Пол Коббаут

Фундаментальные основы Linux

Датапубликации:16октября2014г.
Перевод:А.Панин
Дата перевода: 25 декабря 2014 г.

Paul Cobbaut

Linux Fundamentals

Аннотация

Данная книга предназначена для изучения Linux под руководством инструктора. При использовании ее для самостоятельного изучения Linux следует расположиться в непосредственной близости от работающего под управлением Linux компьютера для того, чтобы иметь возможность незамедлительного исполнения описанных действий путем выполнения каждой из приведенных команд.

Книга может быть рекомендована начинающим системным администраторам Linux (а также может оказаться интересной и полезной для домашних пользователей, которые хотят узнать немного больше о своих системах Linux). Однако, данная книга не содержит рекомендаций по работе с такими приложениями из состава окружений рабочего стола, как текстовые редакторы, браузеры, почтовые клиенты, приложения для работы с мультимедийными данными и офисные приложения.

Дополнительная информация и бесплатный оригинал книги в формате PDF доступны на вебсайте http://linux-training.be.

В случае необходимости вы можете связаться с автором книги:

- Paul Cobbaut: paul.cobbaut@gmail.com, http://www.linkedin.com/in/cobbaut

Также свой вклад в написание данной книги и разработку проекта Linux Training внесли:

- Serge van Ginderachter: serge@ginsys.eu, разработка сценариев для сборки и настройка инфраструктуры
- Ywein Van den Brande: ywein@crealaw.eu, написание главы с информацией о юридических вопросах и лицензиях
- Hendrik De Vloed: hendrik.devloed@ugent.be, разработка сценария buildheader.pl

Кроме того, нам хотелось бы поблагодарить наших рецензентов:

- Wouter Verhelst: wo@uter.be, http://grep.be
- GeertGoossens:mail.goossens.geert@gmail.com, http://www.linkedin.com/in/geertgoossens
- Elie De Brauwer: elie@de-brauwer.be, http://www.de-brauwer.be
- ChristopheVandeplas:christophe@vandeplas.com, http://christophe.vandeplas.com
- Bert Desmet: bert@devnox.be, http://blog.bdesmet.be
- Rich Yonts: richyonts@gmail.com

Авторские права 2007-2014 Netsec BVBA, Paul Cobbaut

Разрешается копировать, распространять и/или модифицировать данный документ в соответствии с условиями версии 1.3 или более поздней версии Лицензии свободной документации проекта GNU, опубликованной Фондом свободного программного обеспечения; при этом не должны переводится отмеченные соответствующим образом разделы, а также тексты на передней и задней обложке. Копия оригинала лицензии размещена в приложении C с названием 'Лицензия'.

Оглавление

Часть I. Вводная информация о Linux....15
Глава 1. История Linux....15
1.1. История Linux....15
Глава 2. Дистрибутивы....17
2.1. RedHat....17
2.2. Ubuntu....17
2.3. Debian....18
2.4. Другие дистрибутивы....18
2.5. Какой дистрибутив следует выбрать?....18
Глава 3. Лицензирование....18
3.1. О лицензировании программного обеспечения....19
3.2. Программное обеспечение, распространяемое как общественное достояние, а также бесплатное программное обеспечение....20
3.3. Свободное программное обеспечение и программное обеспечение с открытым исходным кодом....21
3.4. Универсальная общественная лицензия проекта GNU....22
3.5. Использование программного обеспечения, распространяемого в соответствии с условиями версии 3 лицензии GPL....23
3.6. Лицензия BSD....24
3.7. Другие лицензии....24
3.8. Комбинирование лицензий программного обеспечения....24
Глава 4. Установка Linux в домашних условиях....25
4.1. Загрузка образа установочного диска ОС Linux....26
4.2. Загрузка пакета Virtualbox....26
4.3. Создание виртуальной машины....27
4.4. Присоединение установочного диска ОС....32
4.5. Установка Linux....35
Часть II: Первые шаги в изучении интерфейса командной строки....36
Глава 5. Страницы руководств man....36
5.1. Команда man $команда....36
5.2. Команда man $имя_файла_конфигурации....36
5.3. Команда man $демон....36
5.4. Команда man -k (apropos)....37
5.5. Команда whatis....37
5.6. Команда whereis....37
5.7. Номера справочных разделов....37
5.8. Команда man $раздел $файл....38
5.9. Команда man man....38
5.10. Утилита mandb....38
Глава 6. Работа с директориями....39
6.1. Команда pwd....39
6.2. Команда cd....40
6.2.1. Команда cd ~....40
6.2.2. Команда cd 40
6.2.3. Команда cd -....41
6.3. Абсолютные и относительные пути....41
6.4. Завершение путей....42
6.5. Утилита ls....43
6.5.1. Команда ls -a....43

6.5.2. Команда ls -l....43
6.5.3. Команда ls -lh....44
6.6. Утилита mkdir....45
6.6.1. Команда mkdir -p....46
6.7. Утилита rmdir....46
6.7.1. Команда rmdir -p....47
6.8. Практическое задание: работа с директориями....47
6.9. Корректная процедура выполнения практического задания: работа с директориями....48
Глава 7. Работа с файлами....51
7.1. Все имена файлов регистрозависимы....51
7.2. Все является файлом....51
7.3. Утилита file....51
7.4. Утилита touch....52
7.4.1. Создание пустого файла....52
7.4.2. Команда touch -t....53
7.5. rm....53
7.5.1. Удаление файлов навсегда....53
7.5.2. Команда rm -i....54
7.5.3. Команда rm -rf....54
7.6. Утилита cp....55
7.6.1. Копирование отдельных файлов....55
7.6.2. Копирование файлов в другую директорию....55
7.6.3. Команда cp -r....55
7.6.4. Копирование множества файлов в директорию....56
7.6.5. Команда cp -i....56
7.7. Утилита mv....56
7.7.1. Переименование файлов с помощью утилиты mv....56
7.7.2. Переименование директорий с помощью утилиты mv....56
7.7.3. Команда mv -i....57
7.8. Утилита rename....57
7.8.1. Об утилите rename....57
7.8.2. Утилита rename в дистрибутиве Debian/Ubuntu....58
7.8.3. Утилита rename в дистрибутиве CentOS/RHEL/Fedora....58
7.9. Практическое задание: работа с файлами....59
7.10. Корректная процедура выполнения практического задания: работа с файлами....60
Глава 8. Работа с содержимым файлов....62
8.1. Утилита head....62
8.2. Утилита tail....63
8.3. Утилита cat....63
8.3.1 Объединение файлов....64
8.3.2. Создание файлов....64
8.3.3. Специальный маркер окончания файла....65
8.3.4. Копирование файлов....65
8.4. Утилита tac....66
8.5. Утилиты more и less....66
8.6. Утилита strings....66
8.7. Практическое задание: работа с содержимым файлов....67
8.8. Корректная процедура выполнения практического задания: работа с содержимым файлов....68
Глава 9. Дерево директорий Linux....70
9.1. Стандарт иерархии файловой системы....70

9.2. Страница руководства man hier....70
9.3. Корневая директория /....70
9.4. Директории для хранения бинарных файлов....71
9.4.1. Директория /bin....71
9.4.2. Другие директории /bin....72
9.4.3. Директория /sbin....72
9.4.4. Директория /lib....73
9.4.5. Поддиректория /lib/modules....73
9.4.6. Директории /lib32 и /lib64....73
9.4.7. Директория /opt....74
9.5. Директории для хранения файлов конфигурации....74
9.5.1. Директория /boot....74
9.5.2. Директория /etc....74
9.5.3. Поддиректория /etc/init.d/....75
9.5.4. Поддиректория /etc/X11/....75
9.5.5. Поддиректория /etc/skel/....76
9.5.6. Поддиректория /etc/sysconfig/....76
9.6. Директории для хранения данных....77
9.6.1. Директория /home....77
9.6.2. Директория /root....78
9.6.3. Директория /srv....78
9.6.4. Директория /media....78
9.6.5. Директория /mnt....79
9.6.6. Директория /tmp....79
9.7. Директории в оперативной памяти....79
9.7.1. Директория /dev....79
9.7.2. Стандартные физические устройства....79
9.7.3. Файлы устройств /dev/tty и /dev/pts....80
9.7.4. Файл устройства /dev/null....80
9.7.5. Директория /proc и взаимодействие с ядром ОС....81
9.7.6. Файл /proc/interrupts....84
9.7.7. Файл /proc/kcore....85
9.7.8. Директория /sys для работы с системой горячего подключения устройств ядра Linux 2.6....86
9.8. Директория системных ресурсов Unix /usr....86
9.8.1. Директория /usr/bin....86
9.8.2. Директория /usr/include....87
9.8.3. Директория /usr/lib....87
9.8.4. Директория /usr/local....87
9.8.5. Директория /usr/share....88
9.8.6. Директория /usr/src....88
9.9. Директория для изменяемых данных /var....89
9.9.1. Директория /var/log....89
9.9.2. Файл /var/log/messages....90
9.9.3. Директория /var/cache....90
9.9.4. Директория /var/spool....91
9.9.5. Директория /var/lib....91
9.9.6. Другие директории /var/....91
9.10. Практическое задание: дерево директорий Linux....91
9.11. Корректная процедура выполнения практического задания: дерево директорий Linux....93
Часть III. Раскрытие команд командной оболочкой....97

Глава 10. Команды и аргументы....97
10.1. Аргументы....97
10.2. Удаление пробелов....98
10.3. Одинарные кавычки....98
10.4. Двойные кавычки....99
10.5. Команда echo и кавычки....99
10.6. Команды....100
10.6.1. Внешние или встроенные команды?....100
10.6.2. Команда type....100
10.6.3. Исполнение внешних команд....100
10.6.4. Команда which....101
10.7. Псевдонимы команд....101
10.7.1. Создание псевдонима команды....101
10.7.2. Сокращения команд....101
10.7.3. Стандартные параметры команд....102
10.7.4. Просмотр объявлений псевдонимов команд....102
10.7.5. Команда unalias....102
10.8. Вывод информации о раскрытии команд командной оболочкой....103
10.9. Практическое задание: команды и аргументы....103
10.10. Корректная процедура выполнения практического задания: команды и аргументы 105
Глава 11. Операторы управления....108
11.1. Точка с запятой (;)....108
11.2. Амперсанд (&)....109
11.3. Символ доллара со знаком вопроса ($?)....109
11.4. Двойной амперсанд (&&)....110
11.5. Двойная вертикальная черта (||)....110
11.6. Комбинирование операторов && и ||....111
11.7. Знак фунта (#)....111
11.8. Экранирование специальных символов (\)....112
11.8.1. Обратный слэш в конце строки....112
11.9. Практическое задание: операторы управления....112
11.10. Корректная процедура выполнения практического задания: операторы управления....113
Глава 12. Переменные командной оболочки....115
12.1. Символ доллара ($)....115
12.2. Зависимость от регистра....116
12.3. Создание переменных....116
12.4. Кавычки....116
12.5. Команда set....117
12.6. Команда unset....117
12.7. Переменная окружения $PS1....117
12.8. Переменная окружения $PATH....118
12.9. Команда env....119
12.10. Команда export....120
12.11. Разграничения переменных....121
12.12. Несвязанные переменные....121
12.13. Практическое задание: переменные командной оболочки....122
12.14. Корректная процедура выполнения практического задания: переменные командной оболочки....123
Глава 13. Встраивание и параметры командных оболочек....124
13.1. Встраивание командных оболочек....124

13.1.1. Обратные кавычки....125
13.1.2. Обратные кавычки или одинарные кавычки....125
13.2. Параметры командной оболочки....126
13.3. Практическое задание: встраивание командных оболочек....127
13.4. Корректная процедура выполнения практического задания: встраивание командных оболочек....128
Глава 14. История команд командной оболочки....129
14.1. Повторение последней выполненной команды....129
14.2. Повторение других команд....130
14.3. Команда history....130
14.4. Команда !n....130
14.5. Сочетание клавиш Ctrl-r....131
14.6. Переменная окружения $HISTSIZE....131
14.7. Переменная окружения $HISTFILE....131
14.8. Переменная окружения $HISTFILESIZE....132
14.9. Предотвращение сохранения команд....132
14.10. Регулярные выражения (дополнительная информация)....132
14.11. История команд оболочки Korn Shell (дополнительная информация)....132
14.12. Практическое задание: история команд командной оболочки....133
14.13. Корректная процедура выполнения практического задания: история команд командной оболочки....134
Глава 15. Формирование списков имен файлов на основе шаблонов....135
15.1. Звездочка (*)....136
15.2. Знак вопроса (?)....136
15.3. Квадратные скобки ([])....137
15.4. Диапазоны a-z и 0-9....138
15.5. Переменная окружения $LANG и квадратные скобки....138
15.6. Предотвращение формирования списков имен файлов на основе шаблонов....139
15.7. Практическое задание: формирование списков имен файлов на основе шаблонов....140
15.8. Корректная процедура выполнения практического задания: формирование списков имен файлов на основе шаблонов....142
Часть IV. Программные каналы и команды....145
Глава 16. Перенаправление потоков ввода/вывода....145
16.1. Потоки данных stdin, stdout и stderr....145
16.2. Перенаправление стандартного потока вывода....146
16.2.1. Операция перенаправления потока данных stdout (>)....146
16.2.2. Содержимое выходного файла удаляется....146
16.2.3. Параметр командной оболочки noclobber....147
16.2.4. Нейтрализация влияния параметра командной оболочки noclobber....147
16.2.5. Оператор дополнения >>....148
16.3. Перенаправление стандартного потока ошибок....148
16.3.1. Операция перенаправления потока данных stderr (2>)....148
16.3.2. Операция перенаправления нескольких потоков данных 2>&1....149
16.4. Перенаправление стандартного потока вывода и программные каналы....149
16.5. Объединение стандартных потоков вывода stdout и ошибок stderr....150
16.6. Перенаправление стандартного потока ввода....151
16.6.1. Операция перенаправления потока данных stdin (<)....151
16.6.2. Структура << here document....151
16.6.3. Структура <<< here string....152
16.7. Неоднозначное перенаправление потоков ввода/вывода....152
16.8. Быстрая очистка содержимого файла....152

16.9. Практическое задание: перенаправление потоков ввода/вывода.....153
16.10. Корректная процедура выполнения практического задания: перенаправление потоков ввода/вывода.....153
Глава 17. Фильтры.....155
17.1. Фильтр cat.....155
17.2. Фильтр tee.....155
17.3. Фильтр grep.....156
17.4. Фильтр cut.....158
17.5. Фильтр tr.....159
17.6. Фильтр wc.....161
17.7. Фильтр sort.....161
17.8. Фильтр uniq.....162
17.9. Фильтр comm.....163
17.10. Фильтр od.....164
17.11. Фильтр sed.....165
17.12. Примеры конвейеров.....166
17.12.1. Конвейер who | wc.....166
17.12.2. Конвейер who | cut | sort.....166
17.12.3. Конвейер grep | cut.....166
17.13. Практическое задание: фильтры.....167
17.14. Корректная процедура выполнения практического задания: фильтры.....167
Глава 18. Стандартные инструменты систем Unix.....169
18.1. Утилита find.....169
18.2. Утилита locate.....170
18.3. Утилита date.....171
18.4. Утилита cal.....172
18.5. Утилита sleep.....172
18.6. Команда time.....173
18.7. Утилиты gzip - gunzip.....173
18.8. Утилиты zcat - zmore.....174
18.9. Утилиты bzip2 - bunzip2.....174
18.10. Утилиты bzcat - bzmore.....175
18.11. Практическое задание: стандартные инструменты систем Unix.....175
18.12. Корректная процедура выполнения практического задания: стандартные инструменты систем Unix.....176
Глава 19. Регулярные выражения.....179
19.1. Версии синтаксисов регулярных выражений.....179
19.2. Утилита grep.....180
19.2.1. Вывод строк, совпадающих с шаблоном.....180
19.2.2. Объединение символов.....180
19.2.3. Один или другой символ.....181
19.2.4. Одно или большее количество совпадений.....182
19.2.5. Совпадение в конце строки.....182
19.2.6. Совпадение в начале строки.....183
19.2.7. Разделение слов.....183
19.2.8. Параметры утилиты grep.....184
19.2.9. Предотвращение раскрытия регулярного выражения командной оболочкой 184
19.3. Утилита rename.....185
19.3.1. Реализации утилиты rename.....185
19.3.2. Пакет perl.....185
19.3.3. Хорошо известный синтаксис.....186
19.3.4. Глобальная замена.....187

19.3.5. Замена без учета регистра....188
19.3.6. Изменение расширений....188
19.4. Утилита sed....189
19.4.1. Редактор потока данных....189
19.4.2. Интерактивный редактор....189
19.4.3. Простые обратные ссылки....189
19.4.4. Обратные ссылки....190
19.4.5. Точка для обозначения любого символа....190
19.4.6. Множественные обратные ссылки....190
19.4.7. Пробел....190
19.4.8. Необязательные вхождения....191
19.4.9. Ровно n повторений....191
19.4.10. От n до m повторений....192
19.5. История командной оболочки bash....192
Часть V. Текстовый редактор vi....194
Глава 20. Начальные сведения о текстовом редакторе vi....194
20.1. Режимы ввода команд и ввода текста....194
20.2. Начало редактирования текста (a A i l o O)....194
20.3. Замена и удаление символа (r x X)....195
20.4. Отмена и повторение действий (u .)....195
20.5. Перенос, копирование и вставка строки (dd yy pP)....196
20.6. Перенос, копирование и вставка строк (3dd 2yy)....196
20.7. Переход в начало и конец строки (0 или ^ и $)....197
20.8. Объединение двух (J) и более строк....197
20.9. Слова (w b)....197
20.10. Сохранение (или отказ от сохранения) данных и завершение работы (:w :q :q!) 198
20.11. Поиск (/ ?)....199
20.12. Замена всех найденных строк (:1,$ s/foo/bar/g)....199
20.13. Чтение файлов (:r :r !cmd)....200
20.14. Текстовые буферы....200
20.15. Работа с множеством файлов....200
20.16. Аббревиатуры строк....201
20.17. Соответствия клавиш....201
20.18. Установка значений параметров....201
20.19. Практическое задание: vi(m)....202
20.20. Корректная процедура выполнения практического задания: vi(m)....203
Часть VI. Сценарии....205
Глава 21. Введение в разработку сценариев....205
21.1. Предварительное чтение....205
21.2. Hello world....205
21.3. She-bang....206
21.4. Комментарий....207
21.5. Переменные....207
21.6. Использование рабочей командной оболочки....207
21.7. Отладка сценария....208
21.8. Предотвращение подмены имен файлов сценариев с целью повышения привилегий в системе....209
21.9. Практическое задание: введение в разработку сценариев....209
21.10. Корректная процедура выполнения практического задания: введение в разработку сценариев....210
Глава 22. Циклы в сценариях....211
22.1. Команда test []....211

22.2. Условный переход if then else....213
22.3. Условный переход if then elif....214
22.4. Цикл for....214
22.5. Цикл while....215
22.6. Цикл until....216
22.7. Практическое задание: проверки и циклы в сценариях....216
22.8. Корректная процедура выполнения практического задания: проверки и циклы в сценариях....217
Глава 23. Параметры сценариев....219
23.1. Параметры сценария....219
23.2. Обход списка параметров....220
23.3. Ввод в процессе исполнения сценария....221
23.4. Задействование файла конфигурации....221
23.5. Получение параметров сценария с помощью функции getopts....222
23.6. Получение параметров функционирования командной оболочки с помощью команды shopt....224
23.7. Практическое задание: параметры сценариев и специальные переменные....225
23.8. Корректная процедура выполнения практического задания: параметры сценариев и специальные переменные....226
Глава 24. Дополнительная информация о сценариях....227
24.1. Команда eval....227
24.2. Оператор (( ))....228
24.3. Команда let....228
24.4. Оператор case....229
24.5. Функции сценариев командной оболочки....231
24.6. Практическое задание: дополнительная информация о сценариях....232
24.7. Корректная процедура выполнения практического задания: дополнительная информация о сценариях....233
Часть VII. Управление локальными учетными записями пользователей....236
Глава 25. Вводная информация об учетных записях пользователей....236
25.1. Утилита whoami....236
25.2. Утилита who....236
25.3. Команда who am i....236
25.4. Утилита w....237
25.5. Утилита id....237
25.6. Утилита su для работы от лица другого пользователя....237
25.7. Утилита su для работы от лица пользователя root....237
25.8. Утилита su для пользователя root....238
25.9. Команда su - $имя_пользователя....238
25.10. Команда su -....238
25.11. Запуск приложения от лица другого пользователя....239
25.12. Утилита visudo....239
25.13. Команда sudo su -....239
25.14. Журналирование неудачных попыток использования утилиты sudo....240
25.15. Практическое задание: вводная информация об учетных записях пользователей....241
25.16. Корректная процедура выполнения практического задания: вводная информация об учетных записях пользователей....242
Глава 26. Управление учетными записями пользователей....244
26.1. Управление учетными записями пользователей....244
26.2. Файл /etc/passwd....245
26.3. Пользователь root....246

26.4. Утилита useradd....246
26.5. Файл /etc/default/useradd....246
26.6. Утилита userdel....247
26.7. Утилита usermod....247
26.8. Создание домашних директорий пользователей....247
26.9. Директория /etc/skel/....247
26.10. Удаление домашних директорий пользователей....248
26.11. Командная оболочка, используемая для входа в систему....248
26.12. Утилита chsh....249
26.13. Практическое задание: управление учетными записями пользователей....250
26.14. Корректная процедура выполнения практического задания: управление учетными записями пользователей....251
Глава 27. Пароли пользователей....253
27.1. Утилита passwd....253
27.2. Файл shadow....254
27.3. Шифрование ключевых фраз с помощью утилиты passwd....255
27.4. Шифрование ключевых фраз с помощью утилиты openssl....255
27.5. Шифрование ключевых фраз с помощью функции crypt....256
27.6. Файл /etc/login.defs....258
27.7. Утилита chage....259
27.8. Блокировка учетных записей....259
27.9. Редактирование локальных файлов....261
27.10. Практическое задание: пароли пользователей....261
27.11. Корректная процедура выполнения практического задания: пароли пользователей....262
Глава 28. Профили пользователей....265
28.1. Системный профиль....265
28.2. Файл ~/.bash_profile....266
28.3. Файл ~/.bash_login....267
28.4. Файл ~/.profile....267
28.5. Файл ~/.bashrc....268
28.6. Файл ~/.bash_logout....268
28.7. Обзор сценариев дистрибутива Debian....269
28.8. Обзор сценариев дистрибутива RHEL5....270
28.9. Практическое задание: профили пользователей....270
28.10. Корректная процедура выполнения практического задания: профили пользователей....271
Глава 29. Группы пользователей....271
29.1. Утилита groupadd....272
29.2. Файл group....272
29.3. Команда groups....273
29.4. Утилита usermod....273
29.5. Утилита groupmod....273
29.6. Утилита groupdel....274
29.7. Утилита gpasswd....274
29.8. Утилита newgrp....275
29.9.Утилита vigr....276
29.10. Практическое задание: группы пользователей....276
29.11. Корректная процедура выполнения практического задания: группы пользователей....276
Часть VIII. Механизмы безопасной работы с файлами....278
Глава 30. Стандартные права доступа к файлам....278

30.1. Механизм владения файлами....278
30.1.1. Пользователь и группа, владеющие файлом....278
30.1.2. Вывод списка учетных записей пользователей....278
30.1.3. Утилита chgrp....280
30.1.4. Утилита chown....280
30.1.5. Список специальных типов файлов....280
30.2. Права доступа....281
30.2.1. Символы rwx....281
30.2.2. Три набора символов rwx....282
30.2.3. Примеры прав доступа к файлам....283
30.2.4. Установка прав доступа (chmod)....284
30.2.4. Установка прав доступа в восьмеричном представлении....285
30.2.5. Значение umask....286
30.2.6. Команда mkdir -m....286
30.2.7. Команда cp -p....287
30.3. Практическое задание: стандартные права доступа к файлам....287
30.4. Корректная процедура выполнения практического задания: стандартные права доступа к файлам....289
Глава 31. Расширенные права доступа к файлам....291
31.1. Бит sticky для директорий....291
31.2. Бит setgid для директории....292
31.3. Биты setgid и setuid для обычных файлов....293
31.4. Бит setuid для исполняемого файла sudo....294
31.5. Практическое задание: биты sticky, setuid и setgid....294
31.6. Корректная процедура выполнения практического задания: биты sticky, setuid и setgid....295
Глава 32. Списки контроля доступа....297
32.1. Параметр acl в файле /etc/fstab....297
32.2. Утилита getfacl....298
32.3. Утилита setfacl....298
32.4. Удаление элемента списка контроля доступа....299
32.5. Удаление всего списка контроля доступа....299
32.6. Маска прав списка контроля доступа....299
32.7. Приложение eiciel....300
Глава 33. Ссылки на файлы....301
33.1. Структуры inode....301
33.1.1. Содержимое структуры inode....301
33.1.2. Таблица структур inode....301
33.1.3. Идентификатор структуры inode....302
33.1.4. Структуры inode и содержимое файлов....302
33.2. О директориях....303
33.2.1. Директория является таблицей....303
33.2.2. Директории . и 303
33.3. Жесткие ссылки....304
33.3.1. Создание жестких ссылок....304
33.3.2. Поиск жестких ссылок....304
33.4. Символьные ссылки....305
33.5. Удаление ссылок....305
33.6. Практическое задание: ссылки на файлы....305
33.7. Корректная процедура выполнения практического задания: ссылки на файлы….306
Часть IX. Приложения....308
Приложение A. Раскладки клавиатуры....308

A.1. О раскладках клавиатуры....308
A.2. Раскладка клавиатуры оконной системы X....308
A.3. Раскладка клавиатуры в командной оболочке....309
Приложение B. Аппаратное обеспечение....309
B.1. Шины....309
B.1.1. О шинах....309
B.1.2. Директория /proc/bus....310
B.1.3. Утилита /usr/sbin/lsusb....310
B.1.4. Файл /var/lib/usbutils/usb.ids....310
B.1.5. Утилита /usr/sbin/lspci....311
B.2. Запросы прерываний....311
B.2.1. О запросах прерываний....311
B.2.2. Файл /proc/interrupts....312
B.2.3. Утилита dmesg....313
B.3. Порты ввода-вывода....313
B.3.1. О портах ввода-вывода....313
B.3.2. Файл /proc/ioports....313
B.4. Технология DMA....314
B.4.1. О технологии DMA....314
B.4.2. Файл /proc/dma....314

Часть I. Вводная информация о Linux

Глава 1. История Linux

В данной главе кратко описана история, а также область применения ОС Linux.

Если вам не терпится начать работу с Linux без длительных дискуссий об истории этой ОС, дистрибутивах и условиях их использования, вы можете перейти непосредственно к Части II, а точнее к Главе 6. "Работа с директориями".

1.1. История Linux

Все современные операционные системы уходят своими корнями в 1969 год, в котором Dennis Ritchie и Ken Thompson из AT&T Bell Labs разработали язык программирования C и операционную систему Unix. Они поделились своими наработками в форме исходного кода (да, приложения с открытым исходным кодом появились еще в семидесятых годах прошлого века) со всем миром, включая хиппи из калифорнийского Университета Беркли. К 1975 году, в котором компания AT&T начала распространение ОС UNIX на коммерческой основе, практически половина ее исходного кода была создана сторонними разработчиками. Хиппи не были довольны тем, что коммерческая компания продавала программное обеспечение, в разработке которого они принимали непосредственное участие; баталии (юридического плана) привели к появлению в семидесятых годах двух версий Unix: официальной версии Unix от компании AT&T, а также бесплатной версии BSD Unix.

В восьмидесятых годах многие компании начали разработку своих собственных вариантов ОС Unix: компания IBM создала AIX, компания Sun создала SunOS (которая позднее стала называться Solaris), компания HP создала HP-UX и многие другие компании поступили аналогичным образом. В результате появилось огромное количество разновидностей ОС Unix с множеством вариантов выполнения одного и того же действия. И именно это время можно считать реальным временем зарождения ОС Linux, так как именно тогда Richard Stallman решил приблизить конец эры раздробленности в мире Unix, в которой каждая компания заново изобретает колесо, создав проект GNU (GNU is Not Unix - GNU не Unix). Цель данного проекта заключалась в создании операционной системы, которая была бы доступна для каждого человека и над развитием которой могли бы работать все желающие (так же, как это было в семидеся-

тых годах). Многие инструменты с интерфейсом командной строки, которые вы можете использовать сегодня при работе с ОС Linux или Solaris, являются инструментами, созданными в рамках проекта GNU.

Начало девяностых годов ознаменовалось покупкой говорящим по-шведски финским студентом Linus Torvalds компьютера архитектуры i386 и разработкой им же нового, совместимого со стандартом POSIX ядра операционной системы. Он опубликовал исходный код созданного ядра в сети и заявил, что это ядро никогда не будет поддерживать какое-либо аппаратное обеспечение, кроме уже поддерживаемого аппаратного обеспечения архитектуры i386. Многие люди высоко оценили комбинацию из этого ядра ОС и инструментов, созданных в рамках проекта GNU, а остальное, как говорится, уже стало историей.

На сегодняшний день более 90 процентов суперкомпьютеров (включая 10 наиболее мощных из них), более половины всех смартфонов, миллионы настольных компьютеров, около 70 процентов вебсерверов, большое количество планшетных компьютеров и некоторая бытовая техника (DVD-проигрыватели, стиральные машины, DSL-модемы, маршрутизаторы, ...) работают под управлением ОС Linux. В текущий момент это наиболее часто используемая операционная система в мире.

Версия 3.2 ядра Linux была выпущена в январе 2012 года. Объем исходного кода данной версии ядра ОС вырос практически на двести тысяч строк (по сравнению с объемом исходного кода версии 3.1) благодаря наработкам более 4000 разработчиков, труд которых оплачивался более чем 200 коммерческими компаниями, включая Red Hat, Intel, Broadcom, Texas Instruments, IBM, Novell, Qualcomm, Samsung, Nokia, Oracle, Google и даже Microsoft.

Источники дополнительной информации:

http://en.wikipedia.org/wiki/Dennis_Ritchie
http://en.wikipedia.org/wiki/Richard_Stallman
http://en.wikipedia.org/wiki/Linus_Torvalds
http://kernel.org
http://lwn.net/Articles/472852/
http://www.linuxfoundation.org/
http://en.wikipedia.org/wiki/Linux
http://www.levenez.com/unix/ (большой постер, посвященный истории ОС Unix)

Глава 2. Дистрибутивы

В данной главе представлен краткий обзор наиболее актуальных на данный момент дистрибутивов.

Дистрибутив Linux является коллекцией программного обеспечения (обычно с открытым исходным кодом), работающего под управлением ядра Linux. Дистрибутив (в английском языке для обозначения дистрибутива используется как термин distribution, так и сокращенная версия термина distro) может содержать серверное программное обеспечение, инструменты для управления системой, документацию и множество приложений для работы в окружении рабочего стола, причем эти программные компоненты содержатся в безопасном централизованном репозитории программного обеспечения. Разработчики дистрибутива обычно пытаются максимально унифицировать внешний вид и функции приложений из состава дистрибутива, предоставить инструментарий для простого управления пакетами программного обеспечения и чаще всего формируют набор программного обеспечения для выполнения конкретных задач.

Давайте рассмотрим некоторые популярные дистрибутивы.

2.1. RedHat

RedHat является тесно связанной с Linux коммерческой компанией с бюджетом, исчисляющимся миллиардами долларов, работники которой прикладывают огромные усилия к разработке Linux. В данной компании трудоустроены сотни специалистов в области Linux, поэтому компания оказывает превосходную техническую поддержку своим клиентам. Продукты компании (Red Hat Enterprise Linux и Fedora) распространяются бесплатно. В отличие от дистрибутива Red Hat Enterprise Linux, который тщательно тестируется перед выпуском и поддерживается в течение периода длительностью до семи лет, Fedora является дистрибутивом с более частыми обновлениями, но без платной технической поддержки.

2.2. Ubuntu

Компания Canonical начала работу с распространения по почте бесплатных компакт-дисков с дистрибутивом Ubuntu Linux в 2004 году, в течение короткого периода обеспечив популярность своего дистрибутива среди домашних пользователей (многие пользователи перешли к использованию дистрибутива, отказавшись от Microsoft Windows). Целью компании Canonical является разработка в рамках дистрибутива Ubuntu простого в использовании графического окруже-

ния рабочего стола, работающего под управлением ядра Linux, которое позволит обойтись без командной строки. Конечно же, целью компании также является извлечение прибыли путем продажи услуг технической поддержки дистрибутива Ubuntu.

2.3. Debian

За дистрибутивом Debian не стоит никаких компаний. Вместо персонала компании развитие дистрибутива осуществляется тысячами хорошо организованных разработчиков, которые избирают лидера проекта Debian через каждые два года. Debian зарекомендовал себя как один из самых стабильных дистрибутивов Linux. Также данный дистрибутив является основой каждого из релизов дистрибутива Ubuntu. Существуют три версии дистрибутива Debian: stable (стабильная ветвь), testing (тестовая ветвь) и unstable (нестабильная ветвь). Каждый из выпусков дистрибутива Debian носит имя персонажа фильма "История игрушек" ("Toy Story").

2.4. Другие дистрибутивы

Такие дистрибутивы, как CentOS, Oracle Enterprise Linux и Scientific Linux базируются на дистрибутиве Red Hat Enterprise Linux и используют множество аналогичных принципов, директорий и техник администрирования системы. Дистрибутивы Linux Mint, Edubuntu и многие другие дистрибутивы с именами *buntu базируются на дистрибутиве Ubuntu и, таким образом, очень похожи на дистрибутив Debian. Существуют также сотни других дистрибутивов Linux.

2.5. Какой дистрибутив следует выбрать?

Если вы не работали с Linux до 2014 года, вам следует установить последнюю версию дистрибутива Ubuntu или Fedora. Если же вы просто хотите поработать с командной оболочкой Linux, вы можете установить дистрибутив Ubuntu server и/или дистрибутив CentOS (без графического интерфейса).

Ссылки на сайты дистрибутивов:

redhat.com
ubuntu.com
debian.org
centos.org
distrowatch.com

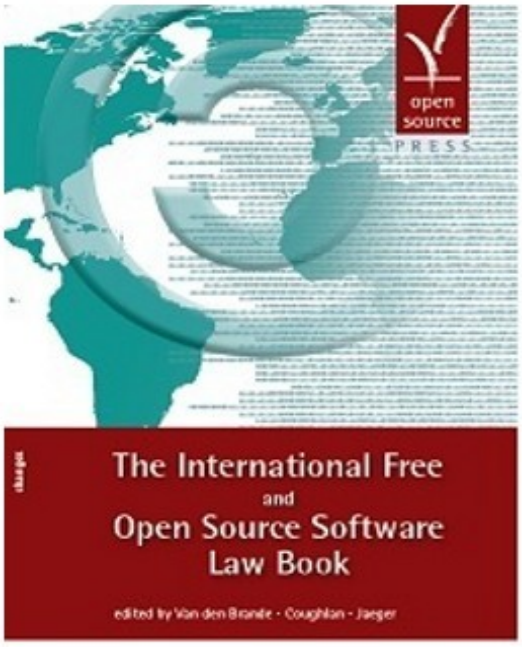
Глава 3. Лицензирование

В данной главе приводятся краткие пояснения относительно условий различных лицензий, в соответствии с которыми распростра-

няется программное обеспечение из состава дистрибутивов операционных систем.

Хотелось бы выразить огромную признательность Ywein Van den Brande за написание большей части текста данной главы.

Ywein является юристом, соавтором книги The International FOSS Law Book, а также автором книги Praktijkboek Informaticarecht (на голландском языке).



http://ifosslawbook.org
http://www.crealaw.eu

3.1. О лицензировании программного обеспечения

Существуют две преобладающих парадигмы лицензирования программного обеспечения: парадигма свободного программного обеспечения и программного обеспечения с открытым исходным кодом (FOSS), а также парадигма собственнического (пропиетарного) программного обеспечения. Критерий различия двух упомянутых парадигм базируется на контроле за распространением и использованием программного обеспечения. В случае собственнического программного обеспечения функции контроля за его распространением и использованием по большей части осуществляется поставщиком программного обеспечения, в то время, как в случае свободного программного обеспечения и программного обеспечения с открытым исходным кодом аналогичные функции контроля в большей степени возлагаются на конечного пользователя. Но, несмотря на то, что описанные парадигмы лицензирования отличаются друг от друга, они используют одни и те же законы об авторском праве для претворения в жизнь и обеспечения соблюдения поставленных условий. С

точки зрения закона свободное программное обеспечение и программное обеспечение с открытым исходным кодом может рассматриваться как программное обеспечение, пользователи которого в общем случае получают больше прав, чем при использовании собственнического программного обеспечения благодаря соответствующей лицензии при условии использования одних и тех же основополагающих механизмов лицензирования.

В соответствии с теорией права, автор свободного программного обеспечения или программного обеспечения с открытым исходным кодом, в отличие от автора программного обеспечения, распространяемого как публичное достояние (public domain), никоим образом не отказывается от своих прав на свое произведение. Парадигма свободного программного обеспечения и программного обеспечения с открытым исходным кодом основывается на правах автора произведения (авторском праве) для придания силы условиям лицензий свободного программного обеспечения и программного обеспечения с открытым исходным кодом. Условия лицензий данного типа должны выполняться пользователем программного обеспечения точно так же, как и условия лицензий собственнического программного обеспечения. Вам следует всегда тщательно знакомиться с лицензионными соглашениями перед использованием любого стороннего программного обеспечения.

Примерами собственнического программного обеспечения являются операционная система AIX от компании IBM, операционная система HP-UX от компании HP и система управления базами данных Oracle Database 11g. Вы не имеете права устанавливать или использовать данное программное обеспечение, не осуществив лицензионных отчислений. Также вы не имеете права распространять копии данного программного обеспечения и модифицировать его, к тому же исходный код данного программного обеспечения не доступен для широкой публики.

3.2. Программное обеспечение, распространяемое как общественное достояние, а также бесплатное программное обеспечение

Оригинальное программное обеспечение, являющееся плодом интеллектуального творчества автора, защищается законом об авторском праве. Не оригинальное программное обеспечение не подпадает под защиту закона об авторском праве и может, в принципе, использоваться бесплатно.

Программное обеспечение, распространяемое как общественное достояние, обычно рассматривается как программное обеспечение, автор которого отказался от всех прав на него, но при этом никто не может предъявить какие-либо права на это программное обеспечение. Оно может использоваться, распространяться или применяться для различных целей абсолютно свободно без получения согласия от автора или осуществления лицензионных отчислений. Программное обеспечение, распространяемое как общественное достояние, в определенных случаях может даже представляться третьими лицами как их собственное произведение, а модифицируя оригинальные версии этого программного обеспечения, третьи лица могут создавать новые версии, которые будут распространяться на условиях, отличных от условий распространения оригинальных версий.

Бесплатное программное обеспечение не является ни программным обеспечением, распространяющимся как общественное достояние, ни свободными программным обеспечением или программным обеспечением с открытым исходным кодом. Это собственническое программное обеспечение, которое вы можете использовать без уплаты лицензионных отчислений. Однако, в случае использования данного программного обеспечения должны соблюдаться обычно строгие условия лицензионных соглашений.

Примерами бесплатного программного обеспечения являются приложение для чтения документов формата PDF Adobe Reader, приложение для работы с IP-телефонией Skype, а также игра Command and Conquer: Tiberian Sun (данная игра распространялась на платной основе как собственническое программное обеспечение с 1999 года, но в 2011 году была переведена в категорию бесплатного программного обеспечения).

3.3. Свободное программное обеспечение и программное обеспечение с открытым исходным кодом

И движение свободного программного обеспечения (называемого Free Software по-английски, vrije software по-голландски и Logiciel Libre по-французски), и движение программного обеспечения с открытым исходным кодом, преследуют аналогичные цели и поддерживают аналогичные лицензии. Но исторически сложилось так, что эти движения по-разному рассматривают основополагающие понятия ввиду различий в расстановке акцентов. В то время, как движение свободного программного обеспечения фокусирует внимание на правах (а именно, четырех свободах), которые программное обеспечение предоставляет пользователям, движение программного обеспече-

ния с открытым исходным кодом ссылается на свое определение открытого исходного кода и акцентирует внимание на преимуществах процесса совместной разработки программного обеспечения.

В последнее время термины "свободное программное обеспечение" и "программное обеспечение с открытым исходным кодом" или FOSS стали практически равнозначными. Реже используемый вариант обозначения свободного программного обеспечения free/libre/open source software (FLOSS) предполагает использование слова libre для уточнения того, что свободное программное обеспечение должно предоставлять определенную свободу, но не обязательно должно распространяется бесплатно.

Примерами свободного программного обеспечения являются коллекция компиляторов gcc, система управления базами данных MySQL, а также редактор растровой графики gimp.

Подробная информация об упомянутых четырех свободах изложена на следующей странице:

http://www.gnu.org/philosophy/free-sw.html

Упомянутое определение программного обеспечения с открытым исходным кодом приведено на следующей странице:

http://www.opensource.org/docs/osd

Данное определение базируется на Критериях Debian по определению Свободного ПО, доступных на странице:

http://www.debian.org/social_contract#guidelines

3.4. Универсальная общественная лицензия проекта GNU

Все больший и больший объем программного обеспечения распространяется в соответствии с условиями лицензии GNU GPL (в 2006 году для нового выпуска Java была использована лицензия GPL). Данная лицензия (а именно, ее версии 2 и 3) является основной лицензией, одобренной Фондом свободного программного обеспечения. Ее главной характеристикой является принцип копилефта (copyleft). Идея этого принципа заключается в том, что каждый из последующих пользователей программного обеспечения в обмен на получение права использования данного программного обеспечения, должен распространять внесенные им улучшения, а также основанные на данном программном обеспечении работы, в соответствии с условиями лицензии оригинального программного обеспечения среди всех других поль-

зователей в том случае, если он примет решение о распространении этих улучшений и производных работ. Другими словами, программное обеспечение, содержащее программные компоненты, которые распространяются в соответствии с условиями лицензии GNU GPL, в свою очередь, должно распространяться в соответствии с условиями лицензии GNU GPL (или совместимой лицензии, о чем будет сказано ниже). При этом невозможно включать программные компоненты, защищенные авторским правом и распространяемые в соответствии с условиями лицензии GNU GPL, в состав собственнической работы. Кроме того, возможность использования лицензии GPL была подтверждена в ходе одного из судебных разбирательств.

3.5. Использование программного обеспечения, распространяемого в соответствии с условиями версии 3 лицензии GPL

Вы можете использовать программное обеспечение, распространяемое в соответствии с условиями версии 3 лицензии GPL, практически без выполнения каких-либо условий. В том случае, если вы исключительно используете такое программное обеспечение по его прямому назначению, вам даже не придется принимать условия лицензии GPL версии 3. Однако, в том случае, если вы используете данное программное обеспечение для других целей, таких, как модификация или повторное распространение, вы автоматически принимаете условия этой лицензии.

В том случае, если вы используете программное обеспечение исключительно для собственных целей (включая работу с программным обеспечением с задействованием функции передачи данных по сети), вы можете свободно модифицировать его и не обязаны распространять модифицированную вами версию. Вы также можете нанять сторонних разработчиков с целью доработки программного обеспечения для решения именно ваших задач под вашим руководством и контролем. Но в том случае, если вы модифицируете программное обеспечение и используете его не только внутри вашей организации, считается, что вы распространяете это программное обеспечение. В этом случае вы должны распространять ваши модификации в соответствии с условиями лицензии GPL версии 3 (в соответствии с принципом копилефта). При распространении программного обеспечения в соответствии с условиями лицензии GPL версии 3, вы должны выполнять и некоторые другие обязательства. Ввиду этого вам следует тщательно ознакомиться с текстом лицензии GPL версии 3.

С помощью программного обеспечения, распространяемого в соответствии с условиями лицензии GPL версии 3, могут создаваться различные работы: условия лицензии GPL версии 3 не будут автоматически распространяться на эти работы.

3.6. Лицензия BSD

Существует несколько версий оригинальной лицензии с описанием условий распространения программного обеспечения, созданной в Университете Беркли. Наиболее часто используемым вариантом лицензии является лицензия из 3 пунктов ("Новая лицензия BSD" или "Модифицированная лицензия BSD").

Это разрешающая лицензия, описывающая условия распространения свободного программного обеспечения. Данная лицензия налагает минимальные ограничения на процесс распространения программного обеспечения. Именно этим она и отличается от таких описанных выше использующих механизм копилефт лицензий, как лицензия GPL версии 3.

Упомянутое отличие не является существенным в том случае, если вы просто используете программное обеспечение по прямому назначению, но приобретает важность в том случае, если вы начинаете повторное распространение копий программного обеспечения в неизменном виде или ваших собственных модифицированных версий.

3.7. Другие лицензии

Существует множество типов лицензий, описывающих условия распространения как свободного, так и несвободного программного обеспечения. Вам следует читать их и разбираться в установленных условиях распространения программного обеспечения перед его использованием.

3.8. Комбинирование лицензий программного обеспечения

В том случае, если вы используете несколько источников программных компонентов или желаете распространять ваше программное обеспечение в соответствии с условиями отличной лицензии, вам придется удостовериться в том, что все используемые лицензии являются совместимыми. Некоторые лицензии, используемые для объявления условий распространения свободного программного обеспечения (такие, как лицензия BSD) совместимы с собственническими лицензиями, но большинство других подобных лицензий не является совместимым. В том случае, если вы столкнулись с несовме-

стимостью лицензий, вам следует связаться с автором программного компонента с несовместимой лицензией для согласования различных условий использования лицензий или отказаться от использования программных компонентов с несовместимыми лицензиями.

Глава 4. Установка Linux в домашних условиях

При написании данной книги предполагалось, что вы имеете доступ к работающему под управлением ОС Linux компьютеру. В большинстве компаний имеется один или несколько серверов на базе ОС Linux и в том случае, если вы имеете доступ к командной оболочке такого сервера, ваше рабочее окружение уже готово (вы можете пропустить данную главу и перейти к чтению следующей главы).

Другой вариант подготовки рабочего окружения заключается во вставке компакт диска с файлами дистрибутива Ubuntu Linux в привод компьютера, на котором уже установлена (или не установлена) ОС Microsoft Windows и следовании инструкциям установщика. Установщик дистрибутива Ubuntu позволяет изменить размер разделов жесткого диска (или создать на нем новые разделы), а также настроить меню загрузчика операционных систем для выбора операционной системы Windows или Linux в процессе загрузки компьютера.

В том случае, если у вас в текущее время нет доступа к компьютеру под управлением ОС Linux и вы не имеете возможности, либо не уверены в возможности установки ОС Linux на ваш компьютер, специально для вас в данной главе описан третий вариант: установка ОС Linux с использованием виртуальной машины.

Установка операционной системы с использованием виртуальной машины (созданной с помощью программных компонентов из пакета Virtualbox) является простой и безопасной операцией. Даже в том случае, если вы допустите ошибки и выведете из строя работающую в виртуальной машине систему на основе ядра Linux, это никак не повлияет на работу реальной системы, под управлением которой работает компьютер.

В данной главе приведены описания простых операций со снимками экрана, которые необходимо выполнить для развертывания рабочего сервера на базе Ubuntu на виртуальной машине, созданной средствами Virtualbox. Эти операции очень похожи на операции, необходимые для установки дистрибутива Fedora, CentOS или даже Debian, кроме того, при желании, вы также можете использовать пакет VMWare вместо Virtualbox для создания виртуальной машины.

4.1. Загрузка образа установочного диска ОС Linux

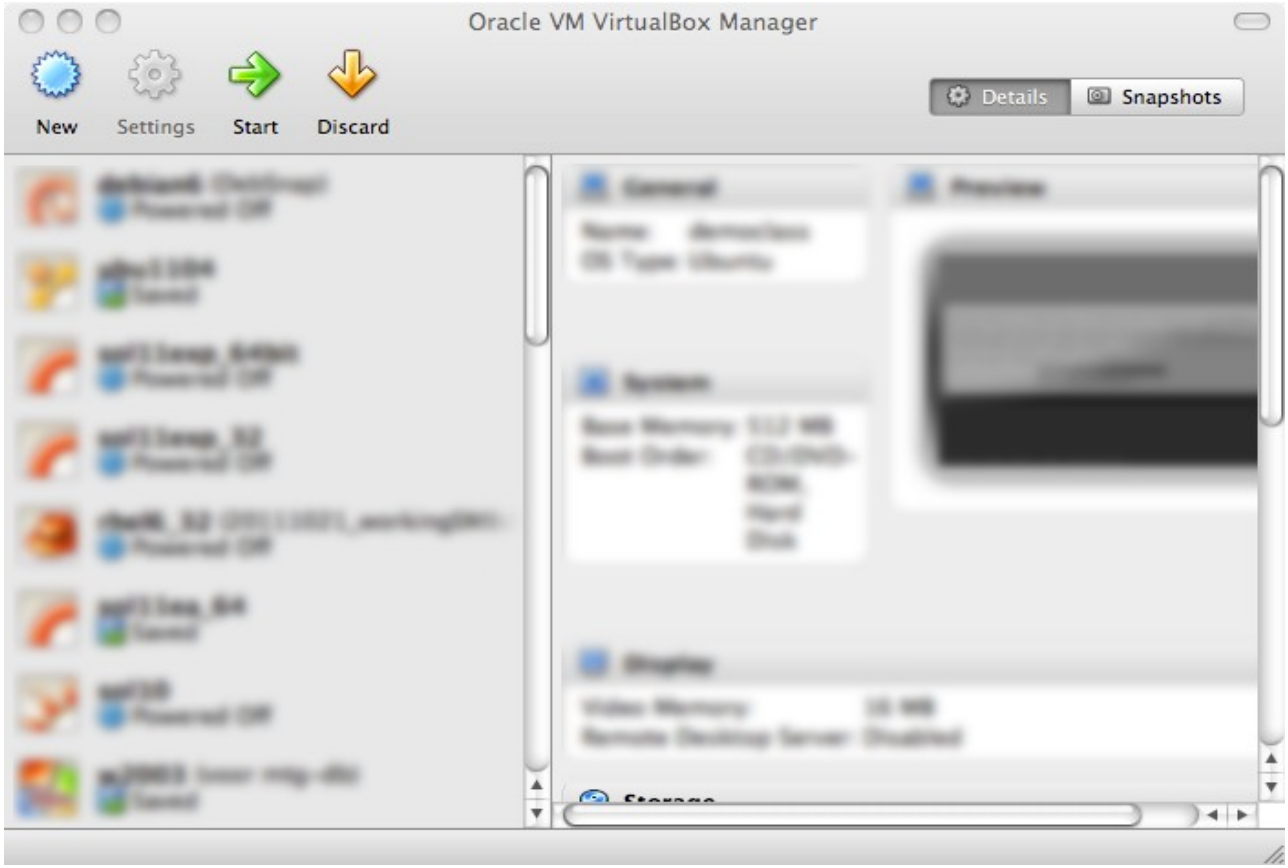
Подготовку рабочего окружения следует начинать с загрузки образа установочного диска ОС Linux (файла с расширением .ISO) с сайта выбранного вами дистрибутива из сети Интернет. Уделите особое внимание выбору корректного образа диска, соответствующего архитектуре центрального процессора вашего компьютера; выбирайте образ диска для архитектуры i386 в том случае, если вы не уверены. В случае выбора образа диска, не соответствующего архитектуре используемого центрального процессора (например, образа диска для архитектуры x86_64 при работе со старым процессором Pentium), вы столкнетесь с неустранимыми ошибками практически стразу же после начала загрузки системы.

4.2. Загрузка пакета Virtualbox

Второй этап подготовки рабочего окружения (который должен выполняться после окончания загрузки файла образа установочного диска с расширением .ISO) заключается в загрузке пакета Virtualbox. В том случае, если вы в данный момент работаете с ОС Microsoft Windows, вам следует скачать и установить пакет Virtualbox для Windows!

4.3. Создание виртуальной машины

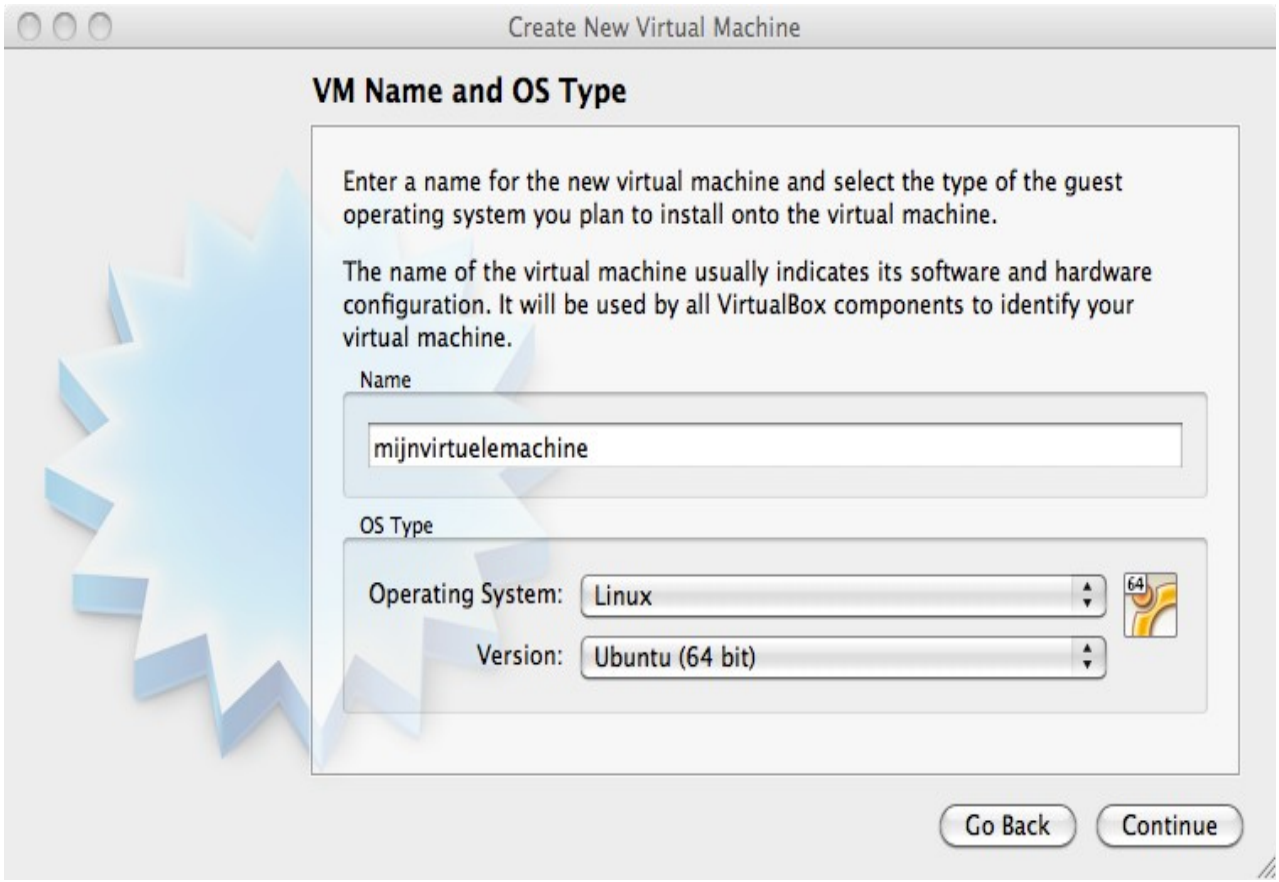
Теперь можно запустить приложение Virtualbox. В отличие от приведенного ниже снимка экрана, левая панель окна вашего приложения должна быть пуста.



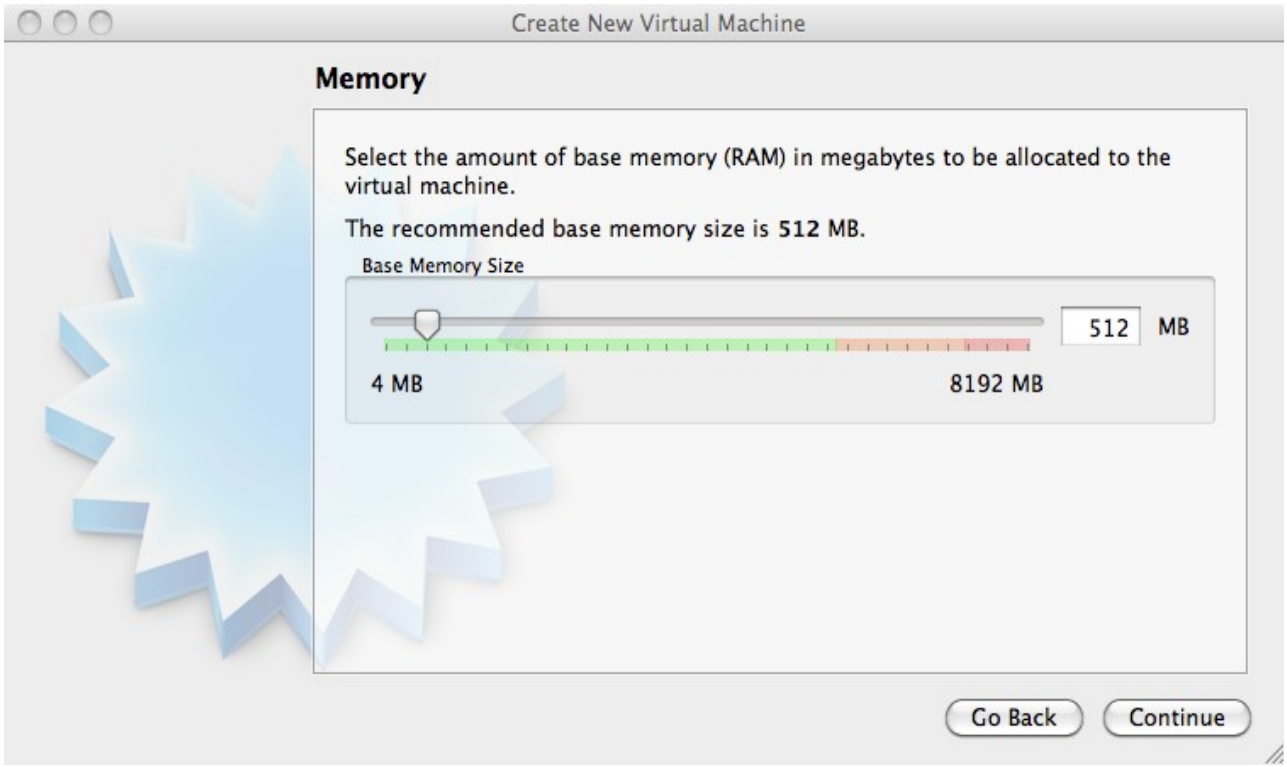
Нажмите на кнопку "New" панели инструментов для того, чтобы создать новую виртуальную машину. Постараемся разобраться вместе с данным мастером подготовки виртуальной машины к работе. Снимки окон приложения были сделаны при работе с ОС Mac OS X; они

могут незначительно отличаться от окон запущенного вами приложения в том случае, если вы работаете с ОС Microsoft Windows.

Присвойте имя вашей виртуальной машине (также, если нужно, выберите 32-битную или 64-битную архитектуру операционной системы).



Выделите оперативную память для виртуальной машины (установите значение 512MB в том случае, если ваш компьютер имеет 2GB или больший объем оперативной памяти, в противном случае установите значение 256MB).

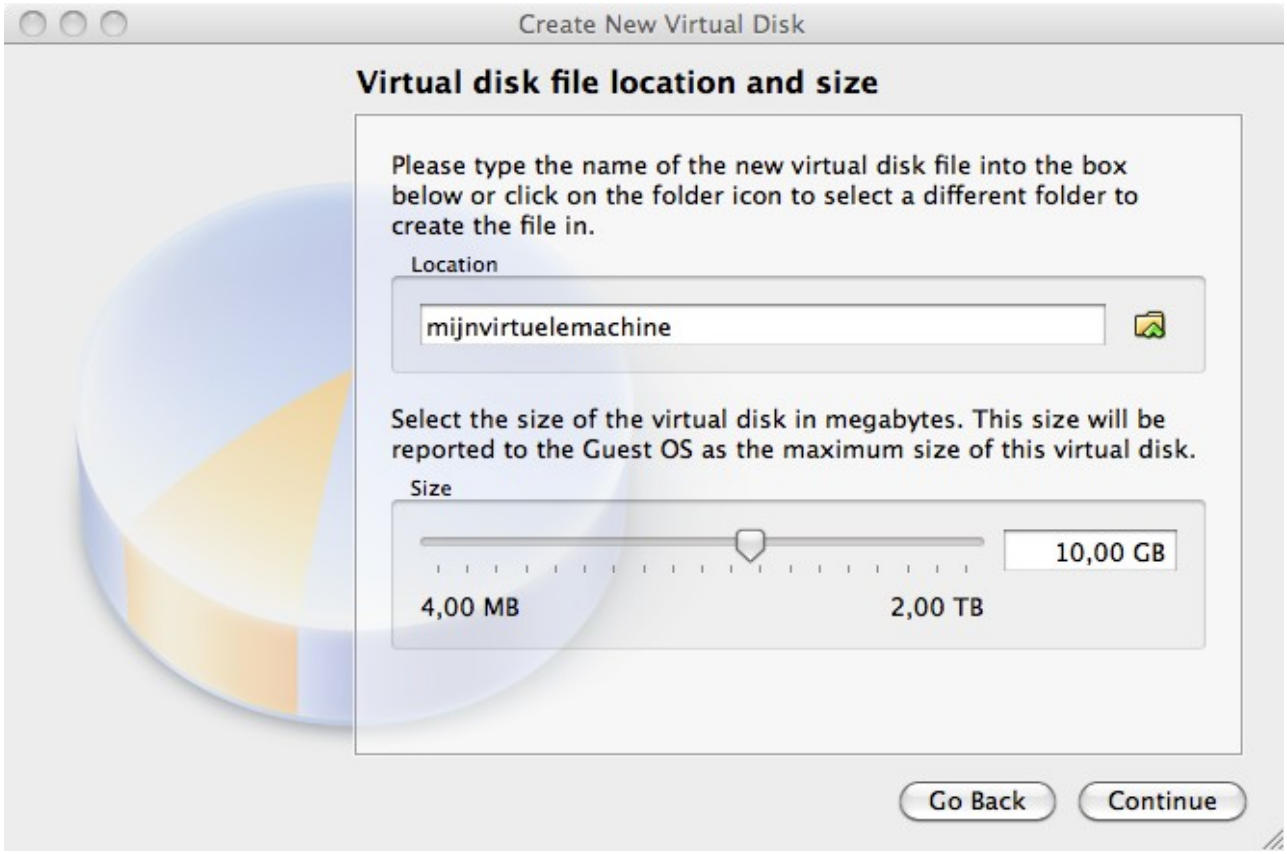


Выберите вариант создания нового диска (помните о том, что в результате будет создан виртуальный диск).

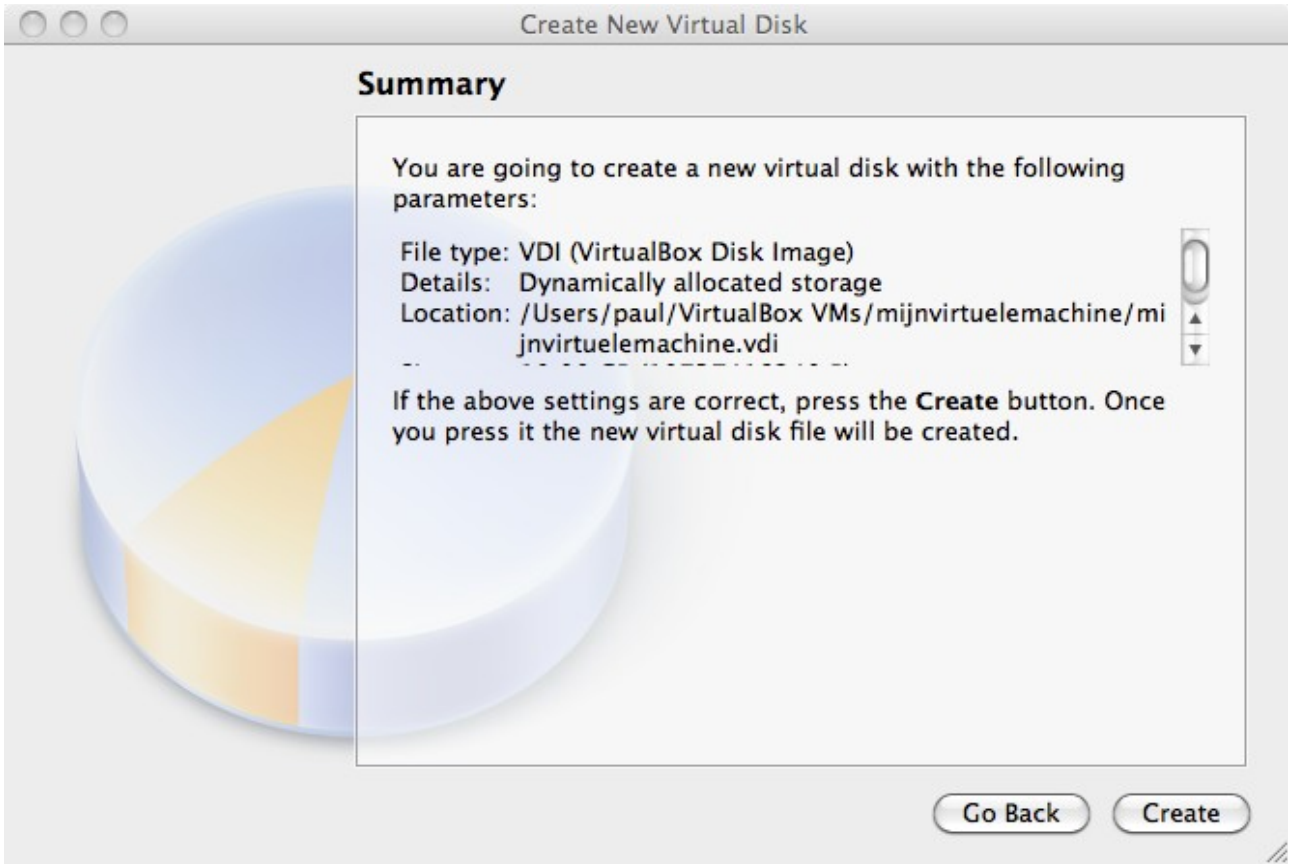
Если вам будет задан представленный ниже вопрос, выберите вариант vdi.

Выберите вариант динамического резервирования объема жесткого диска (dynamically allocated) для размещения данных виртуального диска (вариант fixed size предназначен для резервирования всего установленного объема жесткого диска для размещения данных виртуального диска и полезен исключительно при промышленной эксплуатации виртуальной машины или при работе с очень старым, медленным аппаратным обеспечением).

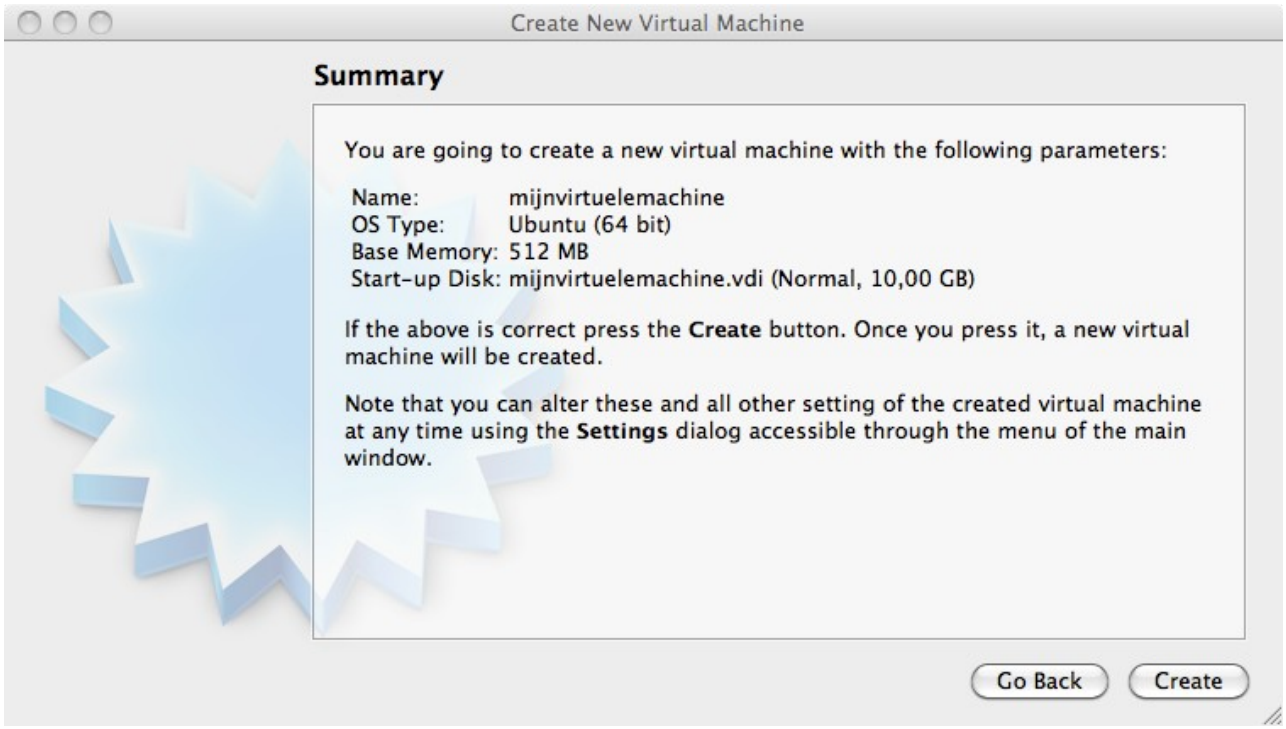
Установите объем виртуального диска, выбрав значение из диапазона от 10GB до 16GB.



Нажмите кнопку "create" для создания виртуального диска.

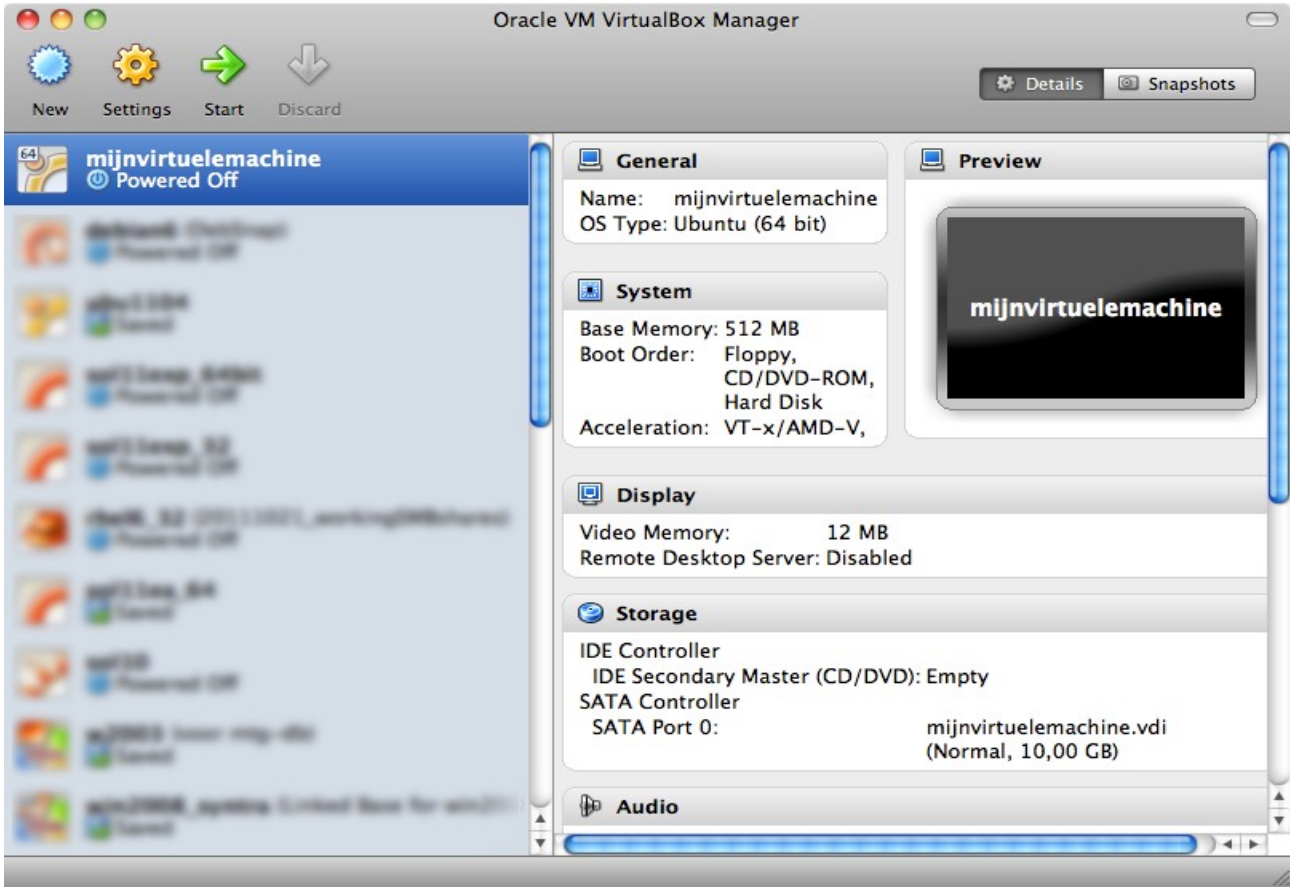


Нажмите кнопку "create" для создания виртуальной машины.

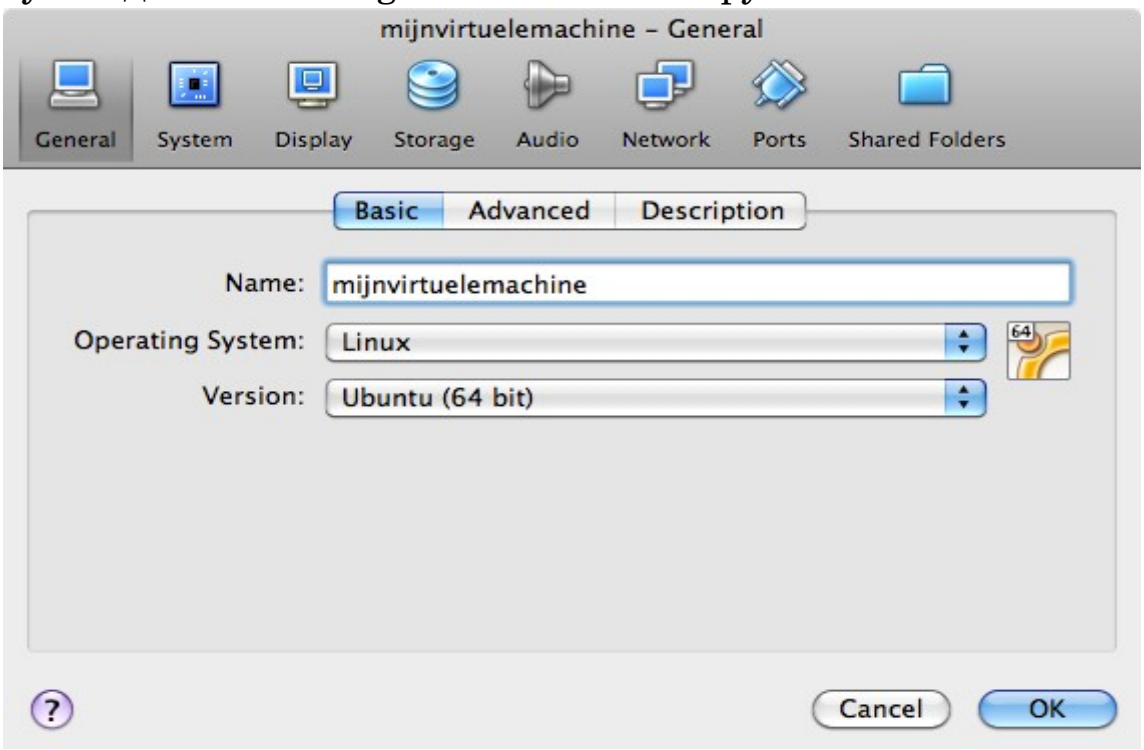


4.4. Присоединение установочного диска ОС

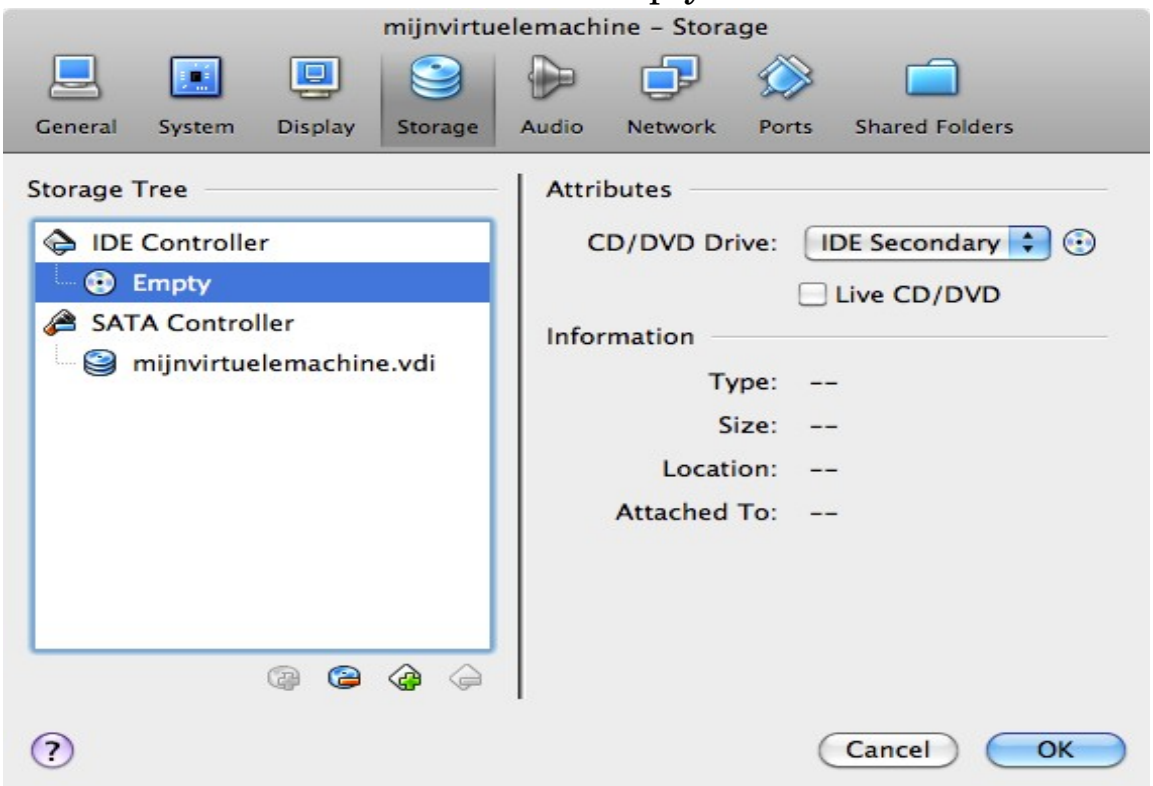
Перед запуском виртуального компьютера следует ознакомиться с некоторыми настройками (нажмите кнопку "Settings" на панели инструментов).



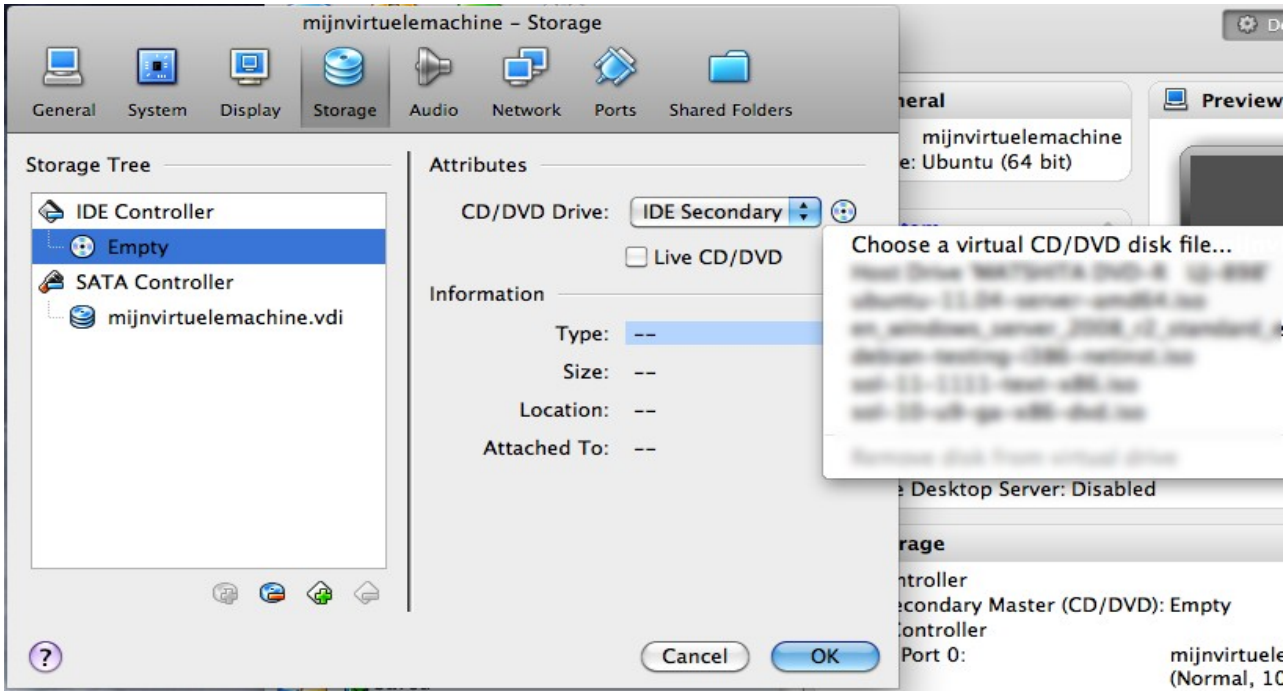
Не следует беспокоиться в том случае, если окно настроек вашего приложения выглядит по-другому, вам нужно всего лишь отыскать кнопку с надписью "Storage" на панели инструментов.



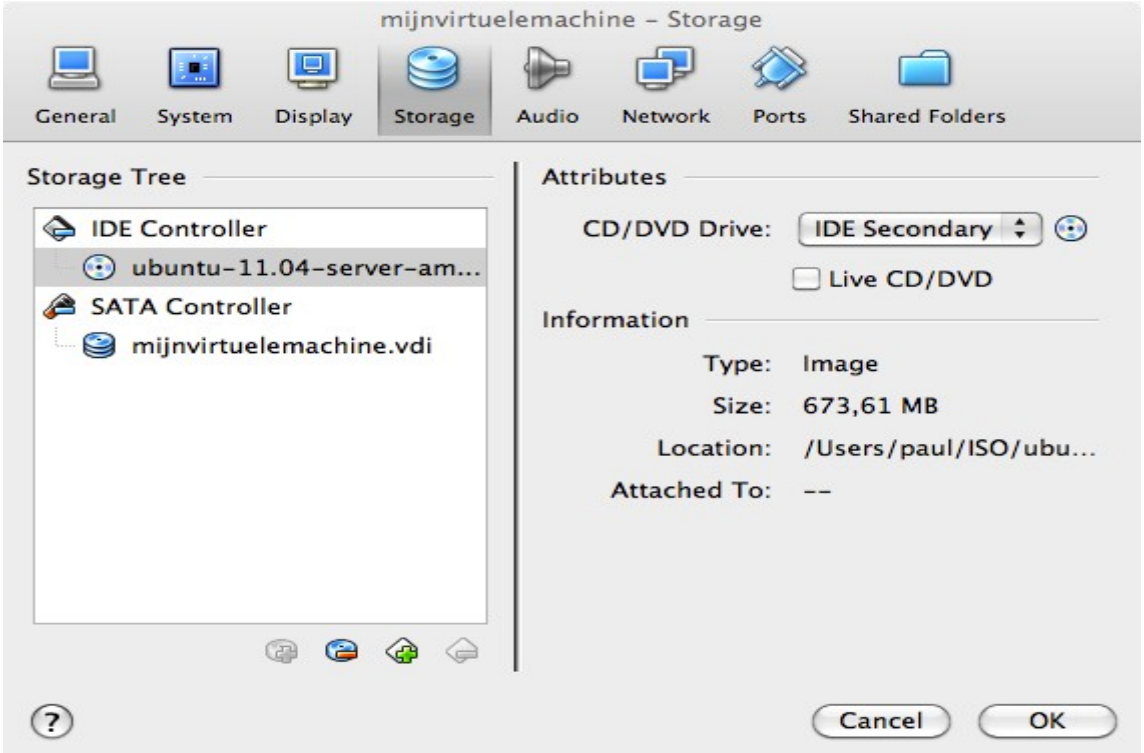
Помните о файле образа установочного диска ОС с расширением .ISO, который вы загрузили в самом начале процесса подготовки рабочего окружения? Для соединения этого файла образа установочного диска ОС с расширением .ISO с настраиваемой виртуальной машиной нажмите на иконку компакт-диска, расположенную рядом с названием элемента списка "Empty".



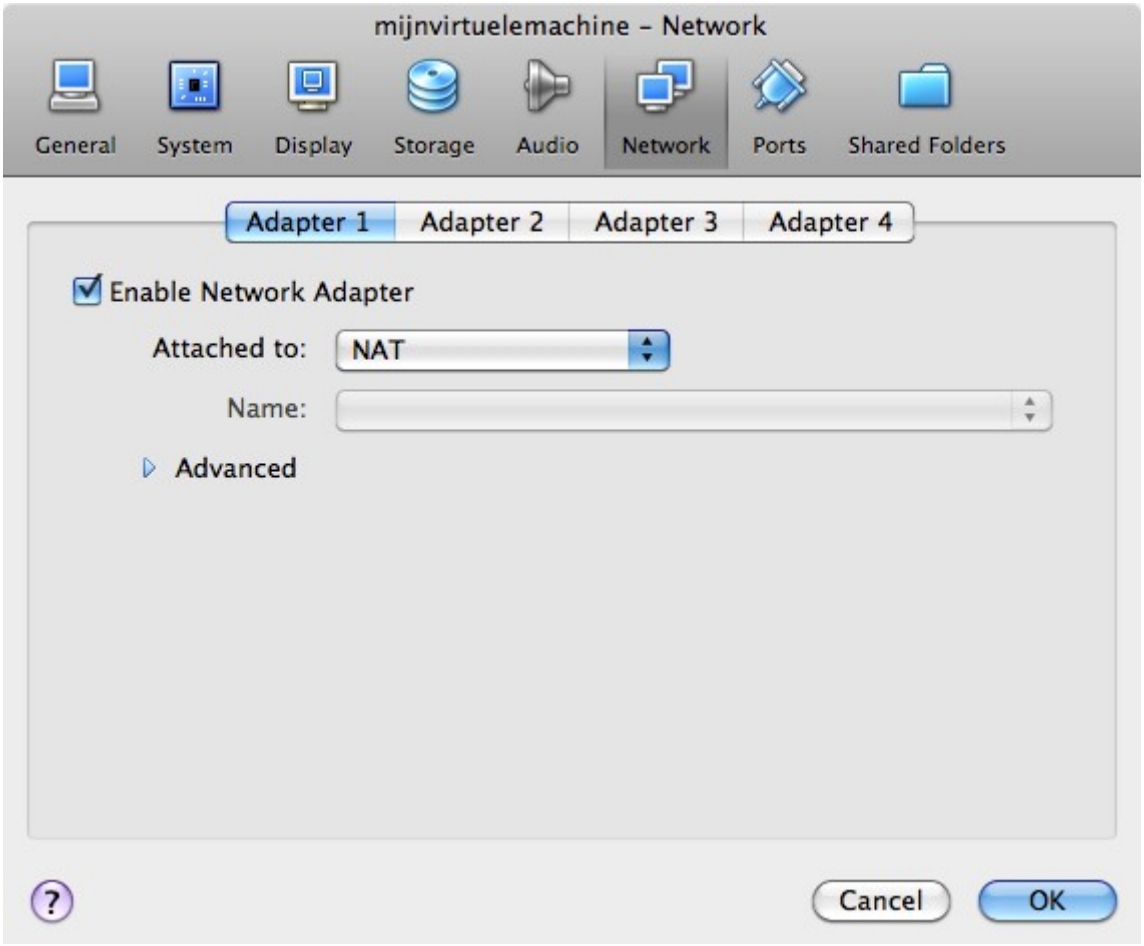
Теперь нажмите на другую иконку компакт-диска и соедините ваш файл образа установочного диска ОС с расширением .ISO с виртуальным приводом для чтения компакт-дисков настраиваемой виртуальной машины.



Убедитесь в том, что загруженный вами образ установочного диска ОС принят приложением для настройки виртуальных машин. В том случае, если приложение Virtualbox сообщает об ошибке на данном этапе настройки, вероятнее всего вы не завершили загрузку данных образа установочного диска ОС (попытайтесь загрузить образ установочного диска ОС снова).



Также может оказаться полезной настройка сетевого адаптера, заключающаяся в переводе последнего в режим моста вместо NAT. При работе виртуального сетевого адаптера режиме моста ваш виртуальный компьютер сможет без лишних сложностей подключиться к сети Интернет.



4.5. Установка Linux

На данном этапе виртуальная машина полностью готова к работе. В процессе загрузки виртуальной машины следует выбрать пункт установки системы и следовать инструкциям, выводимым на экран. По завершении процесса установки вы сможете осуществить вход в систему и начать работать с Linux!

Часть II: Первые шаги в изучении интерфейса командной строки

Глава 5. Страницы руководств man

В данной главе описывается методика использования **man**-страниц (также называемых страницами руководств) при работе с вашим компьютером под управлением Unix или Linux.

Вы узнаете о том, как использовать команду **man**, а также такие связанные с ней команды, как **whereis**, **whatis** и **mandb**.

Большинство файлов и утилит в системах Unix имеет отличные страницы руководств с пояснениями относительно их использования. Страницы руководств становятся особенно полезными в тех случаях, когда вы используете множество различных разновидностей систем Unix или несколько дистрибутивов Linux, так как опции и параметры используемых в них приложений иногда отличаются.

5.1. Команда man $команда

Вы можете ввести команду **man** с последующим именем интересующей команды (для которой вам хотелось бы получить справочную информацию) и начать чтение страницы руководства. Нажмите **q** для выхода из режима чтения страницы руководства. Некоторые страницы руководств содержат примеры (расположенные ближе к концу страницы).

```
paul@laika:~$ man whois

Форматирование страницы whois(1), подождите...
```

5.2. Команда man $имя_файла_конфигурации

Для большинства **файлов конфигурации** имеются отдельные страницы руководств.

```
paul@laika:~$ man syslog.conf
Форматирование страницы syslog.conf(5), подождите...
```

5.3. Команда man $демон

Также данное утверждение справедливо для большинства **демонов** (программ, работающих в фоновом режиме) из вашей системы.

```
paul@laika:~$ man syslogd
Форматирование страницы syslogd(8), подождите...
```

5.4. Команда man -k (apropos)

Команда **man -k** (или **apropos**) позволяет вывести список страниц руководств, содержащих заданную строку.

```
paul@laika:~$ man -k syslog
lm-syslog-setup (8)  - configure laptop mode to switch syslog.conf ...
logger (1)           - a shell command interface to the syslog(3) ...
syslog-facility (8)  - Setup and remove LOCALx facility for sysklogd
syslog.conf (5)      - syslogd(8) configuration file
syslogd (8)          - Linux system logging utilities.
syslogd-listfiles (8) - list system logfiles
```

5.5. Команда whatis

Для ознакомления с описанием страницы руководства следует использовать команду **whatis** с именем интересующей страницы руководства.

```
paul@u810:~$ whatis route
route (8)        - show / manipulate the IP routing table
```

5.6. Команда whereis

Расположение файла страницы руководства в рамках файловой системы может быть определено с помощью команды **whereis**.

```
paul@laika:~$ whereis -m whois
whois: /usr/share/man/man1/whois.1.gz
```

Этот файл может быть непосредственно прочитан с помощью команды **man**.

```
paul@laika:~$ man /usr/share/man/man1/whois.1.gz
```

5.7. Номера справочных разделов

Вы, скорее всего, обратили внимание на числа в круглых скобках. Выполнив команду **man man**, вы можете узнать о том, что эти числа являются номерами справочных разделов. Исполняемые файлы и команды оболочки отнесены к первому справочному разделу.

```
1 Исполняемые программы или команды оболочки (shell)

2 Системные вызовы (функции, предоставляемые ядром)
```

```
3 Библиотечные вызовы (функции, предоставляемые программ-
ными библиотеками)

4 Специальные файлы (обычно находящиеся в каталоге /dev)

5 Форматы файлов и соглашения, например о /etc/passwd

6 Игры

7 Разное (включает пакеты макросов и соглашения), напри-
мер man(7), groff(7)

8 Команды администрирования системы (обычно, запускаемые
только суперпользователем)

9 Процедуры ядра [нестандартный раздел]
```

5.8. Команда man $раздел $файл

Таким образом, при обращении к странице руководства для команды passwd, вы можете обнаружить, что эта страница обозначается как **passwd(1)**; при обращении к странице руководства для файла passwd используется обозначение **passwd(5)**. Приведенные ниже примеры команд иллюстрируют методику открытия страниц руководств из корректных разделов.

```
[paul@RHEL52 ~]$ man passwd      # открывает первую най-
денную страницу руководства
[paul@RHEL52 ~]$ man 5 passwd    # открывает страницу ру-
ководства из раздела 5
```

5.9. Команда man man

Если вы желаете узнать больше о команде **man**, прочитайте это замечательное руководство (Read The Fantastic Manual - RTFM).

К сожалению, на страницах руководств невозможно найти ответы на все вопросы...

```
paul@laika:~$ man woman
Нет справочной страницы для woman
```

5.10. Утилита mandb

Если вы убеждены в том, что страница руководства существует, но при этом вы не можете получить доступ к ней, попробуйте выполнить команду **mandb** в дистрибутиве Debian/Mint.

```
root@laika:~# mandb
```

```
В 0 man-подкаталогах содержатся более новые справочные страницы.
Добавлено 0 справочных страниц.
Добавлено 0 побочных cat-страниц.
Вычищено 0 старых записей базы данных.
```

Или команду **makewhatis** в дистрибутиве CentOS/Redhat.

```
[root@centos65 ~]# apropos scsi
scsi: ничего подходящего не найдено.
[root@centos65 ~]# makewhatis
[root@centos65 ~]# apropos scsi
hpsa                 (4)  - HP Smart Array SCSI driver
lsscsi               (8)  - list SCSI devices (or hosts) and their attributes
sd                   (4)  - Driver for SCSI Disk Drives
st                   (4)  - SCSI tape device
```

Глава 6. Работа с директориями

В данной главе приводится краткий обзор большинства стандартных команд, используемых при работе с директориями: **pwd, cd, ls, mkdir** и **rmdir**. Эти команды доступны при работе с любой системой Linux (или Unix).

Кроме того, в данной главе обсуждаются понятия **абсолютных** и **относительных путей**, а также описывается методика использования механизма **завершения путей** командной оболочки **bash**.

6.1. Команда pwd

С помощью команды **pwd** (расшифровывается как Print Working Directory - вывести информацию о рабочей директории) может быть получена информация о вашем текущем местонахождении в рамках файловой системы. Попробуйте выполнить эту команду: получите доступ к интерфейсу командной строки системы (воспользовавшись одним из приложений со следующими названиями: terminal, console или xterm) и введите команду **pwd**. Командная оболочка выведет путь к вашей **текущей директории**.

```
paul@debian8:~$ pwd
/home/paul
```

6.2. Команда cd

Вы можете изменить вашу текущую директорию с помощью команды **cd** (расшифровывается как Change Directory - изменить директорию).

```
paul@debian8$ cd /etc
paul@debian8$ pwd
/etc
paul@debian8$ cd /bin
paul@debian8$ pwd
/bin
paul@debian8$ cd /home/paul/
paul@debian8$ pwd
/home/paul
```

6.2.1. Команда cd ~

Команда **cd** также может использоваться для быстрого перехода назад в вашу домашнюю директорию. Простое исполнение команды **cd** без задания пути к целевой директории приведет к перемещению в домашнюю директорию. Исполнение команды **cd ~** приведет к аналогичному эффекту.

```
paul@debian8$ cd /etc
paul@debian8$ pwd
/etc
paul@debian8$ cd
paul@debian8$ pwd
/home/paul
paul@debian8$ cd ~
paul@debian8$ pwd
/home/paul
```

6.2.2. Команда cd ..

Для перехода в **родительскую директорию** (ту директорию, которая находится над вашей текущей директорией в дереве директорий) следует использовать команду **cd ..** .

```
paul@debian8$ pwd
/usr/share/games
```

```
paul@debian8$ cd ..
paul@debian8$ pwd
/usr/share
```

Для того, чтобы остаться в текущей директории, просто введите команду ***cd .*** *;-)* Позднее мы все же познакомимся с практическим примером использования символа ., представляющего текущую директорию.

6.2.3. Команда cd -

Другой полезный вариант использования команды **cd** заключается в выполнении простой команды **cd -** для перехода в предыдущую директорию.

```
paul@debian8$ pwd
/home/paul
paul@debian8$ cd /etc
paul@debian8$ pwd
/etc
paul@debian8$ cd -
/home/paul
paul@debian8$ cd -
/etc
```

6.3. Абсолютные и относительные пути

Вы должны иметь представление об **абсолютных и относительных путях** в рамках дерева директорий файловой системы. Если вы вводите путь, начинающийся с символа **слэша** (**/**), подразумевается, что путь будет указан относительно корневой директории файловой системы. Если же вы не начинаете ввод пути с символа слэша, подразумевается, что точкой отсчета будет текущая директория.

В примере ниже показано, что текущей директорией является директория **/home/paul**. Для перехода из этой директории в директорию **/home** вам придется ввести команду **cd /home** вместо команды **cd home**.

```
paul@debian8$ pwd
/home/paul
paul@debian8$ cd home
bash: cd: home: Нет такого файла или каталога
```

```
paul@debian8$ cd /home
paul@debian8$ pwd
/home
```

При нахождении в директории **/home** вам придется ввести команду **cd paul** вместо команды **cd /paul** для перехода в поддиректорию **paul** текущей директории **/home**.

```
paul@debian8$ pwd
/home
paul@debian8$ cd /paul
bash: cd: /paul: Нет такого файла или каталога
paul@debian8$ cd paul
paul@debian8$ pwd
/home/paul
```

В том же случае, если вашей текущей директорией является **корневая директория** /, то и команда **cd /home**, и команда **cd home** позволят вам переместиться в директорию **/home**.

```
paul@debian8$ pwd
/
paul@debian8$ cd home
paul@debian8$ pwd
/home
paul@debian8$ cd /
paul@debian8$ cd /home
paul@debian8$ pwd
/home
```

Этот пример был последним примером, в котором используются команды **pwd**. С этого момента информация о текущей директории будет всегда выводиться в рамках приглашения командной оболочки. Далее в данной книге будут даны пояснения относительно того, как следует использовать переменную командной оболочки **$PS1** для вывода описанной информации.

6.4. Завершение путей

Клавиша Tab может помочь избежать ошибок при вводе путей. Нажатие **клавиши Tab** после ввода части команды **cd /et** приведет к завершению команды до вида **cd /etc/**. При этом нажатие **кла-**

виши Tab после ввода команды **cd /Et** не приведет ни к чему, так как была допущена ошибка при вводе части **пути** (использована буква E в верхнем регистре).

Вам придется нажимать меньше клавиш в случае использования **клавиши Tab**, к тому же, при использовании этой клавиши вы можете быть уверены в том, что введенный **путь** является корректным!

6.5. Утилита ls

Вы можете вывести список содержимого директории с помощью утилиты **ls**.

```
paul@debian8:~$ ls

allfiles.txt  dmesg.txt  services   stuff  summer.txt

paul@debian8:~$
```

6.5.1. Команда ls -a

Часто используемым параметром утилиты **ls** является параметр **-a**, который предназначен для вывода информации обо всех файлах. Под выводом информации обо всех файлах подразумевается вывод информации в том числе и о **скрытых файлах**. В том случае, если имя файла в рамках файловой системы Linux начинается с символа точки, он считается **скрытым файлом** и не включается в обычные списки содержимого директорий.

```
paul@debian8:~$ ls
allfiles.txt  dmesg.txt  services  stuff  summer.txt
paul@debian8:~$ ls -a
.   allfiles.txt   .bash_profile  dmesg.txt   .lesshst
stuff
..  .bash_history  .bashrc        services    .ssh
summer.txt
paul@debian8:~$
```

6.5.2. Команда ls -l

Вам придется многократно использовать параметры утилиты **ls** для вывода информации о содержимом директории в различных форматах или для вывода информации о различных файлах из директории. Команда **ls** без параметров позволяет получить список

файлов, расположенных в директории. Команда **ls -l** (в качестве параметра использована строчная буква L, а не число 1) позволяет получить более подробный список файлов.

```
paul@debian8:~$ ls -l
итого 17296
-rw-r--r-- 1 paul paul 17584442 сен 17 00:03 allfiles.txt
-rw-r--r-- 1 paul paul    96650 сен 17 00:03 dmesg.txt
-rw-r--r-- 1 paul paul    19558 сен 17 00:04 services
drwxr-xr-x 2 paul paul     4096 сен 17 00:04 stuff
-rw-r--r-- 1 paul paul        0 сен 17 00:04 summer.txt
```

6.5.3. Команда ls -lh

Другим периодически используемым параметром утилиты **ls** является параметр **-h**. Он позволяет выводить числовые значения (соответствующие размерам файлов) в формате, лучше читаемом человеком. Также в примере ниже показаны варианты передачи параметров утилите **ls**. Позднее в данной книге будут даны подробные пояснения относительно выводимых данных.

Обратите внимание на то, что мы используем строчную букву L, а не число 1 в качестве параметра утилиты в данном примере.

```
paul@debian8:~$ ls -l -h
итого 17M
-rw-r--r-- 1 paul paul  17M сен 17 00:03 allfiles.txt
-rw-r--r-- 1 paul paul  95K сен 17 00:03 dmesg.txt
-rw-r--r-- 1 paul paul  20K сен 17 00:04 services
drwxr-xr-x 2 paul paul 4.0K сен 17 00:04 stuff
-rw-r--r-- 1 paul paul    0 сен 17 00:04 summer.txt
paul@debian8:~$ ls -lh
итого 17M
-rw-r--r-- 1 paul paul  17M сен 17 00:03 allfiles.txt
-rw-r--r-- 1 paul paul  95K сен 17 00:03 dmesg.txt
-rw-r--r-- 1 paul paul  20K сен 17 00:04 services
drwxr-xr-x 2 paul paul 4.0K сен 17 00:04 stuff
-rw-r--r-- 1 paul paul    0 сен 17 00:04 summer.txt
paul@debian8:~$ ls -hl
```

```
итого 17M
-rw-r--r-- 1 paul paul  17M сен 17 00:03 allfiles.txt
-rw-r--r-- 1 paul paul  95K сен 17 00:03 dmesg.txt
-rw-r--r-- 1 paul paul  20K сен 17 00:04 services
drwxr-xr-x 2 paul paul 4.0K сен 17 00:04 stuff
-rw-r--r-- 1 paul paul    0 сен 17 00:04 summer.txt
paul@debian8:~$ ls -h -l
итого 17M
-rw-r--r-- 1 paul paul  17M сен 17 00:03 allfiles.txt
-rw-r--r-- 1 paul paul  95K сен 17 00:03 dmesg.txt
-rw-r--r-- 1 paul paul  20K сен 17 00:04 services
drwxr-xr-x 2 paul paul 4.0K сен 17 00:04 stuff
-rw-r--r-- 1 paul paul    0 Sep 17 00:04 summer.txt
paul@debian8:~$
```

6.6. Утилита mkdir

Обход дерева директорий файловой системы Unix является интересным занятием, но еще больший интерес представляет создание ваших собственных директорий с помощью утилиты **mkdir**. Вам придется передавать как минимум один параметр утилите **mkdir**, а именно, имя новой директории, которая должна быть создана. При этом следует серьезно задумываться перед использованием начального символа / в именах директорий.

```
paul@debian8:~$ mkdir mydir
paul@debian8:~$ cd mydir
paul@debian8:~/mydir$ ls -al
итого 8
drwxr-xr-x  2 paul paul 4096 сен 17 00:07 .
drwxr-xr-x 48 paul paul 4096 сен 17 00:07 ..
paul@debian8:~/mydir$ mkdir stuff
paul@debian8:~/mydir$ mkdir otherstuff
paul@debian8:~/mydir$ ls -l
итого 8
drwxr-xr-x 2 paul paul 4096 сен 17 00:08 otherstuff
drwxr-xr-x 2 paul paul 4096 сен 17 00:08 stuff
paul@debian8:~/mydir$
```

6.6.1. Команда mkdir -p

Исполнение следующей команды закончится неудачей, так как **родительской директории** для директории **threedirsdeep** не существует.

```
paul@debian8:~$ mkdir mydir2/mysubdir2/threedirsdeep
mkdir: невозможно создать каталог
"mydir2/mysubdir2/threedirsdeep": Нет такого файла или
каталога
```

В случае использования параметра **-p** утилиты **mkdir** при необходимости будут создаваться **родительские директории**.

```
paul@debian8:~$ mkdir -p mydir2/mysubdir2/threedirsdeep
paul@debian8:~$ cd mydir2
paul@debian8:~/mydir2$ ls -l
итого 4
drwxr-xr-x 3 paul paul 4096 сен 17 00:11 mysubdir2
paul@debian8:~/mydir2$ cd mysubdir2
paul@debian8:~/mydir2/mysubdir2$ ls -l
итого 4
drwxr-xr-x 2 paul paul 4096 сен 17 00:11 threedirsdeep
paul@debian8:~/mydir2/mysubdir2$ cd threedirsdeep/
paul@debian8:~/mydir2/mysubdir2/threedirsdeep$ pwd
/home/paul/mydir2/mysubdir2/threedirsdeep
```

6.7. Утилита rmdir

В том случае, если директория пуста, вы можете использовать утилиту rmdir для удаления этой директории.

```
paul@debian8:~/mydir$ ls -l
итого 8
drwxr-xr-x 2 paul paul 4096 сен 17 00:08 otherstuff
drwxr-xr-x 2 paul paul 4096 сен 17 00:08 stuff
paul@debian8:~/mydir$ rmdir otherstuff
paul@debian8:~/mydir$ cd ..
paul@debian8:~$ rmdir mydir
rmdir: не удалось удалить "mydir": Каталог не пуст
paul@debian8:~$ rmdir mydir/stuff
```

```
paul@debian8:~$ rmdir mydir
paul@debian8:~$
```

6.7.1. Команда rmdir -p

И по аналогии с параметром **mkdir -p**, вы также можете использовать утилиту **rmdir** для рекурсивного удаления директорий.

```
paul@debian8:~$ mkdir -p test42/subdir
paul@debian8:~$ rmdir -p test42/subdir
paul@debian8:~$
```

6.8. Практическое задание: работа с директориями

1. Выведите путь к вашей текущей директории.

2. Перейдите в директорию /etc.

3. А теперь перейдите в вашу домашнюю директорию с помощью ровно трех нажатий клавиш.

4. Перейдите в директорию /boot/grub с помощью ровно одиннадцати нажатий клавиш.

5. Перейдите в родительскую директорию для текущей директории.

6. Перейдите в корневую директорию.

7. Выведите список содержимого корневой директории.

8. Выведите подробный список содержимого корневой директории.

9. Оставаясь в текущей рабочей директории, выведите список содержимого директории /etc.

10. Оставаясь в текущей директории, выведите список содержимого директорий /bin и /sbin.

11. Оставаясь в текущей директории, выведите список содержимого директории ~.

12. Выведите список всех файлов (включая скрытые файлы), находящихся в вашей домашней директории.

13. Выведите список файлов, находящихся в директории `/boot`, с использованием формата величин для облегчения чтения человеком.

14. Создайте директорию `testdir` в вашей домашней директории.

15. Перейдите в директорию `/etc` и, оставаясь в ней, создайте директорию newdir в вашей домашней директории.

16. Создайте с помощью одной команды директории `~/dir1/dir1/dir2/dir3` (директория `dir3` является поддиректорией директории `dir2`, а директория `dir2` - поддиректорией директории `dir1`).

17. Удалите директорию `testdir`.

18. Если позволяет время (или вы ждете момента, когда остальные студенты закончат выполнение данного практического задания), попытайтесь воспользоваться командами **`pushd`** и **`popd`** и понять принцип их работы. Обратитесь к странице руководства `man` для командной оболочки **bash** с целью поиска информации о данных командах.

6.9. Корректная процедура выполнения практического задания: работа с директориями

1. Выведите путь к вашей текущей директории.

```
pwd
```

2. Перейдите в директорию `/etc`.

```
cd /etc
```

3. А теперь перейдите в вашу домашнюю директорию с помощью ровно трех нажатий клавиш.

```
cd (и нажатие клавиши Enter)
```

4. Перейдите в директорию `/boot/grub` с помощью ровно одиннадцати нажатий клавиш.

```
cd /boot/grub (Используйте клавишу Tab)
```

5. Перейдите в родительскую директорию для текущей директории.

`cd .. (Между cd и .. должен быть пробел)`

6. Перейдите в корневую директорию.

`cd /`

7. Выведите список содержимого корневой директории.

`ls`

8. Выведите подробный список содержимого корневой директории.

`ls -l`

9. Оставаясь в текущей рабочей директории, выведите список содержимого директории `/etc.`

`ls /etc`

10. Оставаясь в текущей директории, выведите список содержимого директорий `/bin` и `/sbin`.

`ls /bin /sbin`

11. Оставаясь в текущей директории, выведите список содержимого директории ~.

`ls ~`

12. Выведите список всех файлов (включая скрытые файлы), находящихся в вашей домашней директории.

`ls -al ~`

13. Выведите список файлов, находящихся в директории `/boot`, с использованием формата величин для облегчения чтения человеком.

`ls -lh /boot`

14. Создайте директорию `testdir` в вашей домашней директории.

`mkdir ~/testdir`

15. Перейдите в директорию `/etc` и, оставаясь в ней, создайте директорию newdir в вашей домашней директории.

`cd /etc ; mkdir ~/newdir`

16. Создайте с помощью одной команды директории `~/dir1/dir1/dir2/dir3` (директория `dir3` является поддиректорией

директории dir2, а директория dir2 - поддиректорией директории dir1).

```
mkdir -p ~/dir1/dir2/dir3
```

17. Удалите директорию testdir.

```
rmdir testdir
```

18. Если позволяет время (или вы ждете момента, когда остальные студенты закончат выполнение данного практического задания), попытайтесь воспользоваться командами **pushd** и **popd** и понять принцип их работы. Обратитесь к странице руководства man для командной оболочки **bash** с целью поиска информации о данных командах.

```
man bash            # открытие страницы руководства
/pushd              # поиск раздела с описанием команды pushd
n                   # переход к следующему разделу (повторите это действие два/три раза)
```

Командная оболочка Bash поддерживает две встроенных команды с именами **pushd** и **popd**. Обе команды предназначены для работы со стандартным стеком посещенных ранее директорий. Команда pushd позволяет поместить директорию в стек и переместиться в новую текущую директорию, команда popd удаляет директорию из стека и устанавливает предыдущую текущую директорию.

```
paul@debian7:/etc$ cd /bin
paul@debian7:/bin$ pushd /lib
/lib /bin
paul@debian7:/lib$ pushd /proc
/proc /lib /bin
paul@debian7:/proc$ popd
/lib /bin
paul@debian7:/lib$ popd
/bin
```

Глава 7. Работа с файлами

Из данной главы вы узнаете о том, как определять тип файлов, создавать, удалять, копировать и перемещать файлы с помощью таких утилит, как **file, touch, rm, cp, mv** и **rename**.

7.1. Все имена файлов регистрозависимы

Имена файлов в Linux (или в любой системе Unix) являются **регистрозависимыми**. Это означает, что имена **FILE1** и **file1** не являются равноценными, а путь к файлу **/etc/hosts** отличен от пути к файлу **/etc/Hosts** (последнего файла не должно существовать в файловой системе типичного компьютера, работающего под управлением Linux).

В данном примере продемонстрировано различие между двумя файлами, причем имя первого файла начинается с заглавной буквы **W**, а второго - с прописной буквы **w**.

```
paul@laika:~/Linux$ ls
winter.txt  Winter.txt
paul@laika:~/Linux$ cat winter.txt
Холодно.
paul@laika:~/Linux$ cat Winter.txt
Очень холодно!
```

7.2. Все является файлом

Любая **директория** на самом деле является **файлом** (с регистрозависмым именем!), хотя этот **файл** и имеет специальный тип. Каждый терминал (например, **/dev/pts/4**), любой жесткий диск или раздел на нем (например, **/dev/sdb1**) и любой процесс представлены где-либо в рамках ф**айловой системы** с помощью **файла**. После изучения данной главы у вас не останется сомнений в том, что в Linux все является файлом.

7.3. Утилита file

Утилита **file** предназначена для определения типа файла. В Linux для определения типов файлов не используются их расширения. Для инструментов с интерфейсом командной строки абсолютно безразлично, оканчивается ли имя файла на .txt или на .pdf. Исполняя обязанности системного администратора, вы должны использовать команду **file** для установления типа интересующего вас файла.

Ниже приведено несколько примеров использования данной утилиты при работе с типичной системой Linux.

```
paul@laika:~$ file pic33.png
pic33.png: PNG image data, 3840 x 1200, 8-bit/color RGBA, non-interlaced
paul@laika:~$ file /etc/passwd
/etc/passwd: ASCII text
paul@laika:~$ file HelloWorld.c
HelloWorld.c: ASCII C program text
```

Утилита **file** использует файл со списком "магических последовательностей байт", содержащий шаблоны для распознавания типов данных. Файл со списком "магических последовательностей байт" расположен по пути **/usr/share/file/magic**. Используйте команду **man 5 magic** в случае необходимости получения дополнительной информации о нем.

Важно отметить, что для работы с такими специальными файлами, как файлы из директорий **/dev** и **/proc**, следует использовать команду **file -s**.

```
root@debian6~# file /dev/sda
/dev/sda: block special
root@debian6~# file -s /dev/sda
/dev/sda: x86 boot sector; partition 1: ID=0x83, active, starthead...
root@debian6~# file /proc/cpuinfo
/proc/cpuinfo: empty
root@debian6~# file -s /proc/cpuinfo
/proc/cpuinfo: ASCII C++ program text
```

7.4. Утилита touch

7.4.1. Создание пустого файла

Один из простых способов создания пустого фала заключается в использовании утилиты **touch**. (Позднее в данной книге мы рассмотрим множество других способов создания файлов.)

Данный пример начинается с открытия пустой директории, в которой с помощью утилиты **touch** создаются два файла, после чего выводится список созданных файлов.

```
paul@debian7:~$ ls -l
```

```
итого 0
paul@debian7:~$ touch file42
paul@debian7:~$ touch file33
paul@debian7:~$ ls -l
итого 0
-rw-r--r-- 1 paul paul 0 окт 15 08:57 file33
-rw-r--r-- 1 paul paul 0 окт 15 08:56 file42
paul@debian7:~$
```

7.4.2. Команда touch -t

Утилита **touch** позволяет устанавливать набор значений свойств пустых файлов в процессе их создания. Удастся ли вам определить, какие значения свойств были установлены, ознакомившись с приведенным ниже примером? Если вы не смогли разобраться самостоятельно, обратитесь к странице руководства для утилиты **touch**.

```
paul@debian7:~$ touch -t 200505050000 SinkoDeMayo
paul@debian7:~$ touch -t 130207111630 BigBattle.txt
paul@debian7:~$ ls -l
total 0
-rw-r--r-- 1 paul paul 0 июл 11  1302 BigBattle.txt
-rw-r--r-- 1 paul paul 0 окт 15 08:57 file33
-rw-r--r-- 1 paul paul 0 окт 15 08:56 file42
-rw-r--r-- 1 paul paul 0 май  5  2005 SinkoDeMayo
paul@debian7:~$
```

7.5. rm

7.5.1. Удаление файлов навсегда

В том случае, если вам больше не нужен файл, следует использовать утилиту **rm** для его удаления. В отличие от некоторых графических пользовательских интерфейсов, интерфейс командной строки системы не предполагает использования таких хранилищ файлов, как **корзина**, предназначенных для последующего восстановления удаленных файлов. Если вы удалили файл с помощью утилиты **rm**, этот файл будет удален навсегда. Исходя из этого, следует проявлять осторожность при удалении файлов!

```
paul@debian7:~$ ls
BigBattle.txt  file33  file42  SinkoDeMayo
```

```
paul@debian7:~$ rm BigBattle.txt
paul@debian7:~$ ls
file33  file42  SinkoDeMayo
paul@debian7:~$
```

7.5.2. Команда rm -i

Для того, чтобы застраховаться от случайного удаления файла, вы можете использовать команду **rm -i**.

```
paul@debian7:~$ ls
file33  file42  SinkoDeMayo
paul@debian7:~$ rm -i file33
rm: удалить пустой обычный файл "file33"? yes
paul@debian7:~$ rm -i SinkoDeMayo
rm: удалить пустой обычный файл "SinkoDeMayo"? n
paul@debian7:~$ ls
file42  SinkoDeMayo
paul@debian7:~$
```

7.5.3. Команда rm -rf

По умолчанию с помощью команды **rm -r** невозможно удалить непустые директории. Однако, утилита **rm** принимает параметры, которые позволят вам удалять любую директорию. Команда **rm -rf** известна благодаря тому, что она позволяет удалять все элементы файловой системы (конечно же, в том случае, если вы имеете достаточные права для выполнения этой операции). В том случае, если вы вошли в систему, воспользовавшись учетной записью пользователя root, следует проявить особую осторожность при использовании команды **rm -rf** (параметр **f** расшифровывается как **force** (принудительное выполнение операции), а параметр **r** - как **recursive** (рекурсивный обход директорий)), так как в случае работы с учетной записью пользователя root, упомянутые ограничения прав не будут распространяться на вас. Вы сможете в буквальном смысле удалить все файлы вашей системы по неосторожности.

```
paul@debian7:~$ mkdir test
paul@debian7:~$ rm test
rm: невозможно удалить "test": Это каталог
paul@debian7:~$ rm -rf test
paul@debian7:~$ ls test
```

```
ls: невозможно получить доступ к test: Нет такого файла
или каталога
paul@debian7:~$
```

7.6. Утилита cp

7.6.1. Копирование отдельных файлов

Для копирования файла следует использовать утилиту **cp** с аргументами, представленными путями к исходному и целевому файлам.

```
paul@debian7:~$ ls
file42  SinkoDeMayo
paul@debian7:~$ cp file42 file42.copy
paul@debian7:~$ ls
file42  file42.copy  SinkoDeMayo
```

7.6.2. Копирование файлов в другую директорию

В том случае, если в качестве пути к целевому файлу используется путь к директории, исходные файлы будут скопированы в эту целевую директорию.

```
paul@debian7:~$ mkdir dir42
paul@debian7:~$ cp SinkoDeMayo dir42
paul@debian7:~$ ls dir42/
SinkoDeMayo
```

7.6.3. Команда cp -r

Для копирования директорий целиком следует использовать команду **cp -r** (параметр `-r` позволяет осуществлять рекурсивное копирование всех файлов из всех поддиректорий).

```
paul@debian7:~$ ls
dir42  file42  file42.copy  SinkoDeMayo
paul@debian7:~$ cp -r dir42/ dir33
paul@debian7:~$ ls
dir33  dir42  file42  file42.copy  SinkoDeMayo
paul@debian7:~$ ls dir33/
SinkoDeMayo
```

7.6.4. Копирование множества файлов в директорию

Вы также можете использовать утилиту **cp** для копирования множества файлов в одну директорию. В этом случае последний аргумент (аргумент, указывающий на цель) должен быть представлен путем к директории.

```
paul@debian7:~$ cp file42 file42.copy SinkoDeMayo dir42/
paul@debian7:~$ ls dir42/
file42  file42.copy  SinkoDeMayo
```

7.6.5. Команда cp -i

Для предотвращения перезаписи существующих файлов в ходе использования утилиты **cp** следует использовать параметр **-i** (для активации интерактивного режима копирования).

```
paul@debian7:~$ cp SinkoDeMayo file42
paul@debian7:~$ cp SinkoDeMayo file42
paul@debian7:~$ cp -i SinkoDeMayo file42
cp: переписать "file42"? n
paul@debian7:~$
```

7.7. Утилита mv

7.7.1. Переименование файлов с помощью утилиты mv

Утилита **mv** используется для переименования файлов или для перемещения файлов в другие директории.

```
paul@debian7:~$ ls
dir33  dir42  file42  file42.copy  SinkoDeMayo
paul@debian7:~$ mv file42 file33
paul@debian7:~$ ls
dir33  dir42  file33  file42.copy  SinkoDeMayo
paul@debian7:~$
```

В том случае, если вам необходимо переименовать один файл, утилита **mv** является предпочтительным инструментом.

7.7.2. Переименование директорий с помощью утилиты mv

Эта же утилита mv может быть использована и для переименования директорий.

```
paul@debian7:~$ ls -l
итого 8
```

```
drwxr-xr-x 2 paul paul 4096 окт 15 09:36 dir33
drwxr-xr-x 2 paul paul 4096 окт 15 09:36 dir42
-rw-r--r-- 1 paul paul    0 окт 15 09:38 file33
-rw-r--r-- 1 paul paul    0 окт 15 09:16 file42.copy
-rw-r--r-- 1 paul paul    0 май  5  2005 SinkoDeMayo
paul@debian7:~$ mv dir33 backup
paul@debian7:~$ ls -l
итого 8
drwxr-xr-x 2 paul paul 4096 окт 15 09:36 backup
drwxr-xr-x 2 paul paul 4096 окт 15 09:36 dir42
-rw-r--r-- 1 paul paul    0 окт 15 09:38 file33
-rw-r--r-- 1 paul paul    0 окт 15 09:16 file42.copy
-rw-r--r-- 1 paul paul    0 май  5  2005 SinkoDeMayo
paul@debian7:~$
```

7.7.3. Команда mv -i

Утилита **mv** поддерживает параметр **-i** по аналогии с утилитами cp и rm.

В примере ниже показано, как при использовании команды **mv -i** запрашивается подтверждение перезаписи существующего файла.

```
paul@debian7:~$ mv -i file33 SinkoDeMayo
mv: переписать "SinkoDeMayo"? no
paul@debian7:~$
```

7.8. Утилита rename

7.8.1. Об утилите rename

Утилита **rename** является одним из редких случаев, когда в рамках книги "Фундаментальные основы Linux" приходится делать разделение между дистрибутивами Linux. Практически каждая из описанных в данной книге утилит работает практически на каждом компьютере под управлением Linux. Но реализации утилиты **rename** в различных дистрибутивах отличаются.

Следует пытаться использовать утилиту **mv** всегда, когда вам необходимо переименовать несколько файлов.

7.8.2. Утилита rename в дистрибутиве Debian/Ubuntu

Утилита **rename** в дистрибутиве Debian имеет сложный синтаксис (используются регулярные выражения) для единовременного переименования множества файлов.

Ниже приведен пример использования утилиты **rename**, с помощью которой у всех файлов с расширением .txt производится замена расширения на .png.

```
paul@debian7:~/test42$ ls
abc.txt  file33.txt  file42.txt
paul@debian7:~/test42$ rename 's/\.txt/\.png/' *.txt
paul@debian7:~/test42$ ls
abc.png  file33.png  file42.png
```

Во втором примере производится замена всех (первых) вхождений слов "**file**" на "**document**" во всех файлах с расширением .png.

```
paul@debian7:~/test42$ ls
abc.png  file33.png  file42.png
paul@debian7:~/test42$ rename 's/file/document/' *.png
paul@debian7:~/test42$ ls
abc.png  document33.png  document42.png
paul@debian7:~/test42$
```

7.8.3. Утилита rename в дистрибутиве CentOS/RHEL/Fedora

В Red Hat Enterprise Linux синтаксис утилиты **rename** немного отличается. В первом примере, представленном ниже, осуществляется переименование всех файлов с расширением .conf (*.conf) путем замены всех вхождений .conf на .backup.

```
[paul@centos7 ~]$ touch one.conf two.conf three.conf
[paul@centos7 ~]$ rename .conf .backup *.conf
[paul@centos7 ~]$ ls
one.backup  three.backup  two.backup
[paul@centos7 ~]$
```

Во втором примере производится переименование всех файлов (*) путем замены всех вхождений one на ONE.

```
[paul@centos7 ~]$ ls
one.backup  three.backup  two.backup
[paul@centos7 ~]$ rename one ONE *
```

```
[paul@centos7 ~]$ ls
ONE.backup  three.backup  two.backup
[paul@centos7 ~]$
```

7.9. Практическое задание: работа с файлами

1. Выведите список файлов директории /bin.

2. Выведите информацию о типах файлов /bin/cat, /etc/passwd и /usr/bin/passwd.

3a. Загрузите файлы wolf.jpg и LinuxFun.pdf с ресурса http://linux-training.be (с помощью команд wget http://linux-training.be/files/studentfiles/wolf.jpg и wget http://linux-training.be/files/books/LinuxFun.pdf)

```
wget http://linux_training.be/files/studentfiles/wolf.gpj

wget http://linux_training.be/files/studentfiles/wolf.png

wget http://linux_training.be/files/books/LinuxFun.pdf
```

3b. Выведите информацию о типах файлов wolf.jpg и LinuxFun.pdf.

3c. Переименуйте файл wolf.jpg в wolf.pdf (с помощью команды mv).

3d. Выведите информацию о типах файлов wolf.pdf и LinuxFun.pdf.

4. Создайте директорию ~/touched и перейдите в нее.

5. Создайте файлы today.txt и yesterday.txt в директории touched.

6. Измените дату создания файла yesterday.txt таким образом, чтобы она совпадала с датой прошлого дня.

7. Создайте копию файла yesterday.txt с именем copy.yesterday.txt.

8. Переименуйте файл copy.yesterday.txt в kim.

9. Создайте директорию с именем ~/testbackup и скопируйте все файлы из директории ~/touched в нее.

10. Используйте одну команду для удаления директории ~/testbackup и всех файлов из нее.

11. Создайте директорию ~/etcbackup и скопируйте файлы с расширением .conf (*.conf) из директории /etc в нее. Проверьте, был ли осуществлен обход всех поддиректорий директории /etc?

12. Используйте утилиту rename для переименования всех файлов с расширением .conf (*.conf) таким образом, чтобы расширения всех этих файлов изменились на .backup. (Если у вас есть возможность работы более чем со одним дистрибутивом, попробуйте выполнить эту операцию во всех этих дистрибутивах!).

7.10. Корректная процедура выполнения практического задания: работа с файлами

1. Выведите список файлов директории /bin.

```
ls /bin
```

2. Выведите информацию о типах файлов /bin/cat, /etc/passwd и /usr/bin/passwd.

```
file /bin/cat /etc/passwd /usr/bin/passwd
```

3a. Загрузите файлы wolf.jpg и LinuxFun.pdf с ресурса http://linux-training.be (с помощью команд `wget http://linux-training.be/files/studentfiles/wolf.jpg` и `wget http://linux-training.be/files/books/LinuxFun.pdf`)

```
wget http://linux-training.be/files/studentfiles/wolf.jpg
wget http://linux-training.be/files/books/LinuxFun.pdf
```

3b. Выведите информацию о типах файлов wolf.jpg и LinuxFun.pdf.

```
file wolf.jpg LinuxFun.pdf
```

3c. Переименуйте файл wolf.jpg в wolf.pdf (с помощью команды mv).

```
mv wolf.jpg wolf.pdf
```

3d. Выведите информацию о типах файлов wolf.pdf и LinuxFun.pdf.

```
file wolf.pdf LinuxFun.pdf
```

4. Создайте директорию ~/touched и перейдите в нее.

```
mkdir ~/touched ; cd ~/touched
```

5. Создайте файлы today.txt и yesterday.txt в директории touched.

```
touch today.txt yesterday.txt
```

6. Измените дату создания файла yesterday.txt таким образом, чтобы она совпадала с датой прошлого дня.

```
touch -t 200810251405 yesterday.txt (замените 20081025 на дату, соответствующую вчерашнему дню)
```

7. Создайте копию файла yesterday.txt с именем copy.yesterday.txt.

```
cp yesterday.txt copy.yesterday.txt
```

8. Переименуйте файл copy.yesterday.txt в kim.

```
mv copy.yesterday.txt kim
```

9. Создайте директорию с именем ~/testbackup и скопируйте все файлы из директории ~/touched в нее.

```
mkdir ~/testbackup ; cp -r ~/touched ~/testbackup/
```

10. Используйте одну команду для удаления директории ~/testbackup и всех файлов из нее.

```
rm -rf ~/testbackup
```

11. Создайте директорию ~/etcbackup и скопируйте файлы с расширением .conf (*.conf) из директории /etc в нее. Проверьте, был ли осуществлен обход всех поддиректорий директории /etc?

```
cp -r /etc/*.conf ~/etcbackup
```

```
Будут скопированы исключительно файлы с расширением .conf (*.conf), находящиеся непосредственно в директории /etc/.
```

12. Используйте утилиту rename для переименования всех файлов с расширением .conf (*.conf) таким образом, чтобы расширения всех этих файлов изменились на .backup. (Если у вас есть возможность работы более чем со одним дистрибутивом, попробуйте выполнить эту операцию во всех этих дистрибутивах!).

```
В дистрибутиве RHEL: touch 1.conf 2.conf ; rename conf backup *.conf
```

```
В дистрибутиве Debian: touch 1.conf 2.conf ; rename
's/conf/backup/' *.conf
```

Глава 8. Работа с содержимым файлов

В данной главе мы будем работать с содержимым **текстовых файлов**, используя утилиты **head, tail, cat, tac, more, less** и **strings**.

Также мы познакомимся с возможностями таких инструментов с интерфейсом командной строки, как **cat**.

8.1. Утилита head

Вы можете использовать утилиту **head** для вывода первых десяти строк файла.

```
paul@laika:~$ head /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
paul@laika:~$
```

Утилита **head** также может использоваться для вывода первых n строк файла.

```
paul@laika:~$ head -4 /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
```

Кроме того, утилита **head** может использоваться для вывода первых **n** байт файла.

```
paul@laika:~$ head -c4 /etc/passwd
rootpaul@laika:~$
```

8.2. Утилита tail

По аналогии с утилитой **head**, утилита **tail** может использоваться для вывода последних десяти строк файла.

```
paul@laika:~$ tail /etc/services
vboxd           20012/udp
binkp           24554/tcp       # binkp fidonet protocol
asp             27374/tcp       # Address Search Protocol
asp             27374/udp
csync2          30865/tcp       # cluster synchronization
tool
dircproxy       57000/tcp       # Detachable IRC Proxy
tfido           60177/tcp       # fidonet EMSI over
telnet
fido            60179/tcp       # fidonet EMSI over TCP

# Local services
paul@laika:~$
```

Вы можете передать утилите **tail** число, соответствующее количеству строк, которое вы хотите увидеть в выводе.

```
$ tail -3 count.txt
fido            60179/tcp        # fidonet EMSI over TCP

# Local services
paul@laika:~$
```

Утилита **tail** поддерживает и другие полезные параметры, причем некоторые из них мы будем использовать в рамках данного курса.

8.3. Утилита cat

Утилита **cat** является одним из наиболее универсальных инструментов операционной системы. Все, что она делает - это копирует данные из **стандартного потока ввода** в **стандартный поток вывода**. Данная утилита в комбинации с командной оболочкой позволяет реализовать мощные и разнообразные механизмы обработки данных. Воспользуемся примерами для иллюстрации этих механизмов. Первый пример является достаточно простым и вы можете использовать рассмотренную в нем команду в повседневной работе

для вывода содержимого файла на экран. В том случае, если данные из файла не умещаются на экране, будет осуществлена прокрутка до конца файла.

```
paul@laika:~$ cat /etc/resolv.conf
nameserver 194.7.1.4
paul@laika:~$
```

8.3.1 Объединение файлов

Имя утилиты **cat** является сокращенной формой слова **concatenate** (объединить). Одним из наиболее простых сценариев использования данной утилиты является объединение файлов в рамках файла большего объема (или полного файла).

```
paul@laika:~$ echo один >part1
paul@laika:~$ echo два >part2
paul@laika:~$ echo три >part3
paul@laika:~$ cat part1
один
paul@laika:~$ cat part2
два
paul@laika:~$ cat part3
три
paul@laika:~$ cat part1 part2 part3
один
два
три
paul@laika:~$ cat part1 part2 part3 >all
paul@laika:~$ cat all
один
два
три
paul@laika:~$
```

8.3.2. Создание файлов

Также вы можете использовать утилиту **cat** для создания текстовых файлов. Введите команду **cat > winter.txt**, приведенную в примере ниже. После этого введите одну строку или несколько строк текста, завершая ввод каждой из строк нажатием клавиши **Enter**. По-

сле ввода последней строки текста нажмите и удерживайте клавишу **Ctrl** и одновременно с этим кратковременно нажмите клавишу **d**.

```
paul@laika:~/test$ cat > winter.txt
Сегодня очень холодно!
paul@laika:~/test$ cat winter.txt
Сегодня очень холодно!
paul@laika:~/test$
```

Комбинация клавиш **Ctrl+d** позволяет передать исполняющемуся процессу символ окончания файла (**End of File - EOF**), что приведет приведет к завершению исполнения процесса **cat**.

8.3.3. Специальный маркер окончания файла

Вы можете установить специальный маркер окончания файла для утилиты **cat** с помощью параметра **<<** таким образом, как показано в примере. Данная конструкция называется локальной директивой (**here directive**) и позволяет завершать работу процесса **cat**.

```
paul@laika:~/test$ cat > hot.txt <<stop
> Сегодня жарко!
> Да, это лето.
> stop
paul@laika:~/test$ cat hot.txt
Сегодня жарко!
Да, это лето.
paul@laika:~/test$
```

8.3.4. Копирование файлов

При рассмотрении третьего примера вы можете обнаружить, что утилита **cat** также может использоваться для копирования файлов. Подробные пояснения относительно происходящих в этом случае процессов будут даны в главе, посвященной командной оболочке bash.

```
paul@laika:~/test$ cat winter.txt
Сегодня очень холодно!
paul@laika:~/test$ cat winter.txt > cold.txt
paul@laika:~/test$ cat cold.txt
Сегодня очень холодно!
paul@laika:~/test$
```

8.4. Утилита tac

Предназначение утилиты **tac** (в отличие от утилиты **cat**) может быть проиллюстрировано с помощью единственного примера.

```
paul@laika:~/test$ cat count
один
два
три
четыре
paul@laika:~/test$ tac count
четыре
три
два
один
paul@laika:~/test$
```

8.5. Утилиты more и less

Утилита **more** может оказаться полезной в случае возникновения необходимости вывода содержимого файлов, которое не умещается на экране. Утилита **more** позволяет ознакомится с содержимым файла, разделенным на страницы. После открытия файла с использованием данной утилиты следует использовать клавишу "пробел" для перехода к следующей странице или клавишу q для выхода из режима просмотра содержимого файла. Некоторые пользователи предпочитают использовать утилиту **less** вместо утилиты **more**.

8.6. Утилита strings

С помощью утилиты **strings** вы можете осуществить вывод читаемых человеком ascii-строк, которые обнаруживаются в бинарных файлах. В приведенном ниже примере выясняется путь к бинарному файлу **ls**, после чего осуществляется вывод читаемых пользователем строк из этого бинарного файла (вывод сокращен).

```
paul@laika:~$ which ls
/bin/ls
paul@laika:~$ strings /bin/ls
/lib/ld-linux.so.2
librt.so.1
__gmon_start__
```

```
_Jv_RegisterClasses
clock_gettime
libacl.so.1
...
```

8.7. Практическое задание: работа с содержимым файлов

1. Осуществите вывод первых 12 строк файла **/etc/services.**

2. Осуществите вывод последней строки файла /**etc/passwd**.

3. Используйте утилиту cat для создания файла с именем **count.txt**, содержимое которого выглядит следующим образом:

```
Один

Два

Три

Четыре

Пять
```

4. Используйте утилиту **cp** для создания резервной копии этого файла с именем **cnt.txt**.

5. Используйте утилиту **cat** для создания резервной копии этого файла с именем **catcnt.txt**.

6. Осуществите вывод содержимого файла **catcnt.txt** таким образом, чтобы строки из файла были выведены в обратном порядке.

7. Используйте утилиту **more** для вывода содержимого файла **/etc/services**.

8. Осуществите вывод читаемых пользователем строк из бинарного файла **/usr/bin/passwd**.

9. Используйте утилиту **ls** для поиска файла наибольшего размера в директории **/etc**.

10. Откройте два окна терминала (или вкладки) и убедитесь в том, что в каждом из окон открыта одна и та же директория. Введите команду **`echo это первая строка > tailing.txt`** в первом окне терминала, после чего выполните команду **`tail -f tailing.txt`** во втором окне терминала. Теперь перейдите в первое окно терминала и введите команду **`echo это вторая строка >> tailing.txt`** (обратите внимание на два символа >>), после чего убедитесь в том, что средствами команды **`tail -f`** во втором окне терминала выводятся обе строки. Прекратите исполнение команды **`tail -f`** с помощью сочетания клавиш **`Ctrl+C`**.

11. Используйте утилиту **`cat`** для создания файла с именем **`tailing.txt`**, в котором должны содержаться данные из самого файла **`tailing.txt`**, после которых в него должны быть добавлены данные из файла **`/etc/passwd`**.

12. Используйте утилиту **`cat`** для создания файла с именем **`tailing.txt`**, в котором должны содержаться данные из самого этого файла **`tailing.txt`**, перед которыми в него должны быть добавлены данные из файла **`/etc/passwd`**.

8.8. Корректная процедура выполнения практического задания: работа с содержимым файлов

1. Осуществите вывод первых 12 строк файла **`/etc/services`**.

```
head -12 /etc/services
```

2. Осуществите вывод последней строки файла **`/etc/passwd`**.

```
tail -1 /etc/passwd
```

3. Используйте утилиту cat для создания файла с именем **`count.txt`**, содержимое которого выглядит следующим образом:

```
cat > count.txt
Один
Два
Три
Четыре
Пять (ввод должен завершаться с помощью комбинации клавиш Ctrl+d)
```

4. Используйте утилиту **cp** для создания резервной копии этого файла с именем **cnt.txt**.

```
cp count.txt cnt.txt
```

5. Используйте утилиту **cat** для создания резервной копии этого файла с именем **catcnt.txt**.

```
cat count.txt > catcnt.txt
```

6. Осуществите вывод содержимого файла **catcnt.txt** таким образом, чтобы строки из файла были выведены в обратном порядке.

```
tac catcnt.txt
```

7. Используйте утилиту **more** для вывода содержимого файла **/etc/services**.

```
more /var/log/messages
```

8. Осуществите вывод читаемых пользователем строк из бинарного файла **/usr/bin/passwd**.

```
strings /usr/bin/passwd
```

9. Используйте утилиту **ls** для поиска файла наибольшего размера в директории **/etc**.

```
ls -lrS /etc
```

10. Откройте два окна терминала (или вкладки) и убедитесь в том, что в каждом из окон открыта одна и та же директория. Введите команду **echo это первая строка > tailing.txt** в первом окне терминала, после чего выполните команду **tail -f tailing.txt** во втором окне терминала. Теперь перейдите в первое окно терминала и введите команду **echo это вторая строка >> tailing.txt** (обратите внимание на два символа >>), после чего убедитесь в том, что средствами команды **tail -f** во втором окне терминала выводятся обе строки. Прекратите исполнение команды **tail -f** с помощью сочетания клавиш **Ctrl+C**.

11. Используйте утилиту **cat** для создания файла с именем **tailing.txt**, в котором должны содержаться данные из самого файла **tailing.txt**, после которых в него должны быть добавлены данные из файла **/etc/passwd**.

```
cat /etc/passwd >> tailing.txt
```

12. Используйте утилиту **cat** для создания файла с именем **tailing.txt**, в котором должны содержаться данные из самого этого файла **tailing.txt**, перед которыми в него должны быть добавлены данные из файла **/etc/passwd**.

```
mv tailing.txt tmp.txt ; cat /etc/passwd tmp.txt > tailing.txt
```

Глава 9. Дерево директорий Linux

В рамках данной главы производится обзор наиболее часто используемых директорий из **дерева директорий файловой системы Linux**. Также данная глава может служить доказательством утверждения о том, что в Unix-системе все является файлом.

9.1. Стандарт иерархии файловой системы

Многие дистрибутивы Linux частично следуют **Стандарту иерархии файловой системы** (**Filesystem Hierarchy Standard**). Данный стандарт может оказаться полезным для будущего процесса стандартизации деревьев директорий файловых систем Unix/Linux. Стандарт **FHS** доступен в сети по адресу **http://www.pathname.com/fhs/**, причем на данном ресурсе мы можем прочитать: "Стандарт иерархии файловой системы был создан с целью его использования разработчиками дистрибутивов Unix, разработчиками пакетов для распространения программного обеспечения и разработчиками операционных систем. Однако, данный стандарт является в большей степени справочным материалом, нежели руководством по работе с файловой системой Unix или с иерархиями директорий."

9.2. Страница руководства `man hier`

Существуют некоторые различия вt иерархиях файловых систем различных **дистрибутивов Linux**. Для того, чтобы ознакомиться с информацией об иерархии файловой системы вашей машины, используйте команду **man hier**. На данной странице руководства будут приведены пояснения относительно структуры дерева директорий системы, установленной на вашем компьютере.

9.3. Корневая директория /

Структуры директорий всех систем Linux начинаются с **корневой директории**. Корневая директория обозначается с помощью символа **прямого слэша**, а именно, **/**. Все файлы, которые существуют в

вашей системе Linux, находится ниже данной корневой директории в дереве директорий. Давайте рассмотрим содержимое этой корневой директории.

```
[paul@RHELv4u3 ~]$ ls /
bin   dev  home  media  mnt  proc  sbin     srv  tftpboot
usr
boot  etc  lib   misc   opt  root  selinux  sys  tmp
var
```

9.4. Директории для хранения бинарных файлов

Бинарные файлы являются файлами, содержащими скомпилированный исходный код (или машинный код). Бинарные файлы могут **исполняться** на компьютере. Иногда бинарные файлы также называются **исполняемыми** файлами.

9.4.1. Директория /bin

Директория **/bin** содержит **бинарные** файлы, которые могут использоваться всеми пользователями. В соответствии со спецификацией FHS, директория **/bin** должна содержать исполняемые файлы **/bin/cat** и **/bin/date** (помимо других исполняемых файлов).

В примере ниже вы можете увидеть список исполняемых файлов, являющихся реализациями таких команд, как **cat, cp, cpio, date, dd, echo, grep** и.т.д. Многие из упомянутых команд будут рассмотрены в рамках данной книги.

```
paul@laika:~$ ls /bin
archdetect       egrep            mt
setupcon
autopartition    false            mt-gnu           sh
bash             fgconsole        mv
sh.distrib
bunzip2          fgrep            nano             sleep
bzcat            fuser            nc
stralign
bzcmp            fusermount       nc.traditional   stty
bzdiff           get_mountoptions netcat           su
bzegrep          grep             netstat          sync
bzexe            gunzip           ntfs-3g          sysfs
bzfgrep          gzexe            ntfs-3g.probe    tailf
```

```
bzgrep          gzip            parted_devices  tar
bzip2           hostname        parted_server
tempfile
bzip2recover    hw-detect       partman         touch
bzless          ip              partman-commit  true
bzmore          kbd_mode        perform_recipe
ulockmgr
cat             kill            pidof
umount
...
```

9.4.2. Другие директории /bin

Вы можете обнаружить **поддиректорию /bin** во многих других директориях. Например, пользователь с именем **serena** может разместить свои собственные приложения в поддиректории **/home/serena/bin**.

Файлы некоторых приложений, обычно в случае установки путем непосредственной сборки из исходного кода, устанавливаются в директорию **/opt**. К примеру, при установке **сервера samba** для хранения бинарных файлов может быть использована поддиректория **/opt/samba/bin**.

9.4.3. Директория /sbin

Директория **/sbin** содержит бинарные файлы, предназначенные для настройки операционной системы. Многие из **бинарных файлов** для настройки системы требуют наличия привилегий пользователя **root** для выполнения определенных задач.

В примере ниже приведен список **бинарных файлов** для настройки системы, предназначенных для изменения IP-адреса, работы с разделами жестких дисков и создания файловой системы ext4.

```
paul@ubu1010:~$ ls -l /sbin/ifconfig /sbin/fdisk
/sbin/mkfs.ext4
-rwxr-xr-x 1 root root 97172 2011-02-02 09:56 /sbin/fdisk
-rwxr-xr-x 1 root root 65708 2010-07-02 09:27
/sbin/ifconfig
-rwxr-xr-x 5 root root 55140 2010-08-18 18:01
/sbin/mkfs.ext4
```

9.4.4. Директория /lib

Бинарные файлы из директорий **/bin** и **/sbin** обычно используют разделяемые библиотеки, расположенные в директории **/lib**. В примере ниже приведен список некоторых файлов из директории **/lib**.

```
paul@laika:~$ ls /lib/libc*
/lib/libc-2.5.so     /lib/libcfont.so.0.0.0
/lib/libcom_err.so.2.1
/lib/libcap.so.1     /lib/libcidn-2.5.so
/lib/libconsole.so.0
/lib/libcap.so.1.10  /lib/libcidn.so.1
/lib/libconsole.so.0.0.0
/lib/libcfont.so.0   /lib/libcom_err.so.2
/lib/libcrypt-2.5.so
```

9.4.5. Поддиректория /lib/modules

Обычно **ядро Linux** загружает модули из директории **/lib/modules/$версия-ядра/**. Содержимое этой директории будет подробно описано в главе, посвященной ядру Linux.

9.4.6. Директории /lib32 и /lib64

На данный момент осуществляется медленная миграция с **32-битных** на **64-битные** системы. По этой причине вы можете обнаружить в своей системе директории с именами **/lib32** и **/lib64**, которые указывают на размеры регистров, использованные в процессе компиляции расположенных в этих директориях разделяемых библиотек. 64-битный компьютер может содержать некоторые 32-битные бинарные файлы и библиотеки, используемые для достижения совместимости с устаревшими приложениями. В примере ниже утилита **file** используется для демонстрации описанных различий между разделяемыми библиотеками.

```
paul@laika:~$ file /lib32/libc-2.5.so
/lib32/libc-2.5.so: ELF 32-bit LSB shared object, Intel
80386, version 1 (SYSV), for GNU/Linux 2.6.0, stripped
paul@laika:~$ file /lib64/libcap.so.1.10
/lib64/libcap.so.1.10: ELF 64-bit LSB shared object, AMD
x86-64, version 1 (SYSV), stripped
```

Формат ELF (**формат исполняемых и компонуемых файлов - Executable and Linkable Format**) используется практически во

всех Unix-подобных операционных системах с момента выпуска **System V**.

9.4.7. Директория /opt

Директория **/opt** предназначена для хранения **вспомогательного** программного обеспечения. В большинстве случаев данное программное обеспечение устанавливается не из репозитория дистрибутива. В многих системах директория **/opt** пуста.

При установке пакета программного обеспечения большого объема файлы из него могут копироваться в поддиректории **/bin, /lib, /etc** директории **/opt/$имя-пакета/**. Например, в том случае, если пакет программного обеспечения носит имя wp, файлы из него будут устанавливаться в директорию **/opt/wp**, при этом бинарные файлы будут устанавливаться в поддиректорию **/opt/wp/bin**, а файлы страниц руководств - в поддиректорию **/opt/wp/man**.

9.5. Директории для хранения файлов конфигурации

9.5.1. Директория /boot

Директория **/boot** содержит все файлы, необходимые для загрузки компьютера. Эти файлы не изменяются очень часто. В системах Linux в данной директории обычно можно обнаружить поддиректорию **/boot/grub**. Директория **/boot/grub** содержит файл **/boot/grub/grub.cfg** (на более старых системах также может использоваться файл **/boot/grub/grub.conf**), в рамках которого описывается меню загрузки, отображаемое перед загрузкой ядра ОС.

9.5.2. Директория /etc

Все специфичные для машины **конфигурационные файлы** должны быть расположены в директории **/etc**. Изначально имя директории **/etc** была образовано от слова **etcetera** (и так далее), но сегодня люди часто расшифровывают его как **Editable Text Configuration (директория с редактируемыми текстовыми файлами конфигурации).**

Во многих случаях имена конфигурационных файлов совпадают с именами приложений или протоколов, а в качестве расширений этих файлов используется строка **.conf**.

```
paul@laika:~$ ls /etc/*.conf
/etc/adduser.conf       /etc/ld.so.conf
/etc/scrollkeeper.conf
```

```
/etc/brltty.conf          /etc/lftp.conf
/etc/sysctl.conf
/etc/ccertificates.conf   /etc/libao.conf
/etc/syslog.conf
/etc/cvs-cron.conf        /etc/logrotate.conf
/etc/ucf.conf
/etc/ddclient.conf        /etc/ltrace.conf
/etc/uniconf.conf
/etc/debconf.conf         /etc/mke2fs.conf
/etc/updatedb.conf
/etc/deluser.conf         /etc/netscsid.conf
/etc/usplash.conf
/etc/fdmount.conf         /etc/nsswitch.conf
/etc/uswsusp.conf
/etc/hdparm.conf          /etc/pam.conf
/etc/vnc.conf
/etc/host.conf            /etc/pnm2ppa.conf
/etc/wodim.conf
/etc/inetd.conf           /etc/povray.conf
/etc/wvdial.conf
/etc/kernel-img.conf      /etc/resolv.conf
paul@laika:~$
```

В директории **/etc** также можно обнаружить большое количество других важных файлов.

9.5.3. Поддиректория /etc/init.d/

Во многих дистрибутивах Unix/Linux имеется директория **/etc/init.d**, которая содержит сценарии для запуска и остановки **демонов**. Эта поддиректория может исчезнуть в процессе перехода дистрибутивов Linux на системы инициализации, которые заменят старую систему инициализации **init**, используемую для запуска всех **демонов**.

9.5.4. Поддиректория /etc/X11/

Управление системой вывода графики осуществляется средствами программного обеспечения от организации X.org Foundation (а именно, сервера оконной системы **X Window System** или просто **X**). Файл конфигурации для вашего сервера оконной системы носит имя **/etc/X11/xorg.conf**.

9.5.5. Поддиректория /etc/skel/

Содержимое директории **каркаса /etc/skel** копируется в домашнюю директорию при создании учетной записи пользователя. Она обычно содержит такие скрытые файлы, как сценарий **.bashrc**.

9.5.6. Поддиректория /etc/sysconfig/

Данная директория, не упомянутая в спецификации FHS, содержит большое количество файлов конфигурации компонентов дистрибутива **Red Hat Enterprise Linux**. Впоследствии мы будем более подробно рассматривать некоторые из этих файлов. В примере ниже приведен список файлов директории **/etc/sysconfig** дистрибутива RHELv4u4 в случае установки всех связанных с данной директорией программных компонентов.

```
paul@RHELv4u4:~$ ls /etc/sysconfig/
apmd            firstboot     irda              network
saslauthd
apm-scripts     grub          irqbalance        networking
selinux
authconfig      hidd          keyboard          ntpd
spamassassin
autofs          httpd         kudzu             openib.conf
squid
bluetooth       hwconf        lm_sensors        pand
syslog
clock           i18n          mouse             pcmcia
sys-config-sec
console         init          mouse.B           pgsql
sys-config-users
crond           installinfo   named             prelink
sys-logviewer
desktop         ipmi          netdump           rawdevices
tux
diskdump        iptables      netdump_id_dsa    rhn
vncservers
dund            iptables-cfg  netdump_id_dsa.p  samba
xinetd
paul@RHELv4u4:~$
```

Файл **/etc/sysconfig/firstboot** сообщает агенту настройки дистрибутива Red Hat о том, что он не должен запускаться после

загрузки системы. В том случае, если вы желаете использовать агент настройки дистрибутива Red Hat после следующей перезагрузки, вам следует просто удалить данный файл и выполнить команду **chkconfig --level 5 firstboot on**. Агент настройки дистрибутива Red Hat позволяет устанавливать последние обновления системы, создавать учетные записи пользователей, пользоваться функциями портала Red Hat Network, а также выполнять другие действия. После запуска он снова создаст файл /etc/sysconfig/firstboot.

```
paul@RHELv4u4:~$ cat /etc/sysconfig/firstboot
RUN_FIRSTBOOT=NO
```

Файл **/etc/sysconfig/harddisks** содержит дополнительные параметры настройки жестких дисков. Формат файла описан в самом файле.

Вы можете ознакомиться с описанием программного обеспечения, обнаруженного утилитой **kudzu**, которое сохраняется в файле **/etc/sysconfig/hwconf**. Kudzu является приложением от компании Red Hat, предназначенным для автоматического обнаружения и настройки аппаратного обеспечения.

Тип клавиатуры и таблица соответствия символов устанавливаются в файле **/etc/sysconfig/keyboard**. Для получения дополнительной информации о настройках клавиатуры в консоли следует обратиться к следующим страницам руководств **keymaps(5)**, **dumpkeys(1), loadkeys(1),** а также к содержимому директории **/lib/kbd/keymaps/**.

```
root@RHELv4u4:/etc/sysconfig# cat keyboard
KEYBOARDTYPE="pc"
KEYTABLE="us"
```

Файлы для настройки сетевых устройств из данной директории будут обсуждаться в главе, посвященной настройке сети.

9.6. Директории для хранения данных

9.6.1. Директория /home

Пользователи могут хранить персональные данные и данные проектов в директории **/home**. Обычно (но не всегда в соответствии со спецификацией FHS) имя домашней директории пользователя устанавливается в соответствии с полным именем пользователя в формате **/home/$имя_пользователя**. Например:

```
paul@ubu606:~$ ls /home
geert  annik  sandra  paul  tom
```

Помимо выделения каждому пользователю (или каждому проекту или группе) места для хранения персональных данных в рамках домашней директории, в рамках этой же директории выделяется место для хранения данных профиля пользователя. Типичный профиль пользователя Unix содержит множество скрытых файлов (файлов, имена которых начинаются с точки). Скрытые файлы пользовательского профиля Unix содержат параметры, установленные для данного пользователя.

```
paul@ubu606:~$ ls -d /home/paul/.*
/home/paul/.                /home/paul/.bash_profile
/home/paul/.ssh
/home/paul/..               /home/paul/.bashrc
/home/paul/.viminfo
/home/paul/.bash_history    /home/paul/.lesshst
```

9.6.2. Директория /root

Во многих системах директория **/root** является стандартной директорией для хранения персональных данных и данных профиля **пользователя root**. В том случае, если ее не существует по умолчанию, администраторы могут создать ее самостоятельно.

9.6.3. Директория /srv

Вы можете использовать директорию **/srv** для хранения данных, которые **обрабатываются вашей системой**. Спецификация FHS позволяет хранить в этой директории данные **cvs, rsync, ftp** и **www**. Кроме того, спецификация FHS подтверждает возможность использования таких административных имен для поддиректорий, как **/srv/project55/ftp** и **/srv/sales/www**.

В системах Sun Solaris (или Oracle Solaris) для этой цели используется директория **/export**.

9.6.4. Директория /media

Директория **/media** служит точкой монтирования для таких устройств для работы со **съемными носителями**, как приводы CD-ROM, цифровые камеры, а также различные устройства, подключаемые по шине USB. Так как директория **/media** является достаточно новой в мире систем Unix, вы с высокой вероятностью можете встретить системы, не использующие данную директорию. К примеру, несмотря на то, что система Solaris 9 не имеет рассматриваемой директории, эта директория присутствует в системе Solaris 10. Боль-

шинство дистрибутивов Linux на сегодняшний день монтирует все съемные носители в директорию **/media**.

```
paul@debian5:~$ ls /media/
cdrom  cdrom0  usbdisk
```

9.6.5. Директория /mnt

Директория **/mnt** должна быть пустой и использоваться исключительно для создания временных точек монтирования файловых систем (в соответствии со спецификацией FHS).

Администраторы систем Unix и Linux обычно создают в данной директории множество поддиректорий, таких, как **/mnt/something/**. Вы, скорее всего, столкнетесь с системами с более чем одной директорией, созданной и/или смонтированной в рамках директории **/mnt** для работы с различными локальными и удаленными файловыми системами.

9.6.6. Директория /tmp

Приложения и пользователи должны использовать директорию **/tmp** для хранения временных данных при необходимости. Данные, хранимые в директории **/tmp**, могут в реальности храниться как на диске, так и в оперативной памяти. В обоих случаях обслуживание хранилища временных данных осуществляется средствами операционной системы. Никогда не используйте директорию **/tmp** для хранения данных, которые являются важными или которые вы желаете архивировать.

9.7. Директории в оперативной памяти

9.7.1. Директория /dev

Файлы устройств из директории **/dev** выглядят как обычные файлы, но на самом деле не являются обычными файлами, размещенными на жестком диске. Директория **/dev** заполняется файлами в процессе определения устройств средствами ядра операционной системы.

9.7.2. Стандартные физические устройства

Стандартные устройства, такие, как жесткие диски, представлены файлами устройств в директории **/dev**. В примере ниже приведен список файлов устройств SATA ноутбука, а также устройств IDE настольного компьютера. (Подробное описание назначения этих файлов устройств будет приведено ниже.)

```
#
```

```
# Устройства SATA или SCSI или USB
#
paul@laika:~$ ls /dev/sd*
/dev/sda  /dev/sda1  /dev/sda2  /dev/sda3  /dev/sdb
/dev/sdb1  /dev/sdb2


#
# Устройства IDE или ATAPI
#
paul@barry:~$ ls /dev/hd*
/dev/hda  /dev/hda1  /dev/hda2  /dev/hdb  /dev/hdb1
/dev/hdb2  /dev/hdc
```

Помимо представления физических устройств, некоторые файлы устройств выполняют специальные функции. Эти специальные файлы устройств могут оказаться очень полезными.

9.7.3. Файлы устройств /dev/tty и /dev/pts

К примеру, файл устройства **/dev/tty1** представляет терминал или консоль, соединенную с системой. (Не стоит ломать голову над точными значениями терминов 'терминал' или 'консоль', так как в данном случае имеется в виду интерфейс командной строки системы.) При вводе команд в эмуляторе терминала, поставляемом в составе такого графического окружения рабочего стола, как Gnome или KDE, ваш терминал будет представлен файлом устройства **/dev/pts/1** (вместо числа 1 может использоваться другое число).

9.7.4. Файл устройства /dev/null

В Linux вы можете обнаружить и другие файлы специальных устройств, такие, как файл устройства **/dev/null**, которое может рассматриваться как черная дыра; хотя соответствующее устройство и имеет неограниченную емкость, после записи из него не могут быть прочитаны никакие данные. Говоря техническим языком, любые записанные на представленное файлом **/dev/null** устройство данные будут просто отброшены. Представленное файлом **/dev/null** устройство может быть использовано для отбрасывания ненужного вывода различных команд. *Помните о том, что представленное файлом **/dev/null** устройство не является удачным местом для хранения ваших резервных копий данных ;-).*

9.7.5. Директория /proc и взаимодействие с ядром ОС

Директория **/proc** является другой специальной директорией, которая содержит файлы, кажущиеся на первый взгляд обычными файлами, но не занимающие места на диске. На самом деле содержимое данной директории является представлением ядра ОС, а точнее, используемых ядром ОС структур данных и предназначено для непосредственного взаимодействия с ядром ОС. В директорию **/proc** монтируется специальная файловая система procfs.

```
paul@RHELv4u4:~$ mount -t proc
none on /proc type proc (rw)
```

При выводе содержимого директории /proc можно обнаружить множество директорий с именами, представленными числовыми значениями (в любой системе Unix), а также некоторые интересные файлы (в Linux).

```
mul@laika:~$ ls /proc
1      2339   4724  5418  6587  7201       cmdline
mounts
10175  2523   4729  5421  6596  7204       cpuinfo
mtrr
10211  2783   4741  5658  6599  7206       crypto
net
10239  2975   4873  5661  6638  7214       devices
pagetypeinfo
141    29775  4874  5665  6652  7216       diskstats
partitions
15045  29792  4878  5927  6719  7218       dma
sched_debug
1519   2997   4879  6     6736  7223       driver
scsi
1548   3      4881  6032  6737  7224       execdomains
self
1551   30228  4882  6033  6755  7227       fb
slabinfo
1554   3069   5     6145  6762  7260       filesystems
stat
1557   31422  5073  6298  6774  7267       fs
swaps
```

```
1606   3149   5147  6414  6816  7275       ide
sys
180    31507  5203  6418  6991  7282       interrupts
sysrq-trigger
181    3189   5206  6419  6993  7298       iomem
sysvipc
182    3193   5228  6420  6996  7319       ioports
timer_list
18898  3246   5272  6421  7157  7330       irq
timer_stats
19799  3248   5291  6422  7163  7345       kallsyms
tty
19803  3253   5294  6423  7164  7513       kcore
uptime
19804  3372   5356  6424  7171  7525       key-users
version
1987   4      5370  6425  7175  7529       kmsg
version_signature
1989   42     5379  6426  7188  9964       loadavg
vmcore
2      45     5380  6430  7189  acpi       locks
vmnet
20845  4542   5412  6450  7191  asound     meminfo
vmstat
221    46     5414  6551  7192  buddyinfo  misc
zoneinfo
2338   4704   5416  6568  7199  bus        modules
```

Давайте обратим внимание на свойства файлов из директории **/proc**. При рассмотрении даты и времени изменения данных файлов можно отметить, что эти параметры соответствуют текущим значениями даты и времени, из чего можно сделать вывод, что содержимое данных файлов постоянно обновляется (для предоставления доступа к актуальному содержимому используемых ядром ОС структур данных).

```
paul@RHELv4u4:~$ date
Пн янв 29 18:06:32 EST 2007
paul@RHELv4u4:~$ ls -al /proc/cpuinfo
-r--r--r--  1 root root 0 ноя 29 18:06 /proc/cpuinfo
```

```
paul@RHELv4u4:~$
paul@RHELv4u4:~$  ...через некоторое время...
paul@RHELv4u4:~$
paul@RHELv4u4:~$ date
Пн янв 29 18:10:00 EST 2007
paul@RHELv4u4:~$ ls -al /proc/cpuinfo
-r--r--r--  1 root root 0 ноя 29 18:10 /proc/cpuinfo
```

Размер большинства файлов из директории /proc равен 0 байт, но при этом файлы из данной директории содержат данные, а иногда большие объемы данных. Вы можете ознакомиться с этими данными, использовав команду cat по отношению к таким файлам, как файл **/proc/cpuinfo**, который содержит информацию о центральном процессоре.

```
paul@RHELv4u4:~$ file /proc/cpuinfo
/proc/cpuinfo: empty
paul@RHELv4u4:~$ cat /proc/cpuinfo
processor       : 0
vendor_id       : AuthenticAMD
cpu family      : 15
model           : 43
model name      : AMD Athlon(tm) 64 X2 Dual Core
Processor 4600+
stepping        : 1
cpu MHz         : 2398.628
cache size      : 512 KB
fdiv_bug        : no
hlt_bug         : no
f00f_bug        : no
coma_bug        : no
fpu             : yes
fpu_exception   : yes
cpuid level     : 1
wp              : yes
flags           : fpu vme de pse tsc msr pae mce cx8 apic
mtrr pge...
```

```
bogomips        : 4803.54
```

Для сравнения ниже приведено содержимое файла /proc/cpuinfo системы Sun Sunblade 1000...

```
paul@pasha:~$ cat /proc/cpuinfo
cpu : TI UltraSparc III (Cheetah)
fpu : UltraSparc III integrated FPU
promlib : Version 3 Revision 2
prom : 4.2.2
type : sun4u
ncpus probed : 2
ncpus active : 2
Cpu0Bogo : 498.68
Cpu0ClkTck : 000000002cb41780
Cpu1Bogo : 498.68
Cpu1ClkTck : 000000002cb41780
MMU Type : Cheetah
State:
CPU0: online
CPU1: online
```

Большая часть файлов из директории /proc предназначена исключительно для чтения, причем для чтения некоторых из них требуются привилегии пользователя root; в некоторые файлы могут записываться данные, причем в директории **/proc/sys** таких файлов большинство. Давайте поговорим о некоторых файлах из директории /proc.

9.7.6. Файл /proc/interrupts

В системе архитектуры x86 в файле **/proc/interrupts** содержится информация о запросах прерываний.

```
paul@RHELv4u4:~$ cat /proc/interrupts
           CPU0
  0:   13876877    IO-APIC-edge  timer
  1:         15    IO-APIC-edge  i8042
  8:          1    IO-APIC-edge  rtc
  9:          0   IO-APIC-level  acpi
 12:         67    IO-APIC-edge  i8042
```

```
 14:        128    IO-APIC-edge  ide0
 15:     124320    IO-APIC-edge  ide1
169:     111993   IO-APIC-level  ioc0
177:       2428   IO-APIC-level  eth0
NMI:          0
LOC:   13878037
ERR:          0
MIS:          0
```

При использовании машины с двумя центральными процессорами данный файл выглядит следующим образом.

```
paul@laika:~$ cat /proc/interrupts
           CPU0       CPU1
  0:     860013          0  IO-APIC-edge     timer
  1:       4533          0  IO-APIC-edge     i8042
  7:          0          0  IO-APIC-edge     parport0
  8:    6588227          0  IO-APIC-edge     rtc
 10:       2314          0  IO-APIC-fasteoi  acpi
 12:        133          0  IO-APIC-edge     i8042
 14:          0          0  IO-APIC-edge     libata
 15:      72269          0  IO-APIC-edge     libata
 18:          1          0  IO-APIC-fasteoi  yenta
 19:     115036          0  IO-APIC-fasteoi  eth0
 20:     126871          0  IO-APIC-fasteoi  libata,
ohci1394
 21:      30204          0  IO-APIC-fasteoi  ehci_hcd:usb1,
uhci_hcd:usb2
 22:       1334          0  IO-APIC-fasteoi  saa7133[0],
saa7133[0]
 24:     234739          0  IO-APIC-fasteoi  nvidia
NMI:         72         42
LOC:     860000     859994
ERR:          0
```

9.7.7. Файл /proc/kcore

Физическая память представлена файлом **/proc/kcore**. Не пытайтесь использовать команду cat по отношению к этому файлу;

вместо этого при необходимости исследования содержимого оперативной памяти используйте отладчик. Размер файла /proc/kcore совпадает с объемом вашей оперативной памяти плюс четыре байта.

```
paul@laika:~$ ls -lh /proc/kcore
-r-------- 1 root root 2.0G 2007-01-30 08:57 /proc/kcore
paul@laika:~$
```

9.7.8. Директория /sys для работы с системой горячего подключения устройств ядра Linux 2.6

Директория **/sys** была создана в процессе разработки версии 2.6 ядра Linux. С момента выпуска версии 2.6 ядро Linux использует файловую систему **sysfs** для реализации механизма горячего подключения устройств, использующих шины **usb** и **IEEE 1394 (FireWire)**. Обратитесь к страницам руководств udev(8) (данная подсистема пришла на смену подсистеме **devfs**) и hotplug(8) для получения дополнительной информации (или посетите ресурс http://linux-hotplug.sourceforge.net/).

По существу, директория /sys содержит файлы с информацией об используемом аппаратном обеспечении.

9.8. Директория системных ресурсов Unix /usr

Несмотря на то, что имя директории **/usr** напоминает слово user (пользователь), не следует забывать о том, что на самом деле оно расшифровывается как **Unix System Resources** (директория системных ресурсов Unix). Иерархия поддиректорий директории /usr должна содержать разделяемые данные приложений, доступные только для чтения. Некоторые системные администраторы осуществляют монтирование файловой системы **/usr** в режиме только для чтения. В этом случае данная директория должна быть расположена на отдельном разделе жесткого диска или на разделяемом ресурсе NFS.

9.8.1. Директория /usr/bin

Директория **/usr/bin** содержит множество реализаций команд.

```
paul@deb508:~$ ls /usr/bin | wc -l
1395
```

(В системе Solaris директория **/bin** является символьной ссылкой на директорию **/usr/bin**.)

9.8.2. Директория /usr/include

Директория **/usr/include** содержит общедоступные заголовочные файлы для языка программирования C.

```
paul@ubu1010:~$ ls /usr/include/
aalib.h         expat_config.h      math.h
search.h
af_vfs.h        expat_external.h    mcheck.h
semaphore.h
aio.h           expat.h             memory.h
setjmp.h
AL              fcntl.h             menu.h
sgtty.h
aliases.h       features.h          mntent.h
shadow.h
...
```

9.8.3. Директория /usr/lib

Директория **/usr/lib** содержит разделяемые библиотеки, которые не используются непосредственно пользователями или сценариями.

```
paul@deb508:~$ ls /usr/lib | head -7
4Suite
ao
apt
arj
aspell
avahi
bonobo
```

9.8.4. Директория /usr/local

Директория **/usr/local** может использоваться системным администратором для локальной установки программного обеспечения.

```
paul@deb508:~$ ls /usr/local/
bin  etc  games  include  lib  man  sbin  share  src
paul@deb508:~$ du -sh /usr/local/
128K    /usr/local/
```

9.8.5. Директория /usr/share

Директория **/usr/share** содержит независимые от архитектуры данные. Как вы можете заметить, данная директория имеет значительный размер.

```
paul@deb508:~$ ls /usr/share/ | wc -l
263
paul@deb508:~$ du -sh /usr/share/
1.3G    /usr/share/
```

Обычно данная директория содержит поддиректорию **/usr/share/man**, предназначенную для хранения файлов страниц руководств.

```
paul@deb508:~$ ls /usr/share/man
cs  fr            hu            it.UTF-8  man2  man6
pl.ISO8859-2  sv
de  fr.ISO8859-1  id            ja        man3  man7
pl.UTF-8      tr
es  fr.UTF-8      it            ko        man4  man8
pt_BR         zh_CN
fi  gl            it.ISO8859-1  man1      man5  pl    ru
zh_TW
```

Также данная директория содержит поддиректорию **/usr/share/games**, предназначенную для хранения всех статических данных игр (таким образом, в данной директории не могут находиться файлы со списками рекордов или журналами игрового процесса).

```
paul@ubu1010:~$ ls /usr/share/games/
openttd  wesnoth
```

9.8.6. Директория /usr/src

Директория **/usr/src** является рекомендуемой директорией для хранения файлов исходного кода ядра ОС.

```
paul@deb508:~$ ls -l /usr/src/
итого 12
drwxr-xr-x  4 root root 4096 2011-02-01 14:43 linux-
headers-2.6.26-2-686
drwxr-xr-x 18 root root 4096 2011-02-01 14:43 linux-
headers-2.6.26-2-common
```

```
drwxr-xr-x  3 root root 4096 2009-10-28 16:01 linux-
kbuild-2.6.26
```

9.9. Директория для изменяемых данных /var

Файлы заранее неизвестного размера, такие, как файлы журналов, файлы кэша и файлы очереди печати должны сохраняться в директории **/var**.

9.9.1. Директория /var/log

Директория **/var/log** выполняет функции центрального хранилища всех файлов журналов.

```
[paul@RHEL4b ~]$ ls /var/log
acpid           cron.2    maillog.2   quagga
secure.4
amanda          cron.3    maillog.3   radius
spooler
anaconda.log    cron.4    maillog.4   rpmpkgs
spooler.1
anaconda.syslog cups      mailman     rpmpkgs.1
spooler.2
anaconda.xlog   dmesg     messages    rpmpkgs.2
spooler.3
audit           exim      messages.1  rpmpkgs.3
spooler.4
boot.log        gdm       messages.2  rpmpkgs.4
squid
boot.log.1      httpd     messages.3  sa
uucp
boot.log.2      iiim      messages.4  samba
vbox
boot.log.3      iptraf    mysqld.log  scrollkeeper.log
vmware-tools-guestd
boot.log.4      lastlog   news        secure
wtmp
canna           mail      pgsql       secure.1
wtmp.1
cron            maillog   ppp         secure.2
Xorg.0.log
cron.1          maillog.1 prelink.log secure.3
Xorg.0.log.old
```

9.9.2. Файл /var/log/messages

Стандартным файлом, к которому следует обратиться в первую очередь при диагностике дистрибутива от компании Red Hat (и производных дистрибутивов), является файл **/var/log/messages**. По умолчанию данный файл должен содержать информацию о событиях, которые происходят в рамках системы. Файл, выполняющий аналогичные функции в дистрибутивах Debian и Ubuntu, носит имя **/var/log/syslog**.

```
[root@RHEL4b ~]# tail /var/log/messages
Jul 30 05:13:56 anacron: anacron startup succeeded
Jul 30 05:13:56 atd: atd startup succeeded
Jul 30 05:13:57 messagebus: messagebus startup succeeded
Jul 30 05:13:57 cups-config-daemon: cups-config-daemon
startup succeeded
Jul 30 05:13:58 haldaemon: haldaemon startup succeeded
Jul 30 05:14:00 fstab-sync[3560]: removed all generated
mount points
Jul 30 05:14:01 fstab-sync[3628]: added mount point
/media/cdrom for...
Jul 30 05:14:01 fstab-sync[3646]: added mount point
/media/floppy for...
Jul 30 05:16:46 sshd(pam_unix)[3662]: session opened for
user paul by...
Jul 30 06:06:37 su(pam_unix)[3904]: session opened for
user root by paul
```

9.9.3. Директория /var/cache

Директория **/var/cache** может содержать **кэшированные данные** некоторых приложений.

```
paul@ubu1010:~$ ls /var/cache/
apt      dictionaries-common    gdm       man
software-center
binfmts  flashplugin-installer  hald      pm-utils
cups     fontconfig             jockey    pppconfig
debconf  fonts                  ldconfig  samba
```

9.9.4. Директория /var/spool

Директория **/var/spool** обычно содержит поддиректории для хранения файлов с сообщениями электронной почты и данными задач **cron**, причем она также может быть родительской директорией для других файлов очередей (например, файлов очередей печати).

9.9.5. Директория /var/lib

Директория **/var/lib** содержит файлы с данными состояния приложений.

Дистрибутив Red Hat Enterprise Linux, к примеру, хранит файлы, относящиеся к менеджеру пакетов **rpm**, в поддиректории **/var/lib/rpm/**.

9.9.6. Другие директории /var/...

Директория **/var** также содержит файлы с идентификаторами процессов в поддиректории **/var/run** (которая в недалеком будущем будет заменена на директорию **/run**), временные файлы, которые не должны удаляться при перезагрузке, в поддиректории **/var/tmp**, а также файлы блокировок в поддиректории **/var/lock**. Далее в данной книге будут приведены дополнительные примеры использования директории **/var** для хранения данных.

9.10. Практическое задание: дерево директорий Linux

1. Существует ли файл **/bin/ca**t? Как насчет файлов **/bin/dd** и **/bin/echo**. Какого типа данные файлы?

2. Каков общий объем файлов ядра Linux (vmlinu*) в директории **/boot**?

3. Создайте директорию **~/test**. После этого выполните следующие команды:

```
cd ~/test
dd if=/dev/zero of=zeroes.txt count=1 bs=100
od zeroes.txt
```

Утилита **dd** осуществит копирование одного блока (count=1) размером в 100 байт (bs=100) из специального файла **/dev/zero** в файл **~/test/zeroes.txt**. Какие пояснения вы можете дать относительно возможностей специального файла **/dev/zero**?

4. А теперь выполните следующую команду:

```
dd if=/dev/random of=random.txt count=1 bs=100 ; od
random.txt
```

Утилита dd осуществит копирование одного блока (count=1) размером в 100 байт (bs=100) из специального файла **/dev/random** в файл **~/test/random.txt**. Какие пояснения вы можете дать относительно возможностей специального файла **/dev/random**?

5. Выполните две следующие команды и обратите внимание на первый символ вывода каждой из команд.

```
ls -l /dev/sd* /dev/hd*
ls -l /dev/tty* /dev/input/mou*
```

С помощью первой команды будет выведен список файлов блочных устройств, с помощью второй - список файлов символьных устройств. Что вы можете сказать по поводу различий между символьными и блочными устройствами.

6. Используйте команду **cat** для вывода содержимого файлов **/etc/hosts** и **/etc/resolv.conf**. Что вы думаете по поводу предназначения данных файлов?

7. Хранятся ли какие-нибудь файлы в директории **/etc/skel/**? Не забудьте проверить наличие скрытых файлов.

8. Выведите содержимое файла **/proc/cpuinfo**. Машину какой архитектуры вы используете для работы с Linux?

9. Выведите содержимое файла **/proc/interrupts**. Каков размер этого файла? Где хранится данный файл?

10. Можете ли вы перейти в директорию **/root**? Есть ли в этой директории файлы (в том числе скрытые)?

11. Существуют ли бинарные файлы **ifconfig, fdisk, parted, shutdown** и **grup-install** в директории **/sbin**? По какой причине эти бинарные файлы размещены в директории **/sbin**, а не в директории **/bin**?

12. Является ли **/var/log** файлом или директорией? Как насчет **/var/spool**?

13. Откройте два эмулятора терминала (с помощью сочетания клавиш **Ctrl+Shift+T** в gnome-terminal) или терминала (с помощью сочетания клавиш **Ctrl+Alt+F1**, **Ctrl+Alt+F2**, ...) и выполните команду **`who am i`** в обоих. После этого попытайтесь передать слово из одного терминала в другой.

14. Прочитайте страницу руководства **`random`** и попытайтесь на основе полученной информации объяснить разницу между специальными файлами **`/dev/random`** и **`/dev/urandom`**.

9.11. Корректная процедура выполнения практического задания: дерево директорий Linux

1. Существует ли файл **`/bin/cat`**? Как насчет файлов **`/bin/dd`** и **`/bin/echo`**. Какого типа данные файлы?

```
ls /bin/cat ; file /bin/cat
ls /bin/dd ; file /bin/dd
ls /bin/echo ; file /bin/echo
```

2. Каков общий объем файлов ядра Linux (vmlinu*) в директории **`/boot`**?

```
ls -lh /boot/vm*
```

3. Создайте директорию **`~/test`**. После этого выполните следующие команды:

```
cd ~/test
dd if=/dev/zero of=zeroes.txt count=1 bs=100
od zeroes.txt
```

Утилита **`dd`** осуществит копирование одного блока (count=1) размером в 100 байт (bs=100) из специального файла **`/dev/zero`** в файл **`~/test/zeroes.txt`**. Какие пояснения вы можете дать относительно возможностей специального файла **`/dev/zero`**?

Файл **`/dev/zero`** является специальным файлом устройства Linux. Он может рассматриваться как источник нулевых байт. Вы не можете записать какие-либо данные в файл **`/dev/zero`**, но вы можете читать нулевые байты из него.

4. А теперь выполните следующую команду:

```
dd if=/dev/random of=random.txt count=1 bs=100 ; od
random.txt
```

Утилита **dd** осуществит копирование одного блока (count=1) размером в 100 байт (bs=100) из специального файла **/dev/random** в файл **~/test/random.txt**. Какие пояснения вы можете дать относительно возможностей специального файла **/dev/random**?

Файл **/dev/random** выступает в качестве генератора случайных чисел вашей машины, работающей под управлением Linux.

5. Выполните две следующие команды и обратите внимание на первый символ вывода каждой из команд.

```
ls -l /dev/sd* /dev/hd*
ls -l /dev/tty* /dev/input/mou*
```

С помощью первой команды будет выведен список файлов блочных устройств, с помощью второй - список файлов символьных устройств. Что вы можете сказать по поводу различий между символьными и блочными устройствами.

Данные всегда записываются на блочные устройства (или читаются с них) блоками. В случае жестких дисков размер блоков обычно равен 512 байтам. Символьные устройства работают как источники или приемники потоков символов (или байт). Мышь и клавиатура являются типичными символьными устройствами.

6. Используйте команду **cat** для вывода содержимого файлов **/etc/hosts** и **/etc/resolv.conf**. Что вы думаете по поводу предназначения данных файлов?

Файл **/etc/hosts** содержит имена узлов с соответствующими им IP-адресами

Файл **/etc/resolv.conf** должен содержать IP-адреса серверов имен DNS.

7. Хранятся ли какие-нибудь файлы в директории **/etc/skel/**? Не забудьте проверить наличие скрытых файлов.

Выполните команду "**ls -al /etc/skel/**". Да, в данной директории должны храниться скрытые файлы.

8. Выведите содержимое файла **/proc/cpuinfo**. Машину какой архитектуры вы используете для работы с Linux?

Данный файл должен содержать как минимум одну строку с названием модели центрального процессора производства компании Intel или какой-либо другой компании.

9. Выведите содержимое файла **/proc/interrupts**. Каков размер этого файла? Где хранится данный файл?

Размер файла равен нулю байт, но при этом файл содержит данные. Он не хранится где-либо на диске, так как в директорию /proc монтируется виртуальная файловая система, которая позволяет взаимодействовать с ядром ОС. (Ответ "файл хранится в оперативной памяти" также является верным...)

10. Можете ли вы перейти в директорию **/root**? Есть ли в этой директории файлы (в том числе скрытые)?

Попытайтесь выполнить команду "**cd /root**". Директория **/root** не доступна для чтения обычными пользователями в большинстве современных дистрибутивов Linux.

11. Существуют ли бинарные файлы **ifconfig, fdisk, parted, shutdown** и **grup-install** в директории **/sbin**? По какой причине эти бинарные файлы размещены в директории **/sbin**, а не в директории **/bin**?

Да. Так как данные бинарные файлы должны использоваться исключительно системными администраторами.

12. Является ли **/var/bin** файлом или директорией? Как насчет **/var/spool**?

По обоим путям расположены директории.

13. Откройте два эмулятора терминала (с помощью сочетания клавиш **Ctrl+Shift+T** в gnome-terminal) или терминала (с помощью сочетания клавиш **Ctrl+Alt+F1**, **Ctrl+Alt+F2**, ...) и выполните команду **who am i** в обоих. После этого попытайтесь передать слово из одного терминала в другой.

Терминал: tty-terminal: `echo Hello > /dev/tty1`

Эмулятор терминала: pts-terminal: `echo Hello > /dev/pts/1`

14. Прочитайте страницу руководства **random** и попытайтесь на основе полученной информации объяснить разницу между специальными файлами **/dev/random** и **/dev/urandom**.

`man 4 random`

Часть III. Раскрытие команд командной оболочкой

Глава 10. Команды и аргументы

В данной главе вашему вниманию представляется обзор механизма раскрытия команд **командной оболочки** (shell expansion), созданный в ходе подробного рассмотрения методик обработки **команд** и **аргументов**. Понимание принципа работы механизма раскрытия команд командной оболочки является важным ввиду того, что многие команды в вашей системе Linux подвергаются обработке и с высокой вероятностью последующей модификации средствами командной оболочки перед исполнением.

Интерфейс командной строки системы или командная оболочка, используемая в большинстве систем Linux, носит имя **bash**, которое расшифровывается как **Bourne again shell** (название "Born again shell" - "возрожденная командная оболочка" было изменено с целью упоминания автора оригинальной командной оболочки sh Стивена Борна). Командная оболочка **bash** реализует возможности командных оболочек **sh** (оригинальная командная оболочка Стивена Борна), **csh** (командная оболочка Билла Джоя с поддержкой сценариев, синтаксис которых основан на синтаксисе языка программирования C), а также **ksh** (командная оболочка Дэвида Корна).

В данной главе для демонстрации возможностей командной оболочки будет периодически использоваться команда **echo**. Команда **echo** является достаточно простой командой: она всего лишь осуществляет вывод переданных ей данных.

```
paul@laika:~$ echo Burtonville
Burtonville
paul@laika:~$ echo Smurfs are blue
Smurfs are blue
```

10.1. Аргументы

Одной из важнейших возможностей командной оболочки является возможность **обработки строк команд**. При вводе команды после приглашения командной оболочки и нажатии клавиши Enter командная оболочка приступает к обработке строки команды, разделяя ее на **аргументы**. При обработке строки команды командная оболочка может внести множество изменений в переданные вами аргументы.

Данный процесс называется раскрытием команд командной оболочки. После того, как командная оболочка заканчивает обработку и модификацию переданной строки команды, будет осуществляться непосредственное исполнение результирующей команды.

10.2. Удаление пробелов

Части строки команды, которые разделены с помощью одного или нескольких последовательно расположенных символов **пробелов** (или табуляции), рассматриваются как отдельные **аргументы**, причем все пробелы удаляются. Первым **аргументом** является сама команда, которая должна быть исполнена, остальные **аргументы** передаются этой команде. Фактически командная оболочка производит разделение вашей строки команды на один или несколько аргументов.

Это полностью объясняет эквивалентность следующих четырех команд после их раскрытия средствами **командной оболочки**.

```
[paul@RHELv4u3 ~]$ echo Hello World
Hello World
[paul@RHELv4u3 ~]$ echo Hello   World
Hello World
[paul@RHELv4u3 ~]$ echo   Hello   World
Hello World
[paul@RHELv4u3 ~]$    echo      Hello      World
Hello World
```

Команда **echo** будет выводить каждый из принятых от командной оболочки аргументов. Также команда **echo** осуществляет добавление пробелов между всеми принятыми аргументами.

10.3. Одинарные кавычки

Вы можете предотвратить удаление пробелов из строки команды, поместив ее в одинарные кавычки. Содержимое экранированной таким образом строки рассматривается как единый аргумент. В примере ниже команда **echo** принимает только один **аргумент**.

```
[paul@RHEL4b ~]$ echo 'Строка с      одинарными    кавычками'
Строка с      одинарными    кавычками
[paul@RHEL4b ~]$
```

10.4. Двойные кавычки

Вы также можете предотвратить удаление пробелов из строки команды, поместив ее в двойные кавычки. Как и в примере выше, команда **echo** примет только один **аргумент**.

```
[paul@RHEL4b ~]$ echo "Строка с      двойными    кавыч-
ками"
Строка с      двойными    кавычками
[paul@RHEL4b ~]$
```

Позднее при обсуждении **переменных** в рамках данной книги мы разберемся с важными различиями между одинарными и двойными кавычками.

10.5. Команда echo и кавычки

Строки, помещенные в кавычки, могут содержать специальные обозначения символов, идентифицируемые командной **echo** (в случае использования команды **echo -e**). В примере ниже продемонстрирована методика использования обозначения символа **\n** для вставки символа переноса строки, а также обозначения символа **\t** для вставки символа табуляции (обычно эквивалентного восьми символам пробела).

```
[paul@RHEL4b ~]$ echo -e "Строка с \nсимволом переноса
строки"
Строка с
символом переноса строки
[paul@RHEL4b ~]$ echo -e 'Строка с \nсимволом переноса
строки'
Строка с
символом переноса строки
[paul@RHEL4b ~]$ echo -e "Строка с \tсимволом табуляции"
Строка с     символом табуляции
[paul@RHEL4b ~]$ echo -e ' Строка с \tсимволом табуляции'
Строка с     символом табуляции
[paul@RHEL4b ~]$
```

Команда **echo** может генерировать и другие символы помимо символов пробелов, табуляции и переноса строки. Обратитесь к странице руководства для ознакомления со списком допустимых обозначений символов.

10.6. Команды

10.6.1. Внешние или встроенные команды?

Не все исполняемые командной оболочкой команды являются **внешними**; некоторые из них являются **встроенными**. **Внешние команды** реализованы в форме программ, представленных отдельными бинарными файлами, которые размещены в какой-либо директории файловой системы. Многие бинарные фалы, реализующие функции внешних команд, размещаются в директории **/bin** или **/sbin**. **Встроенные команды** являются неотъемлемой частью самого приложения командной оболочки.

10.6.2. Команда type

Для установления того, будет ли переданная командной оболочке команда исполнена как **внешняя команда** или как **встроенная команда**, следует использовать специальную команду **type**.

```
paul@laika:~$ type cd
cd is a shell builtin
paul@laika:~$ type cat
cat is /bin/cat
```

Как вы можете заметить, команда **cd** является **встроенной**, а команда **cat** - **внешней**.

Также вы можете использовать данную команду для установления того, является ли введенная команда **псевдонимом** команды.

```
paul@laika:~$ type ls
ls is aliased to `ls --color=auto'
```

10.6.3. Исполнение внешних команд

Некоторые команды имеют как встроенные, так и внешние реализации. В случае исполнения одной из таких команд приоритет отдается встроенной реализации. Для исполнения внешней реализации вам придется ввести полный путь к бинарному файлу, являющемуся реализацией данной команды.

```
paul@laika:~$ type -a echo
echo is a shell builtin
echo is /bin/echo
paul@laika:~$ /bin/echo Исполнение внешней реализации команды echo...
```

```
Исполнение внешней реализации команды echo...
```

10.6.4. Команда which

Команда **which** осуществляет поиск бинарных файлов в директории, заданной с помощью переменной окружения **$PATH** (переменные будут рассматриваться позднее). В примере ниже устанавливается, что **cd** является **встроенной** командой, а **ls, cp, mkdir, pwd** и **which** - **внешними** командами.

```
[root@RHEL4b ~]# which cp ls cd mkdir pwd
/bin/cp
/bin/ls
/usr/bin/which: no cd in
(/usr/kerberos/sbin:/usr/kerberos/bin:...
/bin/mkdir
/bin/pwd
```

10.7. Псевдонимы команд

10.7.1. Создание псевдонима команды

Командная оболочка позволяет вам создавать **псевдонимы** команд (aliases). Псевдонимы команд обычно используются для создания лучше запоминающихся имен для существующих команд или для упрощения передачи параметров команд.

```
[paul@RHELv4u3 ~]$ cat count.txt
один
два
три
[paul@RHELv4u3 ~]$ alias dog=tac
[paul@RHELv4u3 ~]$ dog count.txt
три
два
один
```

10.7.2. Сокращения команд

Команда создания **псевдонима** команды (alias) также может оказаться полезной в случае необходимости сокращения длины имени существующей команды.

```
paul@laika:~$ alias ll='ls -lh --color=auto'
```

```
paul@laika:~$ alias c='clear'
paul@laika:~$
```

10.7.3. Стандартные параметры команд

Псевдонимы команд могут использоваться для передачи командам стандартных параметров. Например, ниже показана методика передачи параметра **-i** по умолчанию при вводе команды **rm**.

```
[paul@RHELv4u3 ~]$ rm -i winter.txt
rm: удалить обычный файл "winter.txt"? no
[paul@RHELv4u3 ~]$ rm winter.txt
[paul@RHELv4u3 ~]$ ls winter.txt
ls: невозможно получить доступ к winter.txt: Нет такого
файла или каталога
[paul@RHELv4u3 ~]$ touch winter.txt
[paul@RHELv4u3 ~]$ alias rm='rm -i'
[paul@RHELv4u3 ~]$ rm winter.txt
rm:удалить пустой обычный файл "winter.txt"? no
[paul@RHELv4u3 ~]$
```

В некоторых дистрибутивах используются стандартные псевдонимы команд для защиты пользователей от случайного удаления файлов ('rm -i', 'mv -i', 'cp -i').

10.7.4. Просмотр объявлений псевдонимов команд

Вы можете передать один или несколько псевдонимов команд в качестве аргументов команды **alias** для вывода их объявлений. Исполнение команды без аргументов приведет к выводу полного списка используемых на данный момент псевдонимов.

```
paul@laika:~$ alias c ll
alias c='clear'
alias ll='ls -lh --color=auto'
```

10.7.5. Команда unalias

Также вы можете прекратить использование псевдонима команды, воспользовавшись командой **unalias**.

```
[paul@RHEL4b ~]$ which rm
/bin/rm
[paul@RHEL4b ~]$ alias rm='rm -i'
[paul@RHEL4b ~]$ which rm
```

```
alias rm='rm -i'
        /bin/rm
[paul@RHEL4b ~]$ unalias rm
[paul@RHEL4b ~]$ which rm
/bin/rm
[paul@RHEL4b ~]$
```

10.8. Вывод информации о раскрытии команд командной оболочкой

Вы можете активировать режим вывода информации о раскрытии команд командной оболочкой с помощью команды **set -x** и остановить вывод этой информации с помощью команды **set +x**. У вас может возникнуть потребность в использовании данной возможности как при изучении данного курса, так и в случаях, когда возникают сомнения насчет того, как командная оболочка обрабатывает переданную вами команду.

```
[paul@RHELv4u3 ~]$ set -x
++ echo -ne '\033]0;paul@RHELv4u3:~\007'
[paul@RHELv4u3 ~]$ echo $USER
+ echo paul
paul
++ echo -ne '\033]0;paul@RHELv4u3:~\007'
[paul@RHELv4u3 ~]$ echo \$USER
+ echo '$USER'
$USER
++ echo -ne '\033]0;paul@RHELv4u3:~\007'
[paul@RHELv4u3 ~]$ set +x
+ set +x
[paul@RHELv4u3 ~]$ echo $USER
paul
```

10.9. Практическое задание: команды и аргументы

1. Сколько **аргументов** передается с помощью данной строки команды (не считая самой команды)

```
touch '/etc/cron/cron.allow' 'file 42.txt' "file 33.txt"
```

2. Является ли команда **tac** встроенной?

3. Существует ли действующий псевдоним команды **rm**?

4. Прочитайте страницу руководства для команды **rm** и убедитесь в том, что вы поняли предназначение параметра **-i** этой команды. Создайте и удалите файл для проверки работоспособности параметра **-i**.

5. Выполните команду: **alias rm='rm -i'**. Проверьте работоспособность вашего псевдонима команды на тестовом файле. Работает ли он так, как ожидается?

6. Выведите список используемых на данный момент псевдонимов команд.

7a. Создайте псевдоним команды **'city'**, позволяющий вывести название вашего города.

7b. Используйте ваш псевдоним команды для того, чтобы убедиться в его работоспособности.

8. Выполните команду **set -x** для активации режима вывода информации о раскрытии каждой из команд командной оболочкой.

9. Проверьте работоспособность команды **set -x**, воспользовавшись созданными ранее псевдонимами команд **city** и **rm**.

10. Выполните команду **set +x** для прекращения вывода информации о раскрытии команд командной оболочкой.

11. Удалите созданный ранее псевдоним команды **city**.

12. В каких директориях расположены бинарные файлы, являющиеся реализациями команд **cat** и **passwd**?

13. Объясните различие между следующими командами:

```
echo
/bin/echo
```

14. Объясните различие между следующими командами:

```
echo Hello
echo -n Hello
```

15. Выведите строку "**A B C**" с двумя пробелами между буквами B и C.

16 (Необязательное задание). Создайте команду (без использования символов пробелов) для формирования следующего вывода:

```
4+4      =8
10+14    =24
```

17. Используйте команду **echo** для формирования следующего вывода:

```
??\\
```

Найдите два решения с использованием одинарных кавычек, два решения с использованием двойных кавычек и одно решение без использования кавычек (и поблагодарите Rene и Darioush из компании Google за это дополнение).

18. Используйте одну команду **echo** для вывода трех слов в трех строках.

10.10. Корректная процедура выполнения практического задания: команды и аргументы

1. Сколько **аргументов** передается с помощью данной строки команды (не считая самой команды)

```
touch '/etc/cron/cron.allow' 'file 42.txt' "file 33.txt"
```

Ответ: три аргумента

2. Является ли команда **tac** встроенной?

```
type tac
```

3. Существует ли действующий псевдоним команды **rm**?

```
alias rm
```

4. Прочитайте страницу руководства для команды **rm** и убедитесь в том, что вы поняли предназначение параметра **-i** этой команды. Создайте и удалите файл для проверки работоспособности параметра **-i**.

```
man rm
touch testfile
```

```
rm -i testfile
```

5. Выполните команду: **alias rm='rm -i'**. Проверьте работоспособность вашего псевдонима команды на тестовом файле. Работает ли он так, как ожидается?

```
touch testfile
```

```
rm testfile (должен появиться вопрос о подтверждении удаления файла)
```

6. Выведите список используемых на данный момент псевдонимов команд.

```
alias
```

7a. Создайте псевдоним команды **'city'**, позволяющий вывести название вашего города.

```
alias city='echo Antwerp'
```

7b. Используйте ваш псевдоним команды для того, чтобы убедиться в его работоспособности.

```
city (должна быть выведена строка "Antwerp")
```

8. Выполните команду **set -x** для активации режима вывода информации о раскрытии каждой из команд командной оболочкой.

```
set -x
```

9. Проверьте работоспособность команды **set -x**, воспользовавшись созданными ранее псевдонимами команд **city** и **rm**.

Командная оболочка должна вывести информацию о разрешении псевдонима, после чего выполнить команду:

```
paul@deb503:~$ set -x
paul@deb503:~$ city
+ echo antwerp
antwerp
```

10. Выполните команду **set +x** для прекращения вывода информации о раскрытии команд командной оболочкой.

```
set +x
```

11. Удалите созданный ранее псевдоним команды **city**.

```
unalias city
```

12. В каких директориях расположены бинарные файлы, являющиеся реализациями команд **cat** и **passwd**?

```
which cat (вероятно /bin/cat)
which passwd (вероятно /usr/bin/passwd)
```

13. Объясните различие между следующими командами:

```
echo
/bin/echo
```

После интерпретации команды **echo** командной оболочкой будет задействована **встроенная** реализация команды **echo**. Ввод команды **/bin/echo** приведет к исполнению **бинарного** файла **echo**, расположенного в директории **/bin**.

14. Объясните различие между следующими командами:

```
echo Hello
echo -n Hello
```

Параметр **-n** команды **echo** предназначен для предотвращения вывода символа перехода на новую строку в конце переданной строки. Команда **echo Hello** выведет в общей сложности шесть символов, а команда **echo -n Hello** - только пять символов.

(Параметр **-n** может не работать в командной оболочке Korn shell).

15. Выведите строку "**A B C**" с двумя пробелами между буквами B и C.

```
echo "A B  C"
```

16 (Необязательное задание). Создайте команду (без использования символов пробелов) для формирования следующего вывода:

```
4+4     =8
10+14   =24
```

Решение заключается в использовании символов табуляции с помощью специального обозначения **\t**.

```
echo -e "4+4\t=8" ; echo -e "10+14\t=24"
```

17. Используйте команду **echo** для формирования следующего вывода:

```
??\\
```

Найдите два решения с использованием одинарных кавычек, два решения с использованием двойных кавычек и одно решение без использования кавычек (и поблагодарите Rene и Darioush из компании Google за это дополнение).

```
echo '??\\'
echo -e '??\\\\'
echo "??\\\\"
echo -e "??\\\\\\"
echo ??\\\\
```

18. Используйте одну команду **echo** для вывода трех слов в трех строках.

```
echo -e "один \nдва \nтри"
```

Глава 11. Операторы управления

В данной главе мы будем учиться размещать более одной команды в командной строке, используя для этого **операторы управления**. Также мы кратко обсудим связанные с этими операторами параметры (**$?**) и вопросы использования аналогичных операторам специальных символов (**&**).

11.1. Точка с запятой (;)

Вы можете разместить две и более команд в одной и той же строке, разделив эти команды с помощью символа точки с запятой ;. Командная оболочка будет исследовать строку команды до момента достижения символа точки с запятой. Все аргументы перед этим символом точки с запятой будут рассматриваться как аргументы, не относящиеся к команде, находящейся после символа точки с запятой. Все команды с наборами аргументов будут выполнены последовательно, причем командная оболочка будет ожидать завершения исполнения каждой из команд перед исполнением следующей команды.

```
[paul@RHELv4u3 ~]$ echo Hello
Hello
[paul@RHELv4u3 ~]$ echo World
```

```
World
[paul@RHELv4u3 ~]$ echo Hello ; echo World
Hello
World
[paul@RHELv4u3 ~]$
```

11.2. Амперсанд (&)

В том случае, если строка команды оканчивается символом амперсанда **&**, командная оболочка не будет ожидать завершения исполнения этой команды. Сразу же после ввода команды будет выведено новое приглашение командной оболочки, а сама команда будет исполняться в фоновом режиме. В момент завершения исполнения команды в фоновом режиме вы получите соответствующее сообщение.

```
[paul@RHELv4u3 ~]$ sleep 20 &
[1] 7925
[paul@RHELv4u3 ~]$
...ожидание в течение 20 секунд...
[paul@RHELv4u3 ~]$
[1]+  Done                    sleep 20
```

Технические подробности выполняющихся при использовании рассматриваемого оператора операций приведены в разделе, посвященном **процессам**.

11.3. Символ доллара со знаком вопроса ($?)

Код завершения предыдущей команды сохраняется в переменной командной оболочки с именем **$?**. На самом деле **$?** является параметром командной оболочки, а не ее переменной, так как вы не можете присвоить значение переменной **$?**.

```
paul@debian5:~/test$ touch file1
paul@debian5:~/test$ echo $?
0
paul@debian5:~/test$ rm file1
paul@debian5:~/test$ echo $?
0
paul@debian5:~/test$ rm file1
rm: невозможно удалить "file1": Нет такого файла или ка-
талога
```

```
paul@debian5:~/test$ echo $?
1
paul@debian5:~/test$
```

11.4. Двойной амперсанд (&&)

Командная оболочка будет интерпретировать последовательность символов **&&** как **логический оператор "И"**. При использовании оператора **&&** вторая команда будет исполняться только в том случае, если исполнение первой команды успешно завершится (будет возвращен нулевой код завершения).

```
paul@barry:~$ echo первая команда && echo вторая команда
первая команда
вторая команда
paul@barry:~$ zecho первая команда && echo вторая команда
-bash: zecho: команда не найдена...
```

Во втором примере используется тот же принцип работы **логического оператора "И"**. Данный пример начинается с использования работоспособного варианта команды **cd** с последующим исполнением команды **ls**, после чего используется неработоспособный вариант команды **cd**, после которого команда **ls** не исполняется.

```
[paul@RHELv4u3 ~]$ cd gen && ls
file1  file3  File55  fileab  FileAB   fileabc
file2  File4  FileA   Fileab  fileab2
[paul@RHELv4u3 gen]$ cd gen && ls
-bash: cd: gen: Нет такого файла или каталога
```

11.5. Двойная вертикальная черта (||)

Оператор || представляет **логическую операцию "ИЛИ"**. Вторая команда исполняется только тогда, когда исполнение первой команды заканчивается неудачей (возвращается ненулевой код завершения).

```
paul@barry:~$ echo первая команда || echo вторая
команда ; echo третья команда
первая команда
третья команда
paul@barry:~$ zecho первая команда || echo вторая команда
; echo третья команда
```

```
-bash: zecho: команда не найдена...
вторая команда
третья команда
paul@barry:~$
```

В следующем примере используется тот же принцип работы **логического оператора "ИЛИ"**.

```
[paul@RHELv4u3 ~]$ cd gen || ls
[paul@RHELv4u3 gen]$ cd gen || ls
-bash: cd: gen: Нет такого файла или каталога
file1  file3  File55  fileab  FileAB   fileabc
file2  File4  FileA   Fileab  fileab2
```

11.6. Комбинирование операторов && и ||

Вы можете использовать описанные логические операторы "И" и "ИЛИ" для создания структур **условных переходов** в рамках строк команд. В данном примере используется команда **echo** для вывода информации о том, успешно ли отработала команда **rm**.

```
paul@laika:~/test$ rm file1 && echo Команда сработала! ||
echo Исполнение команды завершилось неудачей!
Команда сработала!
paul@laika:~/test$ rm file1 && echo Команда сработала! ||
echo Исполнение команды завершилось неудачей!
rm: невозможно удалить "file1": Нет такого файла или ка-
талога
Исполнение команды завершилось неудачей!
paul@laika:~/test$
```

11.7. Знак фунта (#)

Все написанное после **символа фунта** (**#**) игнорируется командной оболочкой. Это обстоятельство оказывается полезным при возникновении необходимости в написании **комментариев** в сценариях командной оболочки, причем **комментарии** ни коим образом не будут влиять на процесс исполнения команд или процесс раскрытия команд командной оболочкой.

```
paul@debian4:~$ mkdir test    # создаем директорию
paul@debian4:~$ cd test       #### переходим в эту дирек-
торию
```

```
paul@debian4:~/test$ ls       # пуста ли она ?
paul@debian4:~/test$
```

11.8. Экранирование специальных символов (\)

Символ обратного слэша **** позволяет использовать управляющие символы без их интерпретации командной оболочкой; процедура добавления данного символа перед управляющими символами называется **экранированием** символов.

```
[paul@RHELv4u3 ~]$ echo hello \; world
hello ; world
[paul@RHELv4u3 ~]$ echo hello\ \ \ world
hello   world
[paul@RHELv4u3 ~]$ echo экранирование \\\ \#\ \&\ \"\ \'
экранирование \ # & " '
[paul@RHELv4u3 ~]$ echo экранирование \\\?\*\"\'
экранирование \?*"'
```

11.8.1. Обратный слэш в конце строки

Строка команды, заканчивающаяся обратным слэшем, продолжается в следующей строке. Командная оболочка не будет интерпретировать символы перехода на новые строки и отложит исполнение операции раскрытия команды и ее исполнение до момента чтения новой строки команды без обратного слэша в конце.

```
[paul@RHEL4b ~]$ echo Данная строка команды \
> разделена на три \
> части
Данная строка команды разделена на три части
[paul@RHEL4b ~]$
```

11.9. Практическое задание: операторы управления

0. Ответ на каждый из вопросов может быть представлен с помощью единственной строки команды!

1. Какой бинарный файл исполняется при вводе команды **passwd**?

2. Какого типа данный файл?

3. Выполните команду **pwd** два раза. (Помните о пункте 0.)

4. Выполните команду **`ls`** после команды **`cd /etc`**, но только в том случае, если исполнение команды **`cd /etc`** завершилось без ошибок.

5. Выполните команду **`cd /etc`** после команды **`cd etc`**, но только в том случае, если исполнение команды **`cd etc`** завершилось ошибкой.

6. Выведите строку "**сработало**" в случае успешного завершения исполнения команды **`touch test42`** или строку "**не сработало**" в случае неудачного завершения. Все операторы должны находиться в одной строке и исполняться с привилегиями обычного пользователя (не пользователя **`root`**). Протестируйте полученную команду в вашей домашней директории и директории **`/bin/`**.

7. Выполните команду **`sleep 6`**; для чего предназначена эта команда?

8. Выполните команду **`sleep 200`** в фоновом режиме (без ожидая завершения ее исполнения).

9. Создайте строку команды, в рамках которой будет исполняться команда **`rm file55`**. Ваша строка команды должна выводить строку '**удалось**' в том случае, если файл **`file55`** был удален и строку '**не удалось**' в случае возникновения проблем.

10 (необязательное задание). Используйте команду echo для вывода строки **`"Hello World со странными' символами \ * [ } ~ \\ ."`** (включая все кавычки).

11.10. Корректная процедура выполнения практического задания: операторы управления

0. Ответ на каждый из вопросов может быть представлен с помощью единственной строки команды!

1. Какой бинарный файл исполняется при вводе команды **`passwd`**?

```
which passwd
```

2. Какого типа данный файл?

```
file /usr/bin/passwd
```

3. Выполните команду **`pwd`** два раза. (Помните о пункте 0.)

```
pwd ; pwd
```

4. Выполните команду **ls** после команды **cd /etc**, но только в том случае, если исполнение команды **cd /etc** завершилось без ошибок.

```
cd /etc && ls
```

5. Выполните команду **cd /etc** после команды **cd etc**, но только в том случае, если исполнение команды **cd etc** завершилось ошибкой.

```
cd etc || cd /etc
```

6. Выведите строку "**сработало**" в случае успешного завершения исполнения команды **touch test42** или строку "**не сработало**" в случае неудачного завершения. Все операторы должны находиться в одной строке и исполняться с привилегиями обычного пользователя (не пользователя **root**). Протестируйте полученную команду в вашей домашней директории и директории **/bin/**.

```
paul@deb503:~$ cd ; touch test42 && echo сработало ||
echo не сработало
сработало
paul@deb503:~$ cd /bin; touch test42 && echo сработало ||
echo не сработало
touch: невозможно выполнить touch для "test42": Отказано
в доступе
не сработало
```

7. Выполните команду **sleep 6**; для чего предназначена эта команда?

```
Осуществляется ожидание в течение 6 секунд
```

8. Выполните команду **sleep 200** в фоновом режиме (без ожидая завершения ее исполнения).

```
sleep 200 &
```

9. Создайте строку команды, в рамках которой будет исполняться команда **rm file55**. Ваша строка команды должна выводить строку '**удалось**' в том случае, если файл **file55** был удален и строку '**не удалось**' в случае возникновения проблем.

```
rm file55 && echo удалось || echo не удалось
```

10 (необязательное задание). Используйте команду **echo** для вывода строки **"Hello World co странными' символами \ * [} ~ \\ ."** (включая все кавычки).

```
echo \"Hello World co странными\' символами \\ \* \
[ \} \~ \\\\ \. \"
```

или

```
echo \""Hello World co странными' символами \ * [ }
~ \\ . "\"
```

Глава 12. Переменные командной оболочки

В данной главе мы познакомимся с методикой работы с **переменными окружения** с использованием командной оболочки. Эти **переменные** обычно требуются для работы приложений.

12.1. Символ доллара ($)

Еще одним важным интерпретируемым командной оболочкой символом является символ доллара **$**. Командная оболочка будет искать **переменную окружения** с именем, соответствующим размещенной после **символа доллара** строке, и заменять данный символ и имя переменной на значение этой переменной (или ни на что в том случае, если переменной не существует).

Ниже приведено несколько примеров использования переменных **$HOSTNAME, $USER, $UID, $SHELL** и **$HOME**.

```
[paul@RHELv4u3 ~]$ echo Это командная оболочка $SHELL
Это командная оболочка /bin/bash
[paul@RHELv4u3 ~]$ echo Данная командная оболочка $SHELL
используется на компьютере $HOSTNAME
Данная командная оболочка /bin/bash используется на
компьютере RHELv4u3.localdomain
[paul@RHELv4u3 ~]$ echo Идентификатор пользователя $USER
равен $UID
Идентификатор пользователя paul равен 500
[paul@RHELv4u3 ~]$ echo Моей домашней директорией являет-
ся директория $HOME
Моей домашней директорией является директория /home/paul
```

12.2. Зависимость от регистра

В данном примере показано, что имена переменных командной оболочки зависят от регистра!

```
[paul@RHELv4u3 ~]$ echo Привет $USER
Привет paul
[paul@RHELv4u3 ~]$ echo Привет $user
Привет
```

12.3. Создание переменных

В данном примере осуществляется создание переменной **$MyVar** с последующей установкой ее значения. После этого в примере используется команда **echo** для проверки значения созданной переменной.

```
[paul@RHELv4u3 gen]$ MyVar=555
[paul@RHELv4u3 gen]$ echo $MyVar
555
[paul@RHELv4u3 gen]$
```

12.4. Кавычки

Обратите внимание на то, что двойные кавычки также позволяют осуществлять раскрытие переменных в строке команды, в то время, как одинарные кавычки позволяют предотвратить такое раскрытие.

```
[paul@RHELv4u3 ~]$ MyVar=555
[paul@RHELv4u3 ~]$ echo $MyVar
555
[paul@RHELv4u3 ~]$ echo "$MyVar"
555
[paul@RHELv4u3 ~]$ echo '$MyVar'
$MyVar
```

Командная оболочка bash будет заменять переменные на их значения в строках, помещенных в двойные кавычки, но не будет осуществлять такую замену в строках, помещенных в одинарные кавычки.

```
paul@laika:~$ city=Burtonville
paul@laika:~$ echo "Сейчас мы находимся в городе $city."
Сейчас мы находимся в городе Burtonville.
```

```
paul@laika:~$ echo ' Сейчас мы находимся в городе $city.'
Сейчас мы находимся в городе $city.
```

12.5. Команда set

Вы можете использовать команду set для вывода списка переменных окружения. В системах Ubuntu и Debian команда **set** также выведет список функций командной оболочки после списка переменных командной оболочки. Поэтому для ознакомления со всеми элементами списка переменных окружения при работе с данными системами рекомендуется использовать команду **set | more**.

12.6. Команда unset

Следует использовать команду **unset** для удаления переменной из вашего окружения командной оболочки.

```
[paul@RHEL4b ~]$ MyVar=8472
[paul@RHEL4b ~]$ echo $MyVar
8472
[paul@RHEL4b ~]$ unset MyVar
[paul@RHEL4b ~]$ echo $MyVar

[paul@RHEL4b ~]$
```

12.7. Переменная окружения $PS1

Переменная окружения **$PS1** устанавливает формат приветствия вашей командной оболочки. При вводе строки форматирования вы можете использовать обратный слэш для экранирования таких специальных символов, как символ **\u**, предназначенный для вывода имени пользователя, или **\w**, предназначенный для вывода имени рабочей директории. На странице руководства командной оболочки **bash** представлен полный список специальных символов.

В примере ниже мы несколько раз изменяем значение переменной окружения **$PS1**.

```
paul@deb503:~$ PS1=приглашение
приглашение
приглашениеPS1='приглашение '
приглашение
приглашение PS1='> '
>
```

```
> PS1='\u@\h$ '
paul@deb503$
paul@deb503$ PS1='\u@\h:\W$'
paul@deb503:~$
```

Для того, чтобы избежать неисправимых ошибок, вы можете использовать зеленый цвет для приглашений командной оболочки, выводимых обычным пользователям, и красный цвет для приглашений командной оболочки, выводимых пользователю **root**. Добавьте следующие строки в ваш файл **.bashrc** для использования зеленого цвета в приглашениях, выводимых обычным пользователям.

```
# цветное приглашение командной оболочки, созданное paul
RED='\[\033[01;31m\]'
WHITE='\[\033[01;00m\]'
GREEN='\[\033[01;32m\]'
BLUE='\[\033[01;34m\]'
export PS1="${debian_chroot:+($debian_chroot)}
$GREEN\u$WHITE@$BLUE\h$WHITE\w\$ "
```

12.8. Переменная окружения $PATH

Переменная окружения **$PATH** устанавливает директории файловой системы, в которых командная оболочка ищет бинарные файлы, необходимые для исполнения команд (за исключением тех случаев, когда команда является встроенной или представлена псевдонимом команды). Данная переменная содержит список путей к директориям с символами двоеточия в качестве разделителей.

```
[[paul@RHEL4b ~]$ echo $PATH
/usr/kerberos/bin:/usr/local/bin:/bin:/usr/bin:
```

Командная оболочка не будет осуществлять поиск бинарных файлов, которые могут быть исполнены, в текущей директории. (Функция поиска исполняемых файлов в текущей директории являлась простейшим механизмом несанкционированного доступа к данным, хранящимся на компьютерах под управлением PC-DOS). В том случае, если вы хотите, чтобы командная оболочка осуществляла поиск исполняемых файлов в текущей директории, вам следует добавить символ . в конец строки, являющейся значением переменной $PATH вашей командной оболочки.

```
[paul@RHEL4b ~]$ PATH=$PATH:.
[paul@RHEL4b ~]$ echo $PATH
```

```
/usr/kerberos/bin:/usr/local/bin:/bin:/usr/bin:.
[paul@RHEL4b ~]$
```

Значение переменной $PATH вашей командной оболочки может отличаться в случае использования команды **su** вместо команды **su -**, так как последняя команда позволяет дополнительно использовать значения переменных окружения целевого пользователя. К примеру, в представленный значением переменной **$PATH** список директорий пользователя **root** обычно добавляются директории /**sbin**.

```
[paul@RHEL3 ~]$ su
Password:
[root@RHEL3 paul]# echo $PATH
/usr/local/bin:/bin:/usr/bin:/usr/X11R6/bin
[root@RHEL3 paul]# exit
[paul@RHEL3 ~]$ su -
Password:
[root@RHEL3 ~]# echo $PATH
/
usr/local/sbin:/usr/local/bin:/sbin:/bin:/usr/sbin:/usr/b
in:
[root@RHEL3 ~]#
```

12.9. Команда env

Команда **env** в случае использования без параметров выведет список экспортированных **переменных окружения**. Отличие данной команды от команды **set** с параметрами заключается в том, что команда **set** выводит список всех **переменных окружения**, включая те переменные, которые не экспортируются в дочерние командные оболочки.

Кроме того, команда **env** может также использоваться для запуска "чистой" командной оболочки (командной оболочки без наследования какого-либо окружения). Команда **env -i** позволяет очистить окружение дочерней командной оболочки.

При рассмотрении данного примера следует обратить внимание на то, что командная оболочка **bash** установит значение переменной окружения **$SHELL** при запуске.

```
[paul@RHEL4b ~]$ bash -c 'echo $SHELL $HOME $USER'
/bin/bash /home/paul paul
[paul@RHEL4b ~]$ env -i bash -c 'echo $SHELL $HOME $USER'
```

```
/bin/bash
[paul@RHEL4b ~]$
```

Вы можете использовать команду **env** для установки значения переменной **$LANG** или любой другой переменной окружения одного экземпляра командной оболочки **bash** в рамках одной команды. В примере ниже данная возможность используется для демонстрации влияния значения переменной **$LANG** на работу механизма поиска файлов по шаблонам (для получения дополнительной информации о данном механизме следует обратиться к главе, посвященной поиску файлов по шаблонам).

```
[paul@RHEL4b test]$ env LANG=C bash -c 'ls File[a-z]'
Filea  Fileb
[paul@RHEL4b test]$ env LANG=en_US.UTF-8 bash -c 'ls
File[a-z]'
Filea  FileA  Fileb  FileB
[paul@RHEL4b test]$
```

12.10. Команда export

Вы можете экспортировать переменные командной оболочки в другие командные оболочки с помощью команды **export**. В примере ниже с помощью данной команды осуществляется экспорт переменной окружения в дочерние командные оболочки.

```
[paul@RHEL4b ~]$ var3=три
[paul@RHEL4b ~]$ var4=четыре
[paul@RHEL4b ~]$ export var4
[paul@RHEL4b ~]$ echo $var3 $var4
три четыре
[paul@RHEL4b ~]$ bash
[paul@RHEL4b ~]$ echo $var3 $var4
четыре
```

При этом с помощью данной команды переменная не экспортируется в родительскую командную оболочку (ниже приведено продолжение предыдущего примера).

```
[paul@RHEL4b ~]$ export var5=пять
[paul@RHEL4b ~]$ echo $var3 $var4 $var5
четыре пять
[paul@RHEL4b ~]$ exit
```

```
exit
[paul@RHEL4b ~]$ echo $var3 $var4 $var5
три четыре
[paul@RHEL4b ~]$
```

12.11. Разграничения переменных

До текущего момента мы сталкивались с тем, что командная оболочка bash интерпретирует переменную начиная с символа доллара, продолжая интерпретацию до появления первого не алфавитно-цифрового символа, который не является символом подчеркивания. В некоторых ситуациях такое поведение может оказаться проблемой. Для решения этой проблемы могут использоваться фигурные скобки таким образом, как показано в примере ниже.

```
[paul@RHEL4b ~]$ prefix=Super
[paul@RHEL4b ~]$ echo Привет $prefixman и $prefixgirl
Привет  и
[paul@RHEL4b ~]$ echo Привет ${prefix}man и ${prefix}girl
Привет Superman и Supergirl
[paul@RHEL4b ~]$
```

12.12. Несвязанные переменные

В примере ниже представлена попытка вывода значения переменной **$MyVar**, но она не является успешной ввиду того, что переменной не существует. По умолчанию командная оболочка не будет выводить ничего в том случае, если переменная не связана (ее не существует).

```
[paul@RHELv4u3 gen]$ echo $MyVar

[paul@RHELv4u3 gen]$
```

Однако, существует параметр командной оболочки **nounset**, который вы можете использовать для генерации ошибки в том случае, если используемой переменной не существует.

```
paul@laika:~$ set -u
paul@laika:~$ echo $Myvar
bash: Myvar: unbound variable
paul@laika:~$ set +u
paul@laika:~$ echo $Myvar
```

```
paul@laika:~$
```

В командной оболочке bash команда **set -u** идентична команде **set -o nounset** и, по аналогии, команда **set +u** идентична команде **set +o nounset**.

12.13. Практическое задание: переменные командной оболочки

1. Используйте команду **echo** для вывода строки "**Привет**", после которой должно следовать ваше имя. (Используйте переменную командной оболочки bash!)

2. Создайте переменную **answer**, значение которой равно **42**.

3. Скопируйте значение переменной **$LANG** в значение переменной **$MyLANG**.

4. Выведите список используемых в данное время переменных командной оболочки.

5. Выведите список всех экспортируемых переменных командной оболочки.

6. Присутствует ли информация о вашей переменной в выводе команд **env** и **set**?

7. Уничтожьте вашу переменную **answer**.

8. Создайте две переменные и **экспортируйте** одну из них.

9. Выведите значение экспортированной переменной в дочерней интерактивной командной оболочке.

10. Создайте переменную и присвойте ей значение '**Dumb**', после чего аналогичным образом создайте другую переменную с значением '**do**'. Используйте команду `echo` и две созданные переменные для вывода слова '**Dumbledore**'.

11. Найдите список экранированных с помощью обратного слэша управляющих символов на странице руководства командной оболочки

bash. Добавьте управляющий символ в значение переменной PS1 для вывода времени в приветствии командной оболочки.

12.14. Корректная процедура выполнения практического задания: переменные командной оболочки

1. Используйте команду echo для вывода строки "**Привет**", после которой должно следовать ваше имя. (Используйте переменную командной оболочки bash!)

```
echo Привет $USER
```

2. Создайте переменную **answer**, значение которой равно **42**.

```
answer=42
```

3. Скопируйте значение переменной $LANG в значение переменной $MyLANG.

```
MyLANG=$LANG
```

4. Выведите список используемых в данное время переменных командной оболочки.

```
set
set|more в Ubuntu/Debian
```

5. Выведите список всех экспортируемых переменных командной оболочки.

```
env
```

6. Присутствует ли информация о вашей переменной в выводе команд env и set?

```
env | more
set | more
```

7. Уничтожьте вашу переменную answer.

```
unset answer
```

8. Создайте две переменные и **экспортируйте** одну из них.

```
var1=1; export var2=2
```

9. Выведите значение экспортированной переменной в дочерней интерактивной командной оболочке.

```
bash
echo $var2
```

10. Создайте переменную и присвойте ей значение '**Dumb**', после чего аналогичным образом создайте другую переменную с значением '**do**'. Используйте команду **echo** и две созданные переменные для вывода слова '**Dumbledore**'.

```
varx=Dumb; vary=do
```

Решение от Yves из компании Dexia : echo $varx'le'$vary're'

Решение от Erwin из компании Telenet : echo "$varx"le"$vary"re

11. Найдите список экранированных с помощью обратного слэша управляющих символов на странице руководства командной оболочки bash. Добавьте управляющий символ в значение переменной **PS1** для вывода времени в приветствии командной оболочки.

```
PS1='\t \u@\h \W$ '
```

Глава 13. Встраивание и параметры командных оболочек

В данной главе приводится краткий обзор методов использования **дочерних** и **встраиваемых командных оболочек**, а также описываются некоторые **параметры командных оболочек**.

13.1. Встраивание командных оболочек

Командные оболочки могут подвергаться **встраиванию** в рамках строк команд или, другими словами, в ходе разбора строк команд могут создаваться новые процессы, являющиеся копиями процесса текущей командной оболочки. Вы можете использовать переменные для доказательства факта создания новых командных оболочек. В примере ниже переменная **$var1** существует исключительно в рамках дочерней (временной) командной оболочки.

```
[paul@RHELv4u3 gen]$ echo $var1

[paul@RHELv4u3 gen]$ echo $(var1=5;echo $var1)
```

```
5
[paul@RHELv4u3 gen]$ echo $var1


[paul@RHELv4u3 gen]$
```

Вы можете осуществлять встраивание командной оболочки в рамках уже **встроенной командной** оболочки, причем сам описанный процесс называется **многоуровневым встраиванием** командных оболочек.

В примере ниже демонстрируется работа встроенной командной оболочки, функционирующей в рамках другой встроенной командной оболочки.

```
paul@deb503:~$ A='командная оболочка'
paul@deb503:~$ echo $C$B$A $(B='встроенная ';echo $C$B$A;
echo $(C='встроенная ';echo $C$B$A))
командная оболочка встроенная командная оболочка встроен-
ная встроенная командная оболочка
```

13.1.1. Обратные кавычки

Однократное встраивание командной оболочки может оказаться полезным в том случае, если необходимо избежать изменения текущей директории. В примере ниже для встраивания командной оболочки используются **обратные кавычки** вместо рассмотренного ранее символа доллара со скобками.

```
[paul@RHELv4u3 ~]$ echo `cd /etc; ls -d * | grep pass`
passwd passwd- passwd.OLD
[paul@RHELv4u3 ~]$
```

Вы можете использовать нотацию **$()** при необходимости осуществления многоуровневого встраивания командных оболочек, так как эту задачу невозможно решить с помощью **обратных кавычек**.

13.1.2. Обратные кавычки или одинарные кавычки

Размещение встраиваемых команд межу **обратными кавычками** вместо круглых скобок со знаком доллара позволяет отказаться от использования одного дополнительного символа. Однако, следует быть очень осторожным, так как обратные кавычки часто путают с одинарными кавычками. В техническом плане различия между кавычками ' и ` являются значительными.

```
[paul@RHELv4u3 gen]$ echo `var1=5;echo $var1`
```

```
5
[paul@RHELv4u3 gen]$ echo 'var1=5;echo $var1'
var1=5;echo $var1
[paul@RHELv4u3 gen]$
```

13.2. Параметры командной оболочки

И команда **set**, и команда **unset** являются встроенными командами командной оболочки. Они могут использоваться для установки значений параметров самой командной оболочки bash. Проясним это утверждение, рассмотрев следующий пример. По умолчанию командная оболочка будет рассматривать неустановленные переменные как переменные, не имеющие ассоциированных значений. После установки значения параметра **-u** командная оболочка будет рассматривать любые обращения к неустановленным переменным как ошибки. Обратитесь к странице руководства для командной оболочки bash для получения дополнительной информации.

```
[paul@RHEL4b ~]$ echo $var123

[paul@RHEL4b ~]$ set -u
[paul@RHEL4b ~]$ echo $var123
-bash: var123: unbound variable
[paul@RHEL4b ~]$ set +u
[paul@RHEL4b ~]$ echo $var123

[paul@RHEL4b ~]$
```

Для вывода списка всех параметров вашей командной оболочки с установленными значениями следует использовать команду **echo $-**. Параметр **noclobber** (или **-C**) будет описан позднее в рамках данной книги (в главе, посвященной перенаправлению потоков ввода/вывода).

```
[paul@RHEL4b ~]$ echo $-
himBH
[paul@RHEL4b ~]$ set -C ; set -u
[paul@RHEL4b ~]$ echo $-
himuBCH
[paul@RHEL4b ~]$ set +C ; set +u
[paul@RHEL4b ~]$ echo $-
```

```
himBH
[paul@RHEL4b ~]$
```

При исполнении команды **set** без параметров вашему вниманию будет представлен список всех параметров без осуществления каких-либо изменений в том случае, если командная оболочка работает в режиме **posix**. Вы можете перевести командную оболочку в режим **posix**, выполнив команду **set -o posix**.

13.3. Практическое задание: встраивание командных оболочек

1. Найдите список параметров командной оболочки на странице руководства для командной оболочки **bash**. Каковы различия между командами **set -u** и **set -o nounset**?

2. Активируйте параметр **nounset** вашей командной оболочки. Проверьте, выводится ли сообщение об ошибке при использовании несуществующих переменных.

3. Деактивируйте параметр **nounset**.

4. Выполните команды **cd /var** и **ls** в рамках встроенной командной оболочки.

Команда `echo` необходима исключительно для демонстрации вывода команды `ls`. Исключение этой команды приведет к тому, что командная оболочка попытается использовать имя первого файла в качестве команды для исполнения.

5. Создайте переменную **embvar** в рамках встроенной командной оболочки и выведите ее значение. Существует ли данная переменная в рамках вашей текущей командной оболочки.

6. Дайте пояснения относительно назначения команды "**set -x**". Может ли эта команда оказаться полезной?

7 (необязательное задание). Отредактируйте приведенную ниже команду, добавив ровно четыре символа для того, чтобы в результате ее исполнения выводилась строка "**НачалоСерединаКонец**".

```
[paul@RHEL4b ~]$ echo  Начало; echo  Середина; echo
Конец
```

8. Выведите подробный список бинарных файлов (с помощью команды **`ls -l`**) с реализацией команды **`passwd`**, воспользовавшись командой **`which`** в рамках встроенной командной оболочки.

13.4. Корректная процедура выполнения практического задания: встраивание командных оболочек

1. Найдите список параметров командной оболочки на странице руководства для командной оболочки **bash**. Каковы различия между командами **`set -u`** и **`set -o nounset`**?

Прочитайте страницу руководства для командной оболочки bash (`man bash`), найдите описание параметра `nounset` - приведенные команды идентичны.

2. Активируйте параметр **`nounset`** вашей командной оболочки. Проверьте, выводится ли сообщение об ошибке при использовании несуществующих переменных.

```
set -u
```

или

```
set -o nounset
```

3. Деактивируйте параметр **`nounset`**.

```
set +u
```

или

```
set +o nounset
```

4. Выполните команды **`cd /var`** и **`ls`** в рамках встроенной командной оболочки.

Команда **`echo`** необходима исключительно для демонстрации вывода команды **`ls`**. Исключение этой команды приведет к тому, что командная оболочка попытается использовать имя первого файла в качестве команды для исполнения.

```
echo $(cd /var ; ls)
```

5. Создайте переменную `embvar` в рамках встроенной командной оболочки и выведите ее значение. Существует ли данная переменная в рамках вашей текущей командной оболочки.

```
echo $(embvar=emb;echo $embvar) ; echo $embvar #исполне-
ние последней команды echo завершится неудачей

Переменной $embvar не существует в рамках вашей текущей
командной оболочки.
```

6. Дайте пояснения относительно назначения команды "`set -x`". Может ли эта команда оказаться полезной?

Она активирует режим вывода информации о раскрытии команд командной оболочкой, полезный в случае необходимости диагностики вашей команды.

7 (необязательное задание). Отредактируйте приведенную ниже команду, добавив ровно четыре символа для того, чтобы в результате ее исполнения выводилась строка "**НачалоСерединаКонец**".

```
[paul@RHEL4b ~]$ echo  Начало; echo  Середина; echo
Конец
echo -n Начало; echo -n Середина; echo Конец
```

8. Выведите **подробный список** бинарных файлов (с помощью команды **`ls -l`**) с реализацией команды **`passwd`**, воспользовавшись командой **`which`** в рамках встроенной командной оболочки.

```
ls -l $(which passwd)
```

Глава 14. История команд командной оболочки

В командной оболочке реализован механизм для упрощения повторного ввода команд, который в подробностях описан в данной главе.

14.1. Повторение последней выполненной команды

Для повторения последней выполненной команды в командной оболочке bash следует использовать команду **`!!`**. Данная последовательность символов носит имя **bang bang** (бэнг-бэнг).

```
paul@debian5:~/test42$ echo данная команда будет повторно
исполнена > file42.txt
paul@debian5:~/test42$ !!
данная команда будет повторно исполнена > file42.txt
paul@debian5:~/test42$
```

14.2. Повторение других команд

Вы можете повторить другие команды, воспользовавшись символом **!**, после которого должен быть введен один или несколько начальных символов команды. В результате командная оболочка повторит последнюю команду, начинающуюся с введенных символов.

```
paul@debian5:~/test42$ touch file42
paul@debian5:~/test42$ cat file42
paul@debian5:~/test42$ !to
touch file42
paul@debian5:~/test42$
```

14.3. Команда history

Для просмотра списка исполненных ранее команд следует использовать команду **history**, которая позволяет вывести информацию об истории команд, выполненных в рамках данной командной оболочки (или использовать команду **history n** для ознакомления со списком из **n** последних выполненных команд).

```
paul@debian5:~/test$ history 10
38  mkdir test
39  cd test
40  touch file1
41  echo привет > file2
42  echo Сегодня очень холодно > winter.txt
43  ls
44  ls -l
45  cp winter.txt summer.txt
46  ls -l
47  history 10
```

14.4. Команда !n

При вводе символа **!** с последующим вводом номера команды для повторения, командная оболочка выведет строку соответствующей команды и исполнит ее.

```
paul@debian5:~/test$ !43
ls
file1  file2  summer.txt  winter.txt
```

14.5. Сочетание клавиш Ctrl-r

Другой вариант методики осуществления поиска в истории команд заключается в использовании сочетания клавиш **Ctrl-r**. В примере ниже я использовал сочетание клавиш **Ctrl-r** и ввел четыре символа **apti**, после чего в файле истории команд командной оболочки был осуществлен поиск последней команды, содержащей последовательность из этих четырех символов.

```
paul@debian5:~$
(reverse-i-search)`apti': sudo aptitude install screen
```

14.6. Переменная окружения $HISTSIZE

Переменная **$HISTSIZE** устанавливает количество команд, которые будут сохраняться при работе в вашем текущем окружении. Большинство дистрибутивов устанавливает стандартное значение данной переменной, равное 500 или 1000.

```
paul@debian5:~$ echo $HISTSIZE
500
```

Вы можете изменить значение данной переменной на любое желаемое.

```
paul@debian5:~$ HISTSIZE=15000
paul@debian5:~$ echo $HISTSIZE
15000
```

14.7. Переменная окружения $HISTFILE

Переменная **$HISTFILE** указывает на файл, который содержит данные истории команд вашей командной оболочки. В командной оболочке **bash** стандартным значением данной переменной является путь к файлу **~/.bash_history**.

```
paul@debian5:~$ echo $HISTFILE
/home/paul/.bash_history
```

Данные истории команд для сессии работы с командной оболочкой сохраняются в данном файле в момент, когда вы завершаете сессию!

*При закрытии эмулятора терминала **gnome-terminal** с помощью мыши или команды **reboot** в случае работы с учетной записью пользователя **root**, история команд вашего терминала НЕ БУДЕТ сохранена.*

14.8. Переменная окружения $HISTFILESIZE

Количество команд, сохраняемых в вашем файле истории команд, может быть установлено с помощью переменной окружения **$HISTFILESIZE**.

```
paul@debian5:~$ echo $HISTFILESIZE
15000
```

14.9. Предотвращение сохранения команд

Вы можете предотвратить запись команды в **файл истории**, использовав пробел в качестве префикса команды.

```
paul@debian8:~/github$ echo abc
abc
paul@debian8:~/github$  echo def
def
paul@debian8:~/github$ echo ghi
ghi
paul@debian8:~/github$ history 3
 9501  echo abc
 9502  echo ghi
 9503  history 3
```

14.10. Регулярные выражения (дополнительная информация)

Имеется возможность использования **регулярных выражений** в случае использования символа **!** для повтора команд. В примере ниже перед повторным исполнением команды производится замена символа 1 на символ 2 в строке этой команды.

```
paul@debian5:~/test$ cat file1
paul@debian5:~/test$ !c:s/1/2
cat file2
hello
paul@debian5:~/test$
```

14.11. История команд оболочки Korn Shell (дополнительная информация)

Повторение команды при работе с командной оболочкой **Korn Shell** осуществляется похожим образом. Командная оболочка **Korn**

Shell точно так же поддерживает команду **history**, но для извлечения команд из списка ранее исполненных команд в данной командной оболочке используется символ **r**.

В примере ниже продемонстрирована работа команды **history**. Обратите внимание на различие в назначении параметра команды.

```
$ history 17
17  clear
18  echo hoi
19  history 12
20  echo world
21  history 17
```

При повторении команд с использованием символа **r** в качестве аргумента могут быть использованы как номера команд, возвращенные командой **history**, так и последовательности начальных символов команд.

```
$ r e
echo world
world
$ cd /etc
$ r
cd /etc
$
```

14.12. Практическое задание: история команд командной оболочки

1. Выполните команду **echo Ответом на вопросы о смысле жизни, вселенной и всем сущем является 42**.

2. Повторите предыдущую команду, использовав только два символа (существуют два решения!)

3. Выведите список из 5 последних исполненных вами команд.

4. Повторно выполните длинную команду **echo** из вопроса 1, воспользовавшись номерами команд, которые вы получили вместе с выводом команды из вопроса 3.

5. Сколько команд может храниться в файле истории команд в рамках текущей сессии вашей командной оболочки?

6. В какой файл сохраняются команды при при завершении сессии командной оболочки?

7. Сколько команд может быть записано в **файл истории** команд при завершении вашей текущей сессии командной оболочки?

8. Сделайте так, чтобы ваша командная оболочка bash сохранила 5000 введенных вами впоследствии команд.

9. Откройте более одной консоли (используйте сочетание клавиш **Ctrl-shift-t** в эмуляторе терминала gnome-terminal) с использованием одной и той же учетной записи пользователя. Когда данные истории команд запишутся в файл истории команд?

14.13. Корректная процедура выполнения практического задания: история команд командной оболочки

1. Выполните команду **`echo` Ответом на вопросы о смысле жизни, вселенной и всем сущем является 42**.

```
echo Ответом на вопросы о смысле жизни, вселенной и всем сущем является 42
```

2. Повторите предыдущую команду, использовав только два символа (существуют два решения!)

```
!!
```

или

```
!e
```

3. Выведите список из 5 последних исполненных вами команд.

```
paul@ubu1010:~$ history 5
 52  ls -l
 53  ls
 54  df -h | grep sda
 55  echo  Ответом на вопросы о смысле жизни, вселенной и всем сущем является 42
```

```
 56  history 5
```

В вашем случае номера команд будут отличаться.

4. Повторно выполните длинную команду **echo** из вопроса 1, воспользовавшись номерами команд, которые вы получили вместе с выводом команды из вопроса 3.

```
paul@ubu1010:~$ !56
```

5. Сколько команд может храниться в файле истории команд в рамках текущей сессии вашей командной оболочки?

```
echo $HISTSIZE
```

6. В какой файл сохраняются команды при при завершении сессии командной оболочки?

```
echo $HISTFILE
```

7. Сколько команд может быть записано в **файл истории** команд при завершении вашей текущей сессии командной оболочки?

```
echo $HISTFILESIZE
```

8. Сделайте так, чтобы ваша командная оболочка bash сохранила 5000 введенных вами впоследствии команд.

```
HISTSIZE=5000
```

9. Откройте более одной консоли (используйте сочетание клавиш **Ctrl-shift-t** в эмуляторе терминала gnome-terminal) с использованием одной и той же учетной записи пользователя. Когда данные истории команд запишутся в файл истории команд?

В момент исполнения команды **exit**.

Глава 15. Формирование списков имен файлов на основе шаблонов

Командная оболочка также ответственна за **формирование списков имен файлов на основе шаблонов** (или динамическую генерацию списков имен файлов). В данной главе даются пояснения относительно работы данного механизма.

15.1. Звездочка (*)

Звездочка ***** интерпретируется командной оболочкой как символ для генерации списка имен файлов, причем сам символ звездочки может преобразовываться в любую комбинацию символов (или даже в строку без символов). В том случае, если не задано пути к директории для формирования списка имен файлов, командная оболочка будет использовать имена файлов из текущей директории. Обратитесь к странице руководства **glob(7)** для получения дополнительной информации. (Данный вопрос также рассматривается в теме LPI 1.103.3.)

```
[paul@RHELv4u3 gen]$ ls
file1  file2  file3  File4  File55  FileA  fileab  Fileab
FileAB  fileabc
[paul@RHELv4u3 gen]$ ls File*
File4  File55  FileA  Fileab  FileAB
[paul@RHELv4u3 gen]$ ls file*
file1  file2  file3  fileab  fileabc
[paul@RHELv4u3 gen]$ ls *ile55
File55
[paul@RHELv4u3 gen]$ ls F*ile55
File55
[paul@RHELv4u3 gen]$ ls F*55
File55
[paul@RHELv4u3 gen]$
```

15.2. Знак вопроса (?)

Аналогично звездочке, знак вопроса **?** интерпретируется командной оболочкой как символ для генерации списка имен файлов, причем сам знак вопроса соответствует ровно одному символу имени файла.

```
[paul@RHELv4u3 gen]$ ls
file1  file2  file3  File4  File55  FileA  fileab  Fileab
FileAB  fileabc
[paul@RHELv4u3 gen]$ ls File?
File4  FileA
[paul@RHELv4u3 gen]$ ls Fil?4
File4
```

```
[paul@RHELv4u3 gen]$ ls Fil??
File4  FileA
[paul@RHELv4u3 gen]$ ls File??
File55  Fileab  FileAB
[paul@RHELv4u3 gen]$
```

15.3. Квадратные скобки ([ ])

Открывающаяся квадратная скобка **[** интерпретируется командной оболочкой как символ для генерации списка имен файлов, соответствующий любым из символов, находящихся между символом **[** и первым следующим за ним символом **]**. Порядок следования символов в списке между скобками не имеет значения. Каждая пара символов скобок заменяется ровно на один символ.

```
[paul@RHELv4u3 gen]$ ls
file1  file2  file3  File4  File55  FileA  fileab  Fileab
FileAB  fileabc
[paul@RHELv4u3 gen]$ ls File[5A]
FileA
[paul@RHELv4u3 gen]$ ls File[A5]
FileA
[paul@RHELv4u3 gen]$ ls File[A5][5b]
File55
[paul@RHELv4u3 gen]$ ls File[a5][5b]
File55  Fileab
[paul@RHELv4u3 gen]$ ls File[a5][5b][abcdefghijklm]
ls: невозможно получить доступ к File[a5][5b]
[abcdefghijklm]: Нет такого файла или каталога
[paul@RHELv4u3 gen]$ ls file[a5][5b][abcdefghijklm]
fileabc
[paul@RHELv4u3 gen]$
```

Также с помощью символа восклицательного знака ! вы можете исключать символы из списка, расположенного между квадратными скобками. Кроме того, у вас имеется возможность создания комбинаций из описанных выше **шаблонов**.

```
[paul@RHELv4u3 gen]$ ls
file1  file2  file3  File4  File55  FileA  fileab  Fileab
FileAB  fileabc
```

```
[paul@RHELv4u3 gen]$ ls file[a5][!Z]
fileab
[paul@RHELv4u3 gen]$ ls file[!5]*
file1  file2  file3  fileab  fileabc
[paul@RHELv4u3 gen]$ ls file[!5]?
fileab
[paul@RHELv4u3 gen]$
```

15.4. Диапазоны a-z и 0-9

Командная оболочка bash также распознает объявления диапазонов символов между квадратными скобками.

```
[paul@RHELv4u3 gen]$ ls
file1  file3  File55  fileab  FileAB   fileabc
file2  File4  FileA   Fileab  fileab2
[paul@RHELv4u3 gen]$ ls file[a-z]*
fileab  fileab2  fileabc
[paul@RHELv4u3 gen]$ ls file[0-9]
file1  file2  file3
[paul@RHELv4u3 gen]$ ls file[a-z][a-z][0-9]*
fileab2
[paul@RHELv4u3 gen]$
```

15.5. Переменная окружения $LANG и квадратные скобки

В ходе работы с командной оболочкой не стоит забывать о влиянии на процесс генерации имен файлов значения переменной окружения **LANG**. Причина этого влияния заключается в том, что в некоторых языках строчные буквы включаются в диапазон прописных букв (и наоборот).

```
paul@RHELv4u4:~/test$ ls [A-Z]ile?
file1  file2  file3  File4
paul@RHELv4u4:~/test$ ls [a-z]ile?
file1  file2  file3  File4
paul@RHELv4u4:~/test$ echo $LANG
en_US.UTF-8
paul@RHELv4u4:~/test$ LANG=C
```

```
paul@RHELv4u4:~/test$ echo $LANG
C
paul@RHELv4u4:~/test$ ls [a-z]ile?
file1  file2  file3
paul@RHELv4u4:~/test$ ls [A-Z]ile?
File4
paul@RHELv4u4:~/test$
```

В том случае, если в вашей системе устанавливается значение переменной окружения **$LC_ALL**, оно также должно быть сброшено для осуществления корректной генерации списков имен файлов.

15.6. Предотвращение формирования списков имен файлов на основе шаблонов

В примере ниже не должно быть ничего удивительного. При использовании команды **echo *** в пустой директории будет выведен символ *. А при использовании той же команды в директории с файлами будут выведены имена всех файлов.

```
paul@ubu1010:~$ mkdir test42
paul@ubu1010:~$ cd test42
paul@ubu1010:~/test42$ echo *
*
paul@ubu1010:~/test42$ touch file42 file33
paul@ubu1010:~/test42$ echo *
file33 file42
```

Формирование списков имен файлов на основе шаблонов может быть предотвращено путем помещения специальных символов в кавычки, а также экранирования этих символов таким образом, как показано в примере ниже.

```
paul@ubu1010:~/test42$ echo *
file33 file42
paul@ubu1010:~/test42$ echo \*
*
paul@ubu1010:~/test42$ echo '*'
*
paul@ubu1010:~/test42$ echo "*"
*
```

15.7. Практическое задание: формирование списков имен файлов на основе шаблонов

1. Создайте тестовую директорию и перейдите в нее.

2. Создайте следующие файлы:

```
file1
file10
file11
file2
File2
File3
file33
fileAB
filea
fileA
fileAAA
file(
file 2
```

(Имя последнего файла состоит из 6 символов, включая пробел).

3. Выведите список всех имен файлов (с помощью команды `ls`), которые начинаются с строки `file`.

4. Выведите список всех имен файлов (с помощью команды `ls`), которые начинаются с строки `File`.

5. Выведите список всех имен файлов (с помощью команды `ls`), которые начинаются с строки `file` и заканчиваются числовым символом.

6. Выведите список всех имен файлов (с помощью команды `ls`), которые начинаются с строки `file` и заканчиваются буквенным символом.

7. Выведите список всех имен файлов (с помощью команды `ls`), которые начинаются с строки `File` и имеют пятый числовой символ.

8. Выведите список всех имен файлов (с помощью команды `ls`), которые начинаются с строки `File`, имеют пятый числовой символ и никаких символов более.

9. Выведите список всех имен файлов (с помощью команды `ls`), которые начинаются с буквенного символа и заканчиваются числовым символом.

10. Выведите список всех имен файлов (с помощью команды `ls`), которые состоят ровно из пяти символов.

11. Выведите список всех имен файлов (с помощью команды `ls`), которые начинаются с символа `f` или `F` и оканчиваются символом `3` или `A`.

12. Выведите список всех имен файлов (с помощью команды `ls`), которые начинаются с символа `f` и оканчиваются числовым символом, причем вторым символом является символ `i` или `R`.

13. Выведите список всех имен файлов (с помощью команды `ls`), которые не начинаются с символа `F`.

14. Скопируйте значение переменной окружения `$LANG` в значение переменной окружения `$MyLANG`.

15. Продемонстрируйте влияние значения переменной окружения `$LANG` на вывод имен файлов с буквенными символами из диапазонов `A-Z` или `a-z`.

16. Вы получили информацию о том, что один из ваших серверов был взломан, причем взломщик, вероятнее всего, подменил бинарный файл с реализацией команды **`ls`**. Вы знаете о том, что использование команды **`echo`** не несет опасности. Можно ли заменить команду **`ls`** командой **`echo`**? Как вы будете выводить список файлов в текущей директории с помощью команды **`echo`**?

17. Существуют ли другие команды, помимо `cd`, предназначенные для изменения текущей директории?

15.8. Корректная процедура выполнения практического задания: формирование списков имен файлов на основе шаблонов

1. Создайте тестовую директорию и перейдите в нее.

```
mkdir testdir; cd testdir
```

2. Создайте следующие файлы:

```
file1
file10
file11
file2
File2
File3
file33
fileAB
filea
fileA
fileAAA
file(
file 2
```

(Имя последнего файла состоит из 6 символов, включая пробел).

```
touch file1 file10 file11 file2 File2 File3
touch file33 fileAB filea fileA fileAAA
touch "file("
touch "file 2"
```

3. Выведите список всех имен файлов (с помощью команды `ls`), которые начинаются с строки `file`.

```
ls file*
```

4. Выведите список всех имен файлов (с помощью команды `ls`), которые начинаются с строки `File`.

```
ls File*
```

5. Выведите список всех имен файлов (с помощью команды ls), которые начинаются с строки file и заканчиваются числовым символом.

```
ls file*[0-9]
```

6. Выведите список всех имен файлов (с помощью команды ls), которые начинаются с строки file и заканчиваются буквенным символом.

```
ls file*[a-z]
```

7. Выведите список всех имен файлов (с помощью команды ls), которые начинаются с строки File и имеют пятый числовой символ.

```
ls File[0-9]*
```

8. Выведите список всех имен файлов (с помощью команды ls), которые начинаются с строки File, имеют пятый числовой символ и никаких символов более.

```
ls File[0-9]
```

9. Выведите список всех имен файлов (с помощью команды ls), которые начинаются с буквенного символа и заканчиваются числовым символом.

```
ls [a-z]*[0-9]
```

10. Выведите список всех имен файлов (с помощью команды ls), которые состоят ровно из пяти символов.

```
ls ?????
```

11. Выведите список всех имен файлов (с помощью команды ls), которые начинаются с символа f или F и оканчиваются символом 3 или A.

```
ls [fF]*[3A]
```

12. Выведите список всех имен файлов (с помощью команды ls), которые начинаются с символа f и оканчиваются числовым символом, причем вторым символом является символ i или R.

```
ls f[iR]*[0-9]
```

13. Выведите список всех имен файлов (с помощью команды ls), которые не начинаются с символа F.

```
ls [!F]*
```

14. Скопируйте значение переменной окружения $LANG в значение переменной окружения $MyLANG.

```
MyLANG=$LANG
```

15. Продемонстрируйте влияние значения переменной окружения $LANG на вывод имен файлов с буквенными символами из диапазонов A-Z или a-z.

Обратитесь к примеру в книге.

16. Вы получили информацию о том, что один из ваших серверов был взломан, причем взломщик, вероятнее всего, подменил бинарный файл с реализацией команды **ls**. Вы знаете о том, что использование команды **echo** не несет опасности. Можно ли заменить команду **ls** командой **echo**? Как вы будете выводить список файлов в текущей директории с помощью команды **echo**?

```
echo *
```

17. Существуют ли другие команды, помимо cd, предназначенные для изменения текущей директории?

```
pushd popd
```

Часть IV. Программные каналы и команды

Глава 16. Перенаправление потоков ввода/вывода

Одной из мощных возможностей командной оболочки системы Unix является механизм **перенаправления потоков ввода/вывода** с возможностью задействования программных **каналов**.

В данной главе даются пояснения относительно **перенаправления** стандартных потоков ввода, вывода и ошибок.

16.1. Потоки данных stdin, stdout и stderr

Командная оболочка bash поддерживает три типа базовых потоков данных; она принимает данные из стандартного потока ввода **stdin** (поток **0**), отправляет данные в стандартный поток вывода **stdout** (поток **1**), а также отправляет сообщения об ошибках в стандартный поток ошибок **stderr** (поток **2**).

Приведенная ниже иллюстрация является графической интерпретацией этих трех потоков данных.

Клавиатура обычно служит источником данных для стандартного потока ввода **stdin**, в то время, как стандартные потоки вывода **stdout** и ошибок **stderr** используются для вывода данных. Новых пользователей Linux может смущать подобное разделение, так как не существует очевидного способа дифференцирования стандартных потоков вывода **stdout** и ошибок **stderr**. Опытные же пользователи знают о том, что разделение стандартных потоков вывода и ошибок может оказаться весьма полезным.

В следующем разделе будет рассказано о том, как осуществляется перенаправление упомянутых потоков данных.

16.2. Перенаправление стандартного потока вывода

16.2.1. Операция перенаправления потока данных stdout (>)

Перенаправление стандартного потока вывода **stdout** может быть осуществлено с помощью символа знака "**больше**". В том случае, если при разборе строки команды командная оболочка обнаруживает символ знака **>**, она удаляет данные из файла и перенаправлет данные из стандартного потока вывода в него.

Нотация **>** фактически является аббревиатурой для **1>** (в данном случае стандартный поток вывода обозначается как поток номер **1**).

```
[paul@RHELv4u3 ~]$ echo Сегодня холодно!
Сегодня холодно!
[paul@RHELv4u3 ~]$ echo Сегодня холодно! > winter.txt
[paul@RHELv4u3 ~]$ cat winter.txt
Сегодня холодно!
[paul@RHELv4u3 ~]$
```

Обратите внимание на то, что командная оболочка bash фактически **удаляет** описание операции перенаправления потока данных из строки команды перед исполнением этой команды, представленной аргументом 0. Это значит, что в случае исполнения данной команды:

```
echo привет > greetings.txt
```

командная оболочка будет рассматривать только два аргумента (echo = аргумент 0, привет = аргумент 1). Описание операции перенаправления потока данных удаляется перед началом подсчета количества аргументов.

16.2.2. Содержимое выходного файла удаляется

В том случае, если в процессе разбора строки команды командная оболочка обнаружит символ знака **>**, содержимое указанного после него файла **будет удалено!** Ввиду того, что описанная процедура

выполняется перед извлечением **аргумента 0**, содержимое файла будет удалено даже в случае неудачного исполнения команды!

```
[paul@RHELv4u3 ~]$ cat winter.txt
Сегодня холодно!
[paul@RHELv4u3 ~]$ zcho Сегодня холодно! > winter.txt
-bash: zcho: команда не найдена..
[paul@RHELv4u3 ~]$ cat winter.txt
[paul@RHELv4u3 ~]$
```

16.2.3. Параметр командной оболочки noclobber

Удаление содержимого файла при использовании оператора **>** может быть предотвращено путем установки параметра командной оболочки **noclobber**.

```
[paul@RHELv4u3 ~]$ cat winter.txt
Сегодня холодно!
[paul@RHELv4u3 ~]$ set -o noclobber
[paul@RHELv4u3 ~]$ echo Сегодня холодно! > winter.txt
-bash: winter.txt: не могу переписать уже существующий
файл
[paul@RHELv4u3 ~]$ set +o noclobber
[paul@RHELv4u3 ~]$
```

16.2.4. Нейтрализация влияния параметра командной оболочки noclobber

Влияние параметра командной оболочки **noclobber** может быть нейтрализовано с помощью оператора **>|**.

```
[paul@RHELv4u3 ~]$ set -o noclobber
[paul@RHELv4u3 ~]$ echo Сегодня холодно! > winter.txt
-bash: winter.txt: не могу переписать уже существующий
файл
[paul@RHELv4u3 ~]$ echo Сегодня очень холодно! >|
winter.txt
[paul@RHELv4u3 ~]$ cat winter.txt
Сегодня очень холодно!
[paul@RHELv4u3 ~]$
```

16.2.5. Оператор дополнения >>

Следует использовать оператор **>>** для **записи** данных из стандартного потока вывода в конец файла без предварительного удаления содержимого этого файла.

```
[paul@RHELv4u3 ~]$ echo Сегодня холодно! > winter.txt
[paul@RHELv4u3 ~]$ cat winter.txt
Сегодня холодно!
[paul@RHELv4u3 ~]$ echo Когда же наступит лето ? >>
winter.txt
[paul@RHELv4u3 ~]$ cat winter.txt
Сегодня холодно!
Когда же наступит лето ?
[paul@RHELv4u3 ~]$
```

16.3. Перенаправление стандартного потока ошибок

16.3.1. Операция перенаправления потока данных stderr (2>)

Перенаправление стандартного потока ошибок осуществляется с помощью оператора **2>**. Такое перенаправление может оказаться очень полезным для предотвращения заполнения вашего экрана сообщениями об ошибках.

В примере ниже показана методика перенаправления данных из стандартного потока вывода в файл, а данных из стандартного потока ошибок - в специальный файл устройства **/dev/null**. Запись **1>** идентична записи **>**.

```
[paul@RHELv4u3 ~]$ find / > allfiles.txt 2> /dev/null
[paul@RHELv4u3 ~]$
```

16.3.2. Операция перенаправления нескольких потоков данных 2>&1

Для перенаправления данных как из стандартного потока вывода, так и из стандартного потока ошибок в один и тот же файл следует использовать конструкцию **2>&1**.

```
[paul@RHELv4u3 ~]$ find / > allfiles_and_errors.txt 2>&1
[paul@RHELv4u3 ~]$
```

Помните о том, что последовательность операций перенаправления потоков данных имеет значение. К примеру, команда

```
ls > dirlist 2>&1
```

позволяет перенаправить как данные из стандартного потока вывода (с файловым дескриптором 1), так и данные из стандартного потока ошибок (с файловым дескриптором 2) в файл `dirlist`, в то время, как команда

```
ls 2>&1 > dirlist
```

позволяет перенаправить только данные из стандартного потока вывода в файл dirlist, так как с помощью данной команды осуществляется копирование дескриптора стандартного потока вывода в дескриптор стандартного потока ошибок перед тем, как стандартный поток вывода перенаправляется в файл `dirlist`.

16.4. Перенаправление стандартного потока вывода и программные каналы

По умолчанию вы не можете использовать утилиту `grep` для обработки данных стандартного потока ошибок **stderr** приложения при использовании программных каналов в рамках строки команды, так как данная утилита получает данные исключительно из стандартного потока вывода **stdout** приложения.

```
paul@debian7:~$ rm file42 file33 file1201 | grep file42
rm: невозможно удалить "file42": Нет такого файла или каталога
rm: невозможно удалить "file33": Нет такого файла или каталога
rm: невозможно удалить "file1201": Нет такого файла или
каталога
```

С помощью конструкции **2>&1** вы можете переправить данные из стандартного потока ошибок **stderr** в стандартный поток вывода **stdout** приложения. Это обстоятельство позволяет обрабатывать пере-

даваемые посредством программного канала данные из обоих потоков с помощью следующей команды.

```
paul@debian7:~$ rm file42 file33 file1201 2>&1 | grep
file42
rm: невозможно удалить "file42": Нет такого файла или ка-
талога
```

Вы не можете одновременно использовать конструкции **1>&2** и **2>&1** для осуществления обмена файловых дескрипторов между стандартным потоком вывода **stdout** и стандартным потоком ошибок **stderr**.

```
paul@debian7:~$ rm file42 file33 file1201 2>&1 1>&2 |
grep file42
rm: невозможно удалить "file42": Нет такого файла или ка-
талога
paul@debian7:~$ echo file42 2>&1 1>&2 | sed
's/file42/FILE42/'
FILE42
```

Вам потребуется третий поток данных для осуществления обмена файловых дескрипторов между стандартным потоком вывода **stdout** и стандартным потоком ошибок **stderr** перед символом для создания программного канала.

```
paul@debian7:~$ echo file42 3>&1 1>&2 2>&3 | sed
's/file42/FILE42/'
file42
paul@debian7:~$ rm file42 3>&1 1>&2 2>&3 | sed
's/file42/FILE42/'
rm: невозможно удалить "FILE42": Нет такого файла или ка-
талога
```

16.5. Объединение стандартных потоков вывода stdout и ошибок stderr

Конструкция **&>** позволяет объединить стандартные потоки вывода **stdout** и ошибок **stderr** в рамках одного потока данных (причем данные будут сохраняться в файле).

```
paul@debian7:~$ rm file42 &> out_and_err
paul@debian7:~$ cat out_and_err
rm: невозможно удалить "file42": Нет такого файла или ка-
талога
```

```
paul@debian7:~$ echo file42 &> out_and_err
paul@debian7:~$ cat out_and_err
file42
paul@debian7:~$
```

16.6. Перенаправление стандартного потока ввода

16.6.1. Операция перенаправления потока данных stdin (<)

Перенаправление стандартного потока ввода **stdin** осуществляется с помощью оператора **<** (являющегося краткой версией оператора **0<**).

```
[paul@RHEL4b ~]$ cat < text.txt
one
two
[paul@RHEL4b ~]$ tr 'onetw' 'ONEZZ' < text.txt
ONE
ZZO
[paul@RHEL4b ~]$
```

16.6.2. Структура << here document

Структура **here document** (иногда называемая структурой here-is-document) является механизмом для ввода данных до момента обнаружения определенной последовательности символов (обычно EOF). Маркер **EOF** может быть либо введен вручную, либо вставлен автоматически при нажатии комбинации клавиш Ctrl-D.

```
[paul@RHEL4b ~]$ cat < text.txt
> один
> два
> EOF
[paul@RHEL4b ~]$ cat text.txt
один
два
[paul@RHEL4b ~]$ cat < text.txt
> brel
> brol
[paul@RHEL4b ~]$ cat text.txt
brel
```

```
[paul@RHEL4b ~]$
```

16.6.3. Структура <<< here string

Структура **here string** может использоваться для непосредственной передачи строк команде. При использовании данной структуры достигается такой же эффект, как и при использовании команды **echo строка | команда** (но вы сможете избежать создания одного дополнительного процесса).

```
paul@ubu1110~$ base64 <<< linux-training.be
bGludXgtdHJhaW5pbmcuYmUK
paul@ubu1110~$ base64 -d <<< bGludXgtdHJhaW5pbmcuYmUK
linux-training.be
```

Для получения дополнительной информации об алгоритме **base64** следует обратиться к стандарту **rfc 3548**.

16.7. Неоднозначное перенаправление потоков ввода/вывода

Командная оболочка будет осуществлять разбор всей строки команды перед осуществлением перенаправления потоков ввода/вывода. Следующая команда является хорошо читаемой и корректной:

```
cat winter.txt > snow.txt 2> errors.txt
```

Но следующая команды также является корректной, хотя и хуже читается:

```
2> errors.txt cat winter.txt > snow.txt
```

Даже следующая команда будет прекрасно интерпретироваться командной оболочкой:

```
< winter.txt > snow.txt 2> errors.txt cat
```

16.8. Быстрая очистка содержимого файла

Так какой же самый быстрый способ очистки содержимого файла?

```
>foo
```

А какой самый быстрый способ очистки содержимого файла в случае активации параметра командной оболочки **noclobber**?

```
>|bar
```

16.9. Практическое задание: перенаправление потоков ввода/вывода

1. Активируйте параметр командной оболочки **noclobber**.

2. Проверьте, активирован ли параметр **noclobber**, повторив вызов команды вывода содержимого директории **ls** для директории **/etc** с перенаправлением данных из стандартного потока вывода в файл.

3. Какой из символов представляет параметр **noclobber** в списке всех параметров командной оболочки.

4. Деактивируйте параметр **noclobber**.

5. Убедитесь в том, что вы имеете доступ к двум командным оболочкам, открытым на одном компьютере. Создайте пустой файл **tailing.txt**. После этого выполните команду **tail -f tailing.txt**. Используйте вторую командную оболочку для добавления строки текста в этот файл. Убедитесь в том, что эта строка была выведена в первой командной оболочке.

6. Создайте файл, содержащий имена пяти людей. Используйте команду **cat** и механизм перенаправления потоков ввода/вывода для создания файла, а также структуру **here document** для завершения ввода.

16.10. Корректная процедура выполнения практического задания: перенаправление потоков ввода/вывода

1. Активируйте параметр командной оболочки **noclobber**.

```
set -o noclobber
set -C
```

2. Проверьте, активирован ли параметр **noclobber**, повторив вызов команды вывода содержимого директории **ls** для директории **/etc** с перенаправлением данных из стандартного потока вывода в файл.

```
ls /etc > etc.txt
ls /etc > etc.txt (команда не должна работать)
```

3. Какой из символов представляет параметр **noclobber** в списке всех параметров командной оболочки.

```
echo $- (параметр noclobber представлен символом C)
```

4. Деактивируйте параметр **noclobber**.

```
set +o noclobber
```

5. Убедитесь в том, что вы имеете доступ к двум командным оболочкам, открытым на одном компьютере. Создайте пустой файл **tailing.txt**. После этого выполните команду **tail -f tailing.txt**. Используйте вторую командную оболочку для добавления строки текста в этот файл. Убедитесь в том, что эта строка была выведена в первой командной оболочке.

```
paul@deb503:~$ > tailing.txt
paul@deb503:~$ tail -f tailing.txt
hello
world
```

в другой командной оболочке:

```
paul@deb503:~$ echo hello >> tailing.txt
paul@deb503:~$ echo world >> tailing.txt
```

6. Создайте файл, содержащий имена пяти людей. Используйте команду **cat** и механизм перенаправления потоков ввода/вывода для создания файла, а также структуру **here document** для завершения ввода.

```
paul@deb503:~$ cat > tennis.txt << ace
> Justine Henin
> Venus Williams
> Serena Williams
> Martina Hingis
> Kim Clijsters
> ace
paul@deb503:~$ cat tennis.txt
Justine Henin
Venus Williams
Serena Williams
Martina Hingis
Kim Clijsters
```

```
paul@deb503:~$
```

Глава 17. Фильтры

Команды, которые были реализованы для использования совместно с программными **каналами**, называются **фильтрами**. Эти **фильтры** реализуются в виде простейших программ, которые крайне эффективно выполняют одну определенную задачу. Исходя из всего вышесказанного, они могут использоваться в качестве **строительных блоков** при создании сложных конструкций.

В данной главе представлена информация о наиболее часто используемых **фильтрах**. В результате комбинирования простых команд и фильтров с использованием программных **каналов** могут быть созданы элегантные решения.

17.1. Фильтр cat

При размещении **фильтра cat** между двумя программными **каналами** не будет осуществляться какой-либо обработки передающихся через них данных (за исключением передачи этих данных из стандартного потока ввода **stdin** в стандартный поток вывода **stdout** фильтра).

```
[paul@RHEL4b pipes]$ tac count.txt | cat | cat | cat |
cat | cat
пять
четыре
три
два
один
[paul@RHEL4b pipes]$
```

17.2. Фильтр tee

Создание сложных **конвейеров** при работе с интерфейсом командной строки системы Unix является занимательным процессом, но иногда вам могут потребоваться промежуточные результаты работы конвейера. Это именно тот случай, когда фильтр **tee** может оказаться очень полезным. Фильтр **tee** перемещает данные из стандартного потока ввода **stdin** в стандартный поток вывода **stdout**, а также записывает их в файл. Исходя из вышесказанного, фильтр **tee** функционирует аналогично фильтру **cat**, за исключением того, что он имеет два идентичных вывода.

```
[paul@RHEL4b pipes]$ tac count.txt | tee temp.txt | tac
один
два
три
четыре
пять
[paul@RHEL4b pipes]$ cat temp.txt
пять
четыре
три
два
один
[paul@RHEL4b pipes]$
```

17.3. Фильтр grep

Фильтр **grep** снискал известность среди пользователей систем Unix. Наиболее простым сценарием использования фильтра **grep** является фильтрация строк текста, содержащих (или не содержащих) определенную подстроку.

```
[paul@RHEL4b pipes]$ cat tennis.txt
Amelie Mauresmo, Fra
Kim Clijsters, BEL
Justine Henin, Bel
Serena Williams, usa
Venus Williams, USA
[paul@RHEL4b pipes]$ cat tennis.txt | grep Williams
Serena Williams, usa
Venus Williams, USA
```

Вы можете выполнить эту же задачу без задействования фильтра cat.

```
[paul@RHEL4b pipes]$ grep Williams tennis.txt
Serena Williams, usa
Venus Williams, USA
```

Одним из наиболее полезных параметров фильтра grep является параметр **grep -i**, который позволяет производить фильтрацию строк без учета регистра.

```
[paul@RHEL4b pipes]$ grep Bel tennis.txt
Justine Henin, Bel
[paul@RHEL4b pipes]$ grep -i Bel tennis.txt
Kim Clijsters, BEL
Justine Henin, Bel
[paul@RHEL4b pipes]$
```

Другим полезным параметром является параметр **grep -v**, который позволяет осуществлять вывод строк, не содержащих заданную строку.

```
[paul@RHEL4b pipes]$ grep -v Fra tennis.txt
Kim Clijsters, BEL
Justine Henin, Bel
Serena Williams, usa
Venus Williams, USA
[paul@RHEL4b pipes]$
```

И, конечно же, оба описанных выше параметра могут комбинироваться для фильтрации всех строк без учета регистра и вывода тех из них, которые не содержат заданной строки.

```
[paul@RHEL4b pipes]$ grep -vi usa tennis.txt
Amelie Mauresmo, Fra
Kim Clijsters, BEL
Justine Henin, Bel
[paul@RHEL4b pipes]$
```

При использовании параметра **grep -A1** в вывод также будет добавлена одна строка, располагающаяся **после** обнаруженной строки.

```
paul@debian5:~/pipes$ grep -A1 Henin tennis.txt
Justine Henin, Bel
Serena Williams, usa
```

В случае использования параметра **grep -B1** в вывод будет добавлена одна строка, располагающаяся **до** обнаруженной строки.

```
paul@debian5:~/pipes$ grep -B1 Henin tennis.txt
```

```
Kim Clijsters, BEL
Justine Henin, Bel
```

С помощью параметра **grep -C1** (контекст) в вывод может быть добавлена одна строка, находящейся **до** обнаруженной строки, и одна строка, находящаяся **после** нее. Все три параметра (A, B и C) могут быть использованы для вывода произвольного количества дополнительных строк (например, могут быть использованы параметры A2, B4 или C20).

```
paul@debian5:~/pipes$ grep -C1 Henin tennis.txt
Kim Clijsters, BEL
Justine Henin, Bel
Serena Williams, usa
```

17.4. Фильтр cut

Фильтр **cut** может использоваться для извлечения данных из столбцов расположенных в файлах таблиц с указанием разделителя столбцов или количества байт данных в столбцах. В примере ниже фильтр **cut** используется для извлечения имени пользователя и его идентификатора из файла **/etc/passwd**. В качестве разделителя столбцов таблицы из данного файла используется символ двоеточия, при этом производится выборка значений первого и третьего столбцов.

```
[paul@RHEL4b pipes]$ cut -d: -f1,3 /etc/passwd | tail -4
Figo:510
Pfaff:511
Harry:516
Hermione:517
[paul@RHEL4b pipes]$
```

В случае использования фильтра **cut** с символом пробела в качестве разделителя вам придется экранировать этот символ пробела.

```
[paul@RHEL4b pipes]$ cut -d" " -f1 tennis.txt
Amelie
Kim
Justine
Serena
Venus
[paul@RHEL4b pipes]$
```

А в данном примере фильтр **cut** используется для вывода фрагментов строк файла **/etc/passwd** со второго по седьмой символ.

```
[paul@RHEL4b pipes]$ cut -c2-7 /etc/passwd | tail -4
igo:x:
faff:x
arry:x
ermion
[paul@RHEL4b pipes]$
```

17.5. Фильтр tr

Вы можете преобразовывать символы с помощью фильтра **tr**. В примере ниже показана процедура преобразования всех обнаруженных в потоке данных символов e в символы E.

```
[paul@RHEL4b pipes]$ cat tennis.txt | tr 'e' 'E'
AmEliE MaurEsmo, Fra
Kim ClijstErs, BEL
JustinE HEnin, BEl
SErEna Williams, usa
VEnus Williams, USA
```

В данном случае мы переводим все буквенные символы в верхний регистр, указывая два диапазона значений.

```
[paul@RHEL4b pipes]$ cat tennis.txt | tr 'a-z' 'A-Z'
AMELIE MAURESMO, FRA
KIM CLIJSTERS, BEL
JUSTINE HENIN, BEL
SERENA WILLIAMS, USA
VENUS WILLIAMS, USA
[paul@RHEL4b pipes]$
```

А здесь мы преобразовываем все символы новых строк в символы пробелов.

```
[paul@RHEL4b pipes]$ cat count.txt
один
два
три
четыре
```

```
пять
[paul@RHEL4b pipes]$ cat count.txt | tr '\n' ' '
один два три четыре пять [paul@RHEL4b pipes]$
```

Параметр **tr -s** также может использоваться для преобразования последовательностей из множества заданных символов в один символ.

```
[paul@RHEL4b pipes]$ cat spaces.txt
один    два         три
     четыре   пять  шесть
[paul@RHEL4b pipes]$ cat spaces.txt | tr -s ' '
один два три
 четыре пять шесть
[paul@RHEL4b pipes]$
```

Вы можете использовать фильтр **tr** даже для 'шифрования' текстов с использованием алгоритма **rot13**.

```
[paul@RHEL4b pipes]$ cat count.txt | tr 'a-z'
'nopqrstuvwxyzabcdefghijklm'
bar
gjb
guerr
sbhe
svir
[paul@RHEL4b pipes]$ cat count.txt | tr 'a-z' 'n-za-m'
bar
gjb
guerr
sbhe
svir
[paul@RHEL4b pipes]$
```

В последнем примере мы используем параметр **tr -d** для удаления заданного символа.

```
paul@debian5:~/pipes$ cat tennis.txt | tr -d e
Amli Maursmo, Fra
Kim Clijstrs, BEL
Justin Hnin, Bl
```

```
Srna Williams, usa
Vnus Williams, USA
```

17.6. Фильтр wc

Подсчет слов, строк и символов в файле осуществляется достаточно просто в случае использования фильтра **wc**.

```
[paul@RHEL4b pipes]$ wc tennis.txt
  5  15 100 tennis.txt
[paul@RHEL4b pipes]$ wc -l tennis.txt
5 tennis.txt
[paul@RHEL4b pipes]$ wc -w tennis.txt
15 tennis.txt
[paul@RHEL4b pipes]$ wc -c tennis.txt
100 tennis.txt
[paul@RHEL4b pipes]$
```

17.7. Фильтр sort

Фильтр **sort** по умолчанию сортирует строки файла в алфавитном порядке.

```
paul@debian5:~/pipes$ cat music.txt
Queen
Brel
Led Zeppelin
Abba
paul@debian5:~/pipes$ sort music.txt
Abba
Brel
Led Zeppelin
Queen
```

Но при этом фильтр **sort** поддерживает большое количество параметров, позволяющих повлиять на принцип его работы. В следующем примере показана методика сортировки строк на основе значений различных столбцов (столбца 1 и столбца 2 соответственно).

```
[paul@RHEL4b pipes]$ sort -k1 country.txt
Belgium, Brussels, 10
France, Paris, 60
```

```
Germany, Berlin, 100
Iran, Teheran, 70
Italy, Rome, 50
[paul@RHEL4b pipes]$ sort -k2 country.txt
Germany, Berlin, 100
Belgium, Brussels, 10
France, Paris, 60
Italy, Rome, 50
Iran, Teheran, 70
```

В примере ниже продемонстрировано различие между сортировкой в алфавитном порядке и сортировкой по числовым значениям (обе сортировки осуществлены на основе значений из третьего столбца).

```
[paul@RHEL4b pipes]$ sort -k3 country.txt
Belgium, Brussels, 10
Germany, Berlin, 100
Italy, Rome, 50
France, Paris, 60
Iran, Teheran, 70
[paul@RHEL4b pipes]$ sort -n -k3 country.txt
Belgium, Brussels, 10
Italy, Rome, 50
France, Paris, 60
Iran, Teheran, 70
Germany, Berlin, 100
```

17.8. Фильтр uniq

С помощью фильтра **uniq** вы можете удалить повторяющиеся строки из **отсортированного списка** строк.

```
paul@debian5:~/pipes$ cat music.txt
Queen
Brel
Queen
Abba
paul@debian5:~/pipes$ sort music.txt
```

```
Abba
Brel
Queen
Queen
paul@debian5:~/pipes$ sort music.txt |uniq
Abba
Brel
Queen
```

Также в случае использования параметра **-c** фильтр **uniq** может вести подсчет повторений строк.

```
paul@debian5:~/pipes$ sort music.txt |uniq -c
      1 Abba
      1 Brel
      2 Queen
```

17.9. Фильтр comm

Сравнение потоков данных (или файлов) может быть осуществлено с помощью фильтра **comm**. По умолчанию фильтр **comm** будет выводить данные в трех столбцах. В данном примере строки Abba, Cure и Queen присутствуют в списках из обоих файлов, строки Bowie и Sweet только в списке из первого файла, а строка Turnet - только в списке из второго файла.

```
paul@debian5:~/pipes$ cat > list1.txt
Abba
Bowie
Cure
Queen
Sweet
paul@debian5:~/pipes$ cat > list2.txt
Abba
Cure
Queen
Turner
paul@debian5:~/pipes$ comm list1.txt list2.txt
                Abba
```

```
Bowie
                Cure
                Queen
Sweet
        Turner
```

Вывод фильтра **comm** лучше читается в случае формирования одного столбца. При этом с помощью цифровых параметров должны быть указаны столбцы, содержимое которых не должно выводиться.

```
paul@debian5:~/pipes$ comm -12 list1.txt list2.txt
Abba
Cure
Queen
paul@debian5:~/pipes$ comm -13 list1.txt list2.txt
Turner
paul@debian5:~/pipes$ comm -23 list1.txt list2.txt
Bowie
Sweet
```

17.10. Фильтр od

Несмотря на то, что жители Европы предпочитают работать с символами ascii, компьютеры используют байты для хранения данных файлов. В примере ниже создается простой файл, после чего для показа его содержимого в форме шестнадцатеричных значений байт используется фильтр **od**.

```
paul@laika:~/test$ cat > text.txt
abcdefg
1234567
paul@laika:~/test$ od -t x1 text.txt
0000000 61 62 63 64 65 66 67 0a 31 32 33 34 35 36 37 0a
0000020
```

Содержимое этого же файла может быть выведено и в форме восьмеричных значений байт.

```
paul@laika:~/test$ od -b text.txt
0000000 141 142 143 144 145 146 147 012 061 062 063 064
065 066 067 012
0000020
```

А это содержимое рассматриваемого файла в форме символов ascii (или экранированных символов).

```
paul@laika:~/test$ od -c text.txt
0000000   a   b   c   d   e   f   g  \n   1   2   3   4
5   6   7  \n
0000020
```

17.11. Фильтр sed

Фильтр **sed** (расшифровывается как **s**tream **ed**itor - редактор потока данных) позволяет выполнять различные операции редактирования при обработке потока данных с задействованием **регулярных выражений**.

```
paul@debian5:~/pipes$ echo уровень5 | sed 's/5/42/'
уровень42
paul@debian5:~/pipes$ echo уровень5 | sed 's/уровень/пе-
реход/'
переход5
```

Следует добавить флаг регулярного выражения **g** для осуществления глобальной замены (замены всех вхождений заданной строки в строку из потока данных).

```
paul@debian5:~/pipes$ echo уровень5 уровень7 | sed
's/уровень/переход/'
переход5 уровень7
paul@debian5:~/pipes$ echo уровень5 уровень7 | sed
's/уровень/переход/g'
переход5 переход7
```

С помощью флага регулярного выражения **d** вы можете осуществить удаление строк, содержащих заданную последовательность символов, из потока данных.

```
paul@debian5:~/test42$ cat tennis.txt
Venus Williams, USA
Martina Hingis, SUI
Justine Henin, BE
Serena williams, USA
Kim Clijsters, BE
Yanina Wickmayer, BE
paul@debian5:~/test42$ cat tennis.txt | sed '/BE/d'
```

```
Venus Williams, USA
Martina Hingis, SUI
Serena williams, USA
```

17.12. Примеры конвейеров

17.12.1. Конвейер who | wc

Сколькими пользователями был осуществлен вход в систему?

```
[paul@RHEL4b pipes]$ who
root     tty1         июл 25 10:50
paul     pts/0        июл 25 09:29 (laika)
Harry    pts/1        июл 25 12:26 (barry)
paul     pts/2        июл 25 12:26 (pasha)
[paul@RHEL4b pipes]$ who | wc -l
4
```

17.12.2. Конвейер who | cut | sort

Вывод отсортированного списка пользователей, осуществивших вход в систему.

```
[paul@RHEL4b pipes]$ who | cut -d' ' -f1 | sort
Harry
paul
paul
root
```

Вывод отсортированного списка пользователей, осуществивших вход в систему, в котором имя каждого пользователя приводится лишь единожды.

```
[paul@RHEL4b pipes]$ who | cut -d' ' -f1 | sort | uniq
Harry
paul
root
```

17.12.3. Конвейер grep | cut

Вывод списка всех **учетных записей пользователей**, использующих командную оболочку bash на данном компьютере. Учетные записи пользователей будут подробно обсуждаться позднее.

```
paul@debian5:~$ grep bash /etc/passwd
root:x:0:0:root:/root:/bin/bash
```

```
paul:x:1000:1000:paul,,,:/home/paul:/bin/bash
serena:x:1001:1001::/home/serena:/bin/bash
paul@debian5:~$ grep bash /etc/passwd | cut -d: -f1
root
paul
serena
```

17.13. Практическое задание: фильтры

1. Сохраните отсортированный список пользователей командной оболочки bash в файле `bashusers.txt`.

2. Сохраните отсортированный список пользователей, осуществивших вход в систему, в файле `onlineusers.txt`.

3. Создайте список всех имен файлов из директории **`/etc`**, в которых содержится строка **`conf`**.

4. Создайте список всех имен файлов из директории **`/etc`**, в которых содержится строка **`conf`** вне зависимости от регистра символов.

5. Рассмотрите вывод утилиты **`/sbin/ifconfg`**. Создайте команду, с помощью которой будут выводиться исключительно IP-адреса и маски подсетей.

6. Создайте команду, которая позволит удалить все не относящиеся к буквенным символы из потока данных.

7. Создайте команду, которая будет принимать файл и выводить каждое слово из него в отдельной строке.

8. Разработайте систему проверки орфографии с интерфейсом командной строки. (Словарь должен находиться в директории **`/usr/share/dict/`**.)

17.14. Корректная процедура выполнения практического задания: фильтры

1. Сохраните отсортированный список пользователей командной оболочки bash в файле **`bashusers.txt`**.

```
grep bash /etc/passwd | cut -d: -f1 | sort >
bashusers.txt
```

2. Сохраните отсортированный список пользователей, осуществивших вход в систему, в файле **onlineusers.txt**.

```
who | cut -d' ' -f1 | sort > onlineusers.txt
```

3. Создайте список всех имен файлов из директории **/etc**, в которых содержится строка **conf**.

```
ls /etc | grep conf
```

4. Создайте список всех имен файлов из директории **/etc**, в которых содержится строка **conf** вне зависимости от регистра символов.

```
ls /etc | grep -i conf | sort
```

5. Рассмотрите вывод утилиты **/sbin/ifconfg**. Создайте команду, с помощью которой будут выводиться исключительно IP-адреса и маски подсетей.

```
/sbin/ifconfig | head -2 | grep 'inet ' | tr -s ' ' | cut
-d' ' -f3,5
```

6. Создайте команду, которая позволит удалить все не относящиеся к буквенным символы из потока данных.

```
paul@deb503:~$ cat text

Да, это действительно! , текст с ?&* с очень большим
количеством стра$нных# символов ;-)

paul@deb503:~$ cat text | tr -d ',!$?.*&^%#@;()-'

Да это действительно  текст с  очень большим количеством
странных символов
```

7. Создайте команду, которая будет принимать файл и выводить каждое слово из него в отдельной строке.

```
paul@deb503:~$ cat text2
сегодня очень холодно без солнца


paul@deb503:~$ cat text2 | tr ' ' '\n'
сегодня
очень
```

```
холодно
без
солнца
```

8. Разработайте систему проверки орфографии с интерфейсом командной строки. (Словарь должен находиться в директории **/usr/share/dict/**.)

```
paul@rhel ~$ echo "The zun is shining today" > text


paul@rhel ~$ cat > DICT
is
shining
sun
the
today


paul@rhel ~$ cat text | tr 'A-Z ' 'a-z\n' | sort | uniq |
comm -23 - DICT
zun
```

Также вы можете добавить решение из вопроса номер 6 для удаления не относящихся к буквенным символов и фильтр **tr -s ' '** для удаления лишних символов пробелов.

Глава 18. Стандартные инструменты систем Unix

В данной главе описаны утилиты, предназначенные для **поиска файлов** или **установления путей к файлам**, **сжатия файлов**, а также другие стандартные инструменты, которые не обсуждались ранее. Несмотря на то, что рассматриваемые инструменты технически не являются **фильтрами**, они могут использоваться в рамках **конвейеров**.

18.1. Утилита find

Утилита **find** может выполнять полезную работу в начале конвейера в случае возникновения необходимости в поиске фалов. Ниже представлено несколько примеров ее использования. Вы также можете рассмотреть возможность добавления конструкции для пере-

направления стандартного потока ошибок **2>/dev/null** в строки команд для того, чтобы ваш экран не заполнялся сообщениями об ошибках.

Команда для поиска всех файлов в директории **/etc** и сохранения списка имен найденных файлов в файле `etcfiles.txt`:

```
find /etc > etcfiles.txt
```

Команда для поиска файлов во всей файловой системе и сохранения списка имен найденных файлов в файле `allfiles.txt`:

```
find / > allfiles.txt
```

Команда для поиска файлов с расширением `.conf` в текущей директории (и всех поддиректориях):

```
find . -name "*.conf"
```

Команда для поиска обычных файлов с расширением `.conf` и явным указанием типа этих файлов (без директорий, именованных каналов и других файлов специальных типов):

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

Команда для поиска файлов специального типа, представляющих директории, с расширением `.bak`:

```
find /data -type d -name "*.bak"
```

Команда для поиска файлов, созданных позднее файла `file42.txt`:

```
find . -newer file42.txt
```

Утилита **find** также может исполнять произвольные команды для обработки каждого из найденных файлов. В данном примере мы осуществляем поиск файлов с расширением `*.odf` и копируем их в директорию `/backup/`.

```
find /data -name "*.odf" -exec cp {} /backup/ \;
```

Кроме того, утилита **find** может запрашивать подтверждение перед исполнением заданной команды для обработки каждого из найденных файлов. В данном примере файлы с расширением `*.odf` будут удаляться в том случае, если вы подтвердите необходимость исполнения этой операции для каждого из найденных файлов.

```
find /data -name "*.odf" -ok rm {} \;
```

18.2. Утилита `locate`

Утилита **locate** имеет значительное отличие от утилиты **find**, заключающееся в том, что она использует данные индексирования

файловой системы для установления путей к файлам. Несмотря на то, что данный алгоритм поиска гораздо быстрее алгоритма с обходом всех директорий файловой системы, в случае его использования данные о файловой системе в подавляющем большинстве случаев являются устаревшими. В том случае, если данные индексирования файловой системы еще не собраны, вам придется осуществить их сбор путем выполнения команды **updatedb** (в дистрибутиве Red Hat Enterprise Linux для выполнения описанной операции понадобятся привилегии пользователя root).

```
[paul@RHEL4b ~]$ locate Samba
locate: не удалось выполнить stat ()
"/var/lib/mlocate/mlocate.db": Нет такого файла или ка-
талога
[paul@RHEL4b ~]$ updatedb
updatedb: не удалось открыть временный файл для
"/var/lib/mlocate/mlocate.db"
[paul@RHEL4b ~]$ su -
Password:
[root@RHEL4b ~]# updatedb
[root@RHEL4b ~]#
```

В большинстве дистрибутивов Linux для исполнения команды **updatedb** один раз в день используется планировщик задач.

18.3. Утилита date

Утилита **date** может использоваться для вывода информации о дате, времени, часовом поясе, а также дополнительной информации.

```
paul@rhel55 ~$ date
Sat Apr 17 12:44:30 CEST 2010
```

Форматирование строки с информацией о дате может быть изменено в соответствии с вашими предпочтениями. Обратитесь к странице руководства для получения информации о других параметрах форматирования.

```
paul@rhel55 ~$ date +'%A %d-%m-%Y'
Суббота 17-04-2010
```

Во всех системах Unix для подсчета времени используется количество секунд, прошедших с 1969 года (первой секундой является первая секунда первого дня января 1970 года). Для вывода времени Unix в секундах может использоваться команда **date +%s**.

```
paul@rhel55 ~$ date +%s
1271501080
```

Когда же значение этого счетчика достигнет двух миллиардов секунд?

```
paul@rhel55 ~$ date -d '1970-01-01 + 2000000000 seconds'
Ср май 18 03:33:20 MSK 2033
```

18.4. Утилита cal

Утилита **cal** выводит календарь для текущего месяца, в котором выделен текущий день.

```
     Апрель 2010
Пн Вт Ср Чт Пт Сб Вс
              1  2  3
 4  5  6  7  8  9 10
11 12 13 14 15 16 17
18 19 20 21 22 23 24
25 26 27 28 29 30
```

Вы можете выбрать любой месяц из прошлого или будущего.

```
paul@rhel55 ~$ cal 2 1970
   Февраль 1970
Пн Вт Ср Чт Пт Сб Вс
 1  2  3  4  5  6  7
 8  9 10 11 12 13 14
15 16 17 18 19 20 21
22 23 24 25 26 27 28
```

18.5. Утилита sleep

Утилита **sleep** иногда используется в сценариях для перехода в режим ожидания на заданный промежуток времени в секундах. В данном примере показана методика реализации пятисекундного периода ожидания с использованием утилиты **sleep**.

```
paul@rhel55 ~$ sleep 5
paul@rhel55 ~$
```

18.6. Команда time

Команда **time** выводит информацию о том, сколько времени тратится на исполнение заданной команды. На исполнение команды **date** тратится совсем немного времени.

```
paul@rhel55 ~$ time date
Суб Апр 17 13:08:27 CEST 2010

real    0m0.014s
user    0m0.008s
sys     0m0.006s
```

Команда **sleep 5** исполняется в течение пяти секунд **реального** времени, но при этом на ее исполнение тратится совсем немного **процессорного времени**.

```
paul@rhel55 ~$ time sleep 5

real    0m5.018s
user    0m0.005s
sys     0m0.011s
```

А на исполнение команды **bzip2**, осуществляющей сжатие файла, тратится достаточно много **процессорного времени**.

```
paul@rhel55 ~$ time bzip2 text.txt

real    0m2.368s
user    0m0.847s
sys     0m0.539s
```

18.7. Утилиты gzip - gunzip

Пользователям всегда недостаточно дискового пространства, поэтому инструменты для сжатия данных всегда актуальны. Утилита **gzip** позволяет осуществить преобразования файлов, после которых они будут занимать меньше дискового пространства.

```
paul@rhel55 ~$ ls -lh text.txt
-rw-rw-r-- 1 paul paul 6.4M апр 17 13:11 text.txt
paul@rhel55 ~$ gzip text.txt
paul@rhel55 ~$ ls -lh text.txt.gz
```

```
-rw-rw-r-- 1 paul paul 760K апр 17 13:11 text.txt.gz
```

Вы можете получить оригинал вашего файла, воспользовавшись утилитой **gunzip**.

```
paul@rhel55 ~$ gunzip text.txt.gz
paul@rhel55 ~$ ls -lh text.txt
-rw-rw-r-- 1 paul paul 6.4M апр 17 13:11 text.txt
```

18.8. Утилиты zcat - zmore

Содержимое текстовых файлов, сжатых с помощью утилиты **gzip**, может быть просмотрено с помощью утилит **zcat** и **zmore**.

```
paul@rhel55 ~$ head -4 text.txt
/
/opt
/opt/VBoxGuestAdditions-3.1.6
/opt/VBoxGuestAdditions-3.1.6/routines.sh
paul@rhel55 ~$ gzip text.txt
paul@rhel55 ~$ zcat text.txt.gz | head -4
/
/opt
/opt/VBoxGuestAdditions-3.1.6
/opt/VBoxGuestAdditions-3.1.6/routines.sh
```

18.9. Утилиты bzip2 - bunzip2

Файлы также могут сжиматься с помощью утилиты **bzip2**, которая, несмотря на немного большие по сравнению с утилитой **gzip** затраты времени, позволяет достичь лучших показателей сжатия данных.

```
paul@rhel55 ~$ bzip2 text.txt
paul@rhel55 ~$ ls -lh text.txt.bz2
-rw-rw-r-- 1 paul paul 569K апр 17 13:11 text.txt.bz2
```

Декомпрессия результирующих файлов может быть осуществлена с помощью утилиты **bunzip2**.

```
paul@rhel55 ~$ bunzip2 text.txt.bz2
paul@rhel55 ~$ ls -lh text.txt
-rw-rw-r-- 1 paul paul 6.4M apr 17 13:11 text.txt
```

18.10. Утилиты bzcat - bzmore

И, аналогичным образом, с помощью утилит **bzcat** и **bzmore** может осуществляться вывод содержимого файлов, сжатых с использованием утилиты **bzip2**.

```
paul@rhel55 ~$ bzip2 text.txt
paul@rhel55 ~$ bzcat text.txt.bz2 | head -4
/
/opt
/opt/VBoxGuestAdditions-3.1.6
/opt/VBoxGuestAdditions-3.1.6/routines.sh
```

18.11. Практическое задание: стандартные инструменты систем Unix

1. Дайте пояснения относительно различий между следующими двумя командами. Этот вопрос является очень важным. Если вы не знаете ответа на него, вам следует снова обратиться к главе, посвященной работе с **командной оболочкой**.

```
find /data -name "*.txt"
find /data -name *.txt
```

2. Дайте пояснения относительно различий между следующими двумя командами. Будут ли обе команды работоспособны в том случае, если в директории **/data** будут находиться 200 файлов с расширением **.odf**? А что будет в том случае, если в этой директории будут находится два миллиона файлов с расширением **.odf**?

```
find /data -name "*.odf" > data_odf.txt
find /data/*.odf > data_odf.txt
```

3. Создайте команду на основе утилиты `find`, которая будет осуществлять поиск всех файлов, созданных после 30 января 2010 года.

4. Создайте команду на основе утилиты `find`, которая будет осуществлять поиск всех файлов с расширением `.odf`, созданных в сентябре 2009 года.

5. Подсчитайте количество файлов с расширением `*.conf` в директории `/etc` и всех поддиректориях.

6. Ниже приведены две команды, выполняющие одну и ту же операцию: копирование файлов с расширением `*.odf` в директорию `/backup/`. По какой причине первая команда может быть заменена на вторую? И снова данный вопрос является очень важным.

```
cp -r /data/*.odf /backup/
find /data -name "*.odf" -exec cp {} /backup/ \;
```

7. Создайте файл с именем **`loctest.txt`**. Можете ли вы установить путь к данному файлу с помощью утилиты **`locate`**? Почему это невозможно? Что нужно сделать для того, чтобы утилита `locate` возвращала путь к данному файлу.

8. Используйте утилиту `find` с параметром `-exec` для обработки всех файлов с расширением `.htm`, заключающейся в смене расширения на `.html`.

9. Выполните команду **`date`**. После этого выведите информацию о текущей дате в формате YYYY/MM/DD.

10. Выполните команду **`cal`**. Выведите календарь для 1582 и 1752 годов. Заметили ли вы что-нибудь особенное?

18.12. Корректная процедура выполнения практического задания: стандартные инструменты систем Unix

1. Дайте пояснения относительно различий между следующими двумя командами. Этот вопрос является очень важным. Если вы не знаете ответа на него, вам следует снова обратиться к главе, посвященной работе с **командной оболочкой**.

```
find /data -name "*.txt"
find /data -name *.txt
```

В том случае, если описание расширения файлов **`*.txt`** находится в двойных кавычках, командная оболочка не будет преобразовывать его. Инструмент **`find`** будет осуществлять поиск файлов с именами, оканчивающимися на **`.txt`** в директории **`/data`**.

Если же описание расширения файлов **`*.txt`** используется без двойных кавычек, командная оболочка осуществит его раскрытие (разумеется, в том случае, если в директории расположен один или несколько файлов с расширением **`.txt`**). В этом случае утилита **`find`**

может вывести отличный результат или сообщение о синтаксической ошибке.

2. Дайте пояснения относительно различий между следующими двумя командами. Будут ли обе команды работоспособны в том случае, если в директории **/data** будут находиться 200 файлов с расширением **.odf**? А что будет в том случае, если в этой директории будут находится два миллиона файлов с расширением .odf?

```
find /data -name "*.odf" > data_odf.txt
find /data/*.odf > data_odf.txt
```

В результате исполнения первой команды будет выведен список имен всех файлов с расширением **.odf** из директории **/data** и всех поддиректорий. Командная оболочка перенаправит вывод команды в файл с заданным именем.

Вторая команда выведет список, состоящий из имен всех файлов с расширением **.odf** из директории **/data**, а также имен всех файлов из поддиректорий с именами ***.odf** (директории **/data**).

При работе с директорией с двумя миллионами файлов в результате раскрытия команды будет превышено допустимое для командной оболочки количество символов. Таким образом, конец строки результирующей команды будет потерян.

3. Создайте команду на основе утилиты find, которая будет осуществлять поиск всех файлов, созданных после 30 января 2010 года.

```
touch -t 201001302359 marker_date
find . -type f -newer marker_date
Существует и другое решение:
find . -type f -newerat "20100130 23:59:59"
```

4. Создайте команду на основе утилиты find, которая будет осуществлять поиск всех файлов с расширением .odf, созданных в сентябре 2009 года.

```
touch -t 200908312359 marker_start
touch -t 200910010000 marker_end
find . -type f -name "*.odf" -newer marker_start ! -newer
marker_end
```

Символ восклицательного знака `!` перед параметром **`-newer`** может рассматриваться как логический оператор "**НЕ**".

5. Подсчитайте количество файлов с расширением `*.conf` в директории /etc и всех поддиректориях.

```
find /etc -type f -name '*.conf' | wc -l
```

6. Ниже приведены две команды, выполняющие одну и ту же операцию: копирование файлов с расширением `*.odf` в директорию `/backup/`. По какой причине первая команда может быть заменена на вторую? И снова данный вопрос является очень важным.

```
cp -r /data/*.odf /backup/
find /data -name "*.odf" -exec cp {} /backup/ \;
```

Исполнение первой команды может завершиться неудачей в том случае, если в директории находится настолько много файлов, что их имена не уместятся в одну строку команды максимального размера.

7. Создайте файл с именем **`loctest.txt`**. Можете ли вы установить путь к данному файлу с помощью утилиты **`locate`**? Почему это невозможно? Что нужно сделать для того, чтобы утилита `locate` возвращала путь к данному файлу.

Вы не можете установить путь к файлу с помощью утилиты **`locate`**, так как файл не был проиндексирован. Индексация должна осуществляться с помощью команды:

```
updatedb
```

8. Используйте утилиту `find` с параметром `-exec` для обработки всех файлов с расширением `.htm`, заключающейся в смене расширения на `.html`.

```
paul@rhel55 ~$ find . -name '*.htm'
./one.htm
./two.htm
paul@rhel55 ~$ find . -name '*.htm' -exec mv {} {}l \;
paul@rhel55 ~$ find . -name '*.htm*'
./one.html
./two.html
```

9. Выполните команду **date**. После этого выведите информацию о текущей дате в формате YYYY/MM/DD.

```
date +%Y/%m/%d
```

10. Выполните команду **cal**. Выведите календарь для 1582 и 1752 годов. Заметили ли вы что-нибудь особенное?

```
cal 1582
```

Календари отличаются в зависимости от стран, выбранных пользователями при настройке системы. Обратитесь к документу, расположенному по адресу: http://linux-training.be/files/studentfiles/dates.txt.

Глава 19. Регулярные выражения

Механизм **регулярных выражений** являются очень мощным инструментом системы Linux. Регулярные выражения могут использоваться при работе с множеством программ, таких, как `bash, vi, rename, grep, sed` и других.

В данной главе представлены базовые сведения о регулярных выражениях.

19.1. Версии синтаксисов регулярных выражений

Существуют три различных версии синтаксисов регулярных выражений:

BRE: Basic Regular Expressions (Базовый синтаксис регулярных выражений)

ERE: Extended Regular Expressions (Расширенный синтаксис регулярных выражений)

PCRE: Perl Regular Expressions (Синтаксис регулярных выражений языка программирования Perl)

В зависимости от используемого инструмента может использоваться один или несколько упомянутых синтаксисов.

К примеру, инструмент **grep** поддерживает параметр **-E**, позволяющий принудительно использовать расширенный синтаксис регулярных выражений (ERE) при разборе регулярного выражения, в то в время, как параметр **-G** позволяет принудительно использовать базовый синтаксис регулярных выражений (BRE), а параметр **-P** - синтаксис регулярных выражений языка программирования Perl (PCRE).

Учтите и то, что инструмент **grep** также поддерживает параметр **-F**, позволяющий прочитать регулярное выражение без обработки.

Инструмент **sed** также поддерживает параметры, позволяющие выбирать синтаксис регулярных выражений.

Всегда читайте страницы руководств используемых инструментов!

19.2. Утилита grep

19.2.1. Вывод строк, совпадающих с шаблоном

Утилита grep является популярным инструментом систем Linux, предназначенным для поиска строк, которые совпадают с определенным шаблоном. Ниже приведены примеры простейших **регулярных выражений**, которые могут использоваться при работе с ним.

Это содержимое используемого в примерах тестового файла. Данный файл содержит три строки (или три символа **новой строки**).

```
paul@rhel65:~$ cat names
Tania
Laura
Valentina
```

При **поиске** отдельного символа будут выводиться только те строки, которые содержат заданный символ.

```
paul@rhel65:~$ grep u names
Laura
paul@rhel65:~$ grep e names
Valentina
paul@rhel65:~$ grep i names
Tania
Valentina
```

Сравнение с шаблоном, использованным в данном примере, осуществляется очевидным образом; в том случае, если заданный символ встречается в строке, утилита **grep** выведет эту строку.

19.2.2. Объединение символов

Для поиска сочетаний символов в строках символы регулярного выражения должны объединяться аналогичным образом.

В данном примере демонстрируется принцип работы утилиты grep, в соответствии с которым регулярному выражению **ia** будет со-

ответствовать строка Tan**ia**, но не строка V**a**lentina, а регулярному выражению **in** - строка Valent**in**a, но не строка Ta**n**ia.

```
paul@rhel65:~$ grep a names
Tania
Laura
Valentina
paul@rhel65:~$ grep ia names
Tania
paul@rhel65:~$ grep in names
Valentina
paul@rhel65:~$
```

19.2.3. Один или другой символ

Как в синтаксисе PCRE, так и в синтаксисе ERE может использоваться символ создания программного канала, который в данном случае будет представлять логическую операцию "ИЛИ". В данном примере мы будем искать с помощью утилиты grep строки, в которых встречается символ **i** или символ **a**.

```
paul@debian7:~$ cat list
Tania
Laura
paul@debian7:~$ grep -E 'i|a' list
Tania
Laura
```

Обратите внимание на то, что мы используем параметр **-E** утилиты grep для принудительной интерпретации нашего регулярного выражения как выражения, использующего расширенный синтаксис регулярных выражений (ERE).

Нам придется **экранировать** символ создания программного канала в регулярном выражении, использующем базовый синтаксис регулярных выражений (BRE) для аналогичной интерпретации этого символа в качестве логической операции "ИЛИ".

```
paul@debian7:~$ grep -G 'i|a' list
paul@debian7:~$ grep -G 'i\|a' list
Tania
Laura
```

19.2.4. Одно или большее количество совпадений

Символ * соответствует нулю, одному или большему количеству вхождений предыдущего символа, а символ + - последующего символа.

```
paul@debian7:~$ cat list2
ll
lol
lool
loool
paul@debian7:~$ grep -E 'o*' list2
ll
lol
lool
loool
paul@debian7:~$ grep -E 'o+' list2
lol
lool
loool
paul@debian7:~$
```

19.2.5. Совпадение в конце строки

В следующих примерах мы будем использовать данный файл:

```
paul@debian7:~$ cat names
Tania
Laura
Valentina
Fleur
Floor
```

В двух примерах ниже показана методика использования **символа** доллара для поиска совпадения в конце строки.

```
paul@debian7:~$ grep a$ names
Tania
Laura
Valentina
paul@debian7:~$ grep r$ names
```

```
Fleur
Floor
```

19.2.6. Совпадение в начале строки

Символ вставки (**^**) позволяет осуществлять поиск совпадения в начале (или с первых символов) строки.

В данных примерах используется рассмотренный выше файл.

```
paul@debian7:~$ grep ^Val names
Valentina
paul@debian7:~$ grep ^F names
Fleur
Floor
```

Символы доллара и вставки, используемые в регулярных выражениях, называются **якорями** (anchors).

19.2.7. Разделение слов

Последовательность символов **\b** может использоваться в регулярных выражениях в качестве разделителя слов. Рассмотрим в качестве примера следующий файл:

```
paul@debian7:~$ cat text
The governer is governing.
The winter is over.
Can you get over there?
```

При простом поиске строки **over** будет выведено слишком много результирующих строк.

```
paul@debian7:~$ grep over text
The governer is governing.
The winter is over.
Can you get over there?
```

Экранирование разыскиваемых слов с помощью символов пробелов не является удачным решением (так как другие символы также могут использоваться в качестве разделителей слов). В примере ниже показана методика использования последовательности символов **\b** для поиска строк с заданным словом, а не последовательностью символов:

```
paul@debian7:~$ grep '\bover\b' text
The winter is over.
```

```
Can you get over there?
paul@debian7:~$
```

Обратите внимание на то, что утилита **grep** также поддерживает параметр **-w**, предназначенный для осуществления поиска по словам.

```
paul@debian7:~$ cat text
The governer is governing.
The winter is over.
Can you get over there?
paul@debian7:~$ grep -w over text
The winter is over.
Can you get over there?
paul@debian7:~$
```

19.2.8. Параметры утилиты grep

Иногда оказывается проще скомбинировать простое регулярное выражение с параметрами утилиты **grep**, нежели создать более сложное регулярное выражение. Эти параметры обсуждались ранее:

```
grep -i
grep -v
grep -w
grep -A5
grep -B5
grep -C5
```

19.2.9. Предотвращение раскрытия регулярного выражения командной оболочкой

Символ доллара является специальным символом как для регулярного выражения, так и для командной оболочки (вспомните о переменных командной оболочки и встраиваемых командных оболочках). Исходя из этого, рекомендуется при любых обстоятельствах экранировать регулярные выражения, так как экранирование регулярного выражения позволяет предотвратить раскрытие этого выражения командной оболочкой.

```
paul@debian7:~$ grep 'r$' names
Fleur
Floor
```

rename

19.3. Утилита rename

19.3.1. Реализации утилиты rename

В дистрибутиве Debain Linux по пути **/usr/bin/rename** расположена ссылка на сценарий **/usr/bin/prename**, устанавливаемый из пакета **perl**.

```
paul@pi ~ $ dpkg -S $(readlink -f $(which rename))
perl: /usr/bin/prename
```

В дистрибутивах, основанных на дистрибутиве Red Hat, не создается аналогичной символьной ссылки для указания на описанный сценарий (конечно же, за исключением тех случаев, когда создается символьная ссылка на сценарий, установленный вручную), поэтому в данном разделе не будет описываться реализация утилиты **rename** из дистрибутива Red Hat.

В дискуссиях об утилите rename в сети Интернет обычно происходит путаница из-за того, что решения, которые отлично работают в дистрибутиве Debian (а также Ubuntu, xubuntu, Mint, ...), не могут использоваться в дистрибутиве Red Hat (а также CentOS, Fedora, ...).

19.3.2. Пакет perl

Команда **rename** на самом деле реализована в форме сценария, использующего **регулярные выражения языка программирования perl**. С полным руководством по использованию данного сценария можно ознакомиться после ввода команды **perldoc perlrequick** (после установки пакета **perldoc**).

```
root@pi:~# aptitude install perl-doc
Следующие НОВЫЕ пакеты будут установлены:
  perl-doc
0 пакетов обновлено, 1 установлено новых, 0 пакетов отмечено для удаления, и 0 пакетов не обновлено.
Необходимо получить 8,170 kB архивов. После распаковки 13.2 MB будет занято.
Получить: 1 http://mirrordirector.raspbian.org/raspbian/ wheezy/main perl-do...
Получено 8,170 kБ в 19c (412 kБ/c)
Выбор ранее не выбранного пакета perl-doc.
```

```
(Чтение базы данных ... на данный момент установлено
67121 файл и каталог.)
Распаковывается perl-doc (из .../perl-doc_5.14.2-
21+rpi2_all.deb) ...
Adding 'diversion of /usr/bin/perldoc to
/usr/bin/perldoc.stub by perl-doc'
Обрабатываются триггеры для man-db ...
Настраивается пакет perl-doc (5.14.2-21+rpi2) ...

root@pi:~# perldoc perlrequick
```

19.3.3. Хорошо известный синтаксис

Чаще всего утилита **rename** используется для поиска файлов с именами, соответствующими определенному шаблону в форме **строки**, и замены данной строки на **другую строку**.

Обычно данное действие описывается с помощью регулярного выражения **s/строка/другая строка/**, как показано в примере:

```
paul@pi ~ $ ls
abc       allfiles.TXT  bllfiles.TXT  Scratch
tennis2.TXT
abc.conf  backup        cllfiles.TXT  temp.TXT
tennis.TXT
paul@pi ~ $ rename 's/TXT/text/' *
paul@pi ~ $ ls
abc       allfiles.text  bllfiles.text  Scratch
tennis2.text
abc.conf  backup         cllfiles.text  temp.text
tennis.text
```

А ниже приведен другой пример, в котором используется хорошо известный синтаксис утилиты **rename** для повторного изменения расширений тех же файлов:

```
paul@pi ~ $ ls
abc       allfiles.text  bllfiles.text  Scratch
tennis2.text
abc.conf  backup         cllfiles.text  temp.text
tennis.text
paul@pi ~ $ rename 's/text/txt/' *.text
paul@pi ~ $ ls
```

```
abc       allfiles.txt  bllfiles.txt  Scratch
tennis2.txt
abc.conf  backup        cllfiles.txt  temp.txt
tennis.txt
paul@pi ~ $
```

Эти два примера являются работоспособными по той причине, что используемые нами строки встречаются исключительно в расширениях файлов. Не забывайте о том, что расширения файлов не имеют значения при работе с командной оболочкой bash.

В следующем примере продемонстрирована проблема, с которой можно столкнуться при использовании данного синтаксиса.

```
paul@pi ~ $ touch atxt.txt
paul@pi ~ $ rename 's/txt/problem/' atxt.txt
paul@pi ~ $ ls
abc       allfiles.txt  backup        cllfiles.txt
temp.txt     tennis.txt
abc.conf  aproblem.txt  bllfiles.txt  Scratch
tennis2.txt
paul@pi ~ $
```

При исполнении рассматриваемой команды осуществляется замена исключительно первого вхождения разыскиваемой строки.

19.3.4. Глобальная замена

Синтаксис, использованный в предыдущем примере, может быть описан следующим образом: **s/регулярное выражение/строка для замены/**. Это описание является простым и очевидным, так как вам придется всего лишь разместить **регулярное выражение** между двумя первыми слэшами и **строку для замены** между двумя последними слэшами.

В следующем примере данный синтаксис немного расширяется благодаря добавлению **модификатора**.

```
paul@pi ~ $ rename -n 's/TXT/txt/g' aTXT.TXT
aTXT.TXT renamed as atxt.txt
paul@pi ~ $
```

Теперь используемый нами синтаксис может быть описан как **s/регулярное выражение/строка для замены/g**, где модификатор **s** обозначает операцию замены (**switch**), а модификатор **g** - сообщает о необходимости осуществления глобальной замены (**global**).

Обратите внимание на то, что в данном примере был использован параметр **-n** для вывода информации о выполняемой операции (вместо выполнения самой операции, заключающейся в непосредственном переименовании файла).

19.3.5. Замена без учета регистра

Другим **модификатором**, который может оказаться полезным, является модификатор **i**. В примере ниже показана методика замены строки на другую строку без учета регистра.

```
paul@debian7:~/files$ ls
file1.text  file2.TEXT  file3.txt
paul@debian7:~/files$ rename 's/.text/.txt/i' *
paul@debian7:~/files$ ls
file1.txt  file2.txt  file3.txt
paul@debian7:~/files$
```

19.3.6. Изменение расширений

Интерфейс командной строки Linux не имеет представления о расширениях файлов, аналогичных применяемым в операционной системе MS-DOS, но многие пользователи и приложения с графическим интерфейсом используют их.

В данном разделе приведен пример использования утилиты **rename** для изменения исключительно расширений файлов. В примере используется символ доллара для указания на то, что точкой отсчета для замены является окончание имени файла.

```
paul@pi ~ $ ls *.txt
allfiles.txt  bllfiles.txt  cllfiles.txt  really.txt.txt
temp.txt  tennis.txt
paul@pi ~ $ rename 's/.txt$/.TXT/' *.txt
paul@pi ~ $ ls *.TXT
allfiles.TXT  bllfiles.TXT    cllfiles.TXT
really.txt.TXT
temp.TXT      tennis.TXT
paul@pi ~ $
```

Обратите внимание на то, что **символ доллара** в рамках регулярного выражения обозначает **окончание строки**. Без символа доллара исполнение данной команды должно завершиться неудачей в момент обработки имени файла `really.txt.txt`.

19.4. Утилита sed

19.4.1. Редактор потока данных

Редактор потока данных (stream editor) или, для краткости, утилита **sed**, использует **регулярные выражения** для модификации потока данных.

В данном примере утилита **sed** используется для замены строки.

```
echo Понедельник | sed 's/Понедель/Втор/'
Вторник
```

Слэши могут быть заменены на некоторые другие символы, которые могут оказаться более удобными и повысить читаемость команды в ряде случаев.

```
echo Понедельник | sed 's:Понедель:Втор:'
Вторник
echo Понедельник | sed 's_Понедель_Втор_'
Вторник
echo Понедельник | sed 's|Понедель|Втор|'
Вторник
```

19.4.2. Интерактивный редактор

Несмотря на то, что утилита **sed** предназначена для обработки потоков данных, она также может использоваться для интерактивной обработки файлов.

```
paul@debian7:~/files$ echo Понедельник > today
paul@debian7:~/files$ cat today
Понедельник
paul@debian7:~/files$ sed -i 's/Понедель/Втор/' today
paul@debian7:~/files$ cat today
Вторник
```

19.4.3. Простые обратные ссылки

Символ **амперсанда** может использоваться для ссылки на искомую (и найденную) строку.

В данном примере **амперсанд** используется для удвоения количества найденных строк.

```
echo Понедельник | sed 's/Понедель/&&/'
ПонедельПонедельник
```

```
echo Понедельник | sed 's/ник/&&/'
Понедельникник
```

19.4.4. Обратные ссылки

Круглые скобки используются для группировки частей регулярного выражения, на которые впоследствии могут быть установлены ссылки.

Рассмотрите следующий пример:

```
paul@debian7:~$ echo Sunday | sed 's_\(Sun\)_\1ny_'
Sunnyday
paul@debian7:~$ echo Sunday | sed 's_\(Sun\)_\1ny \1_'
Sunny Sunday
```

19.4.5. Точка для обозначения любого символа

В **регулярном выражении** простой символ точки может обозначать любой символ.

```
paul@debian7:~$ echo 2014-04-01 | sed 's/............/YYYY-
MM-DD/'
YYYY-MM-DD
paul@debian7:~$ echo abcd-ef-gh | sed 's/............/YYYY-
MM-DD/'
YYYY-MM-DD
```

19.4.6. Множественные обратные ссылки

В случае использования более чем одной пары **круглых скобок**, ссылка на каждую из них может быть осуществлена путем использования последовательных числовых значений.

```
paul@debian7:~$ echo 2014-04-01 | sed 's/\(....\)-\
(..\)-\(..\)/\1+\2+\3/'
2014+04+01
paul@debian7:~$ echo 2014-04-01 | sed 's/\(....\)-\
(..\)-\(..\)/\3:\2:\1/'
01:04:2014
```

Данная возможность называется **группировкой** (grouping).

19.4.7. Пробел

Последовательность символов `\s` может использоваться для ссылки на такой символ, как символ пробела или табуляции.

В данном примере осуществляется глобальный поиск последовательностей символов пробелов (\s), которые заменяются на 1 символ пробела.

```
paul@debian7:~$ echo -e 'сегодня\tтеплый\tдень'
сегодня теплый  день
paul@debian7:~$ echo -e 'сегодня\tтеплый\tдень' | sed
's_\s_ _g'
сегодня теплый день
```

19.4.8. Необязательные вхождения

Символ знака вопроса указывает на то, что предыдущий символ является **необязательным**.

В примере ниже осуществляется поиск последовательности из трех символов o, причем третий символ o является необязательным.

```
paul@debian7:~$ cat list2
ll
lol
lool
loool
paul@debian7:~$ grep -E 'ooo?' list2
lool
loool
paul@debian7:~$ cat list2 | sed 's/ooo\?/A/'
ll
lol
lAl
lAl
```

19.4.9. Ровно n повторений

Вы можете указать точное количество повторений предыдущего символа.

В данном примере осуществляется поиск строк с ровно тремя символами o.

```
paul@debian7:~$ cat list2
ll
lol
lool
```

```
loool
paul@debian7:~$ grep -E 'o{3}' list2
loool
paul@debian7:~$ cat list2 | sed 's/o\{3\}/A/'
ll
lol
lool
lAl
paul@debian7:~$
```

19.4.10. От n до m повторений

А в данном примере мы четко указываем, что символ должен повторяться от минимального (2) до максимального (3) количества раз.

```
paul@debian7:~$ cat list2
ll
lol
lool
loool
paul@debian7:~$ grep -E 'o{2,3}' list2
lool
loool
paul@debian7:~$ grep 'o\{2,3\}' list2
lool
loool
paul@debian7:~$ cat list2 | sed 's/o\{2,3\}/A/'
ll
lol
lAl
lAl
paul@debian7:~$
```

19.5. История командной оболочки bash

Командная **оболочка bash** также может интерпретировать некоторые регулярные выражения.

В данном примере показана методика манипуляций с символом восклицательного знака в рамках маски поиска в истории командной оболочки bash.

```
paul@debian7:~$ mkdir hist
paul@debian7:~$ cd hist/
paul@debian7:~/hist$ touch file1 file2 file3
paul@debian7:~/hist$ ls -l file1
-rw-r--r-- 1 paul paul 0 апр 15 22:07 file1
paul@debian7:~/hist$ !l
ls -l file1
-rw-r--r-- 1 paul paul 0 апр 15 22:07 file1
paul@debian7:~/hist$ !l:s/1/3
ls -l file3
-rw-r--r-- 1 paul paul 0 Апр 15 22:07 file3
paul@debian7:~/hist$
```

Данная методика также работает в случае использования чисел при чтении истории команд командной оболочки bash.

```
paul@debian7:~/hist$ history 6
 2089  mkdir hist
 2090  cd hist/
 2091  touch file1 file2 file3
 2092  ls -l file1
 2093  ls -l file3
 2094  history 6
paul@debian7:~/hist$ !2092
ls -l file1
-rw-r--r-- 1 paul paul 0 апр 15 22:07 file1
paul@debian7:~/hist$ !2092:s/1/2
ls -l file2
-rw-r--r-- 1 paul paul 0 апр 15 22:07 file2
paul@debian7:~/hist$
```

Часть V. Текстовый редактор vi

Глава 20. Начальные сведения о текстовом редакторе `vi`

Текстовый редактор **`vi`** установлен практически в каждой системе Unix. При установке дистрибутивов Linux очень часто устанавливается текстовый редактор **`vim`** (**улучшенная версия `vi` - vi improved**), который является аналогичным. Каждый системный администратор должен обладать знаниями относительно работы с текстовым редактором **`vi(m)`**, так как данный редактор является простым инструментом, позволяющим решить возникающие проблемы.

Текстовый редактор **`vi`** не является интуитивным, но после того, как вы научитесь с ним работать, он станет еще одним очень мощным инструментом из вашего арсенала. В состав большинства дистрибутивов также включено приложение **`vimtutor`**, которое представляет собой 45-минутный урок по использованию редактора **`vi(m)`**.

20.1. Режимы ввода команд и ввода текста

Текстовый редактор `vi` начинает работу в **режиме ввода команд**. При работе в этом режиме вам предоставляется возможность выполнения поддерживаемых редактором команд. Некоторые из команд позволят вам перейти в **режим ввода текста**. При работе в этом режиме вы можете редактировать текстовые документы. Клавиша **Escape** вернет вас в режим ввода команд.

Таблица 20.1. Переход в режим ввода команд

Клавиша	Действие
Esc	Перевод редактора vi(m) в режим ввода команд.

20.2. Начало редактирования текста (a A i I o O)

Различие между командами A i I o и O заключается в начальной позиции ввода текста. Команда a позволяет продолжить ввода текста после текущего символа, а команда A - после окончания текущей строки. Команда i позволяет начать ввод текста перед текущим символом, а команда I - в начале текущей строки. Команда o позволяет создать новую строку после текущей строки, а команда O - перед текущей строкой.

Таблица 20.2. Переход в режим ввода текста

Команда	Действие
a	Начало ввода текста после текущего символа
A	Начало ввода текста в конце текущей строки
i	Начало ввода текста перед текущим символом
I	Начало ввода текста в начале текущей строки
o	Начало ввода текста в новой строке, созданной после текущей строки
O	Начало ввода текста в новой строке, созданной перед текущей строкой

20.3. Замена и удаление символа (r x X)

При работе в режиме ввода команд (в том случае, если вы нажмете клавишу Escape несколько раз, не произойдет ничего страшного) вы можете использовать клавишу x для удаления текущего символа. Символ X в верхнем регистре (или сочетание клавиш Shift+x) позволяет удалить символ слева от курсора. Также при работе в режиме ввода команд вы можете использовать клавишу r для замены одного единственного символа. В случае использования клавиши r вы будете перемещены в режим ввода текста для осуществления одного нажатия на клавишу, после чего вы будете немедленно перемещены в режим ввода команд.

Таблица 20.3. Замена и удаление символа

Команда	Действие
x	Удаление символа, выделенного с помощью курсора
X	Удаление символа перед курсором
r	Замена символа, выделенного с помощью курсора
p	Вставка символа после курсора (в данном случае будет осуществляться вставка последнего удаленного символа)
xp	Замена местами двух символов

20.4. Отмена и повторение действий (u .)

При работе в режиме ввода команд вы можете отменять ошибочные действия с помощью команды u. Также вы можете повторять эти ошибочные действия с помощью команды . (другими словами, с помо-

щью команды . будет осуществляться повторное выполнение вашей последней команды).

Таблица 20.4. Отмена и повторение действий

Команда	Действие
u	Отменить последнее действие
.	Повторить последнее действие

20.5. Перенос, копирование и вставка строки (dd yy pP)

При работе в режиме ввода команд с помощью команды dd может осуществляться перенос текущей строки. Команда yy позволяет копировать текущую строку. В результате вы сможете осуществить вставку последней скопированной или перенесенной строки после (p) или до (P) текущей строки.

Таблица 20.5. Перенос, копирование и вставка строки

Команда	Действие
dd	Перенос текущей строки
yy	Копирование текущей строки (yank yank)
p	Вставка строки после текущей строки
P	Вставка строки до текущей строки

20.6. Перенос, копирование и вставка строк (3dd 2yy)

При работе в режиме ввода команд перед вводом команд dd или yy вы можете ввести число, соответствующее необходимому количеству повторений команды. Таким образом, с помощью команды 5dd могут быть перемещены 5 строк, а с помощью команды 4yy могут быть скопированы 4 строки. При этом исполнение последней команды будет сопровождаться уведомлением от vi в нижнем левом углу с текстом "4 line yanked".

Таблица 20.6. Перемещение, копирование и вставка строк

Команда	Действие
3dd	Перемещение трех строк
4yy	Копирование четырех строк

20.7. Переход в начало и конец строки (0 или ^ и $)

При работе в режиме ввода команд команда 0 или ^ переместит ваш курсор в начало текущей строки, а команда $ - в конец текущей строки. Вы можете добавлять символы 0 и $ к команде d, в результате чего команда d0 позволит удалить все символы, находящиеся между текущим символом и символом в начале строки. Таким же образом команда d$ позволит удалить все символы в диапазоне от текущего символа до символа в конце строки. Аналогично, команды y0 и y$ позволят осуществлять копирование символов до начала и конца текущей строки соответственно.

Таблица 20.7. Переход в начало и конец строки

Команда	Действие
0	Переход к началу текущей строки
^	Переход к началу текущей строки
$	Переход к концу текущей строки
d0	Удаление символов до начала текущей строки
d$	Удаление символов до конца текущей строки

20.8. Объединение двух (J) и более строк

При вводе команды J в режиме ввода команд будет осуществлено объединение следующей и текущей строк. Кроме того, с помощью команды yyp вы можете дублировать строку, а с помощью команды ddp - поменять местами две строки.

Таблица 20.8. Объединение двух строк

Команда	Действие
J	Объединение двух строк
yyp	Дублирование строки
ddp	Замена местами двух строк

20.9. Слова (w b)

При работе в режиме ввода команд команда w позволяет перейти к следующему слову, а команда b - к предыдущему слову. Команды w и b также могут комбинироваться с командами d и y с целью копирования и переноса слов (dw db yw yb).

Таблица 20.9. Слова

Команда	Действие
w	Переход к следующему слову
b	Переход к предыдущему слову
3w	Переход вперед на три слова
dw	Удаление одного слова
yw	Копирование (yank) одного слова
5yb	Копирование пяти слов до курсора
7dw	Удаление семи слов

20.10. Сохранение (или отказ от сохранения) данных и завершение работы (:w :q :q!)

Ввод символа : позволяет вам передавать инструкции редактору vi (говоря техническим языком, ввод символа двоеточия приводит к открытию редактора ex). Команда :w позволяет записать данные (сохранить) в файл, команда :q - завершить работу с редактором без изменения файла и сохранения данных, а команда :q! - завершить работу с редактором с отклонением всех изменений. При использовании команды :wq данные будут сохранены в файл, после чего работа редактора завершится точно также, как и при использовании команды ZZ в режиме ввода команд.

Таблица 20.10. Сохранение данных и завершение работы с vi

Команда	Действие
:w	Сохранение (запись) данных в файл
:w имя файла	Сохранение данных в файл с заданным именем
:q	Завершение работы с редактором
:wq	Сохранение данных и завершение работы с редактором
ZZ	Сохранение данных и завершение работы с редактором
:q!	Завершение работы с редактором (с отклонением внесенных вами изменений)
:w!	Сохранение данных (и запись данных в защищенный от записи файл)

Последняя команда применяется в особых случаях. В случае использовании команды :w! редактор vi попытается осуществить изменение прав доступа к файлу (**chmod**) с целью получения возможности записи данных в файл (данный трюк сработает в том случае, если вы

являетесь владельцем файла) и вернет прежние права доступа к файлу в случае успешной записи данных. Эта команда должна всегда успешно выполняться, если вы используете учетную запись пользователя root (и имеете возможность осуществления записи данных в любой файл файловой системы).

20.11. Поиск (/ ?)

В режиме ввода команд команда / позволяет вам осуществлять поиск строк в тексте, редактируемом с помощью vi (в качестве поискового запроса может использоваться регулярное выражение). Команда /foo позволяет осуществить поиск строки foo по направлению к концу файла, а команда ?bar - поиск строки bar по направлению к началу файла.

Таблица 20.11. Поиск

Команда	Действие
/строка	Поиск строки по направлению к концу файла
?строка	Поиск строки по направлению к началу файла
n	Переход к следующей найденной строке
/^строка	Поиск строки по направлению к концу файла с начала текущей строки файла
/строка$	Поиск строки по направлению к концу файла с конца текущей строки файла
/br[aeio]l	Поиск слов bral, brel, bril, и brol
/\<he\>	Поиск слова he (но не слов here или the)

20.12. Замена всех найденных строк (:1,$ s/foo/bar/g)

Для замены всех найденных строк foo на строки bar в первую очередь следует перейти в режим ex с помощью команды : . После этого следует сообщить редактору vi о том, какие строки следует использовать, например, в случае использования запроса 1,$ будет осуществляться замена всех найденных строк, начиная с первой и заканчивая последней строкой файла. Вы можете использовать запрос 1,5 для обработки только первых пяти строк файла. Запрос s/foo/bar/g позволяет заменить все строки foo на строки bar.

Таблица 20.12. Замена

Команда	Действие
:4,8 s/foo/bar/g	Заменить строки foo на строки bar начиная с 4 и заканчивая 8 строкой файла
:1,$ s/foo/bar/g	Заменить строки foo на строки bar во всех строках файла

20.13. Чтение файлов (:r :r !cmd)

При работе в режиме ввода команд команда :r foo позволяет осуществлять чтение файла с именем foo, а команда :r !foo - исполнять системную команду foo. Результат исполнения системной команды будет размещен в позиции курсора. Таким образом, команда :r !ls позволяет сохранить список файлов из текущей директории в вашем текстовом файле.

Таблица 20.13. Чтение файлов и вывода команд

Команда	Действие
:r имя файла	Чтение файла с заданным именем и размещение его содержимого в редактируемом текстовом файле
:r !команда	Исполнение заданной команды и размещение ее вывода в редактируемом текстовом файле

20.14. Текстовые буферы

В редакторе vi используется 36 буферов для хранения текста. Вы можете задействовать их, воспользовавшись символом ".

Таблица 20.14. Текстовые буферы

Команда	Действие
"add	Удалить текущую строку и переместить текст в буфер a
"g7yy	Скопировать семь строк в буфер g
"ap	Вставить данные из буфера a

20.15. Работа с множеством файлов

Вы можете осуществлять редактирование множества файлов с помощью текстового редактора vi. Ниже приведены примеры некоторых команд.

Таблица 20.15. Работа с множеством файлов

Команда	Действие
vi файл1 файл2 файл3	Начать редактирование трех фалов с заданными именами
:args	Вывести список файлов и указать активный файл
:n	Начать редактирование следующего файла
:e	Перейти к последнему отредактированному файлу
:rew	Сместить указатель открытых файлов на первый файл

20.16. Аббревиатуры строк

С помощью команды :ab вы можете создавать аббревиатуры строк в рамках редактора vi. Впоследствии следует использовать команду :una для прекращения использования созданной аббревиатуры.

Таблица 20.16. Аббревиатуры строк

Команда	Действие
:ab строка длинная строка	Создать аббревиатуру строка для строки 'длинная строка'
:una строка	Прекратить использование аббревиатуры строка

20.17. Соответствия клавиш

Аналогично аббревиатурам, благодаря наличию поддержки команды :map при работе в режиме ввода команд и команды :map! при работе в режиме ввода текста, вы имеете возможность использовать соответствия клавиш.

В примере ниже показана методика настройки редактора с целью задействования клавиши F6 для активации (выполняющейся с помощью команды set number) и деактивации (выполняющейся с помощью команды set nonumber) режима показа нумерации строк файла. Строка <bar> разделяет две команды, причем команда set number! предназначена для переключения режима показа нумерации строк файла, а команда set number? - для вывода сообщения о текущем состоянии.

```
:map <F6> :set number!<bar>set number?<CR>
```

20.18. Установка значений параметров

Значения некоторых параметров конфигурации редактора vim могут быть установлены непосредственно при работе с ним.

```
:set number  (также попробуйте использовать команду :se
nu)
:set nonumber
:syntax on
:syntax off
:set all  (вывод списка, содержащего имена всех парамет-
ров конфигурации)
:set tabstop=8
:set tx   (окончания строк в стиле CR/LF)
:set notx
```

Также вы можете изменить значения этих (и многих других) параметров конфигурации путем редактирования файла `~/.vimrc` в случае использования редактора vim или `~/.exrc` в случае использования стандартного редактора vi.

```
paul@barry:~$ cat ~/.vimrc
set number
set tabstop=8
set textwidth=78
map <F6> :set number!<bar>set number?<CR>
paul@barry:~$
```

20.19. Практическое задание: vi(m)

1. Запустите приложение `vimtutor` и выполните некоторые или все предложенные задания. При работе с дистрибутивом xubuntu вам, возможно, придется выполнить команду **`aptitude install vim`**.

2. Какая комбинация из 3 клавиш, использованная в режиме ввода команд, позволит создать дубликат текущей строки файла?

3. Какая комбинация из 3 клавиш, использованная в режиме ввода команд, позволит поменять две строки файла местами (в результате чего строка номер пять станет строкой номер шесть, а строка номер шесть - строкой номер пять)?

4. Какая комбинация из 2 клавиш, использованная в режиме ввода команд, позволит поменять местами текущий и следующий символы?

5. Редактор vi может работать с макросами. Запись макроса должна начинаться с символа q, после которого должно записываться имя макроса. Таким образом, команда qa приведет к созданию макроса с именем a. Повторное нажатие клавиши q приведет к завершению записи макроса. После этого вы можете вызвать созданный макрос, воспользовавшись символом @ со следующим за ним именем макроса. Попытайтесь использовать следующий пример: i 1 'Клавиша Escape' qa yyp 'Ctrl a' q5@a. (Сочетание клавиш Ctrl a позволяет увеличить значение на 1).

6. Скопируйте файл `/etc/passwd` в вашу домашнюю директорию (`~/passwd`). Откройте скопированный файл с помощью редактора vi и нажмите Ctrl v. После выделения блока текста с помощью клавиш со стрелками вы сможете скопировать его с помощью команды y или удалить его с помощью команды d. Попытайтесь вставить скопированный блок текста.

7. Какое действие выполняется с помощью команды dwwP, если курсор находится в начале слова из предложения?

20.20. Корректная процедура выполнения практического задания: vi(m)

1. Запустите приложение `vimtutor` и выполните некоторые или все предложенные задания. При работе с дистрибутивом xubuntu вам, возможно, придется выполнить команду **`aptitude install vim`**.

```
vimtutor
```

2. Какая комбинация из 3 клавиш, использованная в режиме ввода команд, позволит создать дубликат текущей строки файла?

```
yyp
```

3. Какая комбинация из 3 клавиш, использованная в режиме ввода команд, позволит поменять две строки файла местами (в результате чего строка номер пять станет строкой номер шесть, а строка номер шесть - строкой номер пять)?

```
ddp
```

4. Какая комбинация из 2 клавиш, использованная в режиме ввода команд, позволит поменять местами текущий и следующий символы?

```
xp
```

5. Редактор vi может работать с макросами. Запись макроса должна начинаться с символа q, после которого должно записываться имя макроса. Таким образом, команда qa приведет к созданию макроса с именем a. Повторное нажатие клавиши q приведет к завершению записи макроса. После этого вы можете вызвать созданный макрос, воспользовавшись символом @ со следующим за ним именем макроса. Попытайтесь использовать следующий пример: i 1 'Клавиша Escape' qa yyp 'Ctrl a' q5@a. (Сочетание клавиш Ctrl a позволяет увеличить значение на 1).

6. Скопируйте файл `/etc/passwd` в вашу домашнюю директорию (`~/passwd`). Откройте скопированный файл с помощью редактора vi и нажмите Ctrl v. После выделения блока текста с помощью клавиш со стрелками вы сможете скопировать его с помощью команды y или удалить его с помощью команды d. Попытайтесь вставить скопированный блок текста.

```
cp /etc/passwd ~
vi passwd
(нажмите Ctrl-V)
```

7. Какое действие выполняется с помощью команды **dwwP**, если курсор находится в начале слова из предложения?

Команда **dwwP** позволяет менять местами текущее и следующее слова в предложении.

Часть VI. Сценарии

Глава 21. Введение в разработку сценариев

Командные оболочки, такие, как **bash** и **Korn shell** поддерживают программные конструкции, которые могут быть сохранены в форме **сценариев**. Эти **сценарии**, в свою очередь, впоследствии могут использоваться в качестве реализаций дополнительных команд **командной оболочки**. Многие команды Linux реализованы в форме **сценариев**. Например, **сценарии для обслуживания профиля пользователя** исполняются при входе пользователя в систему, а **сценарии инициализации** - при остановке и запуске **демона**.

Исходя из вышесказанного, системные администраторы также должны иметь базовое представление о **сценариях командной оболочки** для понимания того, как функционируют их серверы, запускаются, обновляются, исправляются, поддерживаются, настраиваются и удаляются их приложения, а также для понимания устройства программного окружения.

Целью данной главы является предоставление достаточной информации для того, чтобы вы могли читать и понимать сценарии. Для этого вам совершенно не нужно становиться разработчиком сложных сценариев.

21.1. Предварительное чтение

Перед тем, как приступить к чтению данной главы, вам необходимо прочитать и понять главы из **Части III. "Раскрытие команд командной оболочкой"** и **Части IV. "Программные каналы и команды"**.

21.2. Hello world

По аналогии с практически любым курсом по программированию, мы начнем работу с разработки сценария **hello_world**. Следующий сценарий будет выводить строку **Hello World**.

```
echo Hello World
```

После создания этого простого сценария в редакторе **vi** или с помощью команды **echo** вам придется выполнить команду **chmod +x hello_world** для того, чтобы сделать файл сценария исполняемым. В том случае, если вы не будете добавлять путь к директории с вашими сценариями в список директорий из переменной окружения PATH,

вам придется вводить полный путь к сценарию для того, чтобы командная оболочка смогла найти его.

```
[paul@RHEL4a ~]$ echo echo Hello World > hello_world
[paul@RHEL4a ~]$ chmod +x hello_world
[paul@RHEL4a ~]$ ./hello_world
Hello World
[paul@RHEL4a ~]$
```

21.3. She-bang

Давайте немного доработаем наш пример, разместив строку **#!/bin/bash** в начале сценария. Последовательность символов **#!** называется **she-bang** (или иногда **sha-bang**), причем слово **she-bang** составлено из названий двух первых символов сценария.

```
#!/bin/bash
echo Hello World
```

Вы ни при каких обстоятельствах не можете быть уверены в том, какая командная оболочка используется в системе пользователя. Сценарий, превосходно работающий в командной оболочке **bash**, может не работать в командных оболочках **ksh, csh** или **dash**. Для того, чтобы проинструктировать командную оболочку о необходимости запуска вашего сценария в определенной командной оболочке, вы должны начинать ваш сценарий с последовательности символов **she-bang**, после которой должен располагаться путь к бинарному файлу командной оболочки, в которой сценарий должен исполняться. Приведенный ниже сценарий будет исполняться в командной оболочке **bash**.

```
#!/bin/bash
echo -n hello
echo Дочерняя командная оболочка bash `echo -n hello`
```

А этот сценарий будет исполняться в командной оболочке Korn shell (за исключением тех случаев, когда по пути **/bin/ksh** расположена жесткая ссылка на бинарный файл **/bin/bash**). Файл **/etc/shells** содержит список путей к командным оболочкам, установленным в вашей системе.

```
#!/bin/ksh
echo -n hello
```

```
echo Дочерняя командная оболочка Korn shell `echo -n
hello`
```

21.4. Комментарий

Давайте еще немного усовершенствуем наш пример, добавив строки комментариев.

```
#!/bin/bash
#
# Сценарий Hello World
#
echo Hello World
```

21.5. Переменные

Ниже приведен простой пример объявления переменной в сценарии.

```
#!/bin/bash
#
# простая переменная в сценарии
#
var1=4
echo var1 = $var1
```

Сценарии могут содержать переменные, но ввиду того, что сценарии исполняются в своих собственных экземплярах командных оболочек, переменные не смогут пережить момент завершения исполнения сценария.

```
[paul@RHEL4a ~]$ echo $var1

[paul@RHEL4a ~]$ ./vars
var1 = 4
[paul@RHEL4a ~]$ echo $var1

[paul@RHEL4a ~]$
```

21.6. Использование рабочей командной оболочки

К счастью, у вас имеется возможность исполнения сценария в той же рабочей командной оболочке; данная техника называется **использованием рабочей командной оболочки** (sourcing a script).

```
[paul@RHEL4a ~]$ source ./vars
var1 = 4
[paul@RHEL4a ~]$ echo $var1
4
[paul@RHEL4a ~]$
```

Представленная выше команда аналогична следующей команде.

```
[paul@RHEL4a ~]$ . ./vars
var1 = 4
[paul@RHEL4a ~]$ echo $var1
4
[paul@RHEL4a ~]$
```

21.7. Отладка сценария

Другой способ исполнения сценария в отдельной командной оболочке заключается во вводе команды **bash** перед именем сценария, которое в этом случае будет передаваться бинарному файлу командной оболочки в качестве параметра.

```
paul@debian6~/test$ bash runme
42
```

Дополнение данной команды до формы **bash -x** позволит вам ознакомиться со всеми командами, исполняемыми командной оболочкой (после раскрытия команд).

```
paul@debian6~/test$ bash -x runme
+ var4=42
+ echo 42
42
paul@debian6~/test$ cat runme
# сценарий runme
var4=42
echo $var4
paul@debian6~/test$
```

Обратите внимание на отсутствие строки комментария (первым символом которой является символ **#**), а также замену значения переменной перед исполнением команды для вывода данных **echo**.

21.8. Предотвращение подмены имен файлов сценариев с целью повышения привилегий в системе

Какой-либо пользователь может попытаться выполнить сценарий с установленным битом **setuid** с целью повышения привилегий в системе, подменив имя файла этого сценария путем создания ссылки на него (**root spoofing**). Это довольно редкая, но возможная атака. Для повышения безопасности функционирования вашего сценария, а также для предотвращения подмены имен файлов сценариев, вам необходимо добавить после строки **#!/bin/bash** параметр **--**, который позволит деактивировать механизм обработки параметров, благодаря чему командная оболочка не примет дополнительных параметров.

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

или

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

Любые аргументы, расположенные после последовательности символов **--** будут рассматриваться как имена файлов и аргументы. Аргумент **-** эквивалентен аргументу **--**.

21.9. Практическое задание: введение в разработку сценариев

0. Используйте различные имена для файлов ваших сценариев и сохраните их на будущее!

1. Создайте сценарий, который выводит имя города.

2. Сделайте так, чтобы сценарий исполнялся в командной оболочке bash.

3. Сделайте так, чтобы сценарий исполнялся в командной оболочке Korn shell.

4. Создайте сценарий, в котором осуществляется объявление двух переменных с последующим выводом их значений.

5. Предыдущий сценарий не должен оказывать воздействия на вашу рабочую командную оболочку (объявленные в нем переменные не будут существовать вне сценария). А теперь запустите сценарий таким

образом, чтобы он оказал влияние на вашу рабочую командную оболочку.

6. Существует ли более короткая форма команды для **использования рабочей командной оболочки** с целью исполнении сценария?

7. Добавьте комментарии в ваши сценарии для того, чтобы знать о выполняемых ими операциях.

21.10. Корректная процедура выполнения практического задания: введение в разработку сценариев

0. Используйте различные имена для файлов ваших сценариев и сохраните их на будущее!

1. Создайте сценарий, который выводит имя города.

```
$ echo 'echo Antwerp' > first.bash
$ chmod +x first.bash
$ ./first.bash
Antwerp
```

2. Сделайте так, чтобы сценарий исполнялся в командной оболочке bash.

```
$ cat first.bash
#!/bin/bash
echo Antwerp
```

3. Сделайте так, чтобы сценарий исполнялся в командной оболочке Korn shell.

```
$ cat first.bash
#!/bin/ksh
echo Antwerp
```

Обратите внимание на то, что хотя данный сценарий и будет технически исполняться как сценарий командной оболочки Korn shell, расширение файла .bash может смутить пользователя.

4. Создайте сценарий, в котором осуществляется объявление двух переменных с последующим выводом их значений.

```
$ cat second.bash
#!/bin/bash

var33=300
var42=400

echo $var33 $var42
```

5. Предыдущий сценарий не должен оказывать воздействия на вашу рабочую командную оболочку (объявленные в нем переменные не будут существовать вне сценария). А теперь запустите сценарий таким образом, чтобы он оказал влияние на вашу рабочую командную оболочку.

```
source second.bash
```

6. Существует ли более короткая форма команды для **использования рабочей командной оболочки** с целью исполнении сценария?

```
. ./second.bash
```

7. Добавьте комментарии в ваши сценарии для того, чтобы знать о выполняемых ими операциях.

```
$ cat second.bash
#!/bin/bash
# сценарий для проверки работы механизма использования рабочей командной оболочки
# объявление двух переменных
var33=300
var42=400
# вывод значений этих переменных
echo $var33 $var42
```

Глава 22. Циклы в сценариях

22.1. Команда `test []`

Команда **`test`** позволяет установить, является ли какое-либо выражение истинным или ложным. Давайте начнем с проверки, больше ли целочисленное значение `10` целочисленного значения `55`.

```
[paul@RHEL4b ~]$ test 10 -gt 55 ; echo $?
```

```
1
[paul@RHEL4b ~]$
```

Команда test возвращает значение 1, если выражение является ложным. И, как вы увидите в следующем примере, команда test будет возвращать значение 0, если выражение будет являться истинным.

```
[paul@RHEL4b ~]$ test 56 -gt 55 ; echo $?
0
[paul@RHEL4b ~]$
```

Если же вам удобнее работать со строками true (истина) и false (ложь), вы можете использовать команду test таким образом, как показано ниже.

```
[paul@RHEL4b ~]$ test 56 -gt 55 && echo true || echo
false
true
[paul@RHEL4b ~]$ test 6 -gt 55 && echo true || echo false
false
```

Команда test также может заменяться квадратными скобками, поэтому команды из примера ниже полностью аналогичны командам из примера выше.

```
[paul@RHEL4b ~]$ [ 56 -gt 55 ] && echo true || echo false
true
[paul@RHEL4b ~]$ [ 6 -gt 55 ] && echo true || echo false
false
```

Ниже приведены примеры реализаций некоторых проверок. Обратитесь к странице руководства man test для ознакомления с дополнительными возможностями реализации различных проверок.

```
[ -d foo ]              Существует ли директория foo ?
[ -e bar ]              Существует ли файл bar ?
[ '/etc' = $PWD ]       Эквивалентна ли строка /etc значе-
нию переменной $PWD ?
[ $1 != 'secret' ]      Отличается ли значение первого па-
раметра сценария от строки secret ?
[ 55 -lt $bar ]         Меньше ли целочисленное значение
55 значения переменной $bar ?
```

```
[ $foo -ge 1000 ]      Является ли значение переменной
$foo большим или равным целочисленному значению 1000 ?
[ "abc" < $bar ]       Будет ли строка abc расположена
выше значения переменной $bar в списке после сортировки ?
[ -f foo ]             Является ли foo обычным файлом ?
[ -r bar ]             Является ли bar читаемым файлом ?
[ foo -nt bar ]        Новее ли файл foo файла bar ?
[ -o nounset ]         Активирован ли параметр командной
оболочки nounset ?
```

Операторы проверок могут комбинироваться с операторами, соответствующими логическим операциям "И" и "ИЛИ".

```
paul@RHEL4b:~$ [ 66 -gt 55 -a 66 -lt 500 ] && echo true
|| echo false
true
paul@RHEL4b:~$ [ 66 -gt 55 -a 660 -lt 500 ] && echo true
|| echo false
false
paul@RHEL4b:~$ [ 66 -gt 55 -o 660 -lt 500 ] && echo true
|| echo false
true
```

22.2. Условный переход `if then else`

Конструкция **`if then else`** предназначена для выбора варианта кода. В том случае, если определенное условие выполняется, будет исполняться какой либо код, в противном случае будет исполняться какой-либо другой код. В примере ниже осуществляется проверка существования файла, после чего в том случае, если предположение о существовании файла подтверждается, осуществляется вывод соответствующего сообщения.

```
#!/bin/bash

if [ -f isit.txt ]
then echo файл isit.txt существует!
else echo файл isit.txt не найден!
fi
```

В том случае, если мы сохраним данный код сценария в файле с именем 'choice', он сможет быть исполнен аналогичным образом.

```
[paul@RHEL4a scripts]$ ./choice
файл isit.txt не найден!
[paul@RHEL4a scripts]$ touch isit.txt
[paul@RHEL4a scripts]$ ./choice
файл isit.txt существует!
[paul@RHEL4a scripts]$
```

22.3. Условный переход `if then elif`

Вы можете разместить новый оператор условного перехода **if** внутри блока **else**, воспользовавшись оператором **elif**. Ниже приведен простой пример такой записи.

```
#!/bin/bash
count=42
if [ $count -eq 42 ]
then
  echo "42 является корректным значением."
elif [ $count -gt 42 ]
then
  echo "Слишком много."
else
  echo "Не достаточно."
fi
```

22.4. Цикл `for`

В примере ниже представлен синтаксис классического цикла **for** в командной оболочке bash.

```
for i in 1 2 4
do
   echo $i
done
```

Пример использования цикла **for**, скомбинированного с вызовом встраиваемой командной оболочки.

```
#!/bin/ksh
for counter in `seq 1 20`
do
   echo отсчет от 1 до 20, текущее значение $counter
```

```
   sleep 1
done
```

Сценарий, полностью аналогичный представленному выше, может быть создан и без задействования встраиваемой командной оболочки путем использования реализованного в рамках командной оболочки bash объявления диапазона значений **{от значения..до значения}**.

```
#!/bin/bash
for counter in {1..20}
do
   echo отсчет от 1 до 20, текущее значение $counter
   sleep 1
done
```

В данном цикле **for** используется механизм поиска файлов по шаблону (реализованный в рамках механизма раскрытия команд). В случае размещения приведенной инструкции непосредственно в командной строке, она будет функционировать аналогично.

```
kahlan@solexp11$ ls
count.ksh  go.ksh
kahlan@solexp11$ for file in *.ksh ; do cp $file
$file.backup ; done
kahlan@solexp11$ ls
count.ksh  count.ksh.backup  go.ksh  go.ksh.backup
```

22.5. Цикл while

Ниже приведен простой пример использования цикла **while**.

```
i=100;
while [ $i -ge 0 ] ;
do
   echo Обратный отсчет от 100 до 0, текущее значение $i;
   let i--;
done
```

Бесконечные циклы могут реализовываться с помощью объявлений **while true** или **while :** , где символ **:** является эквивалентом **отсутствующей операции** в командных оболочках **Korn shell** и **bash**.

```
#!/bin/ksh
# бесконечный цикл
while :
do
 echo hello
 sleep 1
done
```

22.6. Цикл `until`

Ниже приведен простой пример использования цикла `until`.

```
let i=100;
until [ $i -le 0 ] ;
do
   echo Обратный отсчет от 100 до 1, текущее значение $i;
   let i--;
done
```

22.7. Практическое задание: проверки и циклы в сценариях

1. Разработайте сценарий, который будет использовать цикл **`for`** для отсчета от 3 до 7.

2. Разработайте сценарий, который будет использовать цикл **`for`** для отсчета от 1 до 17000.

3. Разработайте сценарий, который будет использовать цикл **`while`** для отсчета от 3 до 7.

4. Разработайте сценарий, который будет использовать цикл **`until`** для обратного отсчета от 8 до 4.

5. Разработайте сценарий, который будет производить подсчет файлов с расширением **`.txt`** в текущей директории.

6. Используйте оператор **`if`** в созданном сценарии для его корректной работы в случае отсутствия файлов с расширением **`.txt`** в текущей директории.

22.8. Корректная процедура выполнения практического задания: проверки и циклы в сценариях

1. Разработайте сценарий, который будет использовать цикл **for** для отсчета от 3 до 7.

```
#!/bin/bash

for i in 3 4 5 6 7
do
 echo Отсчет от 3 до 7, текущее значение $i
done
```

2. Разработайте сценарий, который будет использовать цикл **for** для отсчета от 1 до 17000.

```
#!/bin/bash

for i in `seq 1 17000`
do
 echo Отсчет от 1 до 17000, текущее значение $i
done
```

3. Разработайте сценарий, который будет использовать цикл **while** для отсчета от 3 до 7.

```
#!/bin/bash

i=3
while [ $i -le 7 ]
do
 echo Отсчет от 3 до 7, текущее значение $i
 let i=i+1
done
```

4. Разработайте сценарий, который будет использовать цикл **until** для обратного отсчета от 8 до 4.

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

```
i=8
until [ $i -lt 4 ]
do
 echo Обратный отсчет от 8 до 4, текущее значение $i
 let i=i-1
done
```

5. Разработайте сценарий, который будет производить подсчет файлов с расширением **.txt** в текущей директории.

```
#!/bin/bash

let i=0
for file in *.txt
do
        let i++
done
echo "В директории $i файлов с расширением .txt"
```

6. Используйте оператор **if** в созданном сценарии для его корректной работы в случае отсутствия файлов с расширением **.txt** в текущей директории.

```
#!/bin/bash

ls *.txt > /dev/null 2>&1
if [ $? -ne 0 ]
then echo "В директории 0 файлов с расширением .txt"
else
        let i=0
        for file in *.txt
        do
                let i++
        done
        echo "В директории $i файлов с расширением .txt"
fi
```

Глава 23. Параметры сценариев

23.1. Параметры сценария

Сценарий командной оболочки **bash** может принимать параметры. Нумерация, которую вы можете увидеть в сценарии ниже, может быть продолжена, если для работы сценария необходимо большее количество параметров. Также в вашем распоряжении имеются специальные параметры, содержащие значения, которые соответствуют количеству параметров, их строковому представлению, а также идентификатору процесса и последнему коду завершения процесса. Полный список специальных параметров размещен на странице руководства командной оболочки **bash**.

```
#!/bin/bash
echo Первым аргументом является $1
echo Вторым аргументом является $2
echo Третьим аргументом является $3

echo \$ $$  - идентификатор процесса (PID) интерпретатора
сценария
echo \# $#  - количество аргументов
echo \? $?  - последний код завершения процесса
echo \* $*  - строковое представление всех аргументов
Ниже представлен вывод данного сценария.
[paul@RHEL4a scripts]$ ./pars one two three
Первым аргументом является one
Вторым аргументом является two
Третьим аргументом является three
$ 5610 - идентификатор процесса (PID) интерпретатора
сценария
# 3  - количество аргументов
? 0  - последний код завершения процесса
* one two three - строковое представление всех аргументов
```

Еще один вариант исполнения данного сценария, но теперь с передачей только двух параметров.

```
[paul@RHEL4a scripts]$ ./pars 1 2
Первым аргументом является 1
```

```
Вторым аргументом является 2
Третьим аргументом является
$ 5612 - идентификатор процесса (PID) интерпретатора сценария
# 2 - количество аргументов
? 0 - последний код завершения процесса
* 1 2 - строковое представление всех аргументов
[paul@RHEL4a scripts]$
```

А это другой пример сценария, в котором мы используем параметр **$0**. Параметр **$0** содержит имя файла исполняющегося сценария.

```
paul@debian6~$ cat myname
echo имя файла этого сценария $0
paul@debian6~$ ./myname
имя файла этого сценария ./myname
paul@debian6~$ mv myname test42
paul@debian6~$ ./test42
имя файла этого сценария ./test42
```

23.2. Обход списка параметров

Оператор обхода списка параметров (**shift**) позволяет произвести разбор всех переданных сценарию **параметров** по очереди. Ниже представлен пример сценария с данным оператором.

```
kahlan@solexp11$ cat shift.ksh
#!/bin/ksh

if [ "$#" == "0" ]
 then
  echo Вы должны передать по крайней мере один параметр сценарию.
  exit 1
fi

while (( $# ))
 do
```

```
  echo Вы передали сценарию параметр $1
  shift
 done
```

А это вариант вывода приведенного выше сценария.

```
kahlan@solexp11$ ./shift.ksh one
Вы передали сценарию параметр one
kahlan@solexp11$ ./shift.ksh one two three 1201 "33 42"
Вы передали сценарию параметр one
Вы передали сценарию параметр two
Вы передали сценарию параметр three
Вы передали сценарию параметр 1201
Вы передали сценарию параметр 33 42
kahlan@solexp11$ ./shift.ksh
Вы должны передать по крайней мере один параметр сцена-
рию.
```

23.3. Ввод в процессе исполнения сценария

Вы можете попросить пользователя ввести данные, воспользовавшись командой **read** в сценарии.

```
#!/bin/bash
echo -n Введите число:
read number
```

23.4. Задействование файла конфигурации

Команда **source** (о которой говорилось в одной из глав, посвященных работе с командной оболочкой) может также использоваться для задействования файла конфигурации сценария.

Ниже приведено содержимое демонстрационного файла конфигурации для приложения.

```
[paul@RHEL4a scripts]$ cat myApp.conf
# Файл конфигурации для приложения myApp

# Путь
myAppPath=/var/myApp

# Количество приложений, осуществляющих саморепликацию
```

```
quines=5
А это код приложения, которое использует данный файл.
[paul@RHEL4a scripts]$ cat myApp.bash
#!/bin/bash
#
# Добро пожаловать в приложение myApp
#

. ./myApp.conf

echo Количество самовоспроизводящихся приложений: $quines
```

В ходе исполнения приложение может использовать значения из подключенного файла конфигурации.

```
[paul@RHEL4a scripts]$ ./myApp.bash
Количество самовоспроизводящихся приложений: 5
[paul@RHEL4a scripts]$
```

23.5. Получение параметров сценария с помощью функции getopts

Функция **getopts** позволяет осуществлять разбор параметров, передаваемых сценарию посредством командного интерфейса системы. Следующий сценарий позволяет использовать любую комбинацию параметров `a`, `f` и `z`.

```
kahlan@solexp11$ cat options.ksh
#!/bin/ksh

while getopts ":afz" option;
do
 case $option in
  a)
   echo принят параметр -a
   ;;
  f)
   echo принят параметр -f
   ;;
```

```
  z)
   echo принят параметр -z
   ;;
  *)
   echo "некорректный параметр -$OPTARG"
   ;;
 esac
done
```

Ниже представлен вариант вывода данного сценария. Сначала мы передаем исключительно допустимые параметры, а затем дважды передаем некорректный параметр.

```
kahlan@solexp11$ ./options.ksh
kahlan@solexp11$ ./options.ksh -af
принят параметр -a
принят параметр -f
kahlan@solexp11$ ./options.ksh -zfg
принят параметр -z
принят параметр -f
некорректный параметр -g
kahlan@solexp11$ ./options.ksh -a -b -z
принят параметр -a
некорректный параметр -b
принят параметр -z
```

Также вы можете обрабатывать параметры, которые подразумевают использование аргумента таким образом, как показано в данном примере.

```
kahlan@solexp11$ cat argoptions.ksh
#!/bin/ksh

while getopts ":af:z" option;
do
 case $option in
  a)
   echo принят параметр -a
```

```
   ;;
  f)
   echo принят параметр -f с аргументом $OPTARG
   ;;
  z)
   echo принят параметр -z
   ;;
  :)
   echo "вместе с параметром -$OPTARG должен быть передан
аргумент"
   ;;
  *)
   echo "некорректный параметр -$OPTARG"
   ;;
 esac
done
```

А это вариант вывода представленного выше сценария.

```
kahlan@solexp11$ ./argoptions.ksh -a -f hello -z
принят параметр -a
принят параметр -f с аргументом hello
принят параметр -z
kahlan@solexp11$ ./argoptions.ksh -zaf 42
принят параметр -z
принят параметр -a
принят параметр -f с аргументом 42
kahlan@solexp11$ ./argoptions.ksh -zf
принят параметр -z
вместе с параметром -f  должен быть передан аргумент
```

23.6. Получение параметров функционирования командной оболочки с помощью команды shopt

Вы можете изменять значения переменных, управляющих аспектами функционирования командной оболочки, с помощью встроенной команды командной оболочки **shopt**. В примере ниже сначала осуществляется проверка того, установлено ли значение переменной

cdspell; оказывается, что оно не установлено. После этого с помощью команды shopt осуществляется установка значения данной переменной, а третья команда, в составе которой используется команда shopt, предназначается для проверки того, действительно ли была осуществлена установка значения рассматриваемой переменной. После установки значения данной переменной вы можете допускать незначительные опечатки в ходе использования команды cd. Страница руководства командной оболочки bash содержит полный список допустимых переменных.

```
paul@laika:~$ shopt -q cdspell ; echo $?
1
paul@laika:~$ shopt -s cdspell
paul@laika:~$ shopt -q cdspell ; echo $?
0
paul@laika:~$ cd /Etc
/etc
```

23.7. Практическое задание: параметры сценариев и специальные переменные

1. Разработайте сценарий, который будет принимать четыре параметра и выводить их в обратном порядке.

2. Разработайте сценарий, который будет принимать два параметра (два имени файла) и выводить информацию о том, существуют ли эти файлы.

3. Разработайте сценарий, который будет предлагать пользователю ввести имя файла. Этот сценарий должен проверять наличие файла, принадлежность файла вам, а также возможность записи в файл. В случае отсутствия возможности записи в файл он должен выполнять все необходимые действия для того, чтобы такая возможность появилась.

4. Создайте конфигурационный файл для предыдущего сценария. Объявите в рамках данного конфигурационного файла логическую переменную, которая позволит управлять функцией журналирования, заключающейся в записи подробной информации обо всех выполняемых сценарием действиях в файл журнала, который расположен в директории /tmp.

23.8. Корректная процедура выполнения практического задания: параметры сценариев и специальные переменные

1. Разработайте сценарий, который будет принимать четыре параметра и выводить их в обратном порядке.

```
echo $4 $3 $2 $1
```

2. Разработайте сценарий, который будет принимать два параметра (два имени файла) и выводить информацию о том, существуют ли эти файлы.

```
#!/bin/bash

if [ -f $1 ]
then echo файл $1 существует!
else echo файл $1 не найден!
fi

if [ -f $2 ]
then echo файл $2 существует!
else echo файл $2 не найден!
fi
```

3. Разработайте сценарий, который будет предлагать пользователю ввести имя файла. Этот сценарий должен проверять наличие файла, принадлежность файла вам, а также возможность записи в файл. В случае отсутствия возможности записи в файл он должен выполнять все необходимые действия для того, чтобы такая возможность появилась.

4. Создайте конфигурационный файл для предыдущего сценария. Объявите в рамках данного конфигурационного файла логическую переменную, которая позволит управлять функцией журналирования, заключающейся в записи подробной информации обо всех выполняемых сценарием действиях в файл журнала, который расположен в директории `/tmp`.

Глава 24. Дополнительная информация о сценариях

24.1. Команда eval

Команда **eval** позволяет интерпретировать переданные аргументы как директивы сценария командной оболочки (результирующие команды исполняются). Данное обстоятельство позволяет использовать значение переменной в качестве переменной.

```
paul@deb503:~/test42$ answer=42
paul@deb503:~/test42$ word=answer
paul@deb503:~/test42$ eval x=\$$word ; echo $x
42
```

Как в командной оболочке **bash**, так и командной оболочке **Korn shell** аргументы могут экранироваться с помощью двойных кавычек.

```
kahlan@solexp11$ answer=42
kahlan@solexp11$ word=answer
kahlan@solexp11$ eval "y=\$$word" ; echo $y
42
```

Иногда команде eval необходимо передавать аргументы таким образом, чтобы командной оболочкой осуществлялся их корректный разбор. Рассмотрите приведенный ниже пример, в котором команда **date** принимает один параметр, являющийся строкой "**1 week ago**".

```
paul@debian6~$ date --date="1 week ago"
Чт мар  8 21:36:25 CET 2012
```

В том случае, если мы сохраняем данную команду в переменную, исполнение команды из этой переменной будет заканчиваться неудачей до того момента, когда мы начнем использовать команду **eval**.

```
paul@debian6~$ lastweek='date --date="1 week ago"'
paul@debian6~$ $lastweek
date: лишний операнд `ago"'

По команде `date --help' можно получить дополнительную информацию.

paul@debian6~$ eval $lastweek
Чт мар  8 21:36:39 CET 2012
```

24.2. Оператор (( ))

Оператор **(( ))** позволяет сравнивать числовые значения.

```
paul@deb503:~/test42$ (( 42 > 33 )) && echo true || echo
false
true
paul@deb503:~/test42$ (( 42 > 1201 )) && echo true ||
echo false
false
paul@deb503:~/test42$ var42=42
paul@deb503:~/test42$ (( 42 == var42 )) && echo true ||
echo false
true
paul@deb503:~/test42$ (( 42 == $var42 )) && echo true ||
echo false
true
paul@deb503:~/test42$ var42=33
paul@deb503:~/test42$ (( 42 == var42 )) && echo true ||
echo false
false
```

24.3. Команда let

Встроенная команда командной оболочки **let** инструктирует командную оболочку о необходимости вычисления значений арифметических выражений. Она будет возвращать значение 0, если результат последней арифметической операции не равен 0.

```
[paul@RHEL4b ~]$ let x="3 + 4" ; echo $x
7
[paul@RHEL4b ~]$ let x="10 + 100/10" ; echo $x
20
[paul@RHEL4b ~]$ let x="10-2+100/10" ; echo $x
18
[paul@RHEL4b ~]$ let x="10*2+100/10" ; echo $x
30
```

Команда **let** также может использоваться для перевода значений в различные системы счисления.

```
[paul@RHEL4b ~]$ let x="0xFF" ; echo $x
```

```
255
[paul@RHEL4b ~]$ let x="0xC0" ; echo $x
192
[paul@RHEL4b ~]$ let x="0xA8" ; echo $x
168
[paul@RHEL4b ~]$ let x="8#70" ; echo $x
56
[paul@RHEL4b ~]$ let x="8#77" ; echo $x
63
[paul@RHEL4b ~]$ let x="16#c0" ; echo $x
192
```

Существует различие между непосредственным присваиванием значения переменной и использованием команды **let** для расчета значений арифметических выражений (даже в том случае, если с помощью данной команды осуществляется исключительно присваивание значения переменной).

```
kahlan@solexp11$ dec=15 ; oct=017 ; hex=0x0f
kahlan@solexp11$ echo $dec $oct $hex
15 017 0x0f
kahlan@solexp11$ let dec=15 ; let oct=017 ; let hex=0x0f
kahlan@solexp11$ echo $dec $oct $hex
15 15 15
```

24.4. Оператор case

В некоторых случаях вы можете упростить конструкции из вложенных условных переходов `if`, воспользовавшись конструкцией **case**.

```
[paul@RHEL4b ~]$ ./help
Какое животное вы видите ? лев
Лучше всего быстро убегать!
[paul@RHEL4b ~]$ ./help
Какое животное вы видите ? собака
Не беспокойтесь, угостите ее печеньем.
[paul@RHEL4b ~]$ cat help
#!/bin/bash
```

```
#
# Советы по обращению с дикими животными
#
echo -n "Какое животное вы видите ? "
read animal
case $animal in
        "лев" | "тигр")
                echo "Лучше всего быстро убегать!"
        ;;
        "кот")
                echo "Выпустите мышь..."
        ;;
        "собака")
                echo "Не беспокойтесь, угостите ее пече-
ньем."
        ;;
        "курица" | "гусь" | "утка" )
                echo "Яйца на завтрак!"
        ;;
        "лигр")
                echo "Подойдите и скажите: 'Ах ты, боль-
шой пушистый котенок...'."
        ;;
        "вавилонская рыбка")
                echo "Она выпала из вашего уха ?"
        ;;
        *)
                echo "Вы обнаружили неизвестное животное,
дайте ему имя!"
        ;;
esac
[paul@RHEL4b ~]$
```

24.5. Функции сценариев командной оболочки

Функции сценариев командной оболочки могут использоваться для логической группировки команд.

```
kahlan@solexp11$ cat funcs.ksh
#!/bin/ksh

function greetings {
echo Hello World!
echo а также приветствуем пользователя $USER!
}

echo Сейчас мы вызовем функцию
greetings
echo Конец
```

А это пример вывода данного сценария командной оболочки с **функцией**.

```
kahlan@solexp11$ ./funcs.ksh
Сейчас мы вызовем функцию
Hello World!
а также приветствуем пользователя kahlan!
Конец
```

Функция сценария командной оболочки также может принимать параметры.

```
kahlan@solexp11$ cat addfunc.ksh
#!/bin/ksh

function plus {
let result="$1 + $2"
echo  $1 + $2 = $result
}

plus 3 10
plus 20 13
plus 20 22
```

Данный сценарий генерирует следующий вывод:

```
kahlan@solexp11$ ./addfunc.ksh
3 + 10 = 13
20 + 13 = 33
20 + 22 = 42
```

24.6. Практическое задание: дополнительная информация о сценариях

1. Разработайте сценарий, который будет запрашивать два числа и выводить их сумму и произведение (также, как показано ниже).

```
Введите число: 5
Введите еще одно число: 2
Сумма:          5 + 2 = 7
Произведение:   5 x 2 = 10
```

2. Доработайте созданный сценарий таким образом, чтобы он осуществлял проверку нахождения чисел в диапазоне между 1 и 100 и завершал работу с выводом сообщения об ошибке при необходимости.

3. Доработайте созданный сценарий таким образом, чтобы при равенстве суммы и произведения чисел выводилось поздравление для пользователя.

4. На основе использующего конструкцию `case` и рассмотренного в главе сценария, разработайте сценарий, не учитывающий регистр символов строк, воспользовавшись для этого параметром `shopt nocasematch`. Значение параметра `nocasematch` должно сбрасываться к установленному перед запуском сценария значению.

5. Если позволяет время (или вы ожидаете, пока остальные студенты закончат выполнение данного практического задания), рассмотрите системные сценарии Linux из директорий `/etc/init.d` и `/etc/rc.d` и попытайтесь понять их. В кокой точке начинается исполнение сценария `/etc/init.d/samba`? Некоторые скрытые файлы сценариев также находятся в директории ~, но о них мы поговорим позднее.

24.7. Корректная процедура выполнения практического задания: дополнительная информация о сценариях

1. Разработайте сценарий, который будет запрашивать два числа и выводить их сумму и произведение (также, как показано ниже).

```
Введите число: 5

Введите еще одно число: 2

Сумма:          5 + 2 = 7

Произведение:   5 x 2 = 10
```

```
#!/bin/bash

echo -n "Введите число : "
read n1

echo -n "Введите еще одно число : "
read n2

let sum="$n1+$n2"
let pro="$n1*$n2"

echo -e "Сумма\t: $n1 + $n2 = $sum"
echo -e "Произведение\t: $n1 * $n2 = $pro"
```

2. Доработайте созданный сценарий таким образом, чтобы он осуществлял проверку нахождения чисел в диапазоне между 1 и 100 и завершал работу с выводом сообщения об ошибке при необходимости.

```
echo -n "Введите число из диапазона от 1 до 100 : "
read n1
if [ $n1 -lt 1 -o $n1 -gt 100 ]
then
       echo Некорректное число...
       exit 1
```

```
fi
```

3. Доработайте созданный сценарий таким образом, чтобы при равенстве суммы и произведения чисел выводилось поздравление для пользователя.

```
if [ $sum -eq $pro ]
then echo Поздравляем\, $sum == $pro
fi
```

4. На основе использующего конструкцию case и рассмотренного в главе сценария, разработайте сценарий, не учитывающий регистр символов строк, воспользовавшись для этого параметром shopt nocasematch. Значение параметра nocasematch должно сбрасываться к установленному перед запуском сценария значению.

```
#!/bin/bash
#
# Советы по обращению с дикими животными без учета
регистра символов
#
if shopt -q nocasematch; then
  nocase=yes;
else
  nocase=no;
  shopt -s nocasematch;
fi

echo -n " Какое животное вы видите ? "
read animal

case $animal in
               лев" | "тигр")
                              echo "Лучше всего быстро
убегать!"
               ;;
               "кот")
                              echo "Выпустите мышь..."
```

```
                ;;
                "собака")
                                echo "Не беспокойтесь,
угостите ее печеньем."
                ;;
                "курица" | "гусь" | "утка" )
                                echo "Яйца на завтрак!"
                ;;
                "лигр")
                                echo "Подойдите и скажи-
те: 'Ах ты, большой пушистый котенок...'."
                ;;
                "вавилонская рыбка")
                                echo "Она выпала из
вашего уха ?"
                ;;
                *)
                                echo "Вы обнаружили неиз-
вестное животное, дайте ему имя!"
                ;;
esac

if [ nocase = yes ] ; then
        shopt -s nocasematch;
else
        shopt -u nocasematch;
fi
```

5. Если позволяет время (или вы ожидаете, пока остальные студенты закончат выполнение данного практического задания), рассмотрите системные сценарии Linux из директорий `/etc/init.d` и `/etc/rc.d` и попытайтесь понять их. В кокой точке начинается исполнение сценария `/etc/init.d/samba`? Некоторые скрытые файлы сценариев также находятся в директории ~, но о них мы поговорим позднее.

Часть VII. Управление локальными учетными записями пользователей

Глава 25. Вводная информация об учетных записях пользователей

По прочтении данной небольшой главы вы научитесь идентифицировать свою учетную запись на компьютере под управлением системы Unix с помощью таких команд, как **who am i**, **id** и других.

Из второй части главы вы узнаете о том, как представиться другим пользователем с помощью команды **su**.

Кроме того, вы узнаете о том, как запустить программу от лица другого пользователя с помощью команды **sudo**.

25.1. Утилита whoami

Утилита **whoami** сообщит вам имя вашей учетной записи.

```
[paul@centos7 ~]$ whoami
paul
[paul@centos7 ~]$
```

25.2. Утилита who

Утилита **who** предоставит вам информацию о том, какие пользователи осуществили вход в систему.

```
[paul@centos7 ~]$ who
root     pts/0        2014-10-10 23:07 (10.104.33.101)
paul     pts/1        2014-10-10 23:30 (10.104.33.101)
laura    pts/2        2014-10-10 23:34 (10.104.33.96)
tania    pts/3        2014-10-10 23:39 (10.104.33.91)
[paul@centos7 ~]$
```

25.3. Команда who am i

В случае использования утилиты **who** в рамках команды **who am i** будет выведена только одна строка с информацией о вашей текущей сессии.

```
[paul@centos7 ~]$ who am i
paul     pts/1        2014-10-10 23:30 (10.104.33.101)
[paul@centos7 ~]$
```

25.4. Утилита w

Утилита **w** предоставляет информацию о пользователях, которые осуществили вход в систему, а также о том, чем они занимаются.

```
[paul@centos7 ~]$ w
 23:34:07 up 31 min,  2 users,  load average: 0.00, 0.01,
0.02
USER     TTY        LOGIN@   IDLE   JCPU   PCPU WHAT
root     pts/0     23:07   15.00s  0.01s  0.01s top
paul     pts/1     23:30    7.00s  0.00s  0.00s w
[paul@centos7 ~]$
```

25.5. Утилита id

Утилита **id** предоставит вам информацию о вашем идентификаторе пользователя, основном идентификаторе группы, а также выведет список групп, в которых вы состоите.

```
paul@debian7:~$ id
uid=1000(paul) gid=1000(paul) группы=1000(paul)
```

В дистрибутивах RHEL/CentOS с помощью данной утилиты вы также можете получить информацию о контексте **SELinux**.

```
[root@centos7 ~]# id
uid=0(root) gid=0(root) группы=0(root)
контекст=unconfined_u:unconfined_r:unconfined_t:s0-
s0:c0.c1023
```

25.6. Утилита su для работы от лица другого пользователя

Утилита **su** позволяет пользователю запустить командную оболочку от лица другого пользователя.

```
laura@debian7:~$ su tania
Password:
tania@debian7:/home/laura$
```

25.7. Утилита su для работы от лица пользователя root

Да, вы также можете использовать утилиту **su** для работы от лица пользователя **root** в том случае, если знаете пароль пользователя **root**.

```
laura@debian7:~$ su root
```

```
Password:
root@debian7:/home/laura#
```

25.8. Утилита su для пользователя root

Вы должны знать пароль пользователя, от лица которого хотите работать, за исключением случая, когда вы входите в систему как пользователь **root**. Пользователь **root** может работать от лица любого существующего в системе пользователя, не зная пароля этого пользователя.

```
root@debian7:~# id
uid=0(root) gid=0(root) группы=0(root)
root@debian7:~# su - valentina
valentina@debian7:~$
```

25.9. Команда su - $имя_пользователя

По умолчанию при смене пользователя утилита **su** осуществляет сохранение переменных окружения командной оболочки. Для того, чтобы работать от лица другого пользователя в окружении командной оболочки целевого пользователя, следует применять команду **su -** с последующим вводом имени целевого пользователя.

```
root@debian7:~# su laura
laura@debian7:/root$ exit
exit
root@debian7:~# su - laura
laura@debian7:~$ pwd
/home/laura
```

25.10. Команда su -

В том случае, если после команды **su** или **su -** не следует имени пользователя, считается, что целевым пользователем является пользователь **root**.

```
tania@debian7:~$ su -
Password:
root@debian7:~#
```

25.11. Запуск приложения от лица другого пользователя

Утилита **sudo** позволяет пользователю осуществлять запуск программ с привилегиями других пользователей. Для того, чтобы данная утилита работала, системный администратор должен отредактировать соответствующим образом файл **/etc/sudoers**. Данная утилита может оказаться полезной в случае возникновения необходимости делегирования административных задач другому пользователю (без передачи этому пользователю пароля пользователя root).

В примере ниже показана методика использования утилиты **sudo**. Пользователь **paul** получил право запускать утилиту **useradd** с привилегиями пользователя **root**. Это позволило пользователю **paul** создавать учетные записи новых пользователей в системе, не работая от лица пользователя **root** и не зная пароля пользователя **root**.

Исполнение первой введенной пользователем **paul** команды завершилось неудачей.

```
paul@debian7:~$ /usr/sbin/useradd -m valentina
useradd: Отказано в доступе
useradd: cannot lock /etc/passwd; try again later.
```

Но в случае использования **sudo** данная команда работает.

```
paul@debian7:~$ sudo /usr/sbin/useradd -m valentina
[sudo] password for paul:
paul@debian7:~$
```

25.12. Утилита visudo

Изучите страницу руководства утилиты **visudo** перед редактированием файла **/etc/sudoers**. Методика редактирования файла **sudoers** выходит за пределы набора тем, рассматриваемых в рамках данной книги.

```
paul@rhel65:~$ apropos visudo
visudo               (8)  - edit the sudoers file
paul@rhel65:~$
```

25.13. Команда sudo su -

В некоторых дистрибутивах Linux, таких, как Ubuntu и Xubuntu пароль пользователя **root** изначально не установлен. Это значит, что не имеется возможности войти в систему под именем пользователя

root (по сути это дополнительная мера безопасности). Для выполнения задач от лица пользователя **root** первому пользователю системы предоставляется возможность использования утилиты **sudo** благодаря добавлению специальной записи в файл **/etc/sudoers**. Фактически все пользователи, являющиеся членами группы admin, также могут использовать утилиту sudo для исполнения команд от лица пользователя root.

```
root@laika:~# grep admin /etc/sudoers
# Members of the admin group may gain root privileges
%admin ALL=(ALL) ALL
```

В результате пользователь может ввести команду **sudo su -** и работать от лица пользователя root без ввода пароля пользователя root. При этом команда sudo требует ввода пароля вызвавшего ее пользователя. Исходя из вышесказанного, можно сделать вывод о том, что запрос пароля из примера ниже выполняется утилитой sudo, а не su.

```
paul@laika:~$ sudo su -
Password:
root@laika:~#
```

25.14. Журналирование неудачных попыток использования утилиты sudo

Использование утилиты sudo без авторизации приведет к выводу строгого предупреждения.

```
paul@rhel65:~$ sudo su -

We trust you have received the usual lecture from the
local System
Administrator. It usually boils down to these three
things:

    #1) Respect the privacy of others.
    #2) Think before you type.
    #3) With great power comes great responsibility.

[sudo] password for paul:
```

```
paul is not in the sudoers file.  This incident will be
reported.
paul@rhel65:~$
```

После этого пользователь root сможет увидеть следующую запись в файле `/var/log/secure` в случае использования дистрибутива Red Hat (или `/var/log/auth.log` в случае использования дистрибутива Debian).

```
root@rhel65:~# tail /var/log/secure | grep sudo | tr -s '
'
Apr 13 16:03:42 rhel65 sudo: paul : user NOT in sudoers ;
TTY=pts/0 ; PWD=\
/home/paul ; USER=root ; COMMAND=/bin/su -
root@rhel65:~#
```

25.15. Практическое задание: вводная информация об учетных записях пользователей

1. Выполните команду, которая выведет информацию только об имени вашей учетной записи, благодаря наличию которой вы осуществили вход в систему.

2. Выведите список всех пользователей, осуществивших вход в систему.

3. Выведите список всех пользователей, осуществивших вход в систему, включая информацию о командах, которые они выполняют в данный момент.

4. Выведите информацию об имени вашей учетной записи и вашем уникальном идентификаторе пользователя (userid).

5. Используйте утилиту **`su`** для перехода к использованию учетной записи другого пользователя (если вы не используете учетную запись пользователя `root`, вам потребуется пароль от учетной записи другого пользователя). После этого вернитесь к использованию предыдущей учетной записи.

6. А теперь используйте команду **`su -`** для перехода к использованию учетной записи другого пользователя и отметьте различия.

Учтите, что команда **su -** переместит вас в домашнюю директорию другого пользователя (в примере это пользователь **Tania**).

7. Попробуйте создать новую учетную запись пользователя (используя для этого вашу обычную учетную запись). Попытка должна завершиться неудачей. (Подробности процесса добавления учетных записей пользователей описаны в следующей главе.)

8. А теперь попытайтесь сделать то же самое, но добавив вызов утилиты **sudo** перед вашей командой.

25.16. Корректная процедура выполнения практического задания: вводная информация об учетных записях пользователей

1. Выполните команду, которая выведет информацию только об имени вашей учетной записи, благодаря наличию которой вы осуществили вход в систему.

```
laura@debian7:~$ whoami
laura
laura@debian7:~$ echo $USER
laura
```

2. Выведите список всех пользователей, осуществивших вход в систему.

```
laura@debian7:~$ who
laura    pts/0        2014-10-13 07:22 (10.104.33.101)
laura@debian7:~$
```

3. Выведите список всех пользователей, осуществивших вход в систему, включая информацию о командах, которые они выполняют в данный момент.

```
laura@debian7:~$ w
 07:47:02 up 16 min,  2 users,  load average: 0.00, 0.00,
0.00
USER     TTY      FROM             LOGIN@   IDLE   JCPU
PCPU WHAT
root     pts/0    10.104.33.101    07:30    6.00s  0.04s
0.00s w
```

```
root     pts/1    10.104.33.101    07:46    6.00s  0.01s
0.00s sleep 42
laura@debian7:~$
```

4. Выведите информацию об имени вашей учетной записи и вашем уникальном идентификаторе пользователя (userid).

```
laura@debian7:~$ id
uid=1005(laura) gid=1007(laura) группы=1007(laura)
laura@debian7:~$
```

5. Используйте утилиту **su** для перехода к использованию учетной записи другого пользователя (если вы не используете учетную запись пользователя root, вам потребуется пароль от учетной записи другого пользователя). После этого вернитесь к использованию предыдущей учетной записи.

```
laura@debian7:~$ su tania
Password:
tania@debian7:/home/laura$ id
uid=1006(tania) gid=1008(tania) группы=1008(tania)
tania@debian7:/home/laura$ exit
laura@debian7:~$
```

6. А теперь используйте команду **su -** для перехода к использованию учетной записи другого пользователя и отметьте различия.

Учтите, что команда **su -** переместит вас в домашнюю директорию другого пользователя (в примере это пользователь **Tania**).

```
laura@debian7:~$ su - tania
Password:
tania@debian7:~$ pwd
/home/tania
tania@debian7:~$ logout
laura@debian7:~$
```

7. Попробуйте создать новую учетную запись пользователя (используя для этого вашу обычную учетную запись). Попытка должна завершиться неудачей. (Подробности процесса добавления учетных записей пользователей описаны в следующей главе.)

```
laura@debian7:~$ useradd valentina
-su: useradd: command not found
laura@debian7:~$ /usr/sbin/useradd valentina
useradd: Отказано в доступе.
useradd: cannot lock /etc/passwd; try again later.
```

Вполне возможно, что утилита `useradd` будет расположена по пути `/sbin/useradd` в файловой системе вашего компьютера.

8. А теперь попытайтесь сделать то же самое, но добавив вызов утилиты **sudo** перед вашей командой.

```
laura@debian7:~$ sudo /usr/sbin/useradd valentina
[sudo] password for laura:
laura is not in the sudoers file.  This incident will be reported.
laura@debian7:~$
```

Обратите внимание на то, что пользователь **laura** не имеет разрешения на использование утилиты **sudo** в данной системе.

Глава 26. Управление учетными записями пользователей

Прочитав данную главу, вы научитесь использовать утилиты **useradd**, **usermod** и **userdel** для создания, модификации и удаления учетных записей пользователей.

Вам понадобятся привилегии пользователя **root** в системе Linux для выполнения действий, описанных в данной главе.

26.1. Управление учетными записями пользователей

Управление учетными записями пользователей в Linux может осуществляться тремя равноценными способами. Во-первых, вы можете использовать инструменты с **графическим интерфейсом**, предоставляемые вашим дистрибутивом. Внешний вид и принцип работы этих инструментов зависит от используемого вами дистрибутива. В том случае, если вы являетесь неопытным пользователем своей домашней системы Linux, используйте инструмент с графическим интерфейсом, предоставляемый вашим дистрибутивом. Такой подход гарантированно позволит избежать проблем.

Другим вариантом является использование таких инструментов с интерфейсом **командной строки**, как `useradd`, `usermod`, `gpasswd`, `passwd` и других. Администраторы серверов с большой вероятностью используют именно эти инструменты, так как они им знакомы, а также поставляются в неизменном виде в составе различных дистрибутивов. В данной главе будут рассматриваться именно эти инструменты с интерфейсом командной строки.

Третий довольно радикальный способ управления учетными записями пользователей заключается в непосредственном **редактировании локальных файлов конфигурации** с помощью текстового редактора vi (или vipw/vigr). Не пытайтесь делать это при работе с системами, находящимися в промышленной эксплуатации, в том случае, если вы не обладаете соответствующими знаниями!

26.2. Файл /etc/passwd

Локальная база данных учетных записей пользователей в Linux (и в большинстве систем Unix) расположена в файле **`/etc/passwd`**.

```
[root@RHEL5 ~]# tail /etc/passwd
inge:x:518:524:art dealer:/home/inge:/bin/ksh
ann:x:519:525:flute player:/home/ann:/bin/bash
frederik:x:520:526:rubius poet:/home/frederik:/bin/bash
steven:x:521:527:roman emperor:/home/steven:/bin/bash
pascale:x:522:528:artist:/home/pascale:/bin/ksh
geert:x:524:530:kernel developer:/home/geert:/bin/bash
wim:x:525:531:master damuti:/home/wim:/bin/bash
sandra:x:526:532:radish stresser:/home/sandra:/bin/bash
annelies:x:527:533:sword fighter:/home/annelies:/bin/bash
laura:x:528:534:art dealer:/home/laura:/bin/ksh
```

Как вы можете заметить, данный файл содержит данные в форме таблицы с семью столбцами, разделенными символом двоеточия. В столбцах содержатся имя пользователя, символ x, идентификатор пользователя, идентификатор основной группы пользователя, описание учетной записи пользователя, путь к домашней директории пользователя, а также путь к исполняемому файлу командной оболочки, используемой для входа пользователя в систему.

Дополнительная информация о данном файле может быть найдена на странице руководства, для получения доступа к которой может использоваться команда **`man 5 passwd`**.

```
[root@RHEL5 ~]# man 5 passwd
```

26.3. Пользователь root

Учетная запись пользователя **root**, также называемого **суперпользователем**, является наиболее привилегированной учетной записью вашей системы Linux. Данный пользователь может делать практически все, включая создание учетных записей других пользователей. Пользователь **root** всегда имеет идентификатор, равный **0** (вне зависимости от имени учетной записи).

```
[root@RHEL5 ~]# head -1 /etc/passwd
root:x:0:0:root:/root:/bin/bash
```

26.4. Утилита useradd

Вы можете добавлять учетные записи пользователей в базу данных с помощью утилиты **useradd**. В примере ниже показана методика добавления учетной записи пользователя с именем **yanina** (последний параметр) с одновременным созданием домашней директории пользователя (**-m**), установкой имени этой домашней директории (**-d**) и добавлением описания учетной записи (**-c**).

```
[root@RHEL5 ~]# useradd -m -d /home/yanina -c "yanina
wickmayer" yanina
[root@RHEL5 ~]# tail -1 /etc/passwd
yanina:x:529:529:yanina wickmayer:/home/yanina:/bin/bash
```

Как видно из примера, пользователь с именем yanina получил идентификатор 529, а также идентификатор **основной группы** 529.

26.5. Файл /etc/default/useradd

Как в дистрибутиве Red Hat Enterprise Linux, так и в дистрибутивах Debian/Ubuntu имеется файл **/etc/default/useradd**, который содержит некоторые стандартные параметры пользовательского окружения. Помимо команды cat, вы можете использовать команду **useradd -D** для ознакомления с содержимым данного файла.

```
[root@RHEL4 ~]# useradd -D
GROUP=100
HOME=/home
INACTIVE=-1
EXPIRE=
SHELL=/bin/bash
SKEL=/etc/skel
```

26.6. Утилита userdel

Вы можете удалить учетную запись пользователя **yanina** с помощью утилиты **userdel**. Параметр **-r** утилиты **userdel** позволяет также удалить домашнюю директорию пользователя.

```
[root@RHEL5 ~]# userdel -r yanina
```

26.7. Утилита usermod

Вы можете модифицировать параметры учетной записи пользователя с помощью утилиты **usermod**. В данном примере утилита **usermod** используется для изменения описания учетной записи пользователя **harry**.

```
[root@RHEL4 ~]# tail -1 /etc/passwd
harry:x:516:520:harry potter:/home/harry:/bin/bash
[root@RHEL4 ~]# usermod -c 'wizard' harry
[root@RHEL4 ~]# tail -1 /etc/passwd
harry:x:516:520:wizard:/home/harry:/bin/bash
```

26.8. Создание домашних директорий пользователей

Простейший способ создания домашней директории пользователя заключается в передаче параметра **-m** утилите **useradd** (вполне вероятно, что данный параметр передается по умолчанию в вашем дистрибутиве Linux).

Менее простой способ заключается в самостоятельном создании домашней директории с помощью команды **mkdir**, что также подразумевает необходимость установки владельца и прав доступа к данной директории с помощью утилит **chmod** и **chown** (обе утилиты подробно обсуждаются в другой главе).

```
[root@RHEL5 ~]# mkdir /home/laura
[root@RHEL5 ~]# chown laura:laura /home/laura
[root@RHEL5 ~]# chmod 700 /home/laura
[root@RHEL5 ~]# ls -ld /home/laura/
drwx------ 2 laura laura 4096 июн 24 15:17 /home/laura/
```

26.9. Директория /etc/skel/

В случае использовании параметра **-m** утилиты **useradd** содержимое директории **/etc/skel/** копируется в создаваемую домашнюю директорию пользователя. В директории **/etc/skel/** находятся некоторые (обычно скрытые) файлы, которые содержат стандартные параметры профиля пользователя и значения параметров приложе-

ний. Таким образом, директория **/etc/skel/** выступает в роли шаблона домашней директории и стандартного профиля пользователя.

```
[root@RHEL5 ~]# ls -la /etc/skel/
итого 48
drwxr-xr-x  2 root root  4096 апр  1 00:11 .
drwxr-xr-x 97 root root 12288 июн 24 15:36 ..
-rw-r--r--  1 root root    24 июл 12  2006 .bash_logout
-rw-r--r--  1 root root   176 июл 12  2006 .bash_profile
-rw-r--r--  1 root root   124 июл 12  2006 .bashrc
```

26.10. Удаление домашних директорий пользователей

В случае использования параметра **-r** утилиты **userdel** вы можете быть уверены в том, что домашняя директория пользователя будет удалена вместе с его учетной записью.

```
[root@RHEL5 ~]# ls -ld /home/wim/
drwx------ 2 wim wim 4096 июн 24 15:19 /home/wim/
[root@RHEL5 ~]# userdel -r wim
[root@RHEL5 ~]# ls -ld /home/wim/
ls: невозможно получить доступ к /home/wim/: Нет такого файла или каталога
```

26.11. Командная оболочка, используемая для входа в систему

В файле **/etc/passwd** содержится информация о **командной оболочке**, **используемой для входа пользователя в систему**. Как вы можете увидеть в примере ниже, пользователь annelies будет использовать для входа в систему командную оболочку **/bin/bash**, а пользователь laura - командную оболочку **/bin/ksh**.

```
[root@RHEL5 ~]# tail -2 /etc/passwd
annelies:x:527:533:sword fighter:/home/annelies:/bin/bash
laura:x:528:534:art dealer:/home/laura:/bin/ksh
```

Вы можете использовать команду usermod для изменения командной оболочки пользователя.

```
[root@RHEL5 ~]# usermod -s /bin/bash laura
[root@RHEL5 ~]# tail -1 /etc/passwd
laura:x:528:534:art dealer:/home/laura:/bin/bash
```

26.12. Утилита chsh

Пользователи могут изменять используемую для входа в систему командную оболочку с помощью утилиты **chsh**. В примере ниже пользователь laura в первую очередь получает список доступных командных оболочек (также данный список может быть получен с помощью команды **cat /etc/shells**), после чего изменяет свою командную оболочку на **Korn shell** (/bin/ksh). При следующем входе в систему пользователю laura по умолчанию будет предоставлена командная оболочка Korn shell вместо bash.

```
[laura@centos7 ~]$ chsh -l
/bin/sh
/bin/bash
/sbin/nologin
/usr/bin/sh
/usr/bin/bash
/usr/sbin/nologin
/bin/ksh
/bin/tcsh
/bin/csh
[laura@centos7 ~]$
```

Учтите, что в дистрибутиве Debian не существует параметра **-l** упомянутой утилиты, а также в данном примере предполагается, что командные оболочки **ksh** и **csh** установлены в системе.

В примере ниже показано, как пользователь **laura** может изменить свою стандартную командную оболочку (которая будет активирована при следующем входе в систему).

```
[laura@centos7 ~]$ chsh -s /bin/ksh
Изменение шелла для laura.
Пароль:
Шелл изменён.
```

26.13. Практическое задание: управление учетными записями пользователей

1. Создайте учетную запись пользователя с именем `serena` и описанием (или комментарием) "**`Serena Williams`**", а также домашнюю директорию этого пользователя. Выполните необходимые действия в рамках одной команды.

2. Создайте учетную запись пользователя с именем **`venus`**, указанием на необходимость использования командной оболочки bash и описанием "**`Venus Williams`**", а также домашнюю директорию этого пользователя с помощью одной команды.

3. Убедитесь в том, что обоим пользователям соответствуют корректные записи в файлах **`/etc/passwd`**, **`/etc/shadow`** и **`/etc/group`**.

4. Проверьте корректность создания домашних директорий пользователей.

5. Создайте учетную запись пользователя с именем **`einstime`** и утилитой **`/bin/date`** в качестве стандартной командной оболочки.

6. Что случится, если вы войдете в систему под именем пользователя **`einstime`**? Можете ли вы сделать предположение о реальной ситуации, в которой было бы полезно заменить стандартную командную оболочку пользователя на приложение?

7. Создайте файл с именем **`welcome.txt`** и убедитесь в том, что каждый новый пользователь будет обнаруживать данный файл в своей домашней директории.

8. Проверьте корректность размещения созданного файла в файловой системе, создав (и удалив) тестовую учетную запись пользователя.

9. Измените стандартную командую оболочку для входа в систему пользователя **`serena`** на командную оболочку **`/bin/bash`**. Осуществите необходимые проверки перед изменением командной оболочки и после него.

26.14. Корректная процедура выполнения практического задания: управление учетными записями пользователей

1. Создайте учетную запись пользователя с именем **serena** и описанием (или комментарием) "**Serena Williams**", а также домашнюю директорию этого пользователя. Выполните необходимые действия в рамках одной команды.

```
root@debian7:~# useradd -m -c 'Serena Williams' serena
```

2. Создайте учетную запись пользователя с именем **venus**, указанием на необходимость использования командной оболочки bash и описанием "**Venus Williams**", а также домашнюю директорию этого пользователя с помощью одной команды.

```
root@debian7:~# useradd -m -c "Venus Williams" -s
/bin/bash venus
```

3. Убедитесь в том, что обоим пользователям соответствуют корректные записи в файлах **/etc/passwd**, **/etc/shadow** и **/etc/group**.

```
root@debian7:~# tail -2 /etc/passwd
serena:x:1008:1010:Serena Williams:/home/serena:/bin/sh
venus:x:1009:1011:Venus Williams:/home/venus:/bin/bash
root@debian7:~# tail -2 /etc/shadow
serena:!:16358:0:99999:7:::
venus:!:16358:0:99999:7:::
root@debian7:~# tail -2 /etc/group
serena:x:1010:
venus:x:1011:
```

4. Проверьте корректность создания домашних директорий пользователей.

```
root@debian7:~# ls -lrt /home | tail -2
drwxr-xr-x 2 serena    serena    4096 окт 15 10:50 serena
drwxr-xr-x 2 venus     venus     4096 окт 15 10:59 venus
root@debian7:~#
```

5. Создайте учетную запись пользователя с именем **einstime** и утилитой **/bin/time** в качестве стандартной командной оболочки.

```
root@debian7:~# useradd -s /bin/date einstime
```

Или даже лучший вариант:

```
root@debian7:~# useradd -s $(which date) einstime
```

6. Что случится, если вы войдете в систему под именем пользователя **einstime**? Можете ли вы сделать предположение о реальной ситуации, в которой было бы полезно заменить стандартную командную оболочку пользователя на приложение?

```
root@debian7:~# su - einstime
Ср окт 15 11:05:56 UTC 2014    # Вы получите вывод команды date
root@debian7:~#
```

Такая замена стандартной командной оболочки может оказаться полезной в том случае, если пользователю необходимо получить доступ только к одному приложению на сервере. Сразу же после входа в систему пользователь получает возможность работы с приложением, а после завершения работы этого приложения автоматически осуществляется выход из системы.

7. Создайте файл с именем **welcome.txt** и убедитесь в том, что каждый новый пользователь будет обнаруживать данный файл в своей домашней директории.

```
root@debian7:~# echo Hello > /etc/skel/welcome.txt
```

8. Проверьте корректность размещения созданного файла в файловой системе, создав (и удалив) тестовую учетную запись пользователя.

```
root@debian7:~# useradd -m test
root@debian7:~# ls -l /home/test
итого 4
-rw-r--r-- 1 test test 6 окт 15 11:16 welcome.txt
root@debian7:~# userdel -r test
root@debian7:~#
```

9. Измените стандартную командую оболочку для входа в систему пользователя **serena** на командную оболочку **/bin/bash**. Осуществите необходимые проверки перед изменением командной оболочки и после него.

```
root@debian7:~# grep serena /etc/passwd
serena:x:1008:1010:Serena Williams:/home/serena:/bin/sh
root@debian7:~# usermod -s /bin/bash serena
root@debian7:~# grep serena /etc/passwd
serena:x:1008:1010:Serena Williams:/home/serena:/bin/bash
root@debian7:~#
```

Глава 27. Пароли пользователей

Данная глава содержит дополнительную информацию о паролях локальных пользователей.

В главе подробно описаны три метода установки паролей; с использованием утилиты **passwd**, с использованием реализации алгоритма шифрования **passwd** из библиотеки **openssl**, а также с использованием функции **crypt** в программе на языке C.

Кроме того, в данной главе обсуждаются параметры паролей и методики отключения, аннулирования или блокировки учетных записей пользователей.

27.1. Утилита passwd

Пароли пользователей могут устанавливаться с помощью утилиты **passwd**. При смене паролей пользователям придется предоставлять утилите свои старые пароли перед двукратным вводом новых паролей.

```
[tania@centos7 ~]$ passwd
Изменяется пароль пользователя tania.
Смена пароля для tania.
(текущий) пароль UNIX:
Новый пароль :
НЕУДАЧНЫЙ ПАРОЛЬ: В пароле должно быть не меньше 8 символов
Новый пароль :
НЕУДАЧНЫЙ ПАРОЛЬ: Пароль является палиндромом
Новый пароль :
НЕУДАЧНЫЙ ПАРОЛЬ: Пароль слишком схож с предыдущим
passwd: Использовано максимальное число попыток, заданное для службы
```

Как вы видите, утилита passwd может выполнять простейшую проверку с целью предотвращения использования слишком простых паролей для учетных записей пользователей. Пользователь **root** не должен выполнять эти правила (хотя он и будет видеть предупреждения). Также пользователь **root** не должен вводить старый пароль перед двукратным вводом нового пароля.

```
root@debian7:~# passwd tania
Новый пароль :
Повторите ввод нового пароля :
passwd: все данные аутентификации успешно обновлены.
```

27.2. Файл shadow

Пароли пользователей хранятся в зашифрованном виде в файле **/etc/shadow**. Файл **/etc/shadow** доступен только для чтения и может читаться исключительно пользователем **root**. В разделе, посвященном правам доступа к файлам, мы поговорим о том, как пользователям удается изменять свои пароли. На текущий момент вам нужно знать лишь о том, что пользователи могут изменять свои пароли с помощью утилиты **/usr/bin/passwd**.

```
[root@centos7 ~]# tail -4 /etc/shadow
paul:
$6$ikp2Xta5BT.Tml.p$2TZjNnOYNNQKpwLJqoGJbVsZG5/Fti8ovBRd.
VzRbiDSl7TEq\
IaSMH.TeBKnTS/SjlMruW8qffC0JNORW.BTW1:16338:0:99999:7:::
tania:
$6$8Z/zovxj$9qvoqT8i9KIrmN.k4EQwAF5ryz5yzNwEvYjAa9L5XVXQu
.z4DlpvMREH\
eQpQzvRnqFdKkVj17H5ST.c79HDZw0:16356:0:99999:7:::
laura:
$6$glDuTY5e$/NYYWLxfHgZFWeoujaXSMcR.Mz.lGOxtcxFocFVJNb98n
bTPhWFXfKWG\
SyYh1WCv6763Wq54.w24Yr3uAZBOm/:16356:0:99999:7:::
valentina:
$6$jrZa6PVI$1uQgqR6En9mZB6mKJ3LXRB4CnFko6LRhbh.v4iqUk9MVr
eui1lv7\
GxHOUDSKA0N55ZRNhGHa6T2ouFnVno/0o1:16356:0:99999:7:::
[root@centos7 ~]#
```

Файл **/etc/shadow** содержит таблицу с девятью разделенными двоеточиями столбцами. Эти девять столбцов (слева направо) со-

держат имя пользователя, зашифрованный пароль, время последнего изменения пароля (первый день соответствует 1 января 1970 года), количество дней, в течение которых пароль должен оставаться неизменным, день истечения срока действия пароля, количество дней перед истечением срока действия пароля, в течение которых должно выводиться предупреждение, количество дней после истечения срока действия пароля, по прошествии которых учетная запись должна быть отключена, а также день, когда учетная запись была отключена (также с начала 1970 года). Последнее поле пока не имеет значения.

Все пароли в примере выше являются хэшами фразы **hunter2**.

27.3. Шифрование ключевых фраз с помощью утилиты passwd

Пароли пользователей системы хранятся в зашифрованном формате. Их шифрование осуществляется средствами функции **crypt**. Простейший (и рекомендованный) способ добавления пользователя с заданным паролем в систему заключается в добавлении пользователя в систему с помощью команды **useradd -m имя_пользователя** с последующей установкой пароля с помощью утилиты **passwd**.

```
[root@RHEL4 ~]# useradd -m xavier
[root@RHEL4 ~]# passwd xavier
Изменяется пароль пользователя xavier.
Новый пароль :
Повторите ввод нового пароля :
passwd: все данные аутентификации успешно обновлены.
[root@RHEL4 ~]#
```

27.4. Шифрование ключевых фраз с помощью утилиты openssl

Другой способ создания учетных записей пользователей с паролями заключается в использовании параметра `-p` утилиты `useradd`, но в случае использования данного параметра утилите необходимо передавать уже зашифрованный пароль. Вы можете зашифровать пароль с помощью команды **openssl passwd**.

Команда `openssl passwd` позволяет сгенерировать несколько отдельных хэшей для одного и того же пароля, причем для этой цели используется значение `salt`.

```
paul@rhel65:~$ openssl passwd hunter2
```

```
86jcUNlnGDFpY
paul@rhel65:~$ openssl passwd hunter2
Yj7mDO9OAnvq6
paul@rhel65:~$ openssl passwd hunter2
YqDcJeGoDbzKA
paul@rhel65:~$
```

Данное значение **salt** может быть выбрано произвольным образом и будет отображаться в виде двух первых символов хэша.

```
paul@rhel65:~$ openssl passwd -salt 42 hunter2
42ZrbtP1Ze8G.
paul@rhel65:~$ openssl passwd -salt 42 hunter2
42ZrbtP1Ze8G.
paul@rhel65:~$ openssl passwd -salt 42 hunter2
42ZrbtP1Ze8G.
paul@rhel65:~$
```

В данном примере показана методика создания учетной записи пользователя с паролем.

```
root@rhel65:~# useradd -m -p $(openssl passwd hunter2)
mohamed
```

Помните о том, что после выполнения данной команды ваш пароль в открытом виде будет сохранен в файле истории команд командной оболочки!

27.5. Шифрование ключевых фраз с помощью функции crypt

Третий вариант заключается в создании вашей собственной программы на языке программирования C, которая будет использовать функцию crypt с ее последующей компиляцией в бинарный файл.

```
paul@rhel65:~$ cat MyCrypt.c
#include <stdio.h>
#define __USE_XOPEN
#include <unistd.h>

int main(int argc, char** argv)
{
```

```
 if(argc==3)
   {
       printf("%s\n", crypt(argv[1],argv[2]));
   }
   else
   {
       printf("Использование: MyCrypt $пароль $salt\n" );
   }
  return 0;
}
```

Эта простая программа может быть скомпилирована средствами компилятора **gcc** с помощью команды, аналогичной следующей:

```
paul@rhel65:~$ gcc MyCrypt.c -o MyCrypt -lcrypt
```

При использовании описанной программы MyCrypt нам придется передавать два параметра. Первым параметром является незашифрованный пароль, а вторым - значение salt. Значение salt используется для изменения алгоритма шифрования в соответствии с одним из 4096 различных вариантов. Данная вариация используется для предотвращения ситуаций, в которых два пользователя с одним и тем же паролем могут использовать одну и ту же запись в файле **/etc/shadow**.

```
paul@rhel65:~$ ./MyCrypt hunter2 42
42ZrbtP1Ze8G.
paul@rhel65:~$ ./MyCrypt hunter2 33
33d6taYSiEUXI
```

Обратили ли вы внимание на то, что первые два символа зашифрованного пароля являются значением **salt**?

Стандартный вывод функции crypt генерируется с использованием алгоритма DES, который давно устарел и может быть взломан за считанные минуты. Более предпочтительным алгоритмом для шифрования паролей является алгоритм **md5**, который может быть распознан по начальным символам значения salt 1.

```
paul@rhel65:~$ ./MyCrypt hunter2 '$1$42'
$1$42$7l6Y3xT5282XmZrtDOF9f0
paul@rhel65:~$ ./MyCrypt hunter2 '$6$42'
```

```
$6$42$OqFFAVnI3gTSYG0yI9TZWX9cpyQzwIop7HwpG1LLEsNBiMr4w6O
vLX1KDa./UpwXfrFk1i...
```

Длина значения salt для алгоритма **md5** может достигать восьми символов. Значения salt хранятся в открытом виде в строках файла **/etc/shadow** между вторым и третьим символами $, поэтому никогда не используйте строку пароля в качестве значения salt!

```
paul@rhel65:~$ ./MyCrypt hunter2 '$1$hunter2'
$1$hunter2$YVxrxDmidq7Xf8Gdt6qM2.
```

27.6. Файл /etc/login.defs

Файл **/etc/login.defs** содержит некоторые стандартные значения параметров паролей пользователей, таких, как период устаревания паролей или ограничения длины паролей. (Там же вы можете обнаружить числовые ограничения идентификаторов пользователей и идентификаторов групп, а также указание на то, должна ли создаваться домашняя директория пользователя по умолчанию).

```
root@rhel65:~# grep ^PASS /etc/login.defs
PASS_MAX_DAYS   99999
PASS_MIN_DAYS   0
PASS_MIN_LEN    5
PASS_WARN_AGE   7
```

В дистрибутиве Debian также имеется данный файл.

```
root@debian7:~# grep PASS /etc/login.defs
#  PASS_MAX_DAYS   Максимальное количество дней, в течение которых может использоваться пароль.
#  PASS_MIN_DAYS   Минимальное количество дней, которые должны пройти между изменениями паролей.
#  PASS_WARN_AGE   Количество дней, в течение которых будут выводиться предупреждения об истечении строка действия пароля.
PASS_MAX_DAYS   99999
PASS_MIN_DAYS   0
PASS_WARN_AGE   7
#PASS_CHANGE_TRIES
#PASS_ALWAYS_WARN
#PASS_MIN_LEN
#PASS_MAX_LEN
```

```
# NO_PASSWORD_CONSOLE
root@debian7:~#
```

27.7. Утилита chage

Утилита **chage** может использоваться для установки даты истечения срока действия пользовательской учетной записи (**-E**), установки минимального (**-m**) и максимального (**-M**) срока действия пароля, даты истечения срока действия пароля, а также установки количества дней, в течение которых выводятся предупреждения об истечении срока действия пароля. Многие из этих функций реализованы также и в рамках утилиты **passwd**. Параметр **-l** утилиты **chage** позволяет ознакомиться с текущими значениями этих параметров для указанного пользователя.

```
root@rhel65:~# chage -l paul
Последний раз пароль был изменён
: мар 27, 2014
Срок действия пароля истекает
: никогда
Пароль будет деактивирован через
: никогда
Срок действия учётной записи истекает
: никогда
Минимальное количество дней между сменой пароля
: 0
Максимальное количество дней между сменой пароля
: 99999
Количество дней с предупреждением перед деактивацией па-
роля       : 7
root@rhel65:~#
```

27.8. Блокировка учетных записей

Пароли из файла **/etc/shadow** не могут начинаться с символа восклицательного знака. Если второе поле в строке из файла **/etc/passwd** начинается с символа восклицательного знака, пароль не может использоваться.

Использование данной особенности обычно называется **блокировкой**, **деактивацией** или **отключением** пользовательской учетной записи. Помимо текстового редактора vi (или vipw) вы можете использовать утилиту **usermod** для этой цели.

Первая команда из следующего примера предназначена для вывода хэшированного пароля пользователя **laura** из файла **/etc/shadow**. Следующая команда позволяет деактивировать пароль пользователя **laura**, после чего laura не сможет пройти фазу аутентификации при условии использования данного пароля.

```
root@debian7:~# grep laura /etc/shadow | cut -c1-70
laura:
$6$JYj4JZqp$stwwWACp3OtE1R2aZuE87j.nbW.puDkNUYVk7mCHfCVMa
3CoDUJV
root@debian7:~# usermod -L laura
```

Как видно в примере ниже, в результате перед хэшем пароля просто добавляется символ восклицательного знака.

```
root@debian7:~# grep laura /etc/shadow | cut -c1-70
laura:!
$6$JYj4JZqp$stwwWACp3OtE1R2aZuE87j.nbW.puDkNUYVk7mCHfCVMa
3CoDUJ
root@debian7:~#
```

Пользователь root (а также пользователи с доступом к команде **su** посредством **sudo**) все так же будут иметь возможность использовать команду **su** для работы с учетной записью пользователя **laura** (так как в этом случае пароль пользователя не требуется). Кроме того, следует учитывать тот факт, что **laura** все еще сможет войти в систему в том случае, если она заранее настроила доступ по ssh без использования пароля!

```
root@debian7:~# su - laura
laura@debian7:~$
```

Вы можете снова разблокировать учетную запись с помощью команды **usermod -U**.

```
root@debian7:~# usermod -U laura
root@debian7:~# grep laura /etc/shadow | cut -c1-70
laura:
$6$JYj4JZqp$stwwWACp3OtE1R2aZuE87j.nbW.puDkNUYVk7mCHfCVMa
3CoDUJV
```

Следует подходить с осторожностью к работе и учитывать различия параметров интерфейса командной строки утилит **passwd**, **usermod** и **useradd** в различных дистрибутивах Linux. Проверяйте содержимое локальных файлов при использовании таких возможно-

стей, как "**деактивация**, **отключение** или **блокировка**" учетных записей пользователей и их паролей.

27.9. Редактирование локальных файлов

В том случае, если вы все еще хотите отредактировать вручную файл **/etc/passwd** или **/etc/shadow**, располагая информацией о командах для управления паролями, используйте утилиту **vipw** вместо непосредственного использования текстового редактора `vi(m)`. Утилита **vipw** осуществляет корректную блокировку данного файла.

```
[root@RHEL5 ~]# vipw /etc/passwd
vipw: the password file is busy (/etc/ptmp present)
```

27.10. Практическое задание: пароли пользователей

1. Установите пароль **hunter2** для пользователя **serena**.

2. Также установите пароль для пользователя **venus**, после чего заблокируйте учетную запись пользователя **venus** с помощью утилиты **usermod**. Проверьте наличие признаков блокировки в файле **/etc/shadow** до и после осуществления блокировки.

3. Используйте команду **passwd -d** для деактивации пароля пользователя **serena**. Проверьте наличие строки для пользователя **serena** в файле **/etc/shadow** до и после осуществления деактивации.

4. Каковы различия между операцией блокировки пользовательской учетной записи и операцией деактивации пароля пользовательской учетной записи, которые мы только что осуществляли с помощью команд **usermod -L** и **passwd -d**?

5. Попытайтесь изменить пароль `serena` на пароль `serena as serena`.

6. Сделайте так, чтобы пользователь **serena** был обязан изменять пароль через каждые 10 дней.

7. Сделайте так, чтобы каждый новый пользователь был обязан изменять свой пароль через каждые 10 дней.

8. Воспользуйтесь учетной записью пользователя `root` для создания резервной копии файла **/etc/shadow**. Используйте текстовый редактор **vi** для копирования хэша пароля **hunter2** из строки пользователя

venus в строку пользователя **serena**. Может ли после этого пользователь **serena** войти в систему с паролем **hunter2**?

9. В каких случаях следует использовать утилиту **vipw** вместо **vi**? Какая проблема может возникнуть при использовании текстового редактора **vi** или **vim** для редактирования файла паролей?

10. Используйте команду **chsh** для вывода списка доступных командных оболочек (данная команда будет работать исключительно в дистрибутивах RHEL/CentOS/Fedora) и сравните вывод с выводом команды **cat /etc/shells**.

11. Какой параметр утилиты **useradd** позволяет установить имя домашней директории пользователя?

12. Как можно определить, заблокирован или разблокирован пароль пользователя **serena**? Предложите решение на основе утилиты **grep**, а также решение на основе утилиты **passwd**.

27.11. Корректная процедура выполнения практического задания: пароли пользователей

1. Установите пароль **hunter2** для пользователя **serena**.

```
root@debian7:~# passwd serena
Изменяется пароль пользователя serena.
Новый пароль :
Повторите ввод нового пароля :
passwd: все данные аутентификации успешно обновлены.
```

2. Также установите пароль для пользователя **venus**, после чего заблокируйте учетную запись пользователя **venus** с помощью утилиты **usermod**. Проверьте наличие признаков блокировки в файле **/etc/shadow** до и после осуществления блокировки.

```
root@debian7:~# passwd venus
Изменяется пароль пользователя venus.
Новый пароль :
Повторите ввод нового пароля :
passwd: все данные аутентификации успешно обновлены.
root@debian7:~# grep venus /etc/shadow | cut -c1-70
```

```
venus:
$6$gswzXICW$uSnKFV1kFKZmTPaMVS4AvNA/KO27OxN0v5LHdV9ed0gTy
XrjUeM/
root@debian7:~# usermod -L venus
root@debian7:~# grep venus /etc/shadow | cut -c1-70
venus:!
$6$gswzXICW$uSnKFV1kFKZmTPaMVS4AvNA/KO27OxN0v5LHdV9ed0gTy
XrjUeM
```

Обратите внимание на то, что в результате выполнения команды **usermod -L** перед хэшем пароля добавляется символ восклицательного знака.

3. Используйте команду **passwd -d** для деактивации пароля пользователя **serena**. Проверьте наличие строки для пользователя serena в файле **/etc/shadow** до и после осуществления деактивации.

```
root@debian7:~# grep serena /etc/shadow | cut -c1-70
serena:
$6$Es/omrPE$F2Ypu8kpLrfKdW0v/UIwA5jrYyBD2nwZ/dt.i/IypRgiP
ZSdB/B
root@debian7:~# passwd -d serena
Удаляется пароль для пользователя serena.
passwd: Успех
root@debian7:~# grep serena /etc/shadow
serena::16358:0:99999:7:::
root@debian7:~#
```

4. Каковы различия между операцией блокировки пользовательской учетной записи и операцией деактивации пароля пользовательской учетной записи, которые мы только что осуществляли с помощью команд **usermod -L** и **passwd -d**?

Блокировка предотвратит вход пользователя в систему с использованием установленного пароля благодаря добавлению символа ! перед хэшем пароля в файле **/etc/shadow**.

Деактивация с помощью утилиты **passwd** приведет к удалению хэша пароля из файла **/etc/shadow**.

5. Попытайтесь изменить пароль serena на пароль serena as serena.

Войдите в систему с именем пользователя serena, после чего выполните команду: passwd serena... Исполнение команды должно завершиться неудачей!

6. Сделайте так, чтобы пользователь **serena** был обязан изменять пароль через каждые 10 дней.

chage -M 10 serena

7. Сделайте так, чтобы каждый новый пользователь был обязан изменять свой пароль через каждые 10 дней.

vi /etc/login.defs (и измените значение переменной PASS_MAX_DAYS на 10)

8. Воспользуйтесь учетной записью пользователя root для создания резервной копии файла **/etc/shadow**. Используйте текстовый редактор **vi** для копирования хэша пароля **hunter2** из строки пользователя **venus** в строку пользователя **serena**. Может ли после этого пользователь **serena** войти в систему с паролем **hunter2**?

Да, может.

9. В каких случаях следует использовать утилиту **vipw** вместо **vi**? Какая проблема может возникнуть при использовании текстового редактора **vi** или **vim** для редактирования файла паролей?

Утилита **vipw** выведет предупреждение в том случае, если кто-либо еще в данный момент редактирует данный файл (с помощью утилиты **vipw**).

10. Используйте команду **chsh** для вывода списка доступных командных оболочек (данная команда будет работать исключительно в дистрибутивах RHEL/CentOS/Fedora) и сравните вывод с выводом команды **cat /etc/shells**.

chsh -l

cat /etc/shells

11. Какой параметр утилиты **useradd** позволяет установить имя домашней директории пользователя?

```
-d
```

12. Как можно определить, заблокирован или разблокирован пароль пользователя **serena**? Предложите решение на основе утилиты **grep**, а также решение на основе утилиты **passwd**.

```
grep serena /etc/shadow
passwd -S serena
```

Глава 28. Профили пользователей

Вошедшие в систему пользователи получают в свое распоряжение множество установленных (и изменяемых) псевдонимов команд, переменных и функций, но откуда они берутся? **Командная оболочка** использует множество загрузочных файлов сценариев, которые исполняются (или **подключаются**) в момент ее вызова. Ниже приводится обзор упомянутых загрузочных сценариев.

28.1. Системный профиль

Как командная оболочка **bash**, так и командная оболочка **ksh** будет проверять существование файла **/etc/profile** и подключать его в случае существования.

При чтении данного файла вы можете обнаружить (как в дистрибутиве Debian, так и в дистрибутиве Red Hat Enterprise Linux), что именно его силами создается переменная окружения PATH (помимо других переменных окружения). Данный сценарий также может изменять значение переменной окружения PS1, устанавливать значение переменной окружения HOSTNAME и исполнять дополнительные сценарии, такие, как **/etc/inputrc**.

В примере ниже для демонстрации механизма манипуляций с значением переменной окружения PATH в файле **/etc/profile** из дистрибутива Debian используется утилита grep.

```
root@debian7:~# grep PATH /etc/profile

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

PATH="/usr/local/bin:/usr/bin:/bin:/usr/local/games:/usr/
games"
```

```
export PATH
root@debian7:~#
```

В следующем примере утилита grep используется для демонстрации аналогичного механизма манипуляций с значением переменной окружения PATH в файле **/etc/profile** из дистрибутива RHEL/CentOS7.

```
[root@centos7 ~]# grep PATH /etc/profile
    case ":${PATH}:" in
                PATH=$PATH:$1
                PATH=$1:$PATH
export PATH USER LOGNAME MAIL HOSTNAME HISTSIZE
HISTCONTROL
[root@centos7 ~]#
```

Пользователь **root** может использовать данный сценарий для объявления псевдонимов команд, функций и переменных для каждого из пользователей в системе.

28.2. Файл ~/.bash_profile

В том случае, если данный файл существует в домашней директории пользователя, командная оболочка **bash** осуществит его подключение. В дистрибутивах Debian 5/6/7 данного файла по умолчанию не существует.

В дистрибутивах RHEL7/CentOS7 используется небольшой сценарий **~/.bash_profile**, который проверяет существование сценария **~/.bashrc** и осуществляет его подключение в случае существования. Также он добавляет путь к директории $HOME/bin к списку путей, хранящемуся в переменной окружения $PATH.

```
[root@rhel7 ~]# cat /home/paul/.bash_profile
# .bash_profile

# Получение псевдонимов команд и функций
if [ -f ~/.bashrc ]; then
        . ~/.bashrc
fi

# Пользовательское окружение и запускаемые программы
```

```
PATH=$PATH:$HOME/.local/bin:$HOME/bin


export PATH
[root@rhel7 ~]#
```

28.3. Файл ~/.bash_login

Если сценария **.bash_profile** не существует, командная оболочка **bash** будет выполнять проверку существования сценария **~/.bash_login** и его подключение в случае существования.

Данного файла не существует по умолчанию ни в дистрибутиве Debian, ни в дистрибутиве Red Hat.

28.4. Файл ~/.profile

Если в домашней директории пользователя не существует ни сценария **~/.bash_profile**, ни сценария **~/bash_login**, командная оболочка bash будет проверять наличие сценария **~/.profile** и исполнять его. По умолчанию данного сценария не существует в дистрибутиве Red Hat.

В дистрибутиве Debain данный сценарий может исполнять сценарий **~/.bashrc** и добавлять путь к директории $HOME/bin в список директорий, хранящийся в переменной окружения $PATH.

```
root@debian7:~# tail -11 /home/paul/.profile
if [ -n "$BASH_VERSION" ]; then
    # подключение сценария .bashrc в случае его существования
    if [ -f "$HOME/.bashrc" ]; then
        . "$HOME/.bashrc"
    fi
fi


# изменение значения переменной PATH с целью включения пути к пользовательской директории бинарных файлов в случае ее существования
if [ -d "$HOME/bin" ] ; then
    PATH="$HOME/bin:$PATH"
fi
```

По умолчанию данного файла не существует в дистрибутивах RHEL/CentOS.

28.5. Файл ~/.bashrc

Сценарий **~/.bashrc** обычно подключается средствами других сценариев. Давайте рассмотрим задачи, решаемые этим сценарием по умолчанию.

В дистрибутиве Red Hat используется очень простой сценарий **~/.bashrc**, проверяющий существование сценария **/etc/bashrc** и подключающий его в случае существования. Также он позволяет объявлять специальные псевдонимы команд и функции.

```
[root@rhel7 ~]# cat /home/paul/.bashrc
# .bashrc

# Подключение файла с глобальными объявлениями
if [ -f /etc/bashrc ]; then
        . /etc/bashrc
fi

# Раскомментируйте следующую строку, если вам не нравится
функция автоматического разделения вывода утилиты
systemctl на страницы:
# export SYSTEMD_PAGER=

# Пользовательские псевдонимы команд и функции
```

В дистрибутиве Debian данный сценарий немного длиннее и также осуществляет установку значений переменной $PS1, некоторых переменных истории, а также множества активных и неактивных псевдонимов команд.

```
root@debian7:~# wc -l /home/paul/.bashrc
110 /home/paul/.bashrc
```

28.6. Файл ~/.bash_logout

При завершении работы командная оболочка **bash** может исполнять сценарий **~/.bash_logout**.

В дистрибутиве Debian данная возможность используется для очистки консоли.

```
serena@deb503:~$ cat .bash_logout
# ~/.bash_logout: исполняется bash(1) при завершении ра-
боты командной оболочки.
```

```
# при завершении работы с консолью в целях повышения без-
опасности экран очищается


if [ "$SHLVL" = 1 ]; then
    [ -x /usr/bin/clear_console ] &&
/usr/bin/clear_console -q
fi
```

В дистрибутиве Red Hat Enterprise Linux 5 данный сценарий осуществляет простой вызов бинарного файла **/usr/bin/clear**.

```
[serena@rhel53 ~]$ cat .bash_logout
# ~/.bash_logout


/usr/bin/clear
```

В дистрибутивах Red Hat Enterprise Linux 6 и 7 данный сценарий создается, но остается пустым (не содержит ничего кроме комментария).

```
paul@rhel65:~$ cat .bash_logout
# ~/.bash_logout
```

28.7. Обзор сценариев дистрибутива Debian

Ниже приведена таблица для дистрибутива Debian с указанием моментов запуска каждого из описанных стартовых сценариев командной оболочки bash.

Таблица 28.1. Пользовательское **окружение дистрибутива Debian**

Сценарий	su	su -	ssh	gdm
~/.bashrc	Не запускается	Запускается	Запускается	Запускается
~/.profile	Не запускается	Запускается	Запускается	Запускается
/etc/profile	Не запускается	Запускается	Запускается	Запускается
/etc/bash.bashrc	Запускается	Не запускается	Не запускается	Запускается

28.8. Обзор сценариев дистрибутива RHEL5

Ниже приведена таблица для дистрибутива Red Hat Entrprise Linux 5 с указанием моментов запуска каждого из описанных стартовых сценариев командной оболочки `bash`.

Таблица 28.2. Пользовательское окружение дистрибутива Red Hat

Сценарий	`su`	`su -`	`ssh`	`gdm`
`~/.bashrc`	Запускается	Запускается	Запускается	Запускается
`~/.bash_profile`	Не запускается	Запускается	Запускается	Запускается
`/etc/profile`	Не запускается	Запускается	Запускается	Запускается
`/etc/bashrc`	Запускается	Запускается	Запускается	Запускается

28.9. Практическое задание: профили пользователей

1. Выведите список всех файлов профиля пользователя в вашей системе.

2. Прочитайте содержимое каждого из этих файлов, обычно в них осуществляется **подключение** дополнительных сценариев.

3. Объявите уникальную переменную, псевдоним команды и функцию в каждом из этих файлов.

4. Воспользуйтесь различными способами получения командной оболочки (su, su -, ssh, tmux, gnome-terminal, Ctrl-Alt-F1, ...) и проверьте, какие из объявленных вами переменных, псевдонимов команд и функций присутствуют в вашем пользовательском окружении.

5. Можете ли вы сходу определить последовательность исполнения обнаруженных стартовых сценариев?

6. Если работа приложения зависит от значения объявленной в файле `$HOME/.profile` переменной, имеет ли значение существование файла `$HOME/.bash_profile` или нет?

28.10. Корректная процедура выполнения практического задания: профили пользователей

1. Выведите список всех файлов профиля пользователя в вашей системе.

```
ls -a ~ ; ls -l /etc/pro* /etc/bash*
```

2. Прочитайте содержимое каждого из этих файлов, обычно в них осуществляется подключение дополнительных сценариев.

3. Объявите уникальную переменную, псевдоним команды и функцию в каждом из этих файлов.

4. Воспользуйтесь различными способами получения командной оболочки (su, su -, ssh, tmux, gnome-terminal, Ctrl-Alt-F1, ...) и проверьте, какие из объявленных вами переменных, псевдонимов команд и функций присутствуют в вашем пользовательском окружении.

5. Можете ли вы сходу определить последовательность исполнения обнаруженных стартовых сценариев?

```
Нет, так как при совпадении имен псевдонимов команд,
функций и переменных производится их перезапись.
```

6. Если работа приложения зависит от значения объявленной в файле `$HOME/.profile` переменной, имеет ли значение существование файла `$HOME/.bash_profile` или нет?

```
Да, имеет. (Обратитесь к странице руководства man bash
/INVOCATION)
```

Глава 29. Группы пользователей

Учетные записи пользователей системы могут объединяться в рамках **групп**. Концепция групп пользователей позволяет устанавливать права доступа на уровне групп пользователей вместо установки аналогичных прав доступа для каждого отдельного пользователя.

В каждом дистрибутиве Unix или Linux имеется инструмент с графическим интерфейсом для управления группами пользователей. Пользователям, не имеющим опыта работы с данными системами, рекомендуется использовать именно эти инструменты. Более опытные пользователи могут использовать инструменты с интерфейсом ко-

мандной строки для управления учетными записями пользователей, проявляя при этом осторожность: некоторые дистрибутивы не позволяют работать одновременно с инструментами для управления группами пользователей с графическим интерфейсом и интерфейсом командной строки (примером может служить инструмент YaST из состава дистрибутива Novell Suse). Опытные системные администраторы могут осуществлять непосредственное редактирование соответствующих файлов с помощью текстового редактора **vi** или утилиты **vigr**.

29.1. Утилита groupadd

Группы пользователей могут создаваться с помощью утилиты **groupadd**. В примере ниже показана методика создания пяти групп (без добавления в них пользователей).

```
root@laika:~# groupadd tennis
root@laika:~# groupadd football
root@laika:~# groupadd snooker
root@laika:~# groupadd formula1
root@laika:~# groupadd salsa
```

29.2. Файл group

Пользователи могут состоять в нескольких группах. Членство пользователей в группах описывается в файле **/etc/group**.

```
root@laika:~# tail -5 /etc/group
tennis:x:1006:
football:x:1007:
snooker:x:1008:
formula1:x:1009:
salsa:x:1010:
root@laika:~#
```

Первым полем в строке с описанием группы пользователей является имя группы. Во втором поле размещается (зашифрованный) пароль группы (это поле может быть пустым). В третьем поле размещается идентификатор группы или значение **GID**. Четвертое поле является списком членов группы, который в данном случае является пустым, так как в группах нет пользователей.

29.3. Команда groups

Пользователь может выполнить команду **groups** для ознакомления со списком групп, в которых он состоит.

```
[harry@RHEL4b ~]$ groups
harry sports
[harry@RHEL4b ~]$
```

29.4. Утилита usermod

Членство пользователя в группах может быть изменено с помощью утилиты useradd или **usermod**.

```
root@laika:~# usermod -a -G tennis inge
root@laika:~# usermod -a -G tennis katrien
root@laika:~# usermod -a -G salsa katrien
root@laika:~# usermod -a -G snooker sandra
root@laika:~# usermod -a -G formula1 annelies
root@laika:~# tail -5 /etc/group
tennis:x:1006:inge,katrien
football:x:1007:
snooker:x:1008:sandra
formula1:x:1009:annelies
salsa:x:1010:katrien
root@laika:~#
```

Проявляйте осторожность при использовании утилиты **usermod** для добавления пользователей в группы. По умолчанию утилита **usermod** будет удалять пользователя из всех групп, в которых он состоял, если имена данных групп не были переданы в составе команды! Использование параметра **-a** (append - дополнение) позволяет избежать данного поведения.

29.5. Утилита groupmod

Вы можете изменить имя группы пользователей с помощью утилиты **groupmod**.

```
root@laika:~# groupmod -n darts snooker
root@laika:~# tail -5 /etc/group
tennis:x:1006:inge,katrien
football:x:1007:
```

```
formula1:x:1009:annelies
salsa:x:1010:katrien
darts:x:1008:sandra
```

29.6. Утилита groupdel

Вы можете навсегда удалить группу пользователей с помощью утилиты **groupdel**.

```
root@laika:~# groupdel tennis
root@laika:~#
```

29.7. Утилита gpasswd

Также вы можете делегировать функции контроля над членством в определенной группе пользователей другому пользователю с помощью утилиты **gpasswd**. В примере ниже мы делегируем права на добавление пользователей в группу sports и удаление их из нее пользователю serena. После этого мы используем команду **su** для добавления пользователя harry в группу sports от лица пользователя serena.

```
[root@RHEL4b ~]# gpasswd -A serena sports
[root@RHEL4b ~]# su - serena
[serena@RHEL4b ~]$ id harry
uid=516(harry) gid=520(harry) группы=520(harry)
[serena@RHEL4b ~]$ gpasswd -a harry sports
Добавление пользователя harry в группу sports
[serena@RHEL4b ~]$ id harry
uid=516(harry) gid=520(harry)
группы=520(harry),522(sports)
[serena@RHEL4b ~]$ tail -1 /etc/group
sports:x:522:serena,venus,harry
[serena@RHEL4b ~]$
```

Администраторы групп пользователей не обязаны быть членами этих групп. Они могут удалить свои учетные записи из администрируемых ими групп пользователей и это никак не повлияет на их возможности добавления пользователей в эти группы или удаления пользователей из них.

```
[serena@RHEL4b ~]$ gpasswd -d serena sports
Удаление пользователя serena из группы sports
```

```
[serena@RHEL4b ~]$ exit
```

Информация об администраторах групп пользователей хранится в файле **/etc/gshadow**.

```
[root@RHEL4b ~]# tail -1 /etc/gshadow
sports:!:serena:venus,harry
[root@RHEL4b ~]#
```

Для удаления всех учетных записей администраторов из группы пользователей следует использовать утилиту **gpasswd** с параметрами для задания пустого списка администраторов.

```
[root@RHEL4b ~]# gpasswd -A "" sports
```

29.8. Утилита newgrp

Вы можете запустить **дочернюю командную оболочку** с новой временной **основной группой пользователя**, воспользовавшись командой **newgrp**.

```
root@rhel65:~# mkdir prigroup
root@rhel65:~# cd prigroup/
root@rhel65:~/prigroup# touch standard.txt
root@rhel65:~/prigroup# ls -l
итого 0
-rw-r--r--. 1 root root 0 апр 13 17:49 standard.txt
root@rhel65:~/prigroup# echo $SHLVL
1
root@rhel65:~/prigroup# newgrp tennis
root@rhel65:~/prigroup# echo $SHLVL
2
root@rhel65:~/prigroup# touch newgrp.txt
root@rhel65:~/prigroup# ls -l
итого 0
-rw-r--r--. 1 root tennis 0 апр 13 17:49 newgrp.txt
-rw-r--r--. 1 root root   0 апр 13 17:49 standard.txt
root@rhel65:~/prigroup# exit
exit
root@rhel65:~/prigroup#
```

29.9.Утилита vigr

По аналогии с утилитой vipw, утилита **vigr** может использоваться для редактирования файла **/etc/group** в ручном режиме, так как она осуществляет корректную блокировку этого файла в процессе редактирования. Текстовый редактор **vi** или утилита **vigr** может использоваться для управления группами пользователей исключительно опытными системными администраторами.

29.10. Практическое задание: группы пользователей

1. Создайте группы пользователей tennis, football и sports.

2. С помощью одной команды сделайте пользователя venus членом групп tennis и sports.

3. Переименуйте группу пользователей fotball в foot.

4. Используйте текстовый редактор vi для добавления пользователя serena в группу пользователей tennis.

5. Используйте команду id для того, чтобы убедиться, что пользователь serena состоит в группе пользователей tennis.

6. Сделайте кого-либо из пользователей ответственным за управление членством пользователей в группах foot и sports. Проверьте работоспособность использованного механизма.

29.11. Корректная процедура выполнения практического задания: группы пользователей

1. Создайте группы пользователей tennis, football и sports.

```
groupadd tennis ; groupadd football ; groupadd sports
```

2. С помощью одной команды сделайте пользователя venus членом групп tennis и sports.

```
usermod -a -G tennis,sports venus
```

3. Переименуйте группу пользователей fotball в foot.

```
groupmod -n foot football
```

4. Используйте текстовый редактор vi для добавления пользователя serena в группу пользователей tennis.

vi /etc/group

5. Используйте команду id для того, чтобы убедиться, что пользователь serena состоит в группе пользователей tennis.

id (после выхода из системы и входа в нее пользователь serena должен быть членом группы)

6. Сделайте кого-либо из пользователей ответственным за управление членством пользователей в группах foot и sports. Проверьте работоспособность использованного механизма.

gpasswd -A (для того, чтобы сделать пользователя ответственным за управление членством в группе пользователей)

gpasswd -a (для того, чтобы сделать пользователя членом группы пользователей)

Часть VIII. Механизмы безопасной работы с файлами

Глава 30. Стандартные права доступа к файлам

В данной главе содержится подробная информация о системе разграничения доступа к файлам на основе механизма **владения файлами** и **прав доступа к файлам**.

30.1. Механизм владения файлами

30.1.1. Пользователь и группа, владеющие файлом

Пользователи и **группы пользователей** в рамках системы могут быть описаны в локальных файлах **/etc/passwd** и **/etc/group** или объявлены на уровне сервера NIS, LDAP или домена Samba. Эти пользователи и группы пользователей могут владеть файлами. На самом деле каждым файлом владеет как **пользователь**, так и **группа пользователей**, что можно увидеть в следующем примере.

```
paul@rhel65:~/owners$ ls -lh
итого 636K
-rw-r--r--. 1 paul snooker 1.1K апр  8 18:47 data.odt
-rw-r--r--. 1 paul paul    626K апр  8 18:46 file1
-rw-r--r--. 1 paul tennis   185 апр  8 18:46 file2
-rw-rw-r--. 1 root root       0 апр  8 18:47 stuff.txt
paul@rhel65:~/owners$
```

Пользователь `paul` владеет тремя файлами; файлом `file1` **владеет пользователь** `paul` и **группа пользователей** `paul`, файлом `data.odt` владеет **группа пользователей** `snooker`, а файлом `file2` - группа пользователей `tennis`.

Последний файл носит имя `stuff.txt` и его владельцами является пользователь `root` и группа пользователей `root`.

30.1.2. Вывод списка учетных записей пользователей

Вы можете воспользоваться следующей командой для вывода списка всех локальных учетных записей пользователей.

```
paul@debian7~$ cut -d: -f1 /etc/passwd | column
root            ntp             sam             bert
naomi
```

daemon	mysql	tom	rino
matthias2			
bin	paul	wouter	antonio
bram			
sys	maarten	robrecht	simon
fabrice			
sync	kevin	bilal	sven
chimene			
games	yuri	dimitri	wouter2
messagebus			
man	william	ahmed	tarik
roger			
lp	yves	dylan	jan
frank			
mail	kris	robin	ian
toon			
news	hamid	matthias	ivan
rinus			
uucp	vladimir	ben	azeddine
eddy			
proxy	abiy	mike	eric
bram2			
www-data	david	kevin2	kamel
keith			
backup	chahid	kenzo	ischa
jesse			
list	stef	aaron	bart
frederick			
irc	joeri	lorenzo	omer
hans			
gnats	glenn	jens	kurt
dries			
nobody	yannick	ruben	steve
steve2			
libuuid	christof	jelle	
constantin	tomas		
Debian-exim	george	stefaan	sam2
johan			

```
statd           joost           marc            bjorn
tom2
sshd            arno            thomas          ronald
```

chgrp

30.1.3. Утилита chgrp

Вы можете изменить имя группы пользователей, владеющей файлом, с помощью утилиты **chgrp**.

```
root@rhel65:/home/paul/owners# ls -l file2
-rw-r--r--. 1 root tennis 185 апр  8 18:46 file2
root@rhel65:/home/paul/owners# chgrp snooker file2
root@rhel65:/home/paul/owners# ls -l file2
-rw-r--r--. 1 root snooker 185 апр  8 18:46 file2
root@rhel65:/home/paul/owners#
```

30.1.4. Утилита chown

Имя пользователя, владеющего файлом, может быть изменено с помощью утилиты **chown**.

```
root@laika:/home/paul# ls -l FileForPaul
-rw-r--r-- 1 root paul 0 2008-08-06 14:11 FileForPaul
root@laika:/home/paul# chown paul FileForPaul
root@laika:/home/paul# ls -l FileForPaul
-rw-r--r-- 1 paul paul 0 2008-08-06 14:11 FileForPaul
```

Вы также можете использовать утилиту **chown** для одновременного изменения имен пользователя и группы пользователей, владеющих файлом.

```
root@laika:/home/paul# ls -l FileForPaul
-rw-r--r-- 1 paul paul 0 2008-08-06 14:11 FileForPaul
root@laika:/home/paul# chown root:project42 FileForPaul
root@laika:/home/paul# ls -l FileForPaul
-rw-r--r-- 1 root project42 0 2008-08-06 14:11
FileForPaul
```

30.1.5. Список специальных типов файлов

При использовании команды **ls -l** в строке с информацией о каждом из файлов вы можете обнаружить десять символов перед именами владеющих файлом пользователя и группы пользователей. Первый символ сообщает нам о типе файла. Для обычных файлов ис-

пользуется символ **-**, для директорий - символ **d**, символьные ссылки обозначаются с помощью символа **l**, именованные каналы - с помощью символа **p**, символьные устройства - с помощью символа **c**, блочные устройства - с помощью символа **b**, а сокеты - с помощью символа **s**.

Таблица 30.1 - Специальные типы файлов Unix

Первый символ	Тип файла
-	Обычный файл
d	Директория
l	Символьная ссылка
p	Именованный канал
b	Блочное устройство
c	Символьное устройство
s	Сокет

В примере ниже выводится информация о файле символьного устройства (консоли) и блочного устройства (жесткого диска).

```
paul@debian6lt~$ ls -ld /dev/console /dev/sda
crw-------   1 root root  5, 1 мар 15 12:45 /dev/console
brw-rw----   1 root disk  8, 0 мар 15 12:45 /dev/sda
```

А в следующем примере вы можете увидеть информацию о директории, обычном файле и символьной ссылке.

```
paul@debian6lt~$ ls -ld /etc /etc/hosts /etc/motd
drwxr-xr-x 128 root root 12288 мар 15 18:34 /etc
-rw-r--r--   1 root root   372 дек 10 17:36 /etc/hosts
lrwxrwxrwx   1 root root    13 дек  5 10:36 /etc/motd
-> /var/run/motd
```

30.2. Права доступа

30.2.1. Символы rwx

Девять символов, следующих после символа типа файла, отображают права доступа к файлу, разделенные на три триплета. Символом права доступа может быть символ **r**, обозначающий возможность чтения данных из файла, символ **w**, обозначающий возможность записи данных в файл, и символ **x**, обозначающий возможность исполнения файла. Вам понадобится право на чтение директории (обозначаемое с помощью символа **r**) для вывода списка содержимого этой директории (например, с помощью команды `ls`). Для входа в ди-

ректорию (например, с помощью команды `cd`) вам понадобится право на исполнение (обозначаемое с помощью символа **`x`**). А для создания новых файлов в директории и удаления существующих файлов из нее вам понадобится право на запись в эту директорию (обозначаемое с помощью символа **`w`**).

Таблица 30.2. Стандартные права доступа к файлам Unix

Право доступа	К файлу	К директории
`r` (чтение)	Чтение содержимого файла (`cat`)	Чтение содержимого директории (`ls`)
`w` (запись)	Изменение содержимого файла (`vi`)	Создание файлов в директории (`touch`)
`x` (исполнение)	Исполнение файла	Вход в директорию (`cd`)

30.2.2. Три набора символов `rwx`

Мы уже знаем о том, что вывод команды **`ls -l`** начинается с десяти символов для каждого из файлов. В примере ниже показан вывод в случае нахождения в директории обычного файла (так как первым символом является символ `-`).

```
paul@RHELv4u4:~/test$ ls -l proc42.bash
-rwxr-xr--  1 paul proj 984 фев  6 12:01 proc42.bash
```

В таблице ниже описано назначение всех десяти символов.

Таблица 30.3. Права доступа к файлам в Unix

Позиция	Символы	Назначение
1	`-`	Указание на то, что это обычный файл.
2-4	`rwx`	Права доступа для пользователя, владеющего файлом.
5-7	`r-x`	Права доступа для пользователей из группы, владеющей файлом.
8-10	`r--`	Права доступа для остальных пользователей.

В том случае, если вы являетесь **владельцем файла**, ваш доступ к содержимому этого файла будет регламентироваться **правами доступа для пользователя, владеющего файлом**. Остальные права доступа не будут оказывать ровным счетом никакого влияния на вашу возможность доступа к содержимому этого файла.

В том случае, если вы состоите в **группе пользователей**, владеющей данным файлом, ваш доступ к содержимому этого файла бу-

дет регламентироваться **правами доступа для пользователей из группы, владеющей файлом**. По аналогии, остальные права доступа не будут оказывать никакого влияния на вашу возможность доступа к содержимому этого файла.

В том же случае, если вы не являетесь **владельцем этого файла**, а также не состоите в **группе пользователей, владеющей этим файлом**, ваш доступ к содержимому файла будет регламентироваться правами доступа для остальных пользователей. Как и раньше, остальные права доступа не будут оказывать никакого влияния на вашу возможность доступа к содержимому этого файла.

30.2.3. Примеры прав доступа к файлам

В примере ниже показаны некоторые комбинации прав доступа к фалам и директориям. Имена файлов и директорий описывают права доступа к ним.

```
paul@laika:~/perms$ ls -lh
итого 12K
drwxr-xr-x 2 paul paul 4.0K 2007-02-07 22:26 Все_пользо-
ватели_могут_осуществлять_вход_Владелец_-
_создавать_и_удалять_файлы
-rwxrwxrwx 1 paul paul    0 2007-02-07 22:21 Все_пользо-
ватели_имеют_полный_контроль_над_файлом.txt
-r--r----- 1 paul paul    0 2007-02-07 22:21 Только_вла-
дельцы_могут_осуществлять_чтение.txt
-rwxrwx--- 1 paul paul    0 2007-02-07 22:21
Владельцы_могут_делать_все_Остальные_-_ничего.txt
dr-xr-x--- 2 paul paul 4.0K 2007-02-07 22:25
Владелец_и_участники_группы_могут_осуществлять_вход
dr-x------ 2 paul paul 4.0K 2007-02-07 22:25 Исключитель-
но_владелец_может_осуществлять_вход
paul@laika:~/perms$
```

Подводя итог, можно сказать, что первый триплет `rwx` представляет права доступа **владельца файла**. Второй триплет соответствует правам доступа **пользователей из группы, владеющей файлом**; он указывает права доступа каждого пользователя из этой группы. Третий триплет описывает права доступа **всех остальных пользователей**, не являющихся владельцами файла или директории и не состоящих в группе пользователей, владеющей файлом или директорией.

30.2.4. Установка прав доступа (chmod)

Права доступа могут быть изменены с помощью утилиты **chmod**. В первом примере владельцу файла permissions.txt дается право на его исполнение.

```
paul@laika:~/perms$ ls -l permissions.txt
-rw-r--r-- 1 paul paul 0 2007-02-07 22:34 permissions.txt
paul@laika:~/perms$ chmod u+x permissions.txt
paul@laika:~/perms$ ls -l permissions.txt
-rwxr--r-- 1 paul paul 0 2007-02-07 22:34 permissions.txt
```

В следующем примере у группы пользователей, владеющей файлом, изымается право на чтение этого файла.

```
paul@laika:~/perms$ chmod g-r permissions.txt
paul@laika:~/perms$ ls -l permissions.txt
-rwx---r-- 1 paul paul 0 2007-02-07 22:34 permissions.txt
```

А в следующем примере право на чтение файла изымается у всех пользователей, не являющихся владельцами этого файла и не входящих в группу, владеющую этим файлом.

```
paul@laika:~/perms$ chmod o-r permissions.txt
paul@laika:~/perms$ ls -l permissions.txt
-rwx------ 1 paul paul 0 2007-02-07 22:34 permissions.txt
```

В примере ниже всем пользователям дается право модификации содержимого файла.

```
paul@laika:~/perms$ chmod a+w permissions.txt
paul@laika:~/perms$ ls -l permissions.txt
-rwx-w--w- 1 paul paul 0 2007-02-07 22:34 permissions.txt
```

В подобных случаях от вас даже не требуется использовать символ a.

```
paul@laika:~/perms$ chmod +x permissions.txt
paul@laika:~/perms$ ls -l permissions.txt
-rwx-wx-wx 1 paul paul 0 2007-02-07 22:34 permissions.txt
```

Кроме того, вы можете задавать права доступа явным образом.

```
paul@laika:~/perms$ chmod u=rw permissions.txt
paul@laika:~/perms$ ls -l permissions.txt
-rw--wx-wx 1 paul paul 0 2007-02-07 22:34 permissions.txt
```

Не стесняйтесь использовать любые комбинации прав доступа.

```
paul@laika:~/perms$ chmod u=rw,g=rw,o=r permissions.txt
paul@laika:~/perms$ ls -l permissions.txt
-rw-rw-r-- 1 paul paul 0 2007-02-07 22:34 permissions.txt
```

Утилита chmod будет принимать даже сомнительные комбинации прав доступа.

```
paul@laika:~/perms$ chmod u=rwx,ug+rw,o=r permissions.txt
paul@laika:~/perms$ ls -l permissions.txt
-rwxrw-r-- 1 paul paul 0 2007-02-07 22:34 permissions.txt
```

30.2.4. Установка прав доступа в восьмеричном представлении

Большинство администраторов систем Unix наверняка предпочтет использовать **классическое** восьмеричное представление при просмотре и установке прав доступа. Рассмотрим триплет на битовом уровне, причем символу `r` будет соответствовать значение 4, символу `w` - 2, а символу `x` - 1.

Таблица 30.4. Права доступа в восьмеричном представлении

Двоичное представление	Восьмеричное представление	Права доступа
000	0	---
001	1	--x
010	2	-w-
011	3	-wx
100	4	r--
101	5	r-x
110	6	rw-
111	7	rwx

Исходя из вышеописанного, значение 777 является эквивалентным правам доступа `rwxrwxrwx` и, по той же логике, значение 654 соответствует правам доступа `rw-r-xr--`. Утилита **chmod** будет принимать и эти числовые значения.

```
paul@laika:~/perms$ chmod 777 permissions.txt
paul@laika:~/perms$ ls -l permissions.txt
-rwxrwxrwx 1 paul paul 0 2007-02-07 22:34 permissions.txt
paul@laika:~/perms$ chmod 664 permissions.txt
paul@laika:~/perms$ ls -l permissions.txt
```

```
-rw-rw-r-- 1 paul paul 0 2007-02-07 22:34 permissions.txt
paul@laika:~/perms$ chmod 750 permissions.txt
paul@laika:~/perms$ ls -l permissions.txt
-rwxr-x--- 1 paul paul 0 2007-02-07 22:34 permissions.txt
```

30.2.5. Значение umask

При создании файла или директории используется набор стандартных прав доступа. Эти стандартные права доступа устанавливаются на основе значения **umask**. Значение **umask** позволяет задать права доступа, которые вы не хотите устанавливать по умолчанию. Вы можете вывести информацию о наборе стандартных прав доступа, выполнив команду **umask**.

```
[Harry@RHEL4b ~]$ umask
0002
[Harry@RHEL4b ~]$ touch test
[Harry@RHEL4b ~]$ ls -l test
-rw-rw-r--  1 Harry Harry 0 июл 24 06:03 test
[Harry@RHEL4b ~]$
```

Как вы можете увидеть, по умолчанию файл также не делается исполняемым. Это стандартная функция, предназначенная для повышения безопасности систем Unix; создаваемые файлы никогда не делаются исполняемыми по умолчанию. Вам придется самостоятельно выполнить команду **chmod +x** для того, чтобы сделать файл исполняемым. Это также означает, что 1 бит значения **umask** не оказывает какого-либо влияния на права доступа к создаваемым файлам - значение **umask** 0022 эквивалентно значению **umask** 0033.

30.2.6. Команда mkdir -m

При создании директорий с помощью команды **mkdir** вы можете использовать параметр **-m** для задания **прав доступа**. Методика использования данного параметра освещена в примере ниже.

```
paul@debian5~$ mkdir -m 700 MyDir
paul@debian5~$ mkdir -m 777 Public
paul@debian5~$ ls -dl MyDir/ Public/
drwx------ 2 paul paul 4096 2011-10-16 19:16 MyDir/
drwxrwxrwx 2 paul paul 4096 2011-10-16 19:16 Public/
```

30.2.7. Команда `cp -p`

Для сохранения прав доступа и меток времени файлов при их копировании следует использовать команду **`cp -p`**.

```
paul@laika:~/perms$ cp file* cp
paul@laika:~/perms$ cp -p file* cpp
paul@laika:~/perms$ ll *
-rwx------ 1 paul paul    0 2008-08-25 13:26 file33
-rwxr-x--- 1 paul paul    0 2008-08-25 13:26 file42

cp:
total 0
-rwx------ 1 paul paul 0 2008-08-25 13:34 file33
-rwxr-x--- 1 paul paul 0 2008-08-25 13:34 file42

cpp:
total 0
-rwx------ 1 paul paul 0 2008-08-25 13:26 file33
-rwxr-x--- 1 paul paul 0 2008-08-25 13:26 file42
```

30.3. Практическое задание: стандартные права доступа к файлам

1. Используя учетную запись обычного пользователя, создайте директорию `~/permissions`. После этого создайте в этой директории файл, владельцем которого будете вы.

2. Скопируйте файл, владельцем которого является пользователь `root`, из директории `/etc/` в вашу директорию `permissions`; кто будет владельцем созданной копии файла?

3. Используя учетную запись пользователя `root`, создайте файл в пользовательской директории `~/permissions`.

4. Используя учетную запись обычного пользователя, получите информацию о владельце данного файла, созданного пользователем `root`.

5. Измените владельца всех файлов из директории `~/permissions` таким образом, чтобы владельцем данных файлов стали вы.

6. Убедитесь в том, что вы имеете все права для работы с этими файлами, а другие пользователи могут исключительно читать их содержимое.

7. Эквивалентно ли значение `770` правам доступа `rwxrwx---` при работе с утилитой `chmod`?

8. Эквивалентно ли значение `664` правам доступа `r-xr-xr--` при работе с утилитой `chmod`?

9. Эквивалентно ли значение `400` правам доступа `r--------` при работе с утилитой `chmod`?

10. Эквивалентно ли значение `734` правам доступа `rwxr-xr--` при работе с утилитой `chmod`?

11a. Выведите значение `umask` в восьмеричной и символьной форме.

11b. Установите значение `umask 077`, использовав символьный формат. Проверьте работоспособность использованной команды.

12. Используя учетную запись пользователя `root`, создайте файл и дайте право на его чтение другим пользователям. Сможет ли обычный пользователь прочитать содержимое этого файла? Попробуйте записать данные в этот файл с помощью текстового редактора `vi`.

13a. Используя учетную запись обычного пользователя, создайте файл и дайте право на его чтение другим пользователям. Сможет ли другой обычный пользователь прочитать содержимое этого файла? Попробуйте записать данные в этот файл с помощью текстового редактора `vi`.

13b. Сможет ли пользователь `root` прочитать содержимое этого файла? Сможет ли пользователь `root` осуществить запись данных в этот файл с помощью текстового редактора `vi`?

14. Создайте директорию, которая будет принадлежать группе пользователей, причем каждый пользователь из этой группы должен иметь возможность читать данные из файлов, записывать данные в

файлы и создавать новые файлы. Сделайте так, чтобы пользователи могли удалять только собственноручно созданные файлы.

30.4. Корректная процедура выполнения практического задания: стандартные права доступа к файлам

1. Используя учетную запись обычного пользователя, создайте директорию `~/permissions`. После этого создайте в этой директории файл, владельцем которого будете вы.

```
mkdir ~/permissions ; touch ~/permissions/myfile.txt
```

2. Скопируйте файл, владельцем которого является пользователь root, из директории `/etc/` в вашу директорию `permissions`; кто будет владельцем созданной копии файла?

```
cp /etc/hosts ~/permissions/
```

Владельцем копии будете являться вы.

3. Используя учетную запись пользователя `root`, создайте файл в пользовательской директории `~/permissions`.

```
(станьте пользователем root)# touch
/home/username/permissions/rootfile
```

4. Используя учетную запись обычного пользователя, получите информацию о владельце данного файла, созданного пользователем `root`.

```
ls -l ~/permissions
```

Владельцем файла, созданного пользователем `root`, будет также пользователь `root`.

5. Измените владельца всех файлов из директории `~/permissions` таким образом, чтобы владельцем данных файлов стали вы.

```
chown user ~/permissions/*
```

Вы не сможете стать владельцем файлов, принадлежащих пользователю `root`.

6. Убедитесь в том, что вы имеете все права для работы с этими файлами, а другие пользователи могут исключительно читать их содержимое.

```
chmod 644 (для файлов)
chmod 755 (для директорий)
```

7. Эквивалентно ли значение 770 правам доступа rwxrwx--- при работе с утилитой chmod?

Да.

8. Эквивалентно ли значение 664 правам доступа r-xr-xr-- при работе с утилитой chmod?

Нет.

9. Эквивалентно ли значение 400 правам доступа r-------- при работе с утилитой chmod?

Да.

10. Эквивалентно ли значение 734 правам доступа rwxr-xr-- при работе с утилитой chmod?

Нет.

11a. Выведите значение umask в восьмеричной и символьной форме.

```
umask ; umask -S
```

11b. Установите значение umask 077, использовав символьный формат. Проверьте работоспособность использованной команды.

```
umask -S u=rwx,go=
```

12. Используя учетную запись пользователя root, создайте файл и дайте право на его чтение другим пользователям. Сможет ли обычный пользователь прочитать содержимое этого файла? Попробуйте записать данные в этот файл с помощью текстового редактора vi.

```
(станьте пользователем root)
# echo hello > /home/username/root.txt
# chmod 744 /home/username/root.txt
(станьте обычным пользователем)
vi ~/root.txt
```

13a. Используя учетную запись обычного пользователя, создайте файл и дайте право на его чтение другим пользователям. Сможет ли другой обычный пользователь прочитать содержимое этого файла? Попробуйте записать данные в этот файл с помощью текстового редактора `vi`.

```
echo hello > file ; chmod 744 file
```

Да, другие пользователи могут осуществлять чтение этого файла.

13b. Сможет ли пользователь `root` прочитать содержимое этого файла? Сможет ли пользователь `root` осуществить запись данных в этот файл с помощью текстового редактора `vi`?

Да, пользователь `root` может осуществлять чтение данных из этого файла и запись данных в этот файл. Права доступа не распространяются на пользователя `root`.

14. Создайте директорию, которая будет принадлежать группе пользователей, причем каждый пользователь из этой группы должен иметь возможность читать данные из файлов, записывать данные в файлы и создавать новые файлы. Сделайте так, чтобы пользователи могли удалять только собственноручно созданные файлы.

```
mkdir /home/project42 ; groupadd project42
chgrp project42 /home/project42 ; chmod 775
/home/project42
```

На данный момент вы не можете выполнить последнюю часть этого задания...

Глава 31. Расширенные права доступа к файлам

31.1. Бит sticky для директорий

Вы можете установить **бит sticky** для директории с целью предотвращения удаления файлов пользователями, которые не являются их непосредственными владельцами. Бит sticky отображается в той же позиции, что и символ права исполнения `x` для пользователей, не являющихся владельцами директории и не состоящих в группе, владеющей директорией. Сам бит sticky отображается с помощью символа **`t`** (в том случае, если подразумевается наличие символа `x`) или символа **`T`** (в том случае, если символ `x` не должен выводиться ввиду отсутствия соответствующих прав доступа к директории).

```
root@RHELv4u4:~# mkdir /project55
root@RHELv4u4:~# ls -ld /project55
drwxr-xr-x  2 root root 4096 фев  7 17:38 /project55
root@RHELv4u4:~# chmod +t /project55/
root@RHELv4u4:~# ls -ld /project55
drwxr-xr-t  2 root root 4096 фев  7 17:38 /project55
root@RHELv4u4:~#
```

Бит sticky также может быть установлен в случае использования восьмеричного значения прав доступа, причем в этом случае должно использоваться двоичное значение 1 в первом из четырех триплетов.

```
root@RHELv4u4:~# chmod 1775 /project55/
root@RHELv4u4:~# ls -ld /project55
drwxrwxr-t  2 root root 4096 фев  7 17:38 /project55
root@RHELv4u4:~#
```

В своей системе вы, скорее всего, обнаружите **бит sticky**, установленный для директории **/tmp**.

```
root@barry:~# ls -ld /tmp
drwxrwxrwt 6 root root 4096 2009-06-04 19:02 /tmp
```

31.2. Бит setgid для директории

Бит **setguid** может устанавливаться для директорий в тех случаях, когда необходимо, чтобы в качестве группы пользователей, владеющей всеми файлами в директории, использовалась группа пользователей, владеющая директорией. Бит **setgid** отображается в той же позиции, что и символ права исполнения `x` для пользователей из группы, владеющей директорией. Сам бит **setgid** отображается с помощью символа `s` (в том случае, если подразумевается наличие символа `x`) или символа **`S`** (в том случае, если символ `x` не должен отображаться ввиду отсутствия соответствующих прав доступа к директории у пользователей из группы, владеющей директорией). Как показано в данном примере, несмотря на то, что пользователь **`root`** не состоит в группе `proj55`, файлы, созданные пользователем `root` в директории `/project55`, будут принадлежать группе пользователей `proj55` ввиду установки бита **setgid**.

```
root@RHELv4u4:~# groupadd proj55
root@RHELv4u4:~# chown root:proj55 /project55/
```

```
root@RHELv4u4:~# chmod 2775 /project55/
root@RHELv4u4:~# touch /project55/fromroot.txt
root@RHELv4u4:~# ls -ld /project55/
drwxrwsr-x  2 root proj55 4096 фев  7 17:45 /project55/
root@RHELv4u4:~# ls -l /project55/
итого 4
-rw-r--r--  1 root proj55 0 фев  7 17:45 fromroot.txt
root@RHELv4u4:~#
```

Вы можете использовать утилиту **find** для поиска всех директорий с установленным битом **setgid**.

```
paul@laika:~$ find / -type d -perm -2000 2> /dev/null
/var/log/mysql
/var/log/news
/var/local
...
```

31.3. Биты setgid и setuid для обычных файлов

Два упомянутых бита позволяют запускать исполняемый файл с правами пользователя, **владеющего файлом**, а не с правами пользователя, **инициировавшего запуск файла**. Это означает, что в том случае, если какой-либо пользователь запустит программу, принадлежащую **пользователю root**, причем для исполняемого файла программы будет установлен бит **setud**, то программа будет запущена от лица пользователя **root**. Такое поведение системы может оказаться опасным, но иногда оказывается и полезным для обхода ограничений безопасности.

Рассмотрим пример с управлением паролями пользователей; они хранятся в файле **/etc/shadow**, содержимое которого может читаться исключительно пользователем **root**. (В любом случае пользователю **root** не требуется разрешения для чтения файлов).

```
root@RHELv4u4:~# ls -l /etc/shadow
-r--------  1 root root 1260 янв 21 07:49 /etc/shadow
```

Для изменения вашего пароля потребуется модифицировать содержимое данного файла, но каким образом обычные пользователи, не обладающие правами пользователя root, могут сделать это? Давайте рассмотрим права доступа к исполняемому файлу **/usr/bin/passwd**.

```
root@RHELv4u4:~# ls -l /usr/bin/passwd
-r-s--x--x  1 root root 21200 июн 17  2005
/usr/bin/passwd
```

Получается, что при запуске программы **passwd** вы инициируете ее исполнение с правами пользователя **root**.

Вы можете использовать утилиту **find** для поиска всех программ с установленным битом **setuid**.

```
paul@laika:~$ find /usr/bin -type f -perm -04000
/usr/bin/arping
/usr/bin/kgrantpty
/usr/bin/newgrp
/usr/bin/chfn
/usr/bin/sudo
/usr/bin/fping6
/usr/bin/passwd
/usr/bin/gpasswd
...
```

В большинстве случаев установка бита **setuid** для исполняемого файла является достаточной мерой. Установка бита **setgid** приведет к исполнению программ с правами пользователей из группы, владеющей исполняемыми файлами.

31.4. Бит setuid для исполняемого файла sudo

Для бинарного файла **sudo** бит **setuid** устанавливается по умолчанию, поэтому любой пользователь может инициировать его исполнение с эффективным идентификатором пользователя `root`.

```
paul@rhel65:~$ ls -l $(which sudo)
---s--x--x. 1 root root 123832 окт  7  2013 /usr/bin/sudo
paul@rhel65:~$
```

31.5. Практическое задание: биты sticky, setuid и setgid

1a. Создайте директорию, которая будет принадлежать группе пользователей `sports`.

1b. Члены группы пользователей `sports` должны иметь возможность создавать файлы в данной директории.

1c. Все файлы, созданные в данной директории, должны принадлежать группе пользователей `sports`.

1d. Пользователи должны иметь возможность удалять из данной директории только принадлежащие им файлы.

1e. Проверьте корректность создания директории в соответствии с изложенными требованиями.

2. Получите информацию о правах доступа к исполняемому файлу **`/usr/bin/passwd`**. Удалите установленный для данного файла бит **setuid**, после чего попытайтесь изменить свой пароль, работая с учетной записью обычного пользователя. Верните изначально установленные права доступа к исполняемому файлу и повторите попытку.

3. Если осталось время (или вы ожидаете завершения выполнения данного практического задания другими студентами) ознакомьтесь с информацией об атрибутах файлов, изложенной на страницах руководств `chattr` и `lsattr`. Попытайтесь установить атрибут `i` для тестового файла и проверьте корректность его установки.

31.6. Корректная процедура выполнения практического задания: биты sticky, setuid и setgid

1a. Создайте директорию, которая будет принадлежать группе пользователей `sports`.

```
groupadd sports
mkdir /home/sports
chown root:sports /home/sports
```

1b. Члены группы пользователей `sports` должны иметь возможность создавать файлы в данной директории.

```
chmod 770 /home/sports
```

1c. Все файлы, созданные в данной директории, должны принадлежать группе пользователей `sports`.

```
chmod 2770 /home/sports
```

1d. Пользователи должны иметь возможность удалять из данной директории только принадлежащие им файлы.

```
chmod +t /home/sports
```

1e. Проверьте корректность создания директории в соответствии с изложенными требованиями.

Войдите в систему с использованием учетных записей различных пользователей (состоящих и не состоящих в группе `sports`, а также пользователя `root`), создайте файлы и получите информацию о правах доступа к ним. Попытайтесь модифицировать и удалить созданные файлы...

2. Получите информацию о правах доступа к исполняемому файлу **`/usr/bin/passwd`**. Удалите установленный для данного файла бит **setuid**, после чего попытайтесь изменить свой пароль, работая с учетной записью обычного пользователя. Верните изначально установленные права доступа к исполняемому файлу и повторите попытку.

```
root@deb503:~# ls -l /usr/bin/passwd
-rwsr-xr-x 1 root root 31704 2009-11-14 15:41
/usr/bin/passwd
root@deb503:~# chmod 755 /usr/bin/passwd
root@deb503:~# ls -l /usr/bin/passwd
-rwxr-xr-x 1 root root 31704 2009-11-14 15:41
/usr/bin/passwd
```

После этих манипуляций обычный пользователь не сможет изменить свой пароль.

```
root@deb503:~# chmod 4755 /usr/bin/passwd
root@deb503:~# ls -l /usr/bin/passwd
-rwsr-xr-x 1 root root 31704 2009-11-14 15:41
/usr/bin/passwd
```

3. Если осталось время (или вы ожидаете завершения выполнения данного практического задания другими студентами) ознакомьтесь с информацией об атрибутах файлов, изложенной на страницах руководств `chattr` и `lsattr`. Попытайтесь установить атрибут i для тестового файла и проверьте корректность его установки.

```
paul@laika:~$ sudo su -
[sudo] password for paul:
root@laika:~# mkdir attr
```

```
root@laika:~# cd attr/
root@laika:~/attr# touch file42
root@laika:~/attr# lsattr
----------------- ./file42
root@laika:~/attr# chattr +i file42
root@laika:~/attr# lsattr
----i------------ ./file42
root@laika:~/attr# rm -rf file42
rm: невозможно удалить "file42": Операция не позволена
root@laika:~/attr# chattr -i file42
root@laika:~/attr# rm -rf file42
root@laika:~/attr#
```

Глава 32. Списки контроля доступа

Стандартных прав доступа Unix может быть недостаточно в случае развертывании систем в некоторых организациях. В данной главе описываются **списки контроля доступа** (**access control lists** или **acls**), предназначенные для дополнительной защиты файлов и директорий от несанкционированного доступа.

32.1. Параметр acl в файле /etc/fstab

Файловые системы, поддерживающие механизм **списков контроля доступа**, должны монтироваться с использованием параметра **acl** в файле **/etc/fstab**. Как вы можете увидеть в примере ниже, поддержка механизма **списков контроля доступа** явно активирована для корневой файловой системы и явно деактивирована для файловой системы, монтируемой в директорию /home/data.

```
root@laika:~# tail -4 /etc/fstab
/dev/sda1       /             ext3     acl,relatime
0  1
/dev/sdb2       /home/data    auto     noacl,defaults
0  0
pasha:/home/r   /home/pasha   nfs      defaults
0  0
wolf:/srv/data  /home/wolf    nfs      defaults
0  0
```

32.2. Утилита getfacl

Чтение списков контроля доступа может осуществляться с помощью утилиты **/usr/bin/getfacl**. В примере ниже показана методика чтения **списка контроля доступа** к файлу с именем **file33** с помощью утилиты **getfacl**.

```
paul@laika:~/test$ getfacl file33
# file: file33
# owner: paul
# group: paul
user::rw-
group::r--
mask::rwx
other::r--
```

32.3. Утилита setfacl

Запись или модификация **списков контроля доступа** может осуществляться с помощью утилиты **/usr/bin/setfacl**. В примерах ниже показана методика модификации **списка контроля доступа** к файлу с именем **file33** с помощью утилиты **setfacl**.

Сначала мы добавляем пользователя **sandra** (о чем говорит символ **u**) в **список контроля доступа** с правами, заданными с помощью восьмеричного значения **7**.

```
paul@laika:~/test$ setfacl -m u:sandra:7 file33
```

После этого мы добавляем группу пользователей (о чем говорит символ **g**) в **список контроля доступа** к тому же файлу с правами, заданными с помощью восьмеричного значения **6**.

```
paul@laika:~/test$ setfacl -m g:tennis:6 file33
```

Результат данных манипуляций будет виден при использовании утилиты **getfacl**.

```
paul@laika:~/test$ getfacl file33
# file: file33
# owner: paul
# group: paul
user::rw-
user:sandra:rwx
group::r--
```

```
group:tennis:rw-
mask::rwx
other::r--
```

32.4. Удаление элемента списка контроля доступа

Параметр **-x** утилиты **setfacl** позволяет удалить элемент **списка контроля доступа** к заданному файлу.

```
paul@laika:~/test$ setfacl -m u:sandra:7 file33
paul@laika:~/test$ getfacl file33 | grep sandra
user:sandra:rwx
paul@laika:~/test$ setfacl -x sandra file33
paul@laika:~/test$ getfacl file33 | grep sandra
```

Обратите внимание на то, что в случае указания элемента **списка контроля доступа** к файлу без использования символа **u** (указывающего на соответствие учетной записи пользователя) или **g** (указывающего на соответствие группе пользователей), по умолчанию будет рассматриваться элемент списка контроля доступа к файлу, соответствующий учетной записи пользователя.

32.5. Удаление всего списка контроля доступа

Параметр **-b** утилиты **setfacl** позволяет удалить весь список контроля доступа к заданному файлу.

```
paul@laika:~/test$ setfacl -b file33
paul@laika:~/test$ getfacl file33
# file: file33
# owner: paul
# group: paul
user::rw-
group::r--
other::r--
```

32.6. Маска прав списка контроля доступа

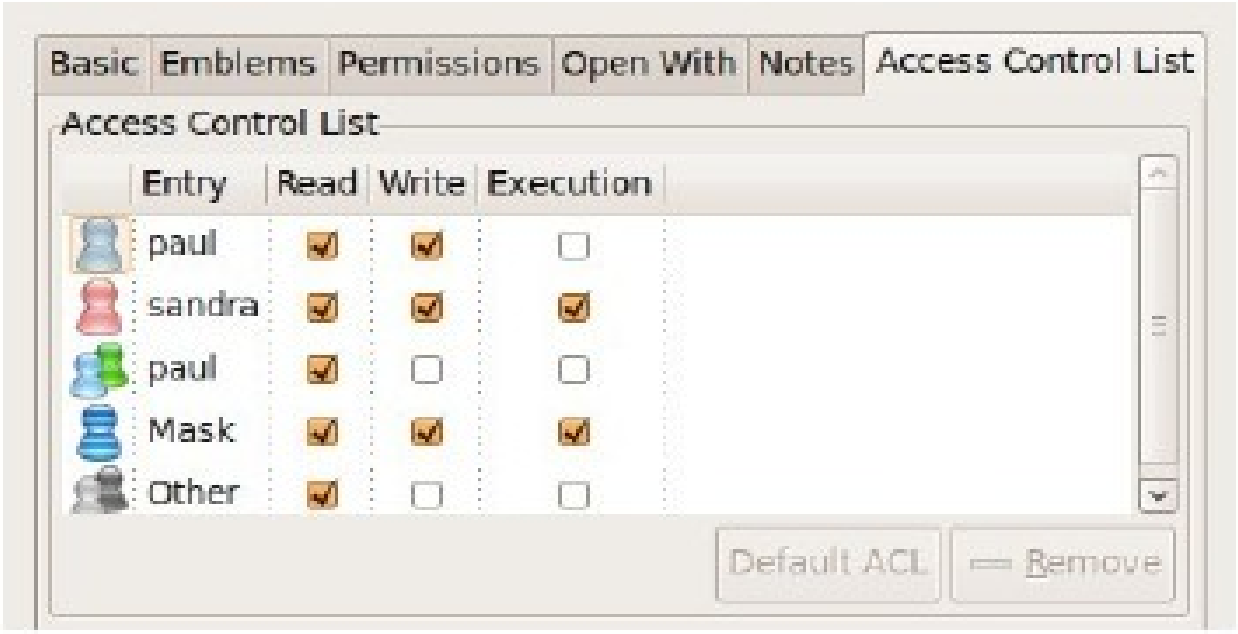
Маска прав списка контроля доступа описывает максимальные эффективные права доступа для любого из элементов этого списка. Данная маска рассчитывается каждый раз, когда вы используете утилиту **setfacl** или **chmod**.

Вы можете предотвратить расчет маски прав списка контроля доступа, воспользовавшись параметром **--no-mask**.

```
paul@laika:~/test$ setfacl --no-mask -m u:sandra:7 file33
paul@laika:~/test$ getfacl file33
# file: file33
# owner: paul
# group: paul
user::rw-
user:sandra:rwx                 #effective:rw-
group::r--
mask::rw-
other::r--
```

32.7. Приложение eiciel

Пользователи настольных систем могут отдать предпочтение приложению **eiciel**, которое является инструментом с графическим интерфейсом для управления **списками контроля доступа**.



Вам придется установить пакеты **eiciel** и **nautilus-actions** для того, чтобы в диалоге изменения свойств файлов и директорий файлового менеджера **nautilus** появилась вкладка для управления **списками контроля доступа**.

```
paul@laika:~$ sudo aptitude install eiciel nautilus-
actions
```

Глава 33. Ссылки на файлы

В файловой системе среднестатистического компьютера, работающего под управлением Linux, используется множество **жестких** и **символьных ссылок** на файлы.

Для понимания концепции ссылок на файлы в рамках файловой системы следует в первую очередь разобраться со структурами **inode**.

33.1. Структуры inode

33.1.1. Содержимое структуры inode

Inode является структурой данных, содержащей относящиеся к файлу метаданные. При сохранении нового файла на диске в рамках файловой системы происходит сохранение не только самого содержимого файла (данных), но и таких дополнительных параметров файла, как имя, дата создания, права доступа и других. Вся эта информация (за исключением имени файла и его содержимого) сохраняется в рамках структуры **inode**, соответствующей данному файлу.

Как видно в следующем примере, при использовании команды **`ls -l`** выводятся некоторые данные, содержащиеся в структуре inode.

```
root@rhel53 ~# ls -ld /home/project42/
drwxr-xr-x 4 root pro42 4.0K мар 27 14:29
/home/project42/
```

33.1.2. Таблица структур inode

Таблица структур inode содержит информацию о всех структурах **inode** и создается в момент создания файловой системы (с помощью утилиты **`mkfs`**). Вы можете использовать команду **`df -i`** для получения информации о количестве используемых и доступных для использования структур **inode** в рамках смонтированных файловых систем.

```
root@rhel53 ~# df -i
Файловая система  Iнодов IИспользовано ICвободно IИсполь-
зовано% Смонтировано в
/dev/mapper/VolGroup00-LogVol00
                 4947968        115326   4832642
3% /
/dev/hda1          26104            45     26059
1% /boot
```

```
tmpfs                64417                1     64416
1% /dev/shm
/dev/sda1           262144             2207    259937
1% /home/project42
/dev/sdb1            74400             5519     68881
8% /home/project33
/dev/sdb5                0                0         0
-  /home/sales
/dev/sdb6           100744               11    100733
1% /home/research
```

В приведенном выше примере вывода команды **df -i** вы можете обнаружить процентные показатели использования структур **inode** для некоторых смонтированных **файловых систем**. Данный показатель не приводится для устройства /**dev/sdb5** ввиду того, что на этом устройстве используется файловая система **fat**.

33.1.3. Идентификатор структуры inode

Каждая структура **inode** имеет уникальный идентификатор (идентификатор структуры inode). Вы можете увидеть идентификаторы структур inode в выводе команды **ls -li**.

```
paul@RHELv4u4:~/test$ touch file1
paul@RHELv4u4:~/test$ touch file2
paul@RHELv4u4:~/test$ touch file3
paul@RHELv4u4:~/test$ ls -li
итого 12
817266 -rw-rw-r--  1 paul paul 0 фев  5 15:38 file1
817267 -rw-rw-r--  1 paul paul 0 фев  5 15:38 file2
817268 -rw-rw-r--  1 paul paul 0 фев  5 15:38 file3
paul@RHELv4u4:~/test$
```

Три этих файла создавались последовательно, причем им соответствуют три различные структуры **inode** (идентификаторы которых выводятся в первом столбце). Вся информация из вывода данной команды **ls**, за исключением имен файлов (которые хранятся в файле директории), содержится в структурах **inode**.

33.1.4. Структуры inode и содержимое файлов

Давайте запишем немного данных в один из рассматриваемых файлов.

```
paul@RHELv4u4:~/test$ ls -li
итого 16
817266 -rw-rw-r--  1 paul paul   0 фев  5 15:38 file1
817270 -rw-rw-r--  1 paul paul 147 фев  5 15:42 file2
817268 -rw-rw-r--  1 paul paul   0 фев  5 15:38 file3
paul@RHELv4u4:~/test$ cat file2
Сейчас зима и очень холодно.
Нам не нравится холод, мы предпочитаем летние ночи.
paul@RHELv4u4:~/test$
```

Данные, выводимые при использовании команды **cat**, содержатся не в структуре **inode**, а где-то на диске. Структура **inode** содержит указатель на эти данные.

33.2. О директориях

33.2.1. Директория является таблицей

Директория является особым видом файла, который содержит таблицу соответствия между именами файлов и структурами inode. При выводе списка содержимого директории с помощью команды **ls -ali** на самом деле будет осуществляться вывод содержимого файла директории.

```
paul@RHELv4u4:~/test$ ls -ali
итого 32
817262 drwxrwxr-x   2 paul paul  4096 фев  5 15:42 .
800768 drwx------  16 paul paul  4096 фев  5 15:42 ..
817266 -rw-rw-r--   1 paul paul     0 фев  5 15:38 file1
817270 -rw-rw-r--   1 paul paul   147 фев  5 15:42 file2
817268 -rw-rw-r--   1 paul paul     0 фев  5 15:38 file3
paul@RHELv4u4:~/test$
```

33.2.2. Директории . и ..

В выводе приведены пять имен файлов, а также информация об их соответствии пяти структурам inode. Имя файла, представленное символом точки (**.**), соответствует текущей директории, а имя файла, представленное двумя символами точки (**..**) - родительской директории. Имена трех других файлов соответствуют различным структурам inode.

33.3. Жесткие ссылки

33.3.1. Создание жестких ссылок

В момент создания **жесткой ссылки** с помощью утилиты **ln** в файл директории добавляется дополнительная запись. Новое имя файла ставится в соответствие существующей структуре inode.

```
paul@RHELv4u4:~/test$ ln file2 hardlink_to_file2
paul@RHELv4u4:~/test$ ls -li
итого 24
817266 -rw-rw-r--  1 paul paul   0 фев  5 15:38 file1
817270 -rw-rw-r--  2 paul paul 147 фев  5 15:42 file2
817268 -rw-rw-r--  1 paul paul   0 фев  5 15:38 file3
817270 -rw-rw-r--  2 paul paul 147 фев  5 15:42
hardlink_to_file2
paul@RHELv4u4:~/test$
```

Оба файла будут использовать одну и ту же структуру inode, поэтому они в любом случае будут иметь одни и те же права доступа, а также одного и того же владельца. Оба этих файла также будут иметь одно и то же содержимое. На самом деле, оба этих файла будут идентичными и это означает, что вы можете безопасно удалить оригинальный файл и это никак не повлияет на существование файла, являющегося жесткой ссылкой. Структура inode содержит счетчик, использующийся для подсчета количества жестких ссылок на нее. В момент, когда значение счетчика уменьшается до нуля, содержимое структуры inode очищается.

33.3.2. Поиск жестких ссылок

Вы можете использовать утилиту **find** для поиска файлов с определенными идентификаторами структур inode. В примере ниже показана методика поиска всех имен файлов, которые соответствуют структуре **inode** с идентификатором 817270. Помните о том, что идентификатор структуры **inode** является уникальным для используемого раздела диска.

```
paul@RHELv4u4:~/test$ find / -inum 817270 2> /dev/null
/home/paul/test/file2
/home/paul/test/hardlink_to_file2
```

33.4. Символьные ссылки

Символьные ссылки (иногда называемые **мягкими ссылками**) не указывают на структуры inode, а являются соответствиями между именами файлов. Символьные ссылки создаются с помощью команды **`ln -s`**. Как вы можете увидеть в примере ниже, **символьная ссылка** имеет собственную структуру inode.

```
paul@RHELv4u4:~/test$ ln -s file2 symlink_to_file2
paul@RHELv4u4:~/test$ ls -li
итого 32
817273 -rw-rw-r--  1 paul paul  13 фев  5 17:06 file1
817270 -rw-rw-r--  2 paul paul 147 фев  5 17:04 file2
817268 -rw-rw-r--  1 paul paul   0 фев  5 15:38 file3
817270 -rw-rw-r--  2 paul paul 147 фев  5 17:04
hardlink_to_file2
817267 lrwxrwxrwx  1 paul paul   5 фев  5 16:55
symlink_to_file2 -> file2
paul@RHELv4u4:~/test$
```

Права доступа к символьной ссылке не имеют значения, так как в итоге будут учитываться права доступа к целевому файлу. Жесткие ссылки ограничены своими разделами дисков (ввиду того, что они указывают на структуры inode), в то время, как символьные ссылки могут указывать на какие угодно файлы (из других файловых систем, даже сетевых).

33.5. Удаление ссылок

Ссылки на файлы могут удаляться с помощью утилиты **`rm`**.

```
paul@laika:~$ touch data.txt
paul@laika:~$ ln -s data.txt sl_data.txt
paul@laika:~$ ln data.txt hl_data.txt
paul@laika:~$ rm sl_data.txt
paul@laika:~$ rm hl_data.txt
```

33.6. Практическое задание: ссылки на файлы

1. Создайте два файла с именами `winter.txt` и `summer.txt`, поместите какие-либо текстовые данные в них.

2. Создайте жесткую ссылку на файл `winter.txt` с именем **`hlwinter.txt`**.

3. Выведите информацию об идентификаторах структур inode, соответствующих этим трем файлам, жесткие ссылки должны использовать структуры inode файлов, на которые они ссылаются.

4. Используйте утилиту `find` для вывода информации о файлах, на которые установлены жесткие ссылки.

5. Все данные файла, помимо двух типов данных, хранятся в соответствующей структуре inode. Назовите эти два типа данных!

6. Создайте символьную ссылку на файл `summer.txt` с именем `slsummer.txt`.

7. Найдите все файлы с идентификатором структуры inode, равным 2. Какой вывод вы можете сделать на основе полученной информации?

8. Исследуйте директории `/etc/init.d/ /etc/rc.d/ /etc/rc3.d/` ..., обнаружили ли вы ссылки на файлы в них?

9. Выведите список файлов директории `/lib` с помощью команды `ls -l`...

10. Используйте утилиту **`find`** для поиска в вашей домашней директории обычных файлов, на которые не установлено (`!`) ни одной жесткой ссылки.

33.7. Корректная процедура выполнения практического задания: ссылки на файлы

1. Создайте два файла с именами `winter.txt` и `summer.txt`, поместите какие-либо текстовые данные в них.

```
echo холодно > winter.txt ; echo жарко > summer.txt
```

2. Создайте жесткую ссылку на файл `winter.txt` с именем `hlwinter.txt`.

```
ln winter.txt hlwinter.txt
```

3. Выведите информацию об идентификаторах структур inode, соответствующих этим трем файлам, жесткие ссылки должны использовать структуры inode файлов, на которые они ссылаются.

```
ls -li winter.txt summer.txt hlwinter.txt
```

4. Используйте утилиту `find` для вывода информации о файлах, на которые установлены жесткие ссылки.

```
find . -inum xyz
```

5. Все данные файла, помимо двух типов данных, хранятся в соответствующей структуре inode. Назовите эти два типа данных!

Имя файла, которое хранится в файле директории и данные файла, которые хранятся где-либо на диске.

6. Создайте символьную ссылку на файл `summer.txt` с именем `slsummer.txt`.

```
ln -s summer.txt slsummer.txt
```

7. Найдите все файлы с идентификатором структуры inode, равным 2. Какой вывод вы можете сделать на основе полученной информации?

Можно сделать вывод о том, что в рамках системы существует более одной таблицы структур inode (по одной для каждого отформатированного раздела + виртуальные файловые системы).

8. Исследуйте директории `/etc/init.d/ /etc/rc.d/ /etc/rc3.d/` ..., обнаружили ли вы ссылки на файлы в них?

```
ls -l /etc/init.d
ls -l /etc/rc.d
ls -l /etc/rc3.d
```

9. Выведите список файлов директории /lib с помощью команды ls -l...

```
ls -l /lib
```

10. Используйте утилиту **`find`** для поиска в вашей домашней директории обычных файлов, на которые не установлено (`!`) ни одной жесткой ссылки.

```
find ~ ! -links 1 -type f
```

Часть IX. Приложения

Приложение A. Раскладки клавиатуры

A.1. О раскладках клавиатуры

Многие люди (например, жители США) предпочитают использовать установленную по умолчанию раскладку клавиатуры US-qwerty. Но если вы не живете в США и желаете использовать привычную раскладку клавиатуры при работе с вашей системой, лучшим вариантом является выбор раскладки клавиатуры в ходе установки системы. После этого в вашей системе будет постоянно использоваться корректная раскладка клавиатуры. Также при использовании ssh для удаленного управления системой Linux будет использоваться ваша локальная раскладка клавиатуры вне зависимости от конфигурации клавиатуры на сервере. Исходя из этого, вы вряд ли обнаружите большой объем информации о динамической смене раскладки клавиатуры в Linux только потому, что это требуется малому количеству пользователей. Ниже приведены некоторые советы по данной теме, которые могут оказаться полезными.

A.2. Раскладка клавиатуры оконной системы X

Ниже приведены примеры фрагментов конфигурационного файла `/etc/X11/xorg.conf`, предназначенные для установки бельгийской раскладки клавиатуры azerty и американской раскладки клавиатуры qwerty соответственно.

```
[paul@RHEL5 ~]$ grep -i xkb /etc/X11/xorg.conf
        Option      "XkbModel" "pc105"
        Option      "XkbLayout" "be"
[paul@RHEL5 ~]$ grep -i xkb /etc/X11/xorg.conf
        Option      "XkbModel" "pc105"
        Option      "XkbLayout" "us"
```

При работе с Gnome, KDE или каким-либо другим графическим окружением рабочего стола следует рассмотреть меню приложения для изменения настроек системы, в котором наверняка найдется раздел конфигурации клавиатуры, предназначенный для выбора предпочтительной раскладки клавиатуры. Попытайтесь использовать этот инструмент с графическим интерфейсом перед редактированием упомянутого файла конфигурации.

A.3. Раскладка клавиатуры в командной оболочке

При работе с командной оболочкой bash следует обратиться к файлу /etc/sysconfig/keyboard. Ниже приведены примеры конфигурации американской раскладки клавиатуры qwerty и бельгийской раскладки клавиатуры azerty соответственно.

```
[paul@RHEL5 ~]$ cat /etc/sysconfig/keyboard
KEYBOARDTYPE="pc"
KEYTABLE="us"
[paul@RHEL5 ~]$ cat /etc/sysconfig/keyboard
KEYBOARDTYPE="pc"
KEYTABLE="be-latin1"
```

Сами файлы с данными соответствия клавиш могут быть найдены в директории /usr/share/keymaps или /lib/kbd/keymaps.

```
[paul@RHEL5 ~]$ ls -l /lib/kbd/keymaps/
итого 52
drwxr-xr-x 2 root root 4096 апр  1 00:14 amiga
drwxr-xr-x 2 root root 4096 апр  1 00:14 atari
drwxr-xr-x 8 root root 4096 апр  1 00:14 i386
drwxr-xr-x 2 root root 4096 апр  1 00:14 include
drwxr-xr-x 4 root root 4096 апр  1 00:14 mac
lrwxrwxrwx 1 root root    3 апр  1 00:14 ppc -> mac
drwxr-xr-x 2 root root 4096 апр  1 00:14 sun
```

Приложение B. Аппаратное обеспечение

B.1. Шины

B.1.1. О шинах

Аппаратные компоненты взаимодействуют с **центральным процессором** (**Central Processing Unit** или **CPU**) посредством **шин**. На сегодняшний день наиболее часто используемыми шинами являются шины **usb**, **pci**, **agp**, **pci-express**, а также шина **pcmcia**, известная, как **pc-card**. Все они являются шинами с поддержкой технологии **Plug and Play**.

Более старые компьютеры архитектуры **x86** обычно имели шины **isa**, которые настраивались с помощью **джамперов** или **dip-переключателей**.

B.1.2. Директория /proc/bus

Для того, чтобы получить список шин вашего компьютера, распознанных ядром Linux, следует обратиться к файлам из директории **/proc/bus/** (ниже приведены примеры списков файлов из данной директории в случае использования дистрибутивов Ubuntu 7.04 и RHEL 4.4 соответственно).

```
root@laika:~# ls /proc/bus/
input  pccard  pci  usb
[root@RHEL4b ~]# ls /proc/bus/
input  pci  usb
```

Можете ли вы догадаться, какой из этих двух списков файлов был создан при работе с ноутбуком?

B.1.3. Утилита /usr/sbin/lsusb

Для получения списка всех устройств usb, соединенных с вашей системой, вам придется прочитать содержимое файла **/proc/bus/usb/devices** (конечно же, в том случае, если этот файл существует), причем вы также можете получить аналогичный лучше читаемый список с помощью утилиты **lsusb**, вывод которой для системы SPARC под управлением дистрибутива Ubuntu приведен ниже.

```
root@shaka:~# lsusb
Bus 001 Device 002: ID 0430:0100 Sun Microsystems, Inc.
3-button Mouse
Bus 001 Device 003: ID 0430:0005 Sun Microsystems, Inc.
Type 6 Keyboard
Bus 001 Device 001: ID 04b0:0136 Nikon Corp. Coolpix 7900
(storage)
root@shaka:~#
```

B.1.4. Файл /var/lib/usbutils/usb.ids

Файл **/var/lib/usbutils/usb.ids** содержит сжатый с помощью утилиты gzip список всех известных идентификаторов и соответствующих им описаний устройств usb.

```
paul@barry:~$ zmore /var/lib/usbutils/usb.ids | head
------> /var/lib/usbutils/usb.ids <------
#
#       List of USB ID's
#
```

```
#       Maintained by Vojtech Pavlik
#       If you have any new entries, send them to the
maintainer.
#       The latest version can be obtained from
#               http://www.linux-usb.org/usb.ids
#
# $Id: usb.ids,v 1.225 2006/07/13 04:18:02 dbrownell Exp
$
```

B.1.5. Утилита /usr/sbin/lspci

Для получения списка всех подсоединенных к системе устройств PCI вы можете либо обратиться к содержимому файла **/proc/bus/pci**, либо использовать утилиту **lspci** (фрагмент вывода которой приведен ниже).

```
paul@laika:~$ lspci
...
00:06.0 FireWire (IEEE 1394): Texas Instruments
TSB43AB22/A IEEE-139...
00:08.0 Ethernet controller: Realtek Semiconductor Co.,
Ltd. RTL-816...
00:09.0 Multimedia controller: Philips Semiconductors
SAA7133/SAA713...
00:0a.0 Network controller: RaLink RT2500 802.11g
Cardbus/mini-PCI
00:0f.0 RAID bus controller: VIA Technologies, Inc. VIA
VT6420 SATA ...
00:0f.1 IDE interface: VIA Technologies, Inc.
VT82C586A/B/VT82C686/A...
00:10.0 USB Controller: VIA Technologies, Inc. VT82xxxxx
UHCI USB 1....
00:10.1 USB Controller: VIA Technologies, Inc. VT82xxxxx
UHCI USB 1....
...
```

B.2. Запросы прерываний

B.2.1. О запросах прерываний

Запрос прерывания (**Interrupt request** или **IRQ**) является запросом, отправляемым устройством центральному процессору. Устройство генерирует запрос прерывания в тот момент, когда ему

должно быть уделено внимание со стороны центрального процессора (этот момент может наступить тогда, когда устройство заканчивает накопление данных, которые должны быть переданы центральному процессору).

С момента презентации шины PCI запросы прерываний могут разделяться между устройствами.

Запрос прерывания с идентификатором 0 всегда зарезервирован за системным таймером, с идентификатором 1 - за клавиатурой, а с идентификатором 2 используется в качестве канала для запросов прерываний с идентификаторами из диапазона от 8 до 15 и, таким образом, совпадает с запросом прерывания с идентификатором 9.

B.2.2. Файл /proc/interrupts

Вы можете ознакомиться со списком идентификаторов запросов прерываний в вашей системе, прочитав содержимое файла **/proc/interrupts**.

```
paul@laika:~$ cat /proc/interrupts
      CPU0     CPU1
0:  1320048     555  IO-APIC-edge      timer
1:    10224       7  IO-APIC-edge      i8042
7:        0       0  IO-APIC-edge      parport0
8:        2       1  IO-APIC-edge      rtc
10:     3062      21  IO-APIC-fasteoi   acpi
12:      131       2  IO-APIC-edge      i8042
15:    47073       0  IO-APIC-edge      ide1
18:        0       1  IO-APIC-fasteoi   yenta
19:    31056       1  IO-APIC-fasteoi   libata, ohci1394
20:    19042       1  IO-APIC-fasteoi   eth0
21:    44052       1  IO-APIC-fasteoi   uhci_hcd:usb1,
uhci_hcd:usb2,...
22:   188352       1  IO-APIC-fasteoi   ra0
23:   632444       1  IO-APIC-fasteoi   nvidia
24:     1585       1  IO-APIC-fasteoi   VIA82XX-MODEM,
VIA8237
```

B.2.3. Утилита dmesg

Также вы можете использовать утилиту **dmesg** для установления идентификаторов запросов прерываний, резервируемых для определенных устройств в процессе загрузки системы.

paul@laika:~$ dmesg | grep "irq 1[45]"

[28.930069] ata3: PATA max UDMA/133 cmd 0x1f0 ctl 0x3f6 bmdma 0x2090 irq 14

[28.930071] ata4: PATA max UDMA/133 cmd 0x170 ctl 0x376 bmdma 0x2098 irq 15

B.3. Порты ввода-вывода

B.3.1. О портах ввода-вывода

Передача данных в противоположном направлении от центрального процессора к устройству осуществляется посредством **портов ввода-вывода** (**IO Ports**). Центральный процессор записывает данные или управляющие коды в порт ввода-вывода устройства. Но данный метод передачи данных не является односторонним, так как центральный процессор может также использовать порт ввода-вывода устройства для чтения информации о состоянии устройства. В отличие от запросов прерываний, порты ввода-вывода не могут разделяться между устройствами!

B.3.2. Файл /proc/ioports

Вы можете получить список портов ввода-вывода устройств вашей системы, прочитав содержимое файла **/proc/interrupts**.

```
[root@RHEL4b ~]# cat /proc/ioports
0000-001f : dma1
0020-0021 : pic1
0040-0043 : timer0
0050-0053 : timer1
0060-006f : keyboard
0070-0077 : rtc
0080-008f : dma page reg
00a0-00a1 : pic2
00c0-00df : dma2
00f0-00ff : fpu
```

```
0170-0177 : ide1
02f8-02ff : serial
...
```

B.4. Технология DMA

B.4.1. О технологии DMA

Устройство, которое передает большие объемы данных, а также требует для своей работы генерации большого количества запросов прерываний и создания большого количества портов ввода-вывода, может создавать значительную нагрузку на центральный процессор. Благодаря технологии **прямого доступа к памяти** (**Direct Memory Access** или **DMA**), устройство может получить доступ (временный) к определенному диапазону адресов **оперативной памяти**.

B.4.2. Файл /proc/dma

Прочитав содержимое файла **/proc/dma**, вы наверняка не получите необходимой информации ввиду того, что данный файл содержит исключительно информацию об используемых в данный момент каналах **прямого доступа к памяти** для устройств **ISA**.

```
root@laika:~# cat /proc/dma
1: parport0
4: cascade
```

Ввиду того, что данные об устройствах **PCI**, использующих технологию прямого доступа к памяти, не приводятся в файле **/proc/dma**, в случае необходимости получения информации о таких устройствах может оказаться полезной утилита **dmesg**. В примере ниже показано, что в процессе загрузки параллельный порт получил канал прямого доступа к памяти номер 1, а инфракрасный порт - канал прямого доступа к памяти номер 3.

```
root@laika:~# dmesg | egrep -C 1 'dma 1|dma 3'
[   20.576000] parport: PnPBIOS parport detected.
[   20.580000] parport0: PC-style at 0x378 (0x778), irq
7, dma 1...
[   20.764000] irda_init()
--
[   21.204000] pnp: Device 00:0b activated.
[   21.204000] nsc_ircc_pnp_probe() : From PnP, found
firbase 0x2F8...
```

```
[    21.204000] nsc-ircc, chip->init
```

t any time before August 1, 2009,
provided the MMC is eligible for relicensing.