "Переименовано файлов: $num"
exit 0
```

12.10.3. Преобразование систем счисления

Сейчас мы напишем простенький сценарий, преобразующий десятичное число в шестнадцатеричное с помощью программы `dc` (листинг 15.5). Самим преобразованием будет заниматься программа `dc`, а наш сценарий только подготовит данные для этой программы.

Листинг 12.5. Сценарий dec_hex

```
#!/bin/bash
B=16      # Основание системы счисления
```

```
# Проверяем, является ли параметр $1 числом
if [ -z "$1" ]
then
echo "Использование: dec_hex число"
exit 1
fi
# Передаем число ($1), систему счисления программе dc
echo ""$1" \"$B" o p" | dc
```

12.10.4. Проверка прав пользователя

Для сценариев, требующих полномочий root, сначала нужно проверить, какой пользователь запустил сценарий. UID пользователя root всегда равен 0. Проверка, является ли пользователь, запустивший сценарий, пользователем root, может выглядеть так, как показано в листинге 12.6.

Листинг 12.6. Проверка полномочий пользователя

```
#!/bin/bash

ROOT_UID=0

if [ "$UID" -eq "$ROOT_UID" ]
then
    echo "Вы – root"
else
    echo "Вы – обычный пользователь"
fi

exit 0
```

12.10.5. Генератор имени временного файла

Довольно часто в процессе обработки данных возникает необходимость в создании временных файлов. Чтобы имя временного файла всегда было уникальным, мы можем использовать программу mcookie, генерирующую случайную строку для аутентификации xauth, но в своих целях. В листинге 12.7 приведен сценарий, генерирующий имя временного файла вида «tmp_случайная_строка».

Листинг 12.7. Генератор случайного имени файла

```
#!/bin/bash

prefix="tmp_"
suffix=`mcookie`'

temp_filename=$prefix.$suffix
```

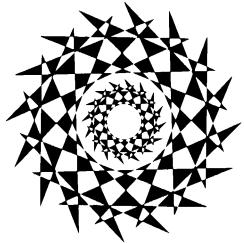
```
echo "Имя файла = \"$temp_filename\""  
exit 0
```

12.10.6. Проверка свободного дискового пространства с уведомлением по электронной почте

Следующий сценарий проверяет свободное дисковое пространство сервера и отправляет уведомление по e-mail администратору, если осталось меньше 500 Мбайт дискового пространства. Сценарий целесообразно запускать через планировщик заданий (например, через cron) с заданной периодичностью (например, каждый час). Код сценария приведен в листинге 12.8.

Листинг 12.8. Проверка свободного дискового пространства

```
#!/bin/bash  
  
# В переменную freespace будет записано свободное пространство  
# на контролируемом диске - /dev/sdal (в Мбайт)  
freespace=`df -m | grep "/dev/sdal" | awk '{print $4}'`  
  
# Если места на диске < 500 Мбайт, то отправляем письмо администратору  
if [ $freespace -lt 500 ];  
then  
echo "Warning!!! Free space on server < 500 MB" | mail -s "Freespace is out!"  
admin@server.ru  
fi
```



ГЛАВА 13

Полезные примеры

13.1. Поиск дубликатов файлов

Представим, что у нас есть каталог с различными изображениями. Среди этих изображений имеются дубликаты, появившиеся по тем или иным причинам: имя файла от подлинника отличается, а вот сам файл является копией какого-то другого файла.

Как искать дубликаты? С помощью команды `md5sum` мы можем вычислить MD5-хеш файлов. Для одного и того же файла MD5-сумма будет одинаковой, следовательно, файлы с одинаковым MD5-хешем считаются дубликатами. Один из таких файлов можно смело удалять. Пример вычисления MD5-хеша:

```
$ md5sum You.jpg  
b4c3f6844679069f8d8567272812b237 You.jpg
```

Вычислить контрольную сумму нескольких файлов можно так:

```
$ md5sum You.jpg Fine.jpg  
b4c3f6844679069f8d8567272812b237 You.jpg  
2c9da76b2bb3e3782716f8dfa01cd025 Fine.jpg
```

Если у вас 3–4 файла, то вы без проблем увидите одинаковые контрольные суммы. Но когда файлов несколько сотен, все сложнее. К счастью, мы можем поручить все это командам Linux. С помощью команды `cut` можно вырезать первые 32 символа в каждой строке — это и есть контрольная сумма. А потом все, что получится, отправить на стандартный ввод команды `sort`, которая отсортирует контрольные суммы. При этом повторяющиеся контрольные суммы будут идти последовательно, что и нужно для использования команды `uniq`:

```
$ md5sum *.jpg | cut -c1-32 | sort | uniq -c
```

Вывод будет таким:

```
1 b932d27060b7179284252abb55473374  
1 c0459a796c5b8ee74254472c235a7460  
1 d1b74ea81a39cd9f66804a5c7a3ef0b6  
2 d2f7aa38442bfcb8ce064545d592d6ed  
...
```

Если слева от контрольной суммы находится число больше 1, значит, есть дубликаты. Останется только выяснить, какому файлу принадлежит та или иная контрольная сумма. Сделать это можно так:

```
$ md5sum *.jpg | grep <контрольная сумма>
```

Например:

```
$ md5sum *.jpg | grep d2f7aa38442bfc8ce064545d592d6ed
```

Очистить вывод `md5sum` (чтобы в нем остались одни только имена файлов) можно так:

```
$ md5sum *.jpg | grep d2f7aa38442bfc8ce064545d592d6ed | cut -c35-
```

На выходе вы получите список имен файлов-дубликатов.

13.2. Сценарий scanner

Для собственного пользования я написал простенький bash-сценарий (листинг 13.1), который ищет файлы, содержащие подозрительные PHP-инструкции, а также файлы `*.html` и `*.php`, изменившиеся за последние 20 дней (рис. 13.1). Не обязательно, что все найденные файлы заражены вирусами, но такие файлы нужно держать на контроле, особенно те, которые вы не изменяли.

Листинг 13.1. Сценарий scanner

```
#!/bin/bash

fld="/var/www/html"

echo "Следующие файлы содержат подозрительный код (инструкции eval, base64_decode,
preg_replace):"

grep -RE 'preg_replace\\(|eval\\(|base64_decode\\(' --include='*.php' $fld | cut -d: -f 1
| sort -u

echo "Файлы, модифицированные за последние 20 дней:"

find $fld -name '*.php' -type f -mtime -20 ! -mtime -1 -printf '%TY-%Tm-%Td %TT %p\n'
find $fld -name '*.html' -type f -mtime -20 ! -mtime -1 -printf '%TY-%Tm-%Td %TT %p\n'
```

/srv/www/	/htdocs/test/analyze/salesplug.php
/srv/www/	/htdocs/test/_Signedint.php
Файлы, модифицированные за последние 20 дней:	
2017-06-21 17:36:36.2327712000	/srv/www/
2017-06-21 17:44:08.5687712000	/srv/www/
2017-06-15 17:22:07.5527712000	/srv/www/
2017-06-21 15:25:53.1727712000	/srv/www/
2017-06-21 14:23:31.4687712000	/srv/www/
	/htdocs/skin/Signedint.php
	/htdocs/salesplug/salesplug.php
	/htdocs/mail.php
	/htdocs/upload2.php
	/htdocs/upload/upload.php

Рис. 13.1. Результат работы сценария scanner

13.3. Изменение прав доступа к файлам и каталогам

Файлы и каталоги требуют различных прав доступа. Например, полный доступ к файлу выглядит так: 666, а к каталогу — 777. Используя параметр `-type` команды `find`, мы можем установить права доступа так, как нам нужно. Пример:

```
find /var/www/html -type f -exec chmod 666 {} \;
find /var/www/html -type d -exec chmod 777 {} \;
```

Конструкция `\;` используется здесь вместо `";"` — она несколько удобнее и короче на один символ.

13.4. Аварийный перезапуск сервисов

В этом разделе мы напишем простенькое средство мониторинга работоспособности сайта. Если сайт «упал», это средство будет перезапускать сервисы Apache и MySQL (или только Apache — все зависит от специфики сайта и причины сбоя).

Решение, конечно же, временное. Нужно найти и устранить причину «падения», но зато такой «костыль» позволяет быстро восстановить работоспособность сайта — руководство даже не заметит сбоя. Что же касается обычных пользователей, то даже если кто-то и получит ошибку 550, а через минуту сайт уже будет работать, вряд ли он будет особо жаловаться.

Проверять работоспособность сайта можно двумя способами: или пытаться получить реальный файл (можно для этого создать пустой файл в «корне» — важно получить ответ 200 от сервера), или же мониторить наличие сервисов apache/mysql в памяти. Второй способ не защищает от ситуации, когда сервисы «подвисли» — в памяти они есть, но на самом деле сайт «лежит». А первый способ не позволяет убедиться в работоспособности MySQL: файл веб-сервер может и «отдать», а база данных может не работать.

13.4.1. Проверка работоспособности веб-сервера

Проще всего проверить работоспособность веб-сервера путем обращения к его главной странице: если получен ответ 200, значит, с сервером все хорошо. Bash-скрипты будут выглядеть так (листинг 13.2).

Листинг 13.2. Проверяем работоспособность веб-сервера

```
#!/bin/bash

if curl -s --head --request GET https://site.name | grep "200 OK" > /dev/null; then
    echo "Site is UP"
else
    echo "site is DOWN, restarting Apache"
    /usr/sbin/service apache2 restart
fi
```

Вызов сценария нужно поместить в расписание cron. Периодичность зависит от частоты «падения» сервера: если сервер сбоят иногда, можно раз в 5–10 минут, если сайт «ложится» часто — раз в минуту. Да, это создаст дополнительную нагрузку на сервер, но зато обеспечит его перезапуск в течение одной минуты.

13.4.2. Проверка работоспособности MySQL

Напишем небольшой PHP-сценарий, подключающийся к БД и возвращающий 1, если соединение удалось, и 0, если соединение не работает. Код этого сценария очень прост (листинг 13.3).

Листинг 13.3. Проверка подключения к MySQL

```
<?php
include "config.php"; // здесь параметры доступа к БД

$mysqli = new mysqli($DBHOST, $DBUSER, $DBPASSWD, $DBNAME);

if ($mysqli->connect_errno) {
echo "0";
}
else "1"; // connect ok
?>
```

Затем создается bash-сценарий, подобный этому (листинг 13.4).

Листинг 13.4. Перезапуск MySQL

```
#!/bin/bash
RESULT=$( /usr/bin/php test-mysql.php )
if [ $RESULT -eq 0 ]; then
echo "Restarting MySQL"
/etc/init.d/mysqld restart
fi
```

Как и в случае с первым нашим bash-сценарием, вызов этого сценария нужно поместить в расписание планировщика. Можно объединить оба bash-сценария в один и вызывать их сразу.

13.4.3. Если «падают» процессы...

Если процессы не виснут, а «падают», т. е. после сбоя в таблице процессов вообще нет процессов Apache/MySQL, тогда поможет следующий сценарий (листинг 13.5).

Листинг 13.5. Если «падают» процессы

```
#!/bin/bash
RESTART="/etc/init.d/apache2 restart"
```

```

PGREP="/usr/bin/pgrep"
HTTPD="apache2"
$PGREP ${HTTPD}
if [ $? -ne 0 ]; then
$RESTART
date >> /var/log/srvmon.log
echo "Apache restarted" >> /var/log/srvmon.log
fi

RESTARTM="/etc/init.d/mysql restart"
MYSQLD="mysqld"
$PGREP ${MYSQLD}
if [ $? -ne 0 ]; then
$RESTART
$RESTARTM
date >> /var/log/srvmon.log
echo "Services restarted" > /var/log/srvmon.log
fi

```

Сценарий не только перезапускает «упавшие» сервисы, но и ведет небольшой лог — записывает время, когда сервисы были перезапущены.

13.5. Поиск битых ссылок с помощью AWK

AWK (GNU AWK) — небольшой язык программирования, ориентированный на обработку текстовых файлов. Его удобно использовать для обработки дампов баз данных, системных файлов. Он подобен языку Perl, но проще в использовании. По сути, AWK является прародителем Perl. Как и Perl, AWK построен на регулярных выражениях и средствах для обработки шаблонов.

Назван он по первым буквам фамилий его создателей: Alfred V. Aho, Peter J. Weinberger и Brian W. Kernighan.

Основное назначение языка, как уже было сказано, — обработка текстовых файлов, поэтому AWK пригодится администраторам для анализа системных журналов и создания различных систем статистики.

Awk — очень гибкий язык, с его помощью можно решить практически все задачи обработки текста. Представим, что нам нужно найти все битые ссылки на нашем сайте. Можно, конечно, перелопатить все страницы и проанализировать каждую ссылку, но гораздо проще проанализировать журнал Apache, чтобы вывести все ошибки 404: мы найдем отсутствующие на нашем сервере ресурсы и сможем исправить ссылки (возможно, ресурсы не отсутствуют, а мы просто допустили ошибку в имени файла). Вывод всех ошибок 404 достигается так:

```
awk '$9 == 404 {print $7}' /var/log/apache2/access_log | uniq -c | sort -rn | more
```

Довольно неплохо, особенно если учесть, что с этой задачей мы справились с помощью одной команды.

13.6. Считаем количество файлов в папке и подпапках

С помощью команд `find` и `wc` можно легко подсчитать количество файлов в каталоге и подкаталогах. Представим, что вы хотите узнать, сколько файлов находится в текущем каталоге и во всех его подкаталогах. Для этого достаточно ввести команду:

```
find . -type f | wc -l
```

Команда `find` найдет все файлы (`-type f`) в текущем каталоге и во всех его подкаталогах и выведет по одному файлу в строке. Все, что нам остается, — это подсчитать количество строк командой `wc -l`. Аналогично можно найти и количество подкаталогов в каталоге:

```
find . -type d | wc -l
```

13.7. Резервное копирование базы данных

Представим, что у нас есть два сервера: основной и резервный. Нам нужно реализовать отправку базы данных с основного сервера на резервный — на тот случай, если что-то случится с основным. Тогда мы сможем использовать резервный, и он будет обладать относительно актуальной версией базы данных.

Для реализации поставленной задачи мы будем действовать двумя сценариями. Рассмотрим первый из них — `send_dump.sh` (листинг 13.6). Этот сценарий посредством команды `mysqldump` создает дамп базы данных и помещает его в файл `db.sql`. Затем этот файл сжимается в архив `db.zip`, который отправляется на резервный сервер посредством команды `scp`. Команда `sshpass` используется для передачи пароля команде `scp`. Такое решение не очень безопасное — лучше использовать авторизацию по ключу, но для упрощения процесса настройки сервера SSH можно пойти на такой шаг. Разумеется, значения, выделенные в листинге полужирным шрифтом, вам нужно будет указать собственные:

- `bitrix0` — имя пользователя БД;
- `PASSWORD` — пароль пользователя БД;
- `sitemanager` — база данных;
- `SSHPASSWORD` — пароль для подключения к SSH;
- `devusr2` — имя пользователя SSH;
- `IP` — IP-адрес резервного сервера;
- `/home/devusr2` — каталог, в который будет загружен архив.

Листинг 13.6. Сценарий `send_dump.sh`

```
#!/bin/bash
cd ~
```

```
mysqldump -u bitrix0 -pPASSWORD sitemanager > db.sql  
zip db.zip db.sql  
sshpass -p "SSHPASSWORD" scp db.zip devusr2@IP:/home/devusr2
```

Для каждого из параметров можно создать свою собственную переменную — так будет правильнее. При желании вы можете сделать это самостоятельно. После создания сценария `send_dump.sh` нужно добавить его вызов с помощью расписания cron для регулярного вызова передачи БД на резервный сервер.

Задача второго сценария — `db.sh` (листинг 13.7) — распаковать архив и передать SQL-файл клиенту `mysql` для разворачивания БД на резервном сервере. В этом сценарии вам также нужно изменить на свои параметры доступа к БД.

Листинг 13.7. Сценарий db.sh

```
#!/bin/bash  
unzip db.zip  
mysql -u USER -pPASSWORD DBNAME < db.sql
```

Предметный указатель

3

3DES 125

I

IDEA 125

A

ASP Linux 112

B

bash 153

BlowFish 125

C

CentOS 112

crontab 149

D

DOS 9, 33

E

EFI 144

ext4 60

F

Fedor a 10

G

GPT 144

J

JFS 135

L

LiveCD 63

M

Mandriva 54

MBR 144

P

PID 72

R

RSA 125

S

SGID 47

Slackware 10

SUID 47

T

Total Commander 133

У

Ubuntu 56
 udev 50
 UFS 135
 UNIX 9, 95
 UUID 50, 56

В

VMware 110

Х

XFS 135

А

Автодополнение 15
 ◇ командной строки 42
 Алгоритмы шифрования 125
 Аутентификация по ключу 126

И

Идентификатор процесса 72
 Исполнимый файл 70
 История команд 29

Б

Безусловные списки 79
 Браузер
 ◇ elinks 118
 ◇ links 118
 ◇ lynx 118
 Буфер 66

К

Канал 16
 Каталог
 ◇ /etc/cron.daily 151
 ◇ /etc/cron.hourly 150
 ◇ /etc/cron.weekly 150
 ◇ домашний 39
 ◇ признак каталога 45
 ◇ родительский 39
 ◇ текущий 39
 Код возврата 79
 Команда
 ◇ at 149
 ◇ atq 149
 ◇ atrm 149
 ◇ cat 36
 ◇ cd 39
 ◇ chattr 48
 ◇ chmod +x 155
 ◇ chmod 46
 ◇ chown 47
 ◇ chroot 63
 ◇ clear 16
 ◇ cmp 90
 ◇ column 91
 ◇ comm 90
 ◇ cp 37
 ◇ curl 104

Г

Главная загрузочная запись (MBR) 66
 Горячие клавиши 102

Д

Дерево процессов 71
 Диск
 ◇ USB 53
 ◇ гибкий 50, 52
 ◇ жесткий 50
 ◇ оптический 52
 Длинные имена дисков 56
 Дочерний процесс 71

Ж

Журнал 60

- ◊ cut 19
- ◊ date 151
- ◊ dd 65
- ◊ dd 66
- ◊ df 152
- ◊ diff 88
- ◊ diff3 89
- ◊ dig 117
- ◊ dmesg 19
- ◊ du 68
- ◊ egrep 91
- ◊ expand 92
- ◊ fdisk 131
- ◊ fdisk 51
- ◊ find 166
- ◊ find 169
- ◊ find 64
- ◊ find 86
- ◊ fmt 93
- ◊ fold 93
- ◊ free 152
- ◊ fsck 55, 62
- ◊ ftp 118
- ◊ ftpwho 146
- ◊ fuser 77
- ◊ gdisk 145
- ◊ gpart 63
- ◊ grep 19, 93
- ◊ halt 14
- ◊ hdparm 63
- ◊ head 18, 94
- ◊ host 116
- ◊ htop 147
- ◊ ifconfig 107, 121
- ◊ ifdown 108
- ◊ ifup 108
- ◊ insmod 109
- ◊ iostat 147
- ◊ kill 72
- ◊ killall 73
- ◊ less 37, 94
- ◊ ln 44
- ◊ locate 37, 65
- ◊ logout 14, 16
- ◊ look 94
- ◊ ls 39
- ◊ lsattr 49
- ◊ md5sum 164
- ◊ mkdir 39
- ◊ mkfs 62
- ◊ more 94
- ◊ mount 50, 53
- ◊ mpstat 147
- ◊ mv 37
- ◊ netstat 109, 110
- ◊ nice 76
- ◊ nmap 127
- ◊ parted 134
- ◊ ping 123
- ◊ poweroff 14
- ◊ ps 73
- ◊ reboot 14
- ◊ renice 76
- ◊ rm 37, 39
- ◊ rmdir 39
- ◊ route 110, 113
- ◊ sed 99
- ◊ seq 105
- ◊ seq 85
- ◊ shutdown 14
- ◊ sort 20, 88
- ◊ split 94
- ◊ startx 10
- ◊ tac 37
- ◊ tail 94, 122
- ◊ tcpdump 125
- ◊ top 74
- ◊ touch 36
- ◊ tracepath 124
- ◊ traceroute 124
- ◊ umount 50
- ◊ unexpand 95
- ◊ uniq 21
- ◊ uptime 145
- ◊ users 145
- ◊ vmstat 147
- ◊ w 146
- ◊ wc 169
- ◊ wc 17, 100
- ◊ wget 104, 120
- ◊ which 37, 65
- ◊ who 146
- ◊ whoami 146
- ◊ whois 116
- ◊ xargs 105
- ◊ xclip 106
- Консоль 9, 10
- Консольные браузеры 104

Конфигуратор pppoeconf 123
Корневая файловая система 34

M

Маска имени файла 38
Маски 23
Массивы 157
Механизм
◊ псевдонимов 30
◊ фонового выполнения 84
Модель OSI 113

O

Оболочка bash 23, 70
Оператор
◊ case 159
◊ if 158

P

Пакет linux-ntfs 135
Переключение рабочих столов 103
Переменная окружения CDPATH 43
Переменные 24, 155
◊ окружения 156
◊ специальные 156
Перенаправление
◊ ввода/вывода 26
◊ команды 82
Планировщик
◊ atd 149
◊ crond 150
◊ cron 80
Подключение по SSH 83
Подстановка
◊ команды 81
◊ процесса 81
Постоянные имена дисков 57
Права доступа 45
Приглашение командной строки 25
Программа
◊ fdisk 131
◊ gparted 135
◊ parted 131, 141
Протокол
◊ SSH 125
◊ Telnet 125
Псевдонимы 25
◊ команд 15, 42

P

Расширение имени файла 33
Регулярные выражения 24
Редактор
◊ ee 98
◊ joe 98
◊ mcedit 98
◊ nano 29, 98
◊ pico 98
◊ vi 29, 95
Родительский процесс 71

C

Сетевые интерфейсы 108
Сеть, отказ работы 121
Сигналы 74
Системный сигнал 74
Специальные права доступа 47
Список 78
Ссылка
◊ жесткая 44
◊ символьическая 44
Стандартный ввод 16
Стандартный вывод 16
Стили редактирования командной строки 31
Сценарии командной оболочки 71

T

Таблица маршрутизации 112
Текстовые FTP-клиенты 119
Терминал 12
Точка монтирования 35, 50, 56, 140

У

Условные списки 78
Утилита DMitry 118

Ф

Файл
◊ .bash_history 153
◊ .bash_profile 15
◊ /etc/crontab 150
◊ /etc/fstab 54
◊ /etc/network/interfaces 113

- ◊ /etc/profile 153
- ◊ /etc/resolv.conf 124
- ◊ /etc/route.conf 112
- ◊ ~/.bash_logout 153
- ◊ ~/.bash_profile 153
- ◊ ~/.bashrc 29, 153
- ◊ fstab 60
- ◊ права доступа 45
- ◊ устройства 50
- Файловая система
 - ◊ ext4 60
 - ◊ журналируемая 60
- Файловый дескриптор 82
- Фигурные скобки 86
- Фоновый режим 83, 104

Ц

- Цикл
- ◊ for 157
 - ◊ while 158

Ш

- Шаблоны 24

Э

- Экстенты 61

Я

- Язык AWK 168



www.bhv.ru

Колисниченко Д.

Linux. От новичка к профессионалу. 8-е издание.

Отдел оптовых поставок:

e-mail: opt@bkhv.ru

Гарантия эффективной работы в Linux



- Fedora 33, zRAM
- Варианты загрузки Linux и управление загрузкой
- Работа с файловой системой и устройствами в Linux
- Файловая система Btrfs, UUID накопителей, загрузчик GRUB2
- Настройка сети, Интернета и популярных серверов Apache, ProFTPD, Samba, BIND и др.
- Настройка SSL-сертификата, ускорение веб-сервера с помощью Memcached и Google PageSpeed
- Настройка VPN-соединения, выбор VPN-провайдера, настройка VPN-сервера
- Брандмауэр ufw
- Выбор VPS/VDS-сервера и его настройка
- Программные системы хранения данных с резервированием

Книга предназначена для широкого круга пользователей Linux и поможет им самостоятельно настроить и оптимизировать эту операционную систему. Даны ответы на все вопросы, возникающие при работе с Linux: от установки и настройки этой ОС до настройки сервера на базе Linux. Материал книги максимально охватывает все сферы применения Linux — от запуска Windows-игр под управлением Linux до настройки собственного веб-сервера. Материал ориентирован на последние версии дистрибутивов Fedora, openSUSE, Slackware, Ubuntu.

В восьмом издании рассмотрены Fedora 33, модуль zRAM, файловая система Btrfs, настройка Apache для работы на нескольких портах, организация поддоменов *.example.com, выбор и настройка VDS, брандмауэр ufw, «лайфхаки» для начинающих администраторов.

Колисниченко Денис Николаевич, инженер-программист, системный администратор и IT консультант. Имеет богатый опыт эксплуатации и создания локальных сетей — от домашней до уровня предприятия на базе операционной системы Linux. Автор более 70 книг компьютерной тематики, в том числе «Самоучитель системного администратора», «Программирование для Android», «Секреты безопасности и анонимности в Интернете» и др.

Командная строка Linux

Колисниченко Дмитрий,
инженер-программист и системный
администратор. Имеет богатый опыт
работы в операционной системе Linux.

Настоящий линуксоид должен уметь работать в консоли, что требует определенной квалификации пользователя. Рассмотрены задачи, которые выполняются из командной строки Linux. Объясняется, как попасть в командную строку, работать в консоли, настраивать систему с помощью программ, обладающих только текстовым интерфейсом. Описаны особенности файловой системы Linux, наиболее полезные команды для работы с текстом, сетью и Интернетом, а также команды системного администратора. Особое внимание уделено написанию сценариев автоматизации рутинных задач на языках командных оболочек bash и tcsh. Рассмотрены способы перенаправления ввода-вывода, маски и псевдонимы, различные варианты запуска программ, эффективные приемы использования клавиатуры, примеры сложных команд и другие вопросы.

**Эффективная
работа
в командной
строке Linux**

СИСАДМИН
СИСТЕМНЫЙ
АДМИНИСТРАТОР

bhv®

КАТЕГОРИЯ: операционные системы

ISBN 978-5-9775-1750-8



9 785977 517508

191036, Санкт-Петербург,
Гончарная ул., 20
Тел.: (812) 717-10-50,
339-54-17, 339-54-28
E-mail: mail@bhv.ru
Internet: www.bhv.ru