

**Linux -**

**ОСНОВЫ СИСТЕМНОГО  
администрирования**

*Савосин В.И.*

Тольятти  
2008 г.



# Лицензионное соглашение

Данный учебник распространяется на основе СТАНДАРТНОЙ ОБЩЕСТВЕННОЙ ЛИЦЕНЗИИ GNU версии 3, так как подпадает под определение ПРОГРАММА.

Из главы «Определения»:

"Программа" подразумевает любое охраноспособное произведение, лицензированное Данной Лицензией.

УЧЕБНИК «LINUX-ОСНОВЫ СИСТЕМНОГО АДМИНИСТРИРОВАНИЯ»

Copyright (C) 2008 Савосин Виталий Иванович

Эта книга является свободным произведением. Вы можете

распространять и/или модифицировать её согласно условиям Стандартной

Общественной Лицензии GNU, опубликованной Фондом Свободного Программного

Обеспечения, версии 3 или, по Вашему желанию, любой более поздней версии.

Это произведение распространяется в надежде, что оно будет полезным, но БЕЗ

ВСЯКИХ ГАРАНТИЙ, в том числе подразумеваемых гарантий ТОВАРНОГО СОСТОЯНИЯ ПРИ

ПРОДАЖЕ и ГОДНОСТИ ДЛЯ ОПРЕДЕЛЁННОГО ПРИМЕНЕНИЯ. Смотрите Стандартную

Общественную Лицензию GNU для получения дополнительной информации.

Вы должны были получить копию Стандартной Общественной Лицензии GNU вместе

с произведением. В случае её отсутствия, посмотрите <<http://www.gnu.org/licenses/gpl.html>>.

НЕофициальные переводы GPL2 и GPL3 на русский:

[http://www.infolex.narod.ru/gpl\\_gnu/gplrus.html](http://www.infolex.narod.ru/gpl_gnu/gplrus.html)

<http://code.google.com/p/gpl3rus/wiki/LatestRelease>

<http://sphynkx.livejournal.com/1571.html> (еще один вариант перевода GPL3, чуть понятней предыдущего)

Официальный текст GNU GPLv3 на английском:

<http://www.gnu.org/copyleft/gpl.html>

Статья «Краткое руководство по GPLv3» Бретт Смит:

<http://www.permlug.org/node/3770/>

Ваши предложения и пожелания прошу направлять на адрес электронной почты [vsavosin@yandex.ru](mailto:vsavosin@yandex.ru)

# Аннотация

Данная книга создавалась как методическое пособие для курса «Linux — системное администрирование и безопасность». Рассчитана на широкий круг читателей. Ознакомление с книгой не требует специальных знаний и навыков. Достаточно базового знакомства с компьютером и с операционной системой, например, MS Windows.

В учебнике и курсе используется дистрибутив Linux Slackware 12.1. Однако Вы можете использовать практически любую версию этого дистрибутива (точно могу сказать, что практически все примеры будут работать и в 8-ой версии), либо любой иной дистрибутив Linux, так как содержание книги практически полностью соответствует стандарту POSIX. Единственно, учитывайте что вывод командного интерпретатора может отличаться от приведенного в книге, однако поведение системы в целом будет аналогичным. Дополнительно, в книге, содержатся материалы по настройке дистрибутивов относящихся к группе RedHat-подобных.

Содержание книги построено таким образом, что поэтапно проводит читателя от знакомства с операционной системой Linux Slackware к созданию рабочей станции. Предполагается, что первые главы, до установки дистрибутива, слушатель курса изучает, подключаясь из среды MS Windows по протоколу ssh к уже существующей рабочей станции с установленной операционной системой Linux.

Если Вы изучаете книгу самостоятельно, то начинайте изучение книги с главы 11 «Установка дистрибутива». И далее возвращайтесь к началу книги.

Знакомится с книгой рекомендуется по порядку, от начала и до конца, не забегая вперед. Однако, если Вы все-таки ушли вперед и какой-то материал Вам не понятен, возвращайтесь назад и Вы наверняка найдете ответы на Ваши вопросы.

Изучив данный учебник вы получите работающую станцию. Но без графического интерфейса. Так как графический интерфейс является клиент-серверной системой его настройка и изучение вынесены в другую книгу - «Linux — сетевое администрирование»

При создании этого учебника были использованы различные документы, находящиеся в свободном доступе в сети Internet. Указанные документы не были просто скопированы и вставлены в текст, но были переработаны и обновлены до текущего состояния системы. Список использованных ресурсов находится в следующем разделе.

К сожалению, для части материалов, использованных в книге не сохранилось адреса источника и автора. Просто эти материалы были ранее сохранены мною на компьютере и в последствии я не смог найти их точного размещения в сети. Авторы этих материалов прошу написать мне по адресу [vsavosin@yandex.ru](mailto:vsavosin@yandex.ru) и указать точные ссылки. Позднее я добавлю эти ссылки в соответствующий раздел.

Так же, в конце книги находятся методические рекомендации к учебному курсу «Linux — системное администрирование и безопасность».

## Использованные ресурсы

1. <http://slackware.pp.ru/> - Linux Slackware Энциклопедия от А до Я
2. [http://wiki.kryukov.biz/wiki/Заглавная\\_страница](http://wiki.kryukov.biz/wiki/Заглавная_страница) — Сетевой учебник по Linux, А.Крюков
3. <http://www.linuxcenter.ru/lib/books/kostromin/> - "Linux для пользователя", В.Костромин
4. <http://lafox.net/docs/slackbook/slackbook/index.html> - Основы Slackware Linux
5. <http://wm-help.net/books-online/book/55570.html> - Linux Network Administrators Guide Russian
6. <http://www.protocols.ru/> - Энциклопедия сетевых протоколов от BiLiM Systems
7. <http://linux.yaroslavl.ru/docs/conf/kernel-2.6-install-1.1.html> - Ставим ядро 2.6, или Ядерная физика для домохозяйки. Версия 1.1.
8. [http://www.linuxrsp.ru/artic/print\\_server.html](http://www.linuxrsp.ru/artic/print_server.html) - Печать в Linux с железными нервами
9. <http://linux-admin.net.ru/content/520> - Служба печати CUPS.
10. <http://www.slackware.org/> - Сайт проекта Slackware Linux
11. <http://www.opennet.ru/cgi-bin/openforum/vsluhboard.cgi> — Форум на opennet.ru
12. <http://sysadmins.ru/> - Форум на sysadmins.ru
13. <http://linuxforum.ru/> - Форум пользователей Linux
14. <http://www.linuxcenter.ru/> - портал про Linux и Unix. Дистрибутивы, книги, статьи о Linux. Mandriva, Ubuntu, SUSE, Fedora, Red Hat, Debian, KNOPPIX, Gentoo, Slackware, FreeBSD, CentOS, Xandros, RedHat, Linux-XP, OpenBSD, Scientific, ASPLinux, ALTLinux, MOPSLinux
15. <http://book.itep.ru/1/intro1.htm> — Общие принципы построения сетей, Семенов Ю.А.
16. <http://www.altlinux.ru/> - Сайт производителя дистрибутива Alt Linux. Один из российских дистрибутивов Linux

# Оглавление

Лицензионное соглашение.....	3
Аннотация.....	4
Использованные ресурсы.....	5
Введение.....	12
Немного истории.....	12
Стандарт POSIX.....	12
Глава 1. Знакомство с операционной системой.....	14
Программы оболочки (командные интерпретаторы).....	14
Файловая система.....	15
Иерархия.....	15
Имена файлов.....	17
Символы подстановки в именах файлов.....	18
Типы файлов.....	18
Глава 2. Знакомство с файловой системой.....	20
Программы для работы с файловой системой.....	20
pwd.....	20
cd.....	20
ls.....	21
Физическое устройство файловой системы.....	23
Программы для работы с файлами.....	24
touch.....	24
cp.....	25
mv.....	26
dd.....	27
mkdir.....	28
rm.....	28
rmdir.....	28
Программы для просмотра текстовых файлов.....	28
cat.....	29
tac.....	30
head и tail.....	30
more.....	30
less.....	31
Глава 3. Знакомство с командным интерпретатором.....	32
Неадекватное поведение терминала.....	32
Комфортная работа в командной строке.....	32
Автоподстановка.....	32
История команд.....	33
Переменные среды окружения.....	34
Псевдонимы.....	36
Стандартные ввод, вывод и вывод ошибки.....	37
Перенаправления. Перенаправление стандартного вывода.....	38
Перенаправление стандартного вывода ошибки.....	38
Перенаправление стандартного ввода.....	38
Конвейеры.....	39
Глава 4. Обработка текстовой информации.....	40
Программы для управления выводом текстовой информации.....	40
tee.....	40
Структурированные текстовые файлы.....	40
cut.....	41
paste.....	42
sort.....	42
wc.....	43
tr.....	43
diff.....	44
Обработка текстовой информации.....	44
grep.....	44
Примеры регулярных выражений.....	45
Примеры использования grep.....	45
sed.....	45
Команда a.....	45
Команда i.....	45
Команда p.....	46
Команды s и g.....	46
Команда q.....	46

Команда d.....	46
Использование командного файла.....	46
Способы адресации.....	47
Глава 5. Ссылки.....	48
ln.....	48
Глава 6. Основы системы безопасности.....	51
Права доступа.....	51
Права доступа к файлам.....	51
Права доступа к директориям.....	52
Права доступа к символьным ссылкам.....	53
chmod.....	53
Числовой формат записи прав доступа.....	53
Символьный формат записи прав доступа.....	54
Копирование прав доступа.....	55
Специальные права.....	55
suid.....	55
sgid.....	57
stiky.....	58
Обозначение.....	58
Права доступа по умолчанию.....	58
POSIX ACL.....	59
getfacl.....	59
setfacl.....	60
Формат записи ACL.....	62
Изменение владельца и группы файлов.....	63
chown.....	63
chgrp.....	63
Глава 7. Процессы.....	64
ps.....	64
pgrep.....	66
Потомок, родитель.....	67
pstree.....	68
pohup.....	69
Демоны.....	69
Сигналы.....	69
kill.....	71
killall.....	72
pkill.....	72
Мониторинг процессов.....	72
top.....	72
Batch-режим.....	76
Приоритет процессов.....	77
nice.....	78
renice.....	79
Временная остановка выполнения процесса.....	79
Глава 8. Система помощи.....	82
man.....	82
Программа просмотра.....	83
Директории.....	83
MANPATH.....	84
Поиск в страницах руководства.....	84
Русские man pages.....	85
info.....	85
--help.....	85
HOWTO.....	86
Документация к программам.....	86
Глава 9. Текстовые редакторы.....	87
Редактор vi.....	87
Режимы работы редактора vi.....	87
Команды редактора vi.....	88
Примеры использования vi.....	88
Вместо заключения.....	89
Глава 10. Основы shell script.....	90
Запуск приложений.....	90
Переменные.....	90
Массивы переменных.....	91
Переменные окружения.....	93
Взаимодействие с пользователем.....	94

Подстановка.....	95
Арифметические выражения.....	96
Условный оператор if.....	97
Проверка условий с помощью оператора test.....	98
Операторы && и   .....	99
Оператор case.....	99
Оператор for.....	101
Получение данных из внешних файлов.....	103
Оператор while.....	104
Оператор select.....	105
Оператор «точка». Функции.....	106
Специальные переменные.....	108
Использование программы getopts.....	110
Оператор trap.....	111
Глава 11. Установка дистрибутива.....	113
Требования к компьютеру.....	113
Загрузка.....	113
Подготовка (разметка) жесткого диска.....	114
Типы файловых систем.....	119
Файловая система ext2.....	119
Журналируемые файловые системы.....	120
Файловая система ext3.....	120
Файловая система ReiserFS.....	121
Файловая система JFS.....	121
Файловая система XFS.....	121
Файловые системы компании Microsoft.....	122
Файловые системы iso9660 и udf.....	122
Виртуальные файловые системы.....	122
Файловая система proc.....	122
Файловая система sysfs.....	128
Файловая система для устройств /dev.....	128
Программа установки.....	130
Глава 12. Управление накопителями и файловыми системами.....	135
Программы для работы с накопителями.....	135
badblocks.....	135
mkfs.....	135
mke2fs.....	139
mkreiserfs.....	139
jfs_mkfs.....	140
mkfs.xfs.....	140
tune2fs.....	141
fsck.....	142
Использование файловых систем.....	144
mount.....	144
Параметры монтирования файловых систем.....	145
Параметры монтирования, общие для всех файловых систем.....	145
Параметры монтирования файловой системы ext3.....	145
Параметры монтирования файловой системы JFS.....	146
Параметры монтирования файловой системы XFS.....	146
Параметры монтирования файловой системы iso9660.....	146
Параметры монтирования файловой системы vfat.....	146
Параметры монтирования файловой системы ntfs.....	147
umount.....	147
Работа с файловыми системами.....	147
Файлы fstab и mtab.....	149
Использование USB накопителей.....	152
df.....	153
du.....	154
Глава 13. Работа с пользователями.....	156
Файл passwd.....	156
Файл shadow.....	157
Файл group.....	158
Программы для работы с учетными записями пользователей.....	159
skel каталог.....	160
vipw.....	160
Программа для смены пароля пользователя passwd.....	160
Управление группами.....	161
Программа su.....	161



Настройка пользовательской среды окружения.....	162
Приглашения интерпретатора.....	163
Файл ~/.bash_logout.....	164
Глава 14. Русификация консоли и консольных приложений (локализация).....	165
Таблицы кодировки символов.....	165
Локализация.....	167
Настройка системных средств локализации.....	168
Проверка наличия средств локализации.....	168
Формат задания значений переменных локализации.....	169
Включение средств локализации.....	169
Локализация консоли.....	170
consoletools.....	171
Как это сделано в дистрибутиве Black Cat.....	172
kbd.....	173
Как это сделано в Slackware Linux.....	174
Переключение кодировок.....	174
Отображение русских букв в файловых системах vfat и ntfs.....	175
Глава 15. Сети в Linux.....	176
Немного истории.....	176
Сети TCP/IP.....	176
Введение в сети TCP/IP.....	177
Ethernet.....	177
Хабы, свитчи и коммутаторы.....	178
Другие типы оборудования.....	178
Internet Protocol.....	179
IP на последовательных линиях.....	180
Transmission Control Protocol.....	180
User Datagram Protocol.....	180
Сетевые порты.....	181
Библиотека сокетов.....	181
Сети и Linux.....	182
Различные направления разработки.....	182
Сетевые интерфейсы.....	183
ip адреса.....	183
Что такое бит?.....	183
Сетевые маски.....	184
Трюки с сетевой маской.....	185
Шестнадцатеричные сетевые маски.....	185
Неиспользуемые адреса.....	186
Преобразование адресов.....	186
IP маршрутизация.....	187
IP сети.....	187
Подсети.....	187
Gateways.....	187
Таблица маршрутов (Routing Table).....	188
Метрические значения.....	189
Internet Control Message Protocol.....	189
Преобразование имен машин.....	190
Обзор сетевых устройств в Linux.....	190
Установка Ethernet.....	191
Автоопределение Ethernet.....	191
Другие типы сетей.....	193
Драйверы сетевых устройств в ядре.....	193
Динамическое подключение драйверов.....	193
Загрузка модулей сетевых карт в Slackware Linux.....	196
Определение HOSTNAME.....	196
Получение сетевого адреса.....	196
Создание подсетей.....	197
Создание файлов hosts и networks.....	197
Настройка сетевых интерфейсов.....	198
Расположение конфигурационных файлов.....	198
Программа ifconfig.....	199
Настройка интерфейса lo.....	200
Настройка Ethernet интерфейса eth0.....	201
Использование нескольких ip адресов на одном интерфейсе.....	201
Интерфейс для последовательного порта.....	202
Настройка маршрутизации.....	203
Лабораторная работа.....	204

Конфигурационный файл rc.inet1.....	206
Программа netstat.....	207
Отображение статистики интерфейса.....	208
Отображение соединений.....	208
Проверка ARP-таблиц.....	208
Программа tcpdump.....	209
Фильтрация при сборе пакетов.....	212
Логические выражения.....	213
Примеры фильтров.....	214
Настройка клиента DNS.....	214
Библиотека Resolver.....	214
Файл host.conf.....	215
Resolver и переменные окружения.....	215
Файл nsswitch.conf.....	216
Файл resolv.conf.....	217
Ошибкоустойчивость Resolver.....	218
Настройка удаленного управления.....	219
Шифрование.....	219
Ключи.....	219
Симметричное шифрование.....	220
Асимметричное шифрование.....	221
Установка и конфигурирование ssh.....	223
Ssh daemon.....	223
Клиент ssh.....	224
Использование ssh.....	225
Глава 16. Поиск файлов в файловой системе.....	227
Программа which.....	227
Программа whereis.....	227
Программа locate.....	227
Программа find.....	228
Глава 17. Архивация и резервное копирование.....	231
Программы для архивации данных.....	231
Программа tar.....	231
Программа cpio.....	234
Сжатие (компрессия) данных.....	238
Программа gzip.....	239
Программа bzip2.....	239
Программа compress.....	240
Сравнение программ сжатия.....	240
Вызов программ сжатия из программы tar.....	241
Резервное копирование и восстановление данных.....	242
Какова Ваша стратегия?.....	242
Что копировать?.....	242
Инструменты резервного копирования.....	243
Глава 18. Управление программным обеспечением.....	245
Менеджер пакетов Slackware.....	245
Формат пакетов.....	245
Программы для работы с пакетами.....	246
pkgtool.....	246
Проверка установленного программного обеспечения.....	246
installpkg.....	247
upgradepkg.....	249
removepkg.....	249
swaret.....	250
rpm2tgz/rpm2targz.....	251
Создание пакетов.....	251
explodepkg.....	251
makepkg.....	252
Скрипты SlackBuild.....	252
Менеджер пакетов RPM.....	252
Компиляция программного обеспечения из исходных текстов.....	255
Необходимые сведения о программировании на языке Си.....	255
Инсталляция программного обеспечения из исходных кодов.....	256
Программа make.....	257
Пример сборки программ из исходных кодов. Программа screen.....	257
Глава 19. Система печати в Linux.....	263
Классические средства печати.....	263
Файл /etc/printcap.....	264

Фильтры.....	265
PostScript и Ghostscript.....	266
Шрифты для Ghostscript.....	266
Печать на удаленный принтер.....	267
Система печати CUPS.....	267
Как это работает?.....	268
Обновление информации о принтерах.....	268
Классификация принтеров.....	268
Интеграция с MS Windows.....	268
Настройка CUPS.....	268
Модули системы печати CUPS.....	269
Пример подключения USB принтера HP LaserJet 1300.....	270
Сетевая печать.....	270
WEB-интерфейс системы CUPS.....	270
Глава 20. Ядро Linux.....	271
Нумерация версий ядра.....	271
Старая система нумерации.....	272
Современная система нумерации.....	272
О компиляции нового ядра.....	272
Перед тем как начать.....	273
Семь шагов к новому ядру.....	273
Получение и разархивация ядра.....	273
Применение заплаток к исходникам ядра.....	274
Конфигурирование будущего ядра.....	275
Компиляция ядра.....	276
Компиляция модулей.....	277
Установка ядра.....	277
Настройка загрузчика ядра.....	278
Глава 21. Система инициализации Linux.....	281
Программа init.....	281
Формат файла /etc/inittab.....	282
Система инициализации BSD.....	285
Файл /etc/rc.d/rc.S.....	285
Заключение.....	290
Файлы /etc/rc.d/rc.modules и /etc/rc.d/rc.netdevice.....	291
Файл /etc/rc.d/rc.M.....	291
Файлы /etc/rc.d/rc.inet1 и /etc/rc.d/inet2.....	294
Файл /etc/rc.d/rc.inet1.....	294
Файл /etc/rc.d/rc.inet2.....	295
Заключение.....	295
Система инициализации SystemV.....	296
Файл /etc/rc.d/rc.....	297
Программа chkconfig.....	298
Запуск и останов сервисов вручную.....	299
Конфигурационные файлы в директории /etc/sysconfig.....	300
Приложение 1. Восстановление пароля пользователя root.....	301
Методические рекомендации.....	302

## Введение

Данный курс проводится с использованием операционной системы **Linux Slackware**. Не имеет особого значения какую unix-подобную операционную систему использовать в самом начале. И именно по этой причине Linux привлекателен для изучения unix-подобных систем. И здесь возникает необходимость внимательно изучить историю возникновения и развития операционной системы Unix в целом и Linux в частности как потомка и соответственно зеркала отражающего в себе своих родителей.

### Немного истории

Итак, в начале был Unix. И даже не Unix, а Multics - проект многопользовательской операционной системы, обеспечивающей удобный доступ большому числу пользователей к вычислительным ресурсам. Этот проект разрабатывался специалистами Bell Labs с 1965 по 1969 год, но так и не был доведен до стадии коммерческого продукта. Bell Labs являлась научно-технической лабораторией крупной компании AT&T. Компания AT&T до сих пор является одной из крупнейших компаний занимающихся телефонией и телекоммуникациями. И в тот период перед Bell Labs была поставлена задача создания операционной системы с разделением времени для современных ATC. Идеи, заложенные в проекте Multics, нашли выход в реализации ОС Unix, написанной сотрудниками Bell Labs - Кеном Томпсоном и Деннисом Ритчи. Первая редакция этой ОС была опубликована в 1971 году. Первая редакция была реализована на языке Assembler и предназначена для машин PDP-11/20. В начале 1973 года вышла уже 3-я редакция системы и в ней уже был встроен первый компилятор для языка C. И следующим шагом в развитии системы становится выход 4-ой редакции в том же году. В ней уже все ядро написано на языке C.

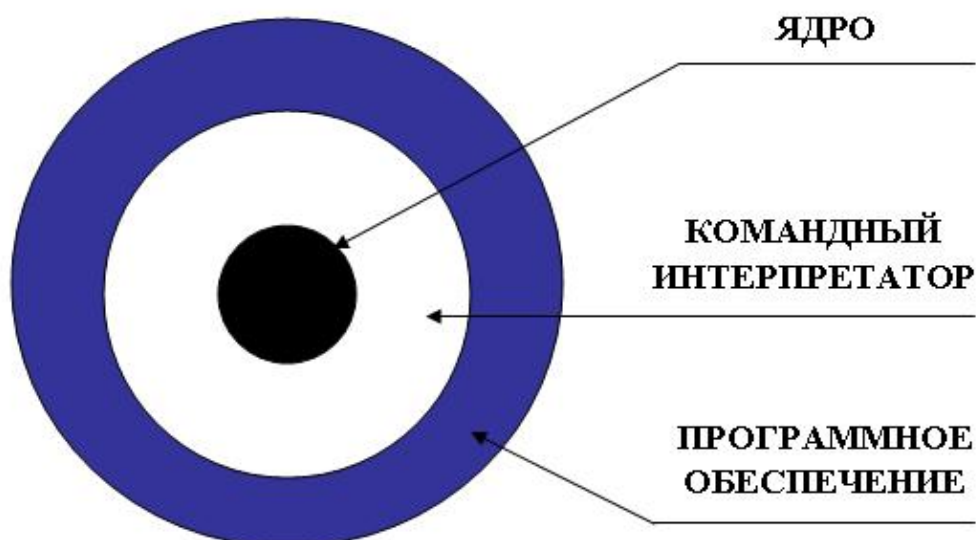
Началом коммерческого использования операционной системы Unix становится выход 6-ой редакции в 1975 году. Теперь вся ОС написана на языке C. В это время Bell Labs занимается исключительно операционной системой Unix. Подготавливает ее коммерческие версии. Стоимость одной лицензии на рынке составляет 21000 долларов США. У Unix фактически нет ни одного конкурента, способного предложить операционную систему сравнимую по возможностям с Unix. И компания AT&T принимает стратегическое решение в конечном итоге повлиявшее на весь современный рынок информационных технологий. Всем высшим учебным заведениям в США Unix предлагается по цене 100 долларов США. Фактически Компания AT&T вырастила поколение пользователей своей операционной системы и вкупе с этим практически бесплатно пользовалась всеми разработками этих университетов. И, вполне естественным, представляется то, что один из университетов в 1977 году выпускает свой программный пакет к ядру Unix, который называется BSD (Berkley System Distribution – дистрибутив Университета Беркли). Компания AT&T подает в суд на Университет Беркли за коммерческое использование ее исходного кода. И следующий дистрибутив BSD увидел свет только 2 года спустя. Но в исходном коде нет уже ни одной строчки из исходного кода AT&T. И более того, с сохранением функциональности и производительности, серьезно изменена система инициализации операционной системы. В дальнейшем мы рассмотрим специфику системы инициализации классического Unix-a и BSD. О функциональности BSD может говорить хотя бы тот факт что в 1981 году по заказу DARPA университетом Беркли разработан стек протоколов TCP/IP, в последующем успешно примененный во всех Unix-системах. Даже компания Microsoft фактически была вынуждена использовать этот стек протоколов, что бы попасть в мир **Internet**, появившийся именно благодаря TCP/IP и среде Unix.

### Стандарт POSIX

Следующим этапом в развитии Unix стал перенос этой операционной системы на другие платформы. Вполне понятным выглядело желание пользователей использовать одинаковое программное обеспечение вне зависимости от платформы, на которой работает Unix. В связи с этим началась разработка стандарта **POSIX** (Portable Operating System for Communication Environment). Стандарт POSIX определял следующее:

- программный интерфейс приложений;
- командный интерпретатор;
- набор утилит.

Прежде чем разобраться со стандартом POSIX, нам необходимо представить структуру операционной системы Unix.



Фактически, ядро отделено от всего программного обеспечения, и, как мы увидим в дальнейшем, от «железа», программным интерпретатором, который обрабатывает все запросы к ядру и «железу». Итак, для всего программного обеспечения написанного под Unix обязательным является соответствие стандарту. Из чего следует что, в общем случае, фактически все программное обеспечение, написанное для среды Unix должно работать в любой Unix-системе, будь то Linux, FreeBSD, Solaris или HP-UX. В любом дистрибутиве Unix обязательным условием поставки является:

- несколько стандартизованных программных интерпретаторов;
- набор стандартных утилит для базовой настройки системы.

Фактически оказавшись в среде любой Unix-системы, вы сможете с помощью одних и тех же команд и утилит осуществить базовую настройку системы или разобраться в текущей конфигурации.

# Глава 1. Знакомство с операционной системой

## Программы оболочки (командные интерпретаторы)

После входа пользователя в систему запускается программа, предоставляющая пользователю командную строку. Это так называемая оболочка (shell). Задача оболочки — дать возможность пользователю запускать на выполнение программы. Кроме того, в shell встроен язык программирования, так называемый shell-scripts. На этом языке можно писать командные файлы аналоги .bat файлов в MS Windows. В дальнейшем будем называть программы, написанные на этом языке просто скриптами.

За время существования UNIX было написано большое количество программ оболочек. Самая первая — это Bourne Shell (программа называется sh). Язык программирования, встроенный в нее не похож на язык C. В дальнейшем мы изучим этот язык и Вы поймете, что он очень не однозначен. Но его знание является обязательным для любого администратора UNIX.

Компания, разработавшая систему BSD, написала свой вариант оболочки — C shell (csh). И с тех пор существуют два направления в программах оболочках: sh и csh.

Время не стоит на месте. В базовые программы вносились дополнения. Сейчас существует много разнообразных вариантов на тему той или иной разновидности оболочек. Если посмотреть на классический sh, то его дальнейшее развитие — это ksh (Корн shell) и bash (Еще один shell господина Борна). Последний написан сообществом GNU и входит в Linux как оболочка по умолчанию. Среди последователей csh наиболее известен tcsh. Эта оболочка по умолчанию применяется в BSD системах. Tcsh так же может использоваться в Linux.

Несмотря на то, что основные принципы работы пользователей одинаковы во всех разновидностях оболочек, все-таки существуют различные нюансы. В некоторых вариантах оболочек были добавлены новые возможности, которые отсутствуют в других вариантах. И это может накладывать ограничения на Вашу работу.

Вам всегда необходимо точно знать в какой оболочке Вы работаете. Существует несколько способов узнать это. Самый простой — это посмотреть специальную переменную среды окружения. В Linux, как и в MS Windows, а если говорить точнее, то в Windows, как и в UNIX, существует такое понятие, как переменные среды окружения. Наиболее известная переменная — это PATH. В Linux есть специальная переменная, в которой хранится имя оболочки, в которой Вы сейчас работаете — SHELL. Для того, что бы посмотреть ее содержимое можно воспользоваться программой echo, которой в качестве аргумента в командной строке следует указать переменную, перед именем которой стоит специальный символ «\$».

**Внимание!** В Linux учитывается регистр букв. То есть если Вы напишите file или File — это будут два разных слова.

**Внимание!** В командной строке Linux, аргументы командной строки обязательно отделяются друг от друга пробелами или табуляциями. Нельзя, как это принято в Windows, писать параметр сразу после имени программы.

Итак, что бы посмотреть содержимое переменной среды окружения SHELL, необходимо выполнить следующую команду:

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

В результате, на экран терминала был выведен полный путь к файлу shell.

**Внимание!** Символ «\$» в начале строки — это приглашение командной строки. Если Вы будете вводить команды в командной строке, «\$» писать не надо. В Вашей системе приглашение командной строки может выглядеть по-другому, все зависит от настроек.

Еще один способ определить shell, в котором Вы работаете — это посмотреть список процессов, выполняемых текущим пользователем на текущем терминале. Для этого можно воспользоваться программой ps.

```
$ ps
  PID TTY          TIME CMD
 2365 pts/0    00:00:00 bash
 2377 pts/0    00:00:00 ps
$
```

На экран выводится список процессов. Самый первый процесс — это обычно shell, в котором Вы сейчас

работаете.

Для завершения работы в системе, можно воспользоваться одной из перечисленных ниже команд:

- logout
- exit
- комбинация клавиш «Ctrl+D».

Logout — это классическая команда, которую можно использовать в любой оболочке. Будет ли работать команда exit или комбинация клавиш «Ctrl+D», зависит от реализации оболочки.

После выхода из системы, сама система продолжает работать. Просто еще один пользователь покинул систему. Для выключения Linux необходимо иметь соответствующие права для использования совсем другой программы.

## Файловая система

Файловая система в Linux имеет огромное значение. Принцип организации файловой системы сильно отличается оттого, что Вы видели в Windows.

В Linux нет такого понятия, как устройства A:, C: и т.д. В Linux есть одна цельная файловая система, которая состоит из физических файловых систем. Представьте себе лоскутное одеяло. В целом — это одеяло, но сшитое из отдельных лоскутков.

У файловой системы Linux есть такое понятие как корень файловой системы, он обозначается символом «/». Корень файловой системы обязательно расположен в какой-либо физической файловой системе. Корневая файловая система (а если точнее — тот раздел, где она расположена) будет видна Linux сразу после старта системы. Для того чтобы были доступны остальные физические файловые системы их необходимо подключать (*Вместо термина подключение, можно использовать термин монтирование. Он происходит от имени программы, которая используется для подключения файловых систем — mount*). Подключать придется все файловые системы, которыми Вы собираетесь пользоваться: разделы жестких дисков, файловые системы флоппи дисков и CD-ROM, сетевые файловые системы и т.д.

Файловая система подключается к любой существующей директории. После подключения все файлы файловой системы будут видны внутри директории, к которой она подключена.

Например, в корневой файловой системе (аналог раздела C: Windows) есть директория dir. На винчестере существует еще один раздел (аналог раздела D: Windows). В этом разделе есть файл с именем file.txt. Для того, что бы получить доступ к файлу второй раздел следует подключить к директории dir. После этого файл будет находиться в директории dir. То есть абсолютный путь к файлу будет выглядеть так:

```
/dir/file.txt
```

Аналогичным образом подключаются любые файловые системы.

## Иерархия

За более чем 30 лет, которые существует UNIX, сложился определенный стандарт на директории, которые должны находиться в корне файловой системы UNIX.

В 1993 году была сформирована группа по стандартизации файловой системы Linux — Linux File System Standard Group (FSSTND). Но, не смотря на то, что этот стандарт был принят, не все дистрибутивы следовали его рекомендациям. В 1997 году был разработан другой стандарт — Linux File Hierarchy Standard (FHS). Найти его можно в Интернет на сайте <http://www.pathname.com/fhs>.

**Внимание!** Вы можете не следовать рекомендациям стандарта. В своей системе, при наличии определенных прав, Вы Царь и Бог и можете делать все, что хотите. Но учтите, что весь остальной мир может быть с Вами не согласен и программы, написанные для стандартных систем, будут работать у Вас только после длительной обработки их напильником.

Давайте посмотрим, какие директории и для чего используются в Linux.

/bin - в этой директории располагаются исполняемые файлы программ, необходимых при старте системы и доступные всем пользователям системы.

/sbin - тут расположены исполняемые файлы программ, необходимых при старте системы и доступные только администратору. Эти программы используются для конфигурации системы.

/etc - в этой директории располагаются конфигурационные файлы системы и различных программ. В одной из поддиректорий находятся стартовые скрипты системы. Почти все конфигурационные файлы — это текстовые

файлы. Поэтому, что бы настроить какую ни будь программу Вам необходимо знать формат файла и любой текстовый редактор. К сожалению, файлы различных программ имеют свои собственные форматы *(Существует проект, который пытается переписать все конфигурационные файлы с использованием формата XML, но я думаю, что у них ничего не получится. Уж слишком консервативен мир UNIX.)*.

`/dev` - директория содержит файлы устройств. Файлы устройств — это особенность UNIX. Дело в том, что в UNIX все представлено в виде файлов. С обыкновенными файлами и директориями дело обстоит так же как и в Windows. А вот для доступа к устройствам (CD-ROM, микшер на звуковой карточке и т.д.) используются специальные файлы. Программа, обратившаяся к таким файлам, начинает общаться с драйвером устройства. Программисты для работы с файлами устройств могут использовать стандартные функции системы: `open`, `read`, `write` и `close`. Что очень упрощает написание программ.

`/lib` - в этой директории находятся файлы динамически загружаемых библиотек и модулей ядра Linux. Обычно в этой директории располагают библиотеки необходимые при старте системы.

`/media` - содержит директории, являющиеся точками подключения съемных устройств. Таких как: CD/DVD, флоппи дисков, USB накопителей.

`/mnt` - содержит директории, являющиеся точками монтирования временных файловых систем. Например, необходимо посмотреть содержимое ISO образа CD-ROM. Файловую систему, находящуюся в этом файле можно подключить непосредственно к директории `/mnt` или к одной из директорий, находящихся в `/mnt`.

`/home` - тут располагаются домашние директории пользователей системы. Имя домашней директории обычно соответствует имени учетной записи пользователя. Например, у пользователя `user` домашняя директория `/home/user`. В Linux все пользователи кроме пользователя `root`, считаются обыкновенными пользователями системы. По умолчанию они могут создавать новые файлы и директории только в пределах своей домашней директории (а так же в директориях `/tmp` и `/var/tmp`).

`/root` - домашняя директория суперпользователя `root`.

`/srv` - содержит директории с данными различных сервисов. Например: WEB сервера Apache, FTP сервера и т.д.

`/tmp` и `/var/tmp` - Директории предназначены для хранения временных файлов. По умолчанию, доступ к этим директориям имеют все пользователи системы. Удалять файлы в этих директориях могут только их хозяин и суперпользователь.

`/usr` - Директория предназначена для хранения файлов, которые не нужны при первоначальной инициализации системы. Например, в этой директории находятся исполняемые файлы WEB сервера Apache. Apache не нужен при первоначальной инициализации системы, это сервис, который будет запущен только после того, как система будет полностью готова к работе. По своей структуре директория почти полностью повторяет содержимое корневой директории. То есть в ней есть такие директории как: `bin`, `sbin`, `etc`, `lib` и т.д. Содержимое этой директории обычно выносят на отдельный раздел винчестера или на сетевой диск и подключают при старте системы к точке монтирования `/usr`.

`/var` - Директория содержит часто изменяемые файлы программ. Говоря проще — директория содержит файлы с данными. Например, в этой директории обычно находятся почтовые ящики пользователей. Почта приходит и уходит, содержимое почтовых ящиков постоянно меняется, следовательно, меняется и содержимое файлов. Тут же располагают кеш прокси сервера и т.д. Содержимое директории `/var` рекомендуется размещать на отдельном разделе винчестера.

`/opt` - По своему назначению эта директория очень напоминает директорию Windows «Program Files». Для каждой программы, размещаемой в `/opt`, создается отдельная поддиректория имя которой соответствует имени программы. По своей структуре эта поддиректория повторяет корневую директорию Linux.

Что бы понять для чего используется директория `/opt`, необходимо разобраться, каким образом можно устанавливать и удалять программы в Linux. Существует два основных способа работы с программами: установка программы из пакета и из исходных кодов.

Пакет — это файл с архивом, в котором содержатся файлы программы. К сожалению, в Linux нет единого стандарта на формат пакетов и в разных дистрибутивах могут использоваться различные форматы, не совместимые друг с другом. Для работы с пакетами используется специальное программное обеспечение, называемое менеджером пакетов. При помощи менеджера можно выполнить основные действия: установить, обновить и удалить программу. Современные менеджеры пакетов, кроме перечисленных действий могут иметь дополнительные возможности.

Менеджер пакетов при установке программы размещает ее файлы по директориям: исполняемые файлы в `bin` или `sbin`, конфигурационные в `etc`, библиотеки в `lib` и т.д. Он запоминает, куда файлы были помещены в процессе установки. И если потребуется удалить программу из системы, менеджер легко найдет и удалит файлы программы.



В случае сборки программ из исходных кодов может возникнуть ситуация, когда разработчики программы не предусмотрели возможность создания пакета для определенного типа Linux. Тогда Вам придется пользоваться стандартным способом, встроенным практически во все программы сборки софта. В этом случае, при установке программы все ее файлы будут помещены в директории заранее заданные программе установки. Но это будет сделано путем простого копирования файлов, минуя менеджер пакетов.

Представьте себе, что у программы, которую Вы установили, было скопировано 500 файлов. Эти файлы, как это принято в Linux, разбросаны по различным директориям. И Вы, по каким либо причинам, хотите эту программу удалить из системы. Обращаться за помощью к менеджеру пакетов бесполезно. Программа была установлена без его участия, и он ничего не знает о файлах программы. Придется самостоятельно искать эти файлы и удалять их вручную (*У некоторых программ присутствует возможность удаления файлов, но для этого необходимо, что бы в системе оставались исходные коды программы.*).

Для того, что бы такие ситуации не возникали, при сборке программ из исходных кодов, для установки можно использовать директорию /opt. Например, если Вы собираете сервер Samba, можно сделать так, что все файлы этого сервера будут помещены в директорию /opt/samba.

Внутри этой директории будут созданы все необходимые директории:

```
/opt/samba/bin
/opt/samba/sbin
/opt/samba/lib
/opt/samba/var
и т.д.
```

Для удаления сервера Samba достаточно удалить директорию /opt/samba.

**Внимание!** Если это возможно, при сборке программ из исходных кодов, не поленитесь, создайте пакет. Гораздо проще управлять системой используя менеджеры пакетов, чем каждый раз вручную заменять и исправлять программы установленные без менеджера пакетов.

Продолжим рассматривать FHS.

/proc - Точка монтирования виртуальной файловой системы proc. Сама по себе файловая система proc отдельная, большая тема, которую необходимо рассматривать специально. Если попытаться кратко рассказать о ней, то все файлы, которые Вы будете видеть в этой директории — это отображение области данных ядра Linux на файловую систему. Ядро Linux — это программа, у которой есть области кода, стека и данных. То есть если Вы смотрите содержимое файла /proc/filesystems, то на самом деле система выводит содержимое области данных ядра, в котором описаны типы поддерживаемых файловых систем. Кроме просмотра содержимого файлов (области данных), есть возможность записи данных в некоторые файлы (область данных). В этом случае Вы можете напрямую влиять на работу системы. Например, есть файл содержащий число — количество файлов, которые программа может одновременно держать открытыми. Записав туда другое число, Вы изменяете количество одновременно открытых программой файлов без перезагрузки системы. Не следует путать registry Windows и директорию /proc Linux. Registry — это файл, /proc — это область памяти ядра. После перезагрузки системы у ядра все параметры устанавливаются по умолчанию. Для определения необходимых Вам параметров их следует записать в файлы директории /proc при старте системы. Для этих целей существуют специальные конфигурационные файлы.

/boot - В директории располагаются файлы ядра и загрузчика операционной системы.

Не все дистрибутивы полностью соблюдают рекомендации FHS. Например, в одних дистрибутивах для подключения съемных устройств до сих пор используется директория /mnt. В некоторых нет директории /srv. Но остальные директории имеют одинаковое предназначение во всех дистрибутивах Linux.

## Имена файлов

**Внимание!** В Linux учитывается регистр букв.

А это значит, что в одной директории могут быть файлы с именами file и File. И это будут два разных файла.

В именах файлов нельзя использовать следующие символы:

```
& ; | * ? ' " ` [ ] ( ) $ < > { } ^ # \ / % !
```

Все остальные символы разрешено использовать (*Существует возможность использовать специальные символы в именах файлов. Но лучше этим не пользоваться, некоторые программы будут очень удивляться встречая такие файлы.*). Но рекомендуется пользоваться только некоторыми из них. Это:

- Английские и, если система русифицирована, русские буквы.

- Цифры.
- Символы тире и подчеркивание.
- Символ точка.

Несмотря на то, что в именах файлов можно использовать пробелы, Вы должны быть готовы к некоторым неудобствам, связанным с их использованием. В командной строке пробел имеет специальное назначение — он разделяет аргументы командной строки. И если в имени встречаются пробелы, имя файла необходимо помещать в двойные кавычки. Пробелы в двойных кавычках интерпретируются как обычный символ, а не разделитель аргументов.

*"The file name"*

Еще один вариант — использование символа обратный слеш — \. Он используется для экранирования значения специального символа, указанного сразу после него. То есть для экранирования специального значения символа пробел, перед каждым пробелом в имени файла необходимо ставить обратный слеш.

*The\ file\ name*

Имя файла должно состоять не более чем из 256 символов.

В файловой системе Linux нет такого понятия как расширение файла. Но в именах файлов можно использовать один или несколько символов точка.

## Символы подстановки в именах файлов

В именах файлов можно использовать символы подстановки:

- \* — звездочка заменяет любое количество любых символов. Причем, символы могут отсутствовать.
- ? — заменяет только один любой символ. Символ обязательно должен существовать.
- [ и ] — заменяет только один символ. Внутри квадратных скобок перечисляются символы, которые могут быть в этой позиции. Например, [abc] или [0a-zA-Z]. Как видно из примеров, внутри скобок можно просто перечислять символы, можно указывать диапазон символов через —, так же можно комбинировать перечисления и диапазоны.

## Типы файлов

Как уже говорилось выше — в Linux все представлено в виде файлов. Но файлы бывают разных типов.

Тип файла можно узнать при помощи программы ls (первый символ первого поля) или при помощи программы stat.

В Linux используются файлы следующих типов:

- Обыкновенный файл — то, что принято считать файлом в Windows: текстовые файлы, файлы баз данных, исполняемые файлы программ и т.д. Программа ls обозначает их символом —.
- Директория. Обозначаются символом **d**.
- Символьная ссылка. Обозначаются символом **l**.
- Файл блочного устройства. Обозначаются символом **b**.
- Файл символьного устройства. Обозначаются символом **c**.
- FIFO. Обозначаются символом **f**.
- Socket (соединение). Обозначаются символом **s**.

С обыкновенными файлами, директориями и символьными ссылками все более или менее ясно. Осталось понять, для чего предназначены другие типы файлов.

Начнём с файлов устройств. В Linux существуют два типа устройств: символьные и блочные. В символьных устройствах информация передается последовательно, по одному биту. Например, COM порт или консоль — это символьные устройства. В блочных устройствах информация передается параллельно. К таким устройствам относятся жесткие диски, CD-ROM и т.д. Все блочные устройства в системе для ускорения работы с ними буферизируются.

При обращении программы к файлу устройства система делает так, что программа начинает общаться с драйвером устройства.

Файлы FIFO (иногда их ещё называют именованными каналами) предназначены для передачи данных между программами. Представьте, что одной программе необходимо передать другой программе большое количество данных. В Linux программы не имеют доступ к оперативной памяти другой программы и не могут передать данные непосредственно в память. Но одна программа может открыть на запись FIFO файл, а другая открыть его на чтение. FIFO — это first input first output — первый вошел, первый вышел. То есть данные передаются как по трубе.

**Внимание!** Если Вы думаете, что передача данных через файловую систему — это медленная операция, то тут Вы немного ошибаетесь. В Linux все файловые системы буферизируются. Реально передача данных между программами происходит через буфера файловой системы, то есть через оперативную память.

У FIFO файлов много достоинств, но есть один серьезный недостаток: данные можно передавать только в одну сторону. Если программам необходимо организовать двухстороннюю передачу данных, то лучше воспользоваться сетевым соединением. Но использовать реальное сетевое соединение имеет смысл только тогда, когда программы выполняются на разных компьютерах в сети. Если они работают на одном и том же компьютере, то лучше передавать данные через специальные файлы, так называемые UNIX domain sockets. При использовании файлов типа socket, как и в случае использования FIFO файлов, данные передаются через буфера файловой системы.

Файлы типа socket в Linux применяются повсеместно. Например, графическая оболочка X Window построена по принципу клиент-сервер. Когда программы X сервер и X клиенты работают на одной машине, они общаются через файлы типа socket. Множество этих файлов можно увидеть в директории /tmp. Или взаимодействие почтового сервера и антивируса. Антивирус ClamAV может принимать запросы, как по реальному сетевому соединению, так и через файл типа socket. Но при настройке почтового сервера приходится явно указывать каким из способов передачи данных необходимо пользоваться.

## Глава 2. Знакомство с файловой системой

### Программы для работы с файловой системой

Почти все программы, которые будут рассмотрены в этом разделе, описаны в стандарте POSIX. Следовательно, они встречаются во всех POSIX-совместимых операционных системах.

#### pwd

Программа `pwd` выводит на экран текущую директорию.

В Linux существует такое понятие как текущая директория. Для каждой программы в состоянии исполнения определена текущая директория. Когда пользователь работает за терминалом с помощью оболочки `bash`, то для `bash` также определена текущая директория, то есть директория, относительно которой вычисляются пути к файлам.

Некоторые программы оболочки имеют встроенную команду `pwd`. Для того, что бы выполнить именно программу, а не встроенную в оболочку команду, при вызове программы следует указать полный путь к ней.

```
$ /bin/pwd
/home/stavr
$
```

#### cd

Программа `cd` используется для изменения текущей директории.

##### cd [директория]

Если программу `cd` выполнить без каких-либо аргументов, Вы перейдете в свою домашнюю директорию (*После входа в систему пользователь автоматически помещается в свою домашнюю директорию.*).

Для перехода в конкретную директорию, при вызове программы `cd` необходимо явно указать путь к директории.

Существуют два способа указания пути к объекту файловой системы: абсолютный и относительный. Если путь указывается от корневой директории, то есть в начале пути присутствует символ слеш — то это абсолютный путь.

```
/usr/local/src
```

Если при указании пути корневая директория не указывается — то это относительный путь. Относительно текущей директории:

```
local/src
```

В каждой директории в файловой системе Linux предусмотрены специальные директории: `.` и `..`. Точка — явно показывает текущую директорию. Две точки — это директория уровнем выше в иерархии директорий.

Директорию точка следует использовать, если необходимо явно указать, что файл или программа находятся в текущей директории. В переменной среды окружения `PATH` в Linux текущая директория обычно не указывается. Поэтому для выполнения программы, находящейся в текущей директории, ее приходится указывать явно:

```
./program
```

Если в качестве имени директории программе `cd` передать символ `-`, то программа переместит Вас в ту директорию, в которой Вы были до того, как оказаться в текущей. Например, находясь в директории `/etc`, Вы решили перейти в директорию `/usr/src`.

```
$ pwd
/etc
$ cd /usr/src
$ pwd
/usr/src
$
```

После выполнения команды `cd` с параметром `-`, Вы возвращаетесь в директорию `/etc`:

```
$ cd -
$ pwd
/etc
$
```

Ещё одна интересная особенность, связанная с директориями в Linux. Если Вы работаете в оболочке bash, то в качестве имени домашней директории можно использовать символ тильды — ~. Тильда определяет домашнюю директорию текущего пользователя. То есть если её использует пользователь user, то вместо тильды будет подставляться /home/user. Если её использует user2, то будет подставлено /home/user2.

Если Вы хотите указать путь относительно своей домашней директории, тогда можете смело использовать тильду. Согласитесь, что набрать в командной строке:

```
cd ~/bin
```

значительно проще и быстрее чем:

```
cd /home/user/bin
```

## ls

Программа ls может показать:

- содержимое директории;
- список файлов;
- подробную информацию о файлах и т.д.

ls [опции] [файл...]

Если программу ls выполнить без каких-либо аргументов, будет показан список всех файлов, находящихся в текущей директории, за исключением скрытых файлов.

Для того, что бы увидеть все файлы в директории, в том числе и скрытые, необходимо использовать опцию -a:

```
$ ls -a
./ ../ .bash_history .lesshst .screenrc .xsession
$
```

В файловой системе Linux у файлов нет атрибута «скрытый файл». Принято считать, что если имя файла начинается с точки — то это скрытый файл. К таким файлам можно отнести директории . и ..

Для того, что бы посмотреть содержимое директории совсем не обязательно сначала переходить в эту директорию. Достаточно явно указать путь к той директории, содержимое которой Вы хотите посмотреть:

```
ls /etc
```

Что бы получить полную информацию о файле/файлах, требуется использовать параметр -l. Если Вы будете указывать путь к директории, ls покажет полную информацию о файлах, находящихся в этой директории:

```
$ ls -l /etc
total 1780
-rw-r--r-- 1 root root 3458 2007-06-09 06:12 DIR_COLORS
-rw-r--r-- 1 root root 18 2008-08-14 20:49 HOSTNAME
drwxr-xr-x 18 root root 4096 2007-02-21 22:31 X11/
-rw-r--r-- 1 root root 2561 2002-02-25 00:37 a2ps-site.cfg
-rw-r--r-- 1 root root 15067 2002-02-25 00:37 a2ps.cfg
drwxr-xr-x 3 root root 41 2004-11-05 12:20 acpi/
-rw-r--r-- 1 root root 47 2008-08-30 11:14 adjtime
-rw-r--r-- 1 root root 265 2008-02-16 23:35 anthy-conf
drwxr-xr-x 4 root root 4096 2008-08-14 19:44 asciidoc/
-rw-r----- 1 root daemon 144 2006-08-03 05:55 at.deny
drwxr-xr-x 3 root root 4096 2008-08-14 20:12 bluetooth/
-rw-r--r-- 1 root root 1229 2006-06-10 04:35 bootptab
.....
$
```

Самое первое значение итога или total (Какое слово будет выводиться на экран зависит от того, какой язык используется в Вашей системе.) показывает, какое количество дискового пространства (в блоках) занимают

файлы, находящиеся в этой директории. Один блок равен одному килобайту.

Затем отображается подробная информация о файлах, один файл — одна строка:

- Первое поле — в этом поле показаны тип файла и права доступа. Обратите внимание на то, что в Linux права доступа не наследуются. То есть нельзя как в Windows или Novell Netware определить права доступа для директории, которые будут автоматически распространяться на все файлы, которые находятся в этой директории. В Linux у каждого объекта файловой системы права доступа свои и их значение не наследуется.
- Второе поле — количество ссылок на файл (*Имеются в виду жесткие ссылки*). Файл существует до тех пор, пока существует хотя бы одна ссылка на него. Что такое ссылка будет рассказано дальше в этой главе.
- Третье поле — владелец файла. В Linux у каждого файла обязательно должен быть владелец. У файла может быть только один владелец.
- Четвертое поле — группа, которой принадлежит данный файл. Это тоже особенность Linux — любой файл должен принадлежать группе пользователей. Файл может принадлежать только одной группе пользователей.
- Пятое поле — размер файла в байтах.
- Шестое поле — время последней модификации (изменения) файла.
- Седьмое поле — имя файла.

Какие поля будут выводиться на экран, зависит от параметров, переданных при вызове программы ls.

Если программе был передан параметр `-F` — тогда после каждого исполняемого файла будет выводиться символ `*`, после директории символ `/`, символической ссылки `->` и т.д. Обычно этот параметр передается по умолчанию.

Если Вы хотите получить информацию о конкретном файле, программе ls следует указать путь к интересующему Вас файлу:

```
$ ls -l /etc/hosts
-rw-r--r-- 1 root root 624 2008-08-14 20:49 /etc/hosts
$
```

Иногда возникает необходимость посмотреть содержимое сразу нескольких директорий. В этом случае, программе ls следует указать пути к этим директориям, разделяя их пробелами.

```
$ ls /etc/ssh /etc/ssl
/etc/ssh:
moduli          ssh_host_dsa_key.pub  ssh_host_rsa_key
ssh_config      ssh_host_key          ssh_host_rsa_key.pub
ssh_host_dsa_key ssh_host_key.pub      sshd_config

/etc/ssl:
certs/ misc/ openssl.cnf private/
$
```

Если необходимо посмотреть содержимое директории и всех поддиректорий, используйте параметр `-R` (Помните? Большие и маленькие буквы в Linux различаются.):

```
ls -R /etc
```

Результат выполнения этой команды слишком громоздкий и по этому не был здесь приведен.

Перечисленные выше опции программы ls определены в стандарте POSIX и могут использоваться в любой POSIX совместимой операционной системе.

В Linux используется вариант программы ls, написанный сообществом GNU. И при ее вызове можно использовать дополнительные параметры.

Среди параметров, свойственных GNU версии программы можно выделить параметр `--color`. Он позволяет различные типы файлов выводить различными цветами. Или наоборот, отключать эту возможность.

При использовании параметра `--color`, требуется указать дополнительный аргумент (*Все GNU параметры обязательно начинаются с двух тире «--»*). Если параметру требуется передавать дополнительный аргумент, он указывается после символа `=`). Можно использовать перечисленные ниже аргументы:

- none — не использовать цвета при выводе;
- auto — использовать цвета только в том случае, если программа выводит данные на терминал;
- always — при выводе всегда использовать цвета.

Какие цвета будут использоваться при выводе, определяет переменная среды окружения `LS_COLORS`. Чтобы посмотреть ее содержимое, можно выполнить следующую команду:

```
$ echo $LS_COLORS
no=00:fi=00:di=01;34:ln=01;36:pi=40;33:so=01;35:do=01;35:bd=40;33;01:cd=40;33;01:or=40;31;01:su=37;41:sg=30;43:tw=30;42:ow=34;42:st=37;44:ex=01;32:*.bat=01;32:*.BAT=01;32:*.btm=01;32:*.BTM=01;32:*.cmd=01;32:*.CMD=01;32:*.com=01;32:*.COM=01;32:*.dll=01;32:*.DLL=01;32:*.exe=01;32:*.EXE=01;32:*.arj=01;31:*.bz2=01;31:*.deb=01;31:*.gz=01;31:*.lzh=01;31:*.rar=01;31:*.RAR=01;31:*.rpm=01;31:*.tar=01;31:*.taz=01;31:*.tb2=01;31:*.tbz2=01;31:*.tbz=01;31:*.tgz=01;31:*.tz2=01;31:*.z=01;31:*.Z=01;31:*.zip=01;31:*.ZIP=01;31:*.zoo=01;31:*.asf=01;35:*.ASF=01;35:*.avi=01;35:*.AVI=01;35:*.bmp=01;35:*.BMP=01;35:*.flac=01;35:*.FLAC=01;35:*.gif=01;35:*.GIF=01;35:*.jpg=01;35:*.JPG=01;35:*.jpeg=01;35:*.JPEG=01;35:*.m2a=01;35:*.M2A=01;35:*.m2v=01;35:*.M2V=01;35:*.m4a=01;35:*.M4A=01;35:*.m4p=01;35:*.M4P=01;35:*.m4v=01;35:*.M4V=01;35:*.mov=01;35:*.MOV=01;35:*.mp3=01;35:*.MP3=01;35:*.mpc=01;35:*.MPC=01;35:*.mpeg=01;35:*.MPEG=01;35:*.mpg=01;35:*.MPG=01;35:*.ogg=01;35:*.OGG=01;35:*.pbm=01;35:*.pgm=01;35:*.png=01;35:*.PNG=01;35:*.ppm=01;35:*.ram=01;35:*.RAM=01;35:*.rm=01;35:*.RM=01;35:*.tga=01;35:*.TGA=01;35:*.tif=01;35:*.TIF=01;35:*.tiff=01;35:*.TIFF=01;35:*.wav=01;35:*.WAV=01;35:*.wma=01;35:*.WMA=01;35:*.wmv=01;35:*.WMV=01;35:*.xbm=01;35:*.xcf=01;35:*.xpm=01;35:*.xwd=01;35:*.XWD=01;35:
$
```

Цифры, которые Вы увидите — это не запись в формате RGB. Для определения цвета используется специальный формат.

В Slackware Linux используется еще одна переменная среды окружения, которая влияет на работу программы `ls` — `LS_OPTIONS`. В этой переменной содержатся параметры программы, которые передаются программе по умолчанию. Что бы посмотреть её содержимое можно выполнить следующую команду:

```
$ echo $LS_OPTIONS
-F -b -T 0 --color=auto
$
```

В других дистрибутивах Linux для передачи параметров по умолчанию используется механизм псевдонимов (aliases), о котором будет рассказано позже.

## Физическое устройство файловой системы

В Linux можно использовать большое количество файловых систем, но все они имеют общий принцип построения.

У каждого файла в файловой системе есть так называемый inode — описатель файла. В inode хранится информация о файле.

В пределах одной физической файловой системы (Помните? В Linux есть одна файловая система, которая состоит из различных физических файловых систем. В данном случае имеется в виду именно физическая файловая система.) каждый inode имеет свой уникальный номер. Таким образом, каждый файл имеет уникальный номер.

В inode хранится информация о:

- Правах доступа. Права представлены в виде набора байт.
- Уникальный номер пользователя (UID), которому принадлежит этот файл. Обратите внимание на то, что в inode хранится номер, а не имя пользователя.
- Уникальный номер группы (GID), которой принадлежит данный файл. Точно так же как и с хозяином файла, в inode хранится номер, а не имя группы.
- Временные метки:
- Время последней модификации файла (modification time).
- Время последнего доступа к файлу (access time). Если Вы прочитали содержимое файла, даже не

изменяя его — это поле будет изменено.

- Время последнего изменения информации в inode файла (change time). Это поле изменяет свое значение когда изменяется информация в inode. Например, при изменении прав доступа.
- Ссылки на блоки, в которых расположены данные. Данные не хранятся в inode, а располагаются в отдельных блоках на диске. Чем больше максимальный размер дискового пространства, отводимого под inode — тем больше возможное количество ссылок на блоки — тем больше максимальный размер файла.

В inode могут храниться другие атрибуты файла. Какие атрибуты будут использованы — зависит от типа файловой системы.

Современные файловые системы имеют усовершенствованную структуру. Например, ссылки на блоки могут храниться не только в inode, но и в самом блоке. То есть получается ссылка на блок ссылок.

В одном блоке может храниться информация, принадлежащая нескольким файлам.

Если информация, хранящаяся в файле, может поместиться в inode, для нее не выделяют отдельный блок и ее размещают непосредственно в inode.

Как будет храниться информация, зависит от типа файловой системы.

Если посмотреть внимательно на информацию, содержащуюся в inode, там нет имени файла. Дело в том, что имена файлов хранятся в директории.

Директория, как и любой файл в Linux, тоже имеет описатель файла — inode. Там хранится такая же информация как и в случае обыкновенных файлов: номер файла, права доступа, хозяин файла, группа, временные метки и т.д. Данные, которые хранятся в файле типа директория — это таблица, связывающая между собой имя файла и inode. То есть физически файл — это набор данных, а директория связывает имя данных и сами данные.

Имя файла — это ссылка на файл. Вспомните, что выводила программа `ls -l`, а точнее второе поле — количество ссылок на файл. Получается, что на одни данные может быть несколько ссылок — несколько имен, и это действительно возможно. Такая ссылка называется жесткой (hard link).

Физически файл (данные) существует до тех пор, пока на этот файл существует хотя бы одна ссылка (*Это утверждение не относится к символическим ссылкам. О символических ссылках будет рассказано ниже.*). То есть пока существует хотя бы одно имя файла — данные существуют.

Увидеть inode можно выполнив следующую команду:

```
$ ls -li
16777354 README          141 dvd/          33595529 memory/
50331784 cdrecorder/    16777353 floppy/   33595528 tmp/
50331783 cdrom/         140 hd/           16777355 zip/
$
```

## Программы для работы с файлами

В этом разделе будут рассмотрены программы для работы с файлами и директориями. Все программы описаны в стандарте POSIX.

### touch

Программа `touch` предназначена для изменения времени последней модификации и/или доступа файла.

`touch [-acm][-r базовый_файл][-t время] [--] файл...`

При вызове программы требуется указать имя файла или нескольких файлов, разделяя их пробелами.

Если программа выполнялась без указания каких-либо параметров, то время последней модификации и доступа к файлу изменяются на текущее время системы на момент выполнения `touch`.

Если необходимо изменить только время последней модификации файла, при вызове программы следует использовать параметр `-m`. При изменении только времени последнего доступа к файлу используют параметр `-a`.

Иногда возникает необходимость в установке времени отличного от текущего. В этом случае используют опцию `-t`. Этой опции следует обязательно указать новое значение времени в следующем формате (в квадратных скобках указаны необязательные параметры):



[ [CC]YY]MMDDhhmm[.SS]

Где:

- CC — столетие. Например: 20 или 19.
- YY — год. Например: 06 или 99. Если не указано столетие, тогда учитывается раздел по 1969 году: 1969-2068.
- MM — месяц.
- DD — день.
- hh — час.
- mm — минуты.
- ss — секунды. Секунды обычно не используются, так как в администрировании системы минимальный используемый квант времени равен минуте.

Например, если Вы хотите записать только время последней модификации равное 12:00 25 Мая 2004 года, командная строка будет выглядеть следующим образом:

```
touch -m -t 200405251200 file
```

У программы `touch` есть интересная особенность, если файл, указанный при вызове программы не существует, программа его создаст! Если Вы не хотите, что бы файл создавался, при вызове программы необходимо указать параметр `-c`.

Например, если в текущей директории файла `test` нет, то первая программа его не создаст, а во втором случае файл будет создан:

```
touch -c test
```

```
touch test
```

## ср

Программа `ср` предназначена для копирования файлов.

`ср [параметры] файл_источник файл_назначения`

`ср [параметры] файл_источник... директория_назначения`

Существуют два варианта использования программы. В первом случае указывается файл и путь, куда это файл необходимо скопировать. Например:

```
cp test test.bak
```

В этом примере содержимое файла с именем `test` копируется в файл с именем `test.bak`. Причем, если файл `test.bak` существовал — он создается снова. То есть старые данные, которые находились в нем, теряются.

Во втором варианте, в качестве последнего аргумента используется директория. Тогда указанный файл или файлы будут скопированы с тем же самым именем в указанную директорию:

```
cp test test.bak anydir
```

Если копируется один файл, то можно указать его новое имя в этой директории:

```
cp test anydir/newtest
```

Если необходимо скопировать содержимое директории, следует использовать опцию `-R` или `-r`. В GNU реализации программы `ср` — эти опции являются синонимами.

Например, необходимо скопировать директорию `dir1` со всем содержимым в директорию `dir2`. Командная строка будет выглядеть следующим образом:

```
cp -R dir1 dir2
```

Свои действия программы в Linux обычно выполняют без вывода лишней информации на экран. То есть при написании программ предусматривается, что пользователь, выполняющий ее, сначала подумал о последствиях, прежде чем запустил программу. Как уже говорилось выше, `ср` переписывает файл назначения. При этом на экран не выводится никаких предупреждений.

Если Вы хотите, что бы программа выводила предупреждение о том, что файл назначения существует, при ее вызове следует использовать параметр `-i`.

В некоторых дистрибутивах при вызове, программе по умолчанию передается параметр `-i`. При копировании большого количества файлов приходится отвечать на вопрос программы о каждом файле. Если Вам не хочется этого делать, можно воспользоваться параметром `-f`. В этом случае параметр `-i`, если он был определен, игнорируется. Предположим, что Вам требуется произвести замену жесткого диска. В этом случае сначала необходимо скопировать все данные, находящиеся на нем во временную директорию:

```
cp -R hddir tmpdir
```

Затем происходит замена диска, его форматирование и подключение файловой системы к прежней точке монтирования. После чего данные копируются обратно:

```
cp -R tmpdir hddir
```

И Вы обнаруживаете, что программы, хранящие свои данные на этом диске, перестали работать. Для того, что бы понять, почему это произошло, следует разобраться с тем «что в лесу шишки».

Вы работаете в системе как пользователь `user1` и копируете файл, принадлежащий пользователю `user2`:

```
cp file_user2 newfile
```

Вопрос заключается в следующем: кому принадлежит вновь созданный файл `newfile`? Пользователю `user1` или `user2`?

Ответ на этот вопрос очень простой. В Linux пользователь может создавать файлы принадлежащие только ему. То есть все создаваемые пользователем `user1` файлы будут принадлежать ему. Соответственно, у файла `newfile` хозяином будет `user1`. Теперь вернемся к случаю, когда мы копировали содержимое диска. Эту операцию мог делать только суперпользователь `root`, так как на диске файлы принадлежали различным пользователям с различными правами доступа.

**Внимание!** В Linux только пользователь `root` имеет право доступа к любым файлам любых пользователей системы.

После того, как файлы были скопированы, все файлы в новой директории стали принадлежать пользователю `root`, так как при копировании именно он их создавал. Более того, всем создаваемым файлам в Linux присваиваются права доступа по умолчанию. Получается, что после копирования, все файлы принадлежат пользователю `root` и имеют права доступа по умолчанию. С другой стороны, программы, работающие с этими файлами, будут выполняться с правами других пользователей системы. И после копирования эти программы не смогут получить доступ к файлам.

Для того, что бы при копировании у файлов не изменялись их атрибуты: хозяин, группа, права доступа и т.д. При выполнении программы `cp` надо использовать опцию `-r`. Но учтите, что если Вы копируете файлы нескольких пользователей, эту опцию может использовать только пользователь `root`. Потому что — это единственный пользователь в системе кому разрешено создавать файлы, принадлежащие другим пользователям. Обыкновенные пользователи системы опцию `-r` могут использовать только когда они копируют файлы принадлежащие им.

## mv

Программа `mv` предназначена для переименования и/или переноса файлов и/или директорий.

```
mv [параметры...] файл_источник файл_назначения
```

```
mv [параметры...] файл_источник... файл_назначения
```

Если в качестве файла источника и файла назначения указаны файлы, находящиеся в одной физической файловой системе, перенос данных не происходит. Просто изменяется имя (ссылка) на существующие данные. Если файлы находятся в различных физических файловых системах — тогда данные переносятся.

У программы `mv` есть параметр `-i`. Его значение точно такое же, как и у `cp`.

При переносе или переименовании директорий программе не надо указывать никаких дополнительных опций. Параметр `-f` имеет такое же значение, как и аналогичный параметр программы `cp`.

Используя программу `mv` файл можно одновременно перемещать из одной директории в другую и переименовывать. Например, следующая команда перенесет файл с именем `test` в директорию `dir`, одновременно изменив его имя на `newtest`:

```
mv test dir/newtest
```

## dd

Программа dd предназначена для копирования и конвертирования файлов.

### dd [параметры]

Если пользоваться этой программой только в Linux, возможности конвертирования файлов Вам не потребуются. Конвертирование обычно используется при переносе данных между различными архитектурами компьютеров.

С точки зрения копирования данных программа dd отличается от cp в первую очередь возможностью копирования не всего файла целиком, а только части файла. Например, определенного количества блоков с начала файла, или предварительно пропустив некоторое количество блоков.

Если программу запустить без каких либо параметров, она читает данные со стандартного ввода и передает их на стандартный вывод.

Параметр if определяет имя входного файла. Параметр of — выходного. Например, что бы скопировать данные из файла file1 в file2, необходимо выполнить следующую команду:

```
dd if=file1 of=file2
```

Обратите внимание на то, что в программе dd (по историческим причинам) перед параметром не ставится символ тире.

Изначально программа dd предназначалась для непосредственной работы с накопителями. И это наложило отпечаток на работу программы — она копирует данные блоками. Размер блока по умолчанию равен 512 байтам. При помощи параметра bs можно установить размер блока. При помощи параметра count — определить количество копируемых блоков. Например, если необходимо скопировать 100 блоков размером в 1024 байта, можно воспользоваться следующей командой:

```
dd if=/dev/zero of=swap bs=1024 count=100
```

Если создать файл при помощи программы touch у файла будет нулевой размер.

В приведенном примере есть один интересный момент — в качестве источника данных используется файл /dev/zero. Судя по тому, что он находится в директории /dev можно предположить, что это файл устройства. И это действительно так. Устройство /dev/zero предназначено для чтения бесконечного количества абсолютных нулей (в шестнадцатеричном виде — 0x00). Если при выполнении этой команды забыть указать параметр count, то, теоретически, можно получить файл swap бесконечного размера, содержащий только нули. Но в реальной жизни размер файла будет ограничен размером файловой системы. Или, если администратор действительно следит за своей системой, квотами, наложенными на файловую систему.

Если выполнить следующую команду:

```
dd if=/dev/fd0 of=floppy.img
```

информация будет читаться из файла устройства /dev/fd0. В Linux этот файл соответствует дисководу гибких дисков. Когда программа будет читать из этого файла, драйвер дисковода будет выдавать содержимое гибкого диска по секторам, то есть не в виде отдельных файлов, а именно посекторная копия диска. Соответственно, файл floppy.img — это образ гибкого диска.

Если при вызове программы поменять эти файлы местами:

```
dd if=floppy.img of=/dev/fd0
```

то происходит запись данных в файл устройства /dev/fd0. В этом случае драйвер дисковода всю входящую информации посекторно записывает на гибкий диск. То есть происходит посекторное копирование дисков.

Аналогичным образом можно поступить, когда необходимо сделать копию жесткого диска или раздела на жестком диске. В этом случае указываются файлы соответствующих устройств. Например, что бы посекторно скопировать раздел /dev/hda1 в /dev/hdb1 (*Файл устройства /dev/hda1 соответствует первому разделу на мастер IDE жестком диске, подключенном к первому IDE контроллеру. А файл /dev/hdb1 — первому разделу slave жесткого диска, подключенному к первому IDE контроллеру*), необходимо выполнить следующую команду:

```
dd if=/dev/hda1 of=/dev/hdb1
```

Следующая команда позволит получить ISO образ CD-ROM:

```
dd if=/dev/cdrom of=cdrom.iso
```

Как Вы видите, совсем необязательно иметь специальное программное обеспечение для получения такого образа. К сожалению, поменять местами файлы /dev/cdrom и cdrom.iso нельзя. Для записи CD-R/RW

необходимо пользоваться специальной программой, например `cdrecord`.

Если на устройстве, откуда читаются данные, предполагается наличие сбойных блоков, программе `dd` обязательно указывают параметр `conv=noerror, sync`. `Noerror` заставляет программу продолжать читать данные после обнаружения ошибки чтения. `Sync` дополняет каждый входной блок до размера по умолчанию или определенного параметром `bs`.

## mkdir

Программа `mkdir` предназначена для создания директорий.

`mkdir [-p][-m mode] dir...`

При вызове программы необходимо указать имя или имена создаваемых директорий, разделяя их пробелами. Например:

```
mkdir dir1
```

```
mkdir dir2 dir3 dir4
```

У создаваемых директорий устанавливаются права по умолчанию. Если Вы хотите при создании указать другие права, необходимо использовать опцию `-m`:

```
mkdir -m 750 dir1
```

Если попытаться выполнить следующую команду:

```
mkdir d1/d2/d3
```

В случае отсутствия в текущей директории директории `d1`, Вы получите сообщение об ошибке. Дело в том, что `d1/d2/d3` — это путь к объекту в файловой системе. `d1/d2` — путь, а `d3` — имя объекта. Поскольку директории `d1` нет, то невозможно создать директорию `d3`, расположенную по указанному пути. Для того чтобы программа `mkdir` создавала все необходимые родительские директории, при ее вызове необходимо использовать опцию `-p`.

```
mkdir -p d1/d2/d3
```

В этом случае сообщения об ошибке не будет. Будут созданы директории `d1`, `d2` и `d3`.

## rm

Программа `rm` предназначена для удаления файлов или директорий.

`rm [параметры] файл...`

По умолчанию программа не может удалять директории. Для этого ей требуется указать параметр `-R` или `-r`. В этом случае удаляется директория со всем содержимым.

Как и подавляющее большинство программ в Linux, `rm` удаляет файлы без вывода на экран информации о процессе удаления. Сообщения программы можно увидеть только при возникновении ошибки.

**Внимание!** Учтите, что в Linux нет мусорной корзины, а процесс восстановления удаленных файлов — задача не для слабоверных. Хорошо подумайте, прежде чем нажмете на кнопку `Enter`.

Если Вы хотите, что бы при удалении файла программа сначала задавала вопрос, при ее запуске необходимо указать параметр `-i`.

## rmdir

Программа предназначена для удаления пустых директорий.

`rmdir [параметры] директория...`

Программа досталась нам в наследство и остается в системе для совместимости с предыдущими версиями.

## Программы для просмотра текстовых файлов

Большая часть базовых утилит, поставляемых с Linux, предназначена для работы с текстовыми файлами. Сказывается историческое наследие. UNIX разрабатывался для управления патентной документацией, то есть как система для работы базы данных. В то время практически не существовало движков БД таких как, например, Oracle или DB2. Базы данных представляли собой набор структурированных текстовых файлов. С

UNIX поставлялся набор программ для работы с такими текстовыми файлами. Этот набор программ описан в стандарте POSIX и поэтому считается обязательным, поэтому он так же присутствует в Linux.

Здесь будут показаны только основные программы, которые могут пригодиться в реальной работе. В стандарте POSIX их гораздо больше. Кроме того, сертификационные экзамены предусматривают знание таких программ.

Это далеко не все утилиты, которые применяются для обработки текстовых файлов. Мы продолжим рассмотрение таких программ далее.

## cat

Программа cat предназначена для объединения файлов.

cat [параметры] [файл]...

Каким образом при помощи cat можно объединять файлы мы посмотрим несколько позже, после того как изучим необходимый для понимания используемых технологий материал. Сейчас нас интересует одна из особенностей программы — просмотр содержимого небольших текстовых файлов.

При помощи программы рекомендуется просматривать только небольшие файлы, т.к. при просмотре большого файла Вы увидите только его последние строки. Даже если пользоваться буфером терминала (*Для перемещения по буферу терминала используются комбинации клавиш «Ctrl+PgUp» и «Ctrl+PgDn»*), некоторые файлы настолько большие, что не помещаются целиком в этот буфер и будут видны только последние строки файла.

Для того, что бы посмотреть содержимое файла /etc/hosts, необходимо выполнить следующую команду:

```
$ cat /etc/hosts
# For loopbacking.
127.0.0.1          localhost
10.10.0.21        master.unix.class master
# End of hosts.
$
```

У программы cat есть много других параметров. Мы рассмотрим только три:

- -b — при выводе файла программа нумерует строки.
- -E — в конце строки выводится символ \$.
- -T — вместо символа табуляции выводятся символы ^I.

**Внимание!** Вы должны запомнить, что cat, как и большинство рассматриваемых программ, не будет изменять содержимое файла. То есть файл — это источник данных, а результат работы программы выводится только на экран терминала.

Явное обозначение конца строки — это проблема программ написанных плохими программистами. Иногда встречаются программы некорректно обрабатывающие свои конфигурационные файлы — параметры в файле написаны правильно, а программа выдает сообщение об ошибке в этой строке. Проблема в лишние пробелах, которые могут присутствовать в конце строки. Такие пробелы обнаружить достаточно трудно, ведь этот символ никак не отображается в большинстве редакторов и программ показывающих содержимое текстовых файлов. В обнаружении лишних пробелов в конце строки поможет программа cat.

```
$ cat -E /etc/hosts
# For loopback# For loopbacking
127.0.0.1          localhost$
10.10.0.21        master.unix.class master$
$
# End of hosts.$
$
```

Если после последнего параметра в конце строки перед символом \$ есть пробелы, они будут видны.

Параметр -T помогает обнаружить, где в тексте стоят символы табуляции. Например, в Linux в конфигурационном файле системы отвечающей за журнальную регистрацию (SYSLOG) поля в строке отделяются символом табуляции и только им. Если среди табуляций будет введен хотя бы один пробел, программа будет выдавать сообщение об ошибке. Табуляции, как и пробелы, тоже никак не отображаются в редакторах. Если пропустить такой конфигурационный файл через cat с параметром -T, сразу будут видны лишние пробелы.

## tac

Программа `tac` аналогична программе `cat`, но выводит содержимое файла по строкам в обратном порядке.

`tac [параметры] [файл]...`

Я не вижу особой необходимости в этой программе, но она определена в стандарте POSIX и о ней могут спрашивать на сертификационных экзаменах.

Пример использования программы `tac`:

```
$ tac /etc/hosts
# End of hosts.
10.10.0.21          master.unix.class master
127.0.0.1          localhost
# For loopbacking.
$
```

## head и tail

Программа `head` показывает первые десять строк файла. Программа `tail` последние десять строк.

`head [параметры] [файл]...`

`tail [параметры] [файл]...`

При вызове программ можно указать количество строк, которые будут выводить программы. Это можно сделать двумя способами. В обоих случаях будут показаны первые три строки файла `/etc/hosts`:

```
$ head -3 /etc/hosts
# For loopbacking.
127.0.0.1          localhost
10.10.0.21          master.unix.class master
$ head -n 3 /etc/hosts
# For loopbacking.
127.0.0.1          localhost
10.10.0.21          master.unix.class master
$
```

Программы умеют выводить файлы не только построчно, но и посимвольно. В этом случае используется параметр `-c` с указанием количества символов. Например:

```
$ head -c50 /etc/hosts
# For loopbacking.
127.0.0.1          localhost
10.10.0.21
$
```

У программы `tail` есть параметр `-f`. Если `tail` запустить с этим параметром она переходит в бесконечный цикл, в котором раз в секунду перечитывает конец указанного файла. Что бы выйти из этого режима, используйте комбинацию клавиш `Ctrl+C`.

Зачем нужен такой режим программы `tail`? Почти все файлы журнальной регистрации в Linux — это текстовые файлы. Новая информация добавляется в конец файла. Если запустить `tail` с параметром `-f`, то Вы будете почти в реальном времени видеть, как в файл добавляется информация.

## more

Программа предназначена для просмотра текстовых файлов.

`more [параметры]... [файл]`

Пример:

```
more /usr/share/doc/Linux-FAQ/FTP-FAQ
```

`More`, в отличие от `cat`, выводит содержимое файлов постранично. Фактически, `more`, является оболочкой для просмотра текстовых файлов. Управление выводом весьма ограничено, в связи с чем в Linux обычно

пользуются программой `less`, имеющей более удобным интерфейсом. Однако у программы `more` есть неоспоримое преимущество — она описана в стандарте POSIX и существует по умолчанию во всех unix-подобных системах, в отличие от `less` или `most`.

Команды:

- Enter — сдвиг на одну строку вниз;
- пробел — сдвиг на одну страницу вниз;
- b — сдвиг на одну страницу вверх;
- /слово — поиск слова вниз по тексту;
- n — продолжить поиск вниз по тексту.

В некоторых реализациях `more` существует более удобное перемещение вверх и вниз по тексту. FreeBSD - с помощью клавиш управления курсором (вверх и вниз). HP-UX — с помощью клавиш j и k.

## less

Программа предназначена для просмотра текстовых файлов.

`less [параметры]... [файл]`

Less, так же как и `more`, выводит содержимое файлов постранично. Для того, что бы посмотреть содержимое файла, его имя передают в качестве аргумента при запуске программы. Например:

```
less /usr/share/doc/Linux-FAQ/FTP-FAQ
```

В `less` работают клавиши управления курсором (вверх и вниз).

У `less` есть встроенные команды, которые можно использовать во время работы:

- q — выход из программы.
- h — вывести страницу с информацией о командах.
- /слово — поиск слова в тексте.
- ?слово — поиск слова вверх по тексту.
- n — продолжить поиск слова.
- N — продолжить поиск слова в обратном направлении.

В некоторых дистрибутивах поставляется еще одна программа — `most`. Она аналогична `less`, но имеет расширенные возможности, в том числе поддержку цвета при выводе.

## Глава 3. Знакомство с командным интерпретатором

### Неадекватное поведение терминала

Когда я писал этот раздел, мне сразу вспомнился «Ералаш» в котором мальчик принес свою собаку на прием к ветеринару.

— Доктор. Моя собака неадекватно реагирует на мои команды...

При использовании `cat` и некоторых других программ терминал, на котором Вы работаете, может начать вести себя неадекватно. У него вместо букв будут отображаться символы псевдографики. Перестанут работать некоторые клавиши. Обычно такие изменения возникают при попытке посмотреть содержимое бинарного файла. Все дело в том, что у терминалов есть встроенные команды. Они начинаются с символа `escape`, после которого идет сама команда. Команды передаются в том же потоке данных, что и символы, которые выводит терминал. При просмотре файлов их содержимое просто копируется на терминал. Если это текстовый файл проблем не будет, так как данные представляют собой набор печатаемых символов, которые терминал выводит на свой экран. Если это бинарный файл, то его содержимое не ограничено только печатаемыми символами, в нем возможны любые данные от `0x00` до `0xff` (таким образом обозначают шестнадцатеричные цифры), в том числе и символ `escape`. То есть существует вероятность, что будет выполнена команда смены шрифта или переназначения функции клавиши.

Если Ваш терминал начинает работать неправильно, в командной строке необходимо набрать команду `reset` и нажать `Enter`. Поскольку терминал не корректно отображает символы, `reset` придется набирать «на ощупь». Если была переопределена клавиша `Enter`, воспользуйтесь комбинацией клавиш `Ctrl+J` — это то же самое, что и `Enter`.

Команда `reset` заставляет систему послать соответствующую команду терминалу. После этого он восстанавливает свои настройки по умолчанию, в том числе шрифт и функции клавиш.

### Комфортная работа в командной строке

Командная строка Linux может показаться очень неудобной, особенно когда Вы все время работали в графическом интерфейсе Windows. Но, если использовать все возможности, предоставляемые командной строкой, то работа в ней будет очень быстрой и комфортной.

### Автоподстановка

Одной из таких расширенных возможностей является автоматическая подстановка в командной строке. Например, необходимо ввести следующую команду (*Приведенный пример работает в Slackware Linux. В SuSE Linux можно использовать путь `/usr/share/doc/howto/en/txt/FTP.gz`.*):

```
cat /usr/share/doc/Linux-FAQs/FTP-FAQ
```

Путь к файлу FTP-FAQ достаточно длинный и для его набора требуется время. Кроме того, при наборе такого пути легко ошибиться. Тут Вам поможет автоподстановка. Для того, что бы оболочка автоматически продолжила путь следует использовать символ табуляции.

Итак, что бы правильно и быстро ввести путь к файлу последовательность действий будет такой:

В командной строке сначала необходимо набрать команду и начало пути:

```
$ cat /u
```

и нажать символ табуляции. В результате оболочка автоматически продолжит путь. На экране появится продолжение пути:

```
$ cat /usr/
```

Что произошло? Нажав на табуляцию, мы сказали оболочке: в корне файловой системы / нас интересует файл, начинающийся с символа `u`. Если можешь, продолжи строку. Поскольку в корне файловой системы существует только один файл с именем, начинающимся на `u` — директория `/usr`, оболочка дописала имя и поскольку это директория, дописала слеш в конце имени.

Дальше от нас требуется ввести продолжение пути.

```
$ cat /usr/s
```

При первом нажатии на символ табуляции ничего не произошло. Это значит, что в этой директории нет файлов



начинающихся на `s`, либо их несколько. Поэтому необходимо второй раз нажать на табуляцию. Если файлов нет — тогда Вы ничего не увидите, если есть — то будет показан список возможных файлов:

```
$ cat /usr/s
sbin/  share/ spool/ src/
```

Придется ввести еще один символ `h` и нажать на табуляцию. В результате получим:

```
$ cat /usr/share/
```

Продолжим ввод пути. Допишем сразу два символа `do` и нажмем на табуляцию.

```
$ cat /usr/share/doc/
```

Наберем `Li` и нажмем табуляцию.

```
$ cat /usr/share/doc/Linux-
```

Дальше `F` — табуляция.

```
$ cat /usr/share/doc/Linux-FAQ/
```

Вводим `FTP`, табуляция и получаем то, что нам необходимо.

```
$ cat /usr/share/doc/Linux-FAQ/FTP-FAQ
```

Первое время достаточно тяжело заставить себя использовать табуляцию. Зато со временем, благодаря табуляции, скорость набора в командной строке значительно возрастет. И Вы не будете допускать ошибки при наборе. Затем Вы увидите, что Вам для работы не требуется графическая оболочка. А потом не будете представлять, как это я раньше обходился без командной строки.

## История команд

Вторая особенность, облегчающая работу в командной строке — это история команд.

Для просмотра истории команд можно пользоваться клавишами стрелка вверх и стрелка вниз. Или встроенной в shell командой `history`.

Если Вы при помощи клавиш управления курсором выбрали какую-нибудь команду из истории, эту команду сначала можно отредактировать и только потом нажать на `Enter`.

`History` выводит пронумерованный список команд из истории. Для того, что бы повторно запустить программу, в командной строке следует ввести символ восклицательный знак и номер команды, затем нажать на `Enter`. Например:

```
$ history
 1  exit
 2  su -
 3  echo $LS_COLORS
 4  less /etc/inittab
 5  exit
 6  echo $SHELL
 7  ps
 8  pwd
 9  whereis pwd
10  ls -a
11  ls -l /etc
12  ls -l /etc/hosts
13  ls -l /etc/ssh /etc/ssl
14  ls /etc/ssh /etc/ssl
15  echo $LS_COLOR
16  echo $LS_COLORS
17  echo $LS_OPTIONS
18  ls -i
19  ls /var/
20  ls /opt/
21  ls /mnt/
22  cd /mnt/
23  ls -i
24  cd -
```

```
25 touch -c test
26 ls -l
27 touch test
28 ls -l
29 cat /etc/hosts
30 cat -E /etc/hosts
31 history
$
```

Чтобы выполнить команду под номером 28, сначала вводим восклицательный знак и без пробелов номер команды в истории.

```
$ !28
ls -l
total 0
-rw-r--r-- 1 stavr users 0 2008-09-04 21:50 test
$
```

**Внимание!** Обратите внимание на то, что в истории сохраняются вызовы программ со всеми параметрами командной строки. Никогда не передавайте пароли программам как аргумент командной строки. Это небезопасно!

История команд текущего сеанса хранится в оперативной памяти. По умолчанию в истории может храниться до 500 строк. После выхода из системы, история сохраняется в файле `.bash_history`, расположенном в домашней директории пользователя. Количество строк в файле ограничено 500 строками. Итого при работе история команд может достигать 500+500 — тысяча строк. Это очень много! Для ограничения количества команд в оперативной памяти используется переменная среды окружения `HISTSIZE`, в файле — `HISTFILESIZE`.

```
$ echo $HISTSIZE
500
$ echo $HISTFILESIZE
500
$
```

## Переменные среды окружения

Любая программа, работающая в Linux, имеет так называемое окружение — набор переменных. У каждой программы есть свой собственный набор переменных среды окружения. Программы имеют полный доступ только к своим переменным, доступ к переменным среды окружения другой программы запрещен.

У переменных есть имена. В имени можно использовать только английские буквы, символы `-` и `_`, цифры (но не в начале имени). Большие и маленькие буквы отличаются. Имена переменных среды окружения принято писать, используя большие буквы, но Вы можете не следовать этому правилу.

Переменные среды окружения — это еще один способ управления программами. Программы имеют доступ к переменным и могут использовать значения переменных в своей работе. Например, программам, работающим в графической оболочке в Linux, можно указать X-сервер, с помощью которого они будут выводить свои данные. Это можно сделать двумя способами:

- передать программе при запуске специальный параметр;
- определить переменную среды окружения `DISPLAY`.

Во втором случае не требуется каждой программе при ее запуске передавать параметр. Все они будут использовать содержимое переменной `DISPLAY` и выводить данные с помощью X-сервера, указанного в этой переменной.

То, какие переменные среды окружения использует программа, зависит от ее автора.

Вы как пользователь системы можете определять, изменять и удалять переменные среды окружения.

Давайте рассмотрим пример работы с переменными. Что бы добавить новую переменную в окружение программы надо написать ее имя, затем без пробелов символ `=` и значение переменной.

```
$CAR=запорожец
$
```

Все переменные среды окружения не имеют типа, и считается, что они содержат строку. Если в строке

встречаются пробелы, значение переменной берут в двойные кавычки.

```
$ CAR="запорожец - это машина?"
$
```

Для того, что бы получить значение переменной, перед ее именем необходимо поставить символ \$. Вместо \$CAR shell подставит значение переменной CAR.

Используйте программу echo, если Вы хотите увидеть значение переменной. Echo выводит на стандартный вывод все, что Вы передадите ей в командной строке. Например, что бы посмотреть содержимое переменной CAR следует выполнить следующую команду:

```
$ echo $CAR
запорожец - это машина?
$
```

Содержимое переменной было выведено на экран в следующей строке.

В то же время:

```
$ echo CAR
CAR
$
```

Предположим, что мы присвоили переменной CAR значение bmw. И запустили еще одну оболочку — sh.

```
$ CAR=bmw
$ sh
$
```

Внешний вид командной строки не изменится, но Вы будете работать совсем в другой программе, имеющей свой собственный набор переменных среды окружения. И если теперь посмотреть содержимое переменной, CAR на экран ничего выведено не будет, так как в окружении программы sh переменной CAR не существует.

```
$ echo $CAR

$
```

В Linux есть возможность заранее определить переменные, которые будут помещены в окружение программы при ее запуске. Ниже приведен пример того, что необходимо сделать, что бы переменная CAR была доступна программе sh.

```
$ CAR=bmw
$ export CAR
$ sh
$ echo $CAR
bmw
$
```

Переменную нужно экспортировать при помощи оператора export. Только экспортированные переменные будут видны в программах-потомках. То есть во всех программах, которые будут запущены из программы, где переменная была определена. Эта переменная будет автоматически определяться только у программ потомков и только у них. На остальные программы, работающие в системе, операция экспорта не повлияет.

После входа пользователя в систему у пользователя существует большое количество заранее определенных переменных среды окружения. Их список можно посмотреть при помощи команды set. Некоторые из них определяются самой оболочкой, а некоторые определены в специальных конфигурационных файлах. Вы можете самостоятельно определять новые переменные или изменять старые.

Существует набор файлов, которые выполняются при входе пользователя в систему. Именно в этих файлах можно осуществить добавление или переопределение переменных среды окружения. Если говорить о конкретном пользователе, то используют файл ~/.bash\_profile.

Например, необходимо в переменную PATH добавить директорию ~/bin. Для этого можно отредактировать файл .bash\_profile и добавить в него следующую строку:

```
export PATH=$PATH:~/bin
```

Для удаления переменной используется команда unset. Например:

```
$ unset CAR
$ echo $CAR

$
```

## Псевдонимы

Если при работе в командной строке Linux Вы часто используете определенные программы с одними и теми же аргументами командной строки, пора задуматься об использовании псевдонимов.

Для работы с псевдонимами используются команды (*Это действительно не программы, а команды, встроенные в оболочку.*) `alias` и `unalias`.

- `alias` — показывает список псевдонимов, создает новые псевдонимы.
- `unalias` — удаляет псевдоним.

Например, Вы часто используете программу `ls` со следующими параметрами:

```
ls -al --color=no
```

Каждый раз набирать все аргументы командной строки долго. Поэтому можно определить псевдоним. Он определяется следующим образом:

```
$ alias ll='ls -l --color=no'
$ alias
alias d='dir'
alias dir='/bin/ls $LS_OPTIONS --format=vertical'
alias ll='ls -al --color=no'
alias ls='/bin/ls $LS_OPTIONS'
alias mc='. /usr/share/mc/bin/mc-wrapper.sh'
alias v='vdir'
alias vdir='/bin/ls $LS_OPTIONS --format=long'
$
```

В приведенном примере определяется псевдоним с именем `ll`. Теперь, если в командной строке набрать `ll` и нажать Enter, будет выполнена программа `ls` со всеми перечисленными параметрами.

Как работают псевдонимы? Дело в том, что оболочка после нажатия пользователем на Enter считывает строку, но не сразу выполняет написанную в ней команду. В первую очередь она пытается обработать специальные символы и псевдонимы. То есть она сначала ищет `ll` в списке псевдонимов и если находит, запускает на выполнение то, что определено в псевдониме. Если не находит, то производит поиск программы `ll` в файловой системе.

Этой особенностью оболочки пользуются в некоторых дистрибутивах Linux, заранее определяя псевдонимы. Как уже говорилось выше, программы в Linux стараются не задавать лишних вопросов пользователю. Считается, что он знает, что делает. И если пользователь хочет удалить файл, значит он хорошо подумал перед вызовом программы `rm`. Работая в некоторых из дистрибутивов, Вы увидите, что при выполнении `rm`, программа начинает задавать вопросы. Если программы становятся разговорчивыми, значит хозяин дистрибутива заранее «позаботился» о Вас и определил псевдоним для программы `rm`. Например, так:

```
alias rm='/bin/rm -i'
```

То есть оболочка, сначала смотрит список псевдонимов, обнаруживает псевдоним с именем `rm` и выполняет `rm` с параметром `-i`. Поэтому программа начинает задавать вопросы.

Для удаления псевдонима используется команда `unalias`. Например, что бы удалить псевдоним `rm` следует выполнить команду:

```
$ unalias rm
$
```

Если Вы в командной строке определили псевдоним, потом вышли из системы и вернулись обратно, Вашего псевдонима в списке не будет. Чудес не бывает, все следует описывать в конфигурационных файлах. Для того чтобы псевдонимы оставались, их определение следует написать в файле, который выполняется при входе пользователя в систему. Какой файл использовать зависит от программы оболочки. При использовании `bash` псевдонимы записывают в файл `.bash_profile` в домашней директории пользователя. В этом же файле можно написать команду удаления определенного в системе псевдонима.

Старайтесь регулярно проверять список текущих псевдонимов, явно вызывая команду `alias` без аргументов.

Если в списке появляются нежелательные псевдонимы их лучше удалять. Псевдонимами могут пользоваться хакеры для сбора информации. Им достаточно поставить в систему свою программу (*На самом деле это не так просто сделать. Но вероятность взлома системы существует.*), выполняющую действия аналогичные стандартной программы и написать псевдоним. Например, самой желанной является информация о паролях. Для смены пароля пользователи системы применяют программу `passwd`. Если хакер просто заменит стандартную программу своей, администратор это почти сразу обнаружит. Поэтому можно поставить в систему свою программу, назвать ее `mypasswd` и прописать псевдоним:

```
$ alias passwd='mypasswd'
$
```

Таким образом, при вызове программы `passwd` на самом деле будет выполняться программа `mypasswd`. А это как раз и нужно хакеру. Поэтому — следите за списком псевдонимов.

## Стандартные ввод, вывод и вывод ошибки

Стандартный ввод, вывод и вывод ошибки — это одна из базовых технологий Linux, которая будет использоваться Вами повсеместно.

**Внимание!** Материал, который будет дан в этом разделе, обязателен к изучению и... пониманию. Если не понять концепцию стандартных ввода, вывода и вывода ошибки, которые используются почти везде, работать в системе будет сложно.

Для работы с файлами в Linux программисты могут использовать четыре базовых функции:

- `open` — открывает файл и возвращает индексный дескриптор открытого файла.
- `read` — читает данные из файла.
- `write` — записывает данные в файл.
- `close` — закрывает файл.

Когда функция `open` открывает файл, она возвращает индексный дескриптор — номер открытого файла. У каждой программы своя нумерация файлов, начинающаяся с нуля. При вызове функций `read`, `write` и `close` им в качестве параметра передается этот номер.

В Linux есть интересная особенность. У любой программы, работающей в системе, автоматически открываются три файла с индексными дескрипторами 0, 1 и 2. Этих файлов в файловой системе не существует, они являются виртуальными и не имеют реальных имен. Но у них есть названия:

- `stdin` (стандартный ввод) — файл с индексным дескриптором 0.
- `stdout` (стандартный вывод) — файл с индексным дескриптором 1.
- `stderr` (стандартный вывод ошибки) — файл с индексным дескриптором 2.

Эти файлы не надо открывать при помощи функции `open`, они уже итак открыты системой. И их не надо закрывать после использования, они виртуальные и будут автоматически закрыты после завершения выполнения программы.

В чем особенность этих файлов? Они связаны с терминалом на котором выполняется данная программа:

- Если записать данные в файл стандартного вывода, данные будут выведены на экран терминала.
- Если записать данные в файл стандартного вывода ошибки, они тоже будут выведены на экран терминала.
- Если читать из файла стандартного ввода, будет получено то, что пользователь вводил на клавиатуре.

Если необходимо вывести данные на экран терминала, данные достаточно записать в файл стандартного вывода. Например, программа `ls` записывает список файлов в файл стандартного вывода и поэтому мы видим его на экране. Или программа `cat`, она тоже записывает содержимое файла в стандартный вывод.

Если во время выполнения программы возникают ошибки, сообщения об ошибках записывают в файл стандартного вывода ошибки и мы их видим на экране терминала.

Таким образом, программистам, разрабатывающим программы, не приходится выдумывать специальные функции для общения с различными терминалами.

Какие возможности эти файлы дают пользователям системы? Вы можете перенаправлять эти файлы в реальные файлы файловой системы. То есть вместо того, что бы данные выводились на экран, они будут попадать в

указанный файл. Или наоборот, читать данные не с клавиатуры, а из указанного файла файловой системы.

## **Перенаправления. Перенаправление стандартного вывода**

Для перенаправления стандартного вывода программы при ее запуске используется символ `>`. Например, если необходимо, что бы список файлов, выводимый программой `ls`, попадал не на экран, а в файл с именем `list`, следует выполнить такую команду:

```
$ ls > list
```

Если файла `list` не было, он будет создан. Если файл `list` существовал, предыдущие данные будут потеряны и заменены списком файлов.

Для того чтобы добавить данные в конец файла используют два символа `>>`.

```
$ ls >> list
```

Стандартный вывод перенаправляется только у той программы, у которой при запуске использовались символы перенаправления. На другие программы, запускаемые позже, это не распространяется.

## **Перенаправление стандартного вывода ошибки**

Стандартный вывод ошибки — это файл с индексным дескриптором 2. Поэтому, для его перенаправления в файл используются те же самые символы, что и в случае перенаправления стандартного вывода, но с явным указанием номера дескриптора.

```
$ ls 2> file
```

```
$ ls 2>> file
```

В стандартный вывод ошибки программы записывают ошибки, возникающие при их выполнении. Предположим, что при вызове программы `cat` ей в качестве параметра был указан не существующий файл. Тогда на экране появится соответствующее сообщение об ошибке.

```
$ cat nofile
cat: nofile: No such file or directory
$
```

Если Вы хотите, что бы при работе программы сообщения об ошибке не выводились на экран терминала, перенаправьте стандартный вывод ошибки в файл. Как вариант, вместо имени файла можно использовать специальное устройство `/dev/null`.

```
$ cat nofile 2> /dev/null
$
```

Устройство `/dev/null` предназначено для потери информации, то есть любая информация, записываемая в этот файл, пропадает. Читать из этого файла не имеет смысла, т.к. сразу возвращается символ EOF.

Для того чтобы перенаправить стандартный вывод и вывод ошибки в один и тот же файл необходимо вывод ошибки перенаправить в файл с индексным дескриптором 1, а стандартный вывод в интересующий Вас файл. Если необходимо сделать так, что бы программа не выводила на экран терминала никакой информации, сделайте так как показано в следующем примере.

```
$ cat file1 file2 2>&1 > /dev/null
$
```

## **Перенаправление стандартного ввода**

Для перенаправления стандартного ввода программы используют специальный символ `<`. Например:

```
$ cat < file
```

Программа `cat` выводит на экран содержимое файла `file`. Пример не очень удачный, этого можно добиться, выполнив просто:

```
$ cat file
```

Но в первом примере программа не открывает файл `file`, это делает система и передает содержимое файла на стандартный ввод. Во втором примере, файл `file` открывает сама `cat`, явно используя функцию `open`.

## Конвейеры

Предположим, что нам необходимо выполнить две программы: `prog1` и `prog2`. Причем, то, что `prog1` выводит на стандартный вывод необходимо передать на стандартный ввод `prog2`. Для этого можно выполнить следующие команды:

```
$ prog1 > tmp
$ prog2 < tmp
$ rm tmp
$
```

То есть сначала сохраняем во временном файле стандартный вывод программы `prog1`. А при запуске `prog2` перенаправляем содержимое файла на ее стандартный ввод. Затем удаляем временный файл.

Для передачи данных со стандартного вывода одной программы на стандартный ввод другой программы в Linux предусмотрен механизм называемый конвейером команд. Для создания такого конвейера используется символ `|`.

```
prog1 | prog2
```

Рассмотрим простой пример.

```
$ ls -l
total 8
-rw-r--r-- 1 stavr users 110 2008-09-05 18:36 afile
-rw-r--r-- 1 stavr users  0 2008-09-04 21:50 test
-rw-r--r-- 1 stavr users 110 2008-09-05 18:34 testfile
$ ls -l | sort > sorted
$ cat sorted
-rw-r--r-- 1 stavr users  0 2008-09-04 21:50 test
-rw-r--r-- 1 stavr users  0 2008-09-05 18:47 sorted
-rw-r--r-- 1 stavr users 110 2008-09-05 18:34 testfile
-rw-r--r-- 1 stavr users 110 2008-09-05 18:36 afile
total 8
$
```

Результат работы программы `ls` (список файлов) передается по конвейеру на стандартный вход программы `sort`. При этом список файлов не попадает на экран терминала. Программа `sort` сортирует файл по строкам. Поскольку у `sort` перенаправлен стандартный вывод, информация на экран не попадает, а передается в файл `sorted`. То есть в результате выполнения этого конвейера команд, в файле `sorted` сохраняется отсортированный список файлов.

Длина конвейера команд ограничена только размером командной строки. То есть можно добавить к конвейеру третью, четвертую, пятую и т.д. программы.

На примере конвейеров команд хорошо виден так называемый путь UNIX (*UNIX way*). В UNIX редко встречаются универсальные программы, которые умеют делать все. Существует большое количество небольших программ. Каждая программа хорошо делает определенное действие: хорошо сортирует файлы, хорошо фильтрует данные и т.д. Все программы умеют работать со стандартными вводом и выводами. Объединяя эти программы в конвейер команд, в результате мы получаем обработку данных без написания новой программы. Таким образом, по-разному комбинируя программы в конвейере, можно получить различные результаты.

## Глава 4. Обработка текстовой информации

### Программы для управления выводом текстовой информации

Теперь, когда Вы знаете что такое конвейер команд, можно продолжить рассмотрение программ работающих с текстовыми файлами. Некоторые из них применяются только в конвейерах.

#### tee

Программа tee читает данные со стандартного ввода и без изменения копирует их на стандартный вывод. Копию проходящих данных сохраняет в указанном при ее запуске файле или файлах.

tee [параметры] [файл]...

Программа используется только в конвейерах.

Рассмотри пример применения программы tee.

```
$ ls -l | tee nosort | sort > sorted
$ cat nosort
total 8
-rw-r--r-- 1 stavr users 110 2008-09-05 18:36 afile
-rw-r--r-- 1 stavr users  0 2008-09-05 20:31 nosort
-rw-r--r-- 1 stavr users  0 2008-09-05 20:31 sorted
-rw-r--r-- 1 stavr users  0 2008-09-04 21:50 test
-rw-r--r-- 1 stavr users 110 2008-09-05 18:34 testfile
$ cat sorted
-rw-r--r-- 1 stavr users  0 2008-09-04 21:50 test
-rw-r--r-- 1 stavr users  0 2008-09-05 20:31 nosort
-rw-r--r-- 1 stavr users  0 2008-09-05 20:31 sorted
-rw-r--r-- 1 stavr users 110 2008-09-05 18:34 testfile
-rw-r--r-- 1 stavr users 110 2008-09-05 18:36 afile
total 8
$
```

Список файлов, выдаваемый программой ls, передается по конвейеру программе tee. Tee сохраняет этот список в файле nosort и передает его дальше программе sort. Sort сортирует список файлов. В конце строки стандартный вывод последней программы в конвейере (sort) перенаправляется в файл sorted. Таким образом, после выполнения команды в файле nosort сохраняется не сортированный список файлов, а в файле sorted отсортированный.

Программа tee каждый раз создает по новой, указанный в ее командной строке файл, то есть данные теряются. Если Вы хотите, что бы данные добавлялись в конец файла, используйте параметр -a.

### Структурированные текстовые файлы

Прежде чем мы продолжим рассмотрение программ, необходимо разобраться, что такое структурированный файл и где его применяют.

UNIX создавался как система, предназначенная для обработки патентной документации. В те времена, когда разрабатывался UNIX, не было таких БД как Oracle или DB2. Базы данных представляли собой набор текстовых файлов. Каждый файл — это таблица базы. Строка в файле — это запись в таблице. Поля в записи (строке) разделялись каким либо символом: табуляцией, двоеточием или любым другим. То есть файл имел структуру.

Для работы со структурированными текстовыми файлами был разработан специальный набор программ. Каждая программа, как это принято в UNIX, делала только одно действие и делала его хорошо. Все программы работали со стандартными вводом и выводом. Для обработки файлов достаточно было собрать программы в конвейер в необходимой последовательности и в результате работы получали выборку данных, внесение изменений или другое действие.

Набор программ оказался настолько удачно написан, что он до сих пор присутствует в UNIX, и, следовательно, в Linux. Эти программы также присутствуют в списке набора обязательных утилит стандарта POSIX.



## cut

Программа `cut` выбирает столбцы из каждой строки структурированных текстовых файлов.

`cut [параметры] [файл]...`

Если при вызове программы не указывается файл, `cut` берет данные со стандартного ввода.

Программа может выбирать информацию, как по символам, так и по столбцам. В первом случае следует использовать параметр `-с`, во втором параметр `-f`.

По умолчанию, символом разделителем полей считается символ табуляции. Если в файле использовался другой символ разделителя, при вызове `cut` необходимо явно передавать параметр `-d` с указанием символа разделителя в файле. Например, нам необходимо получить только первые десять символов в каждой строке выдаваемой программой `ls` с опцией `-l` (права доступа к файлам). Тогда выполняемая команда будет выглядеть следующим образом:

```
$ ls -l | cut -c 1-10
total 16
-rw-r--r--
-rw-r--r--
-rw-r--r--
-rw-r--r--
-rw-r--r--
$
```

1-10 — это диапазон символов, которые необходимо оставить в каждой строке. Можно использовать различные варианты записи диапазона. Например:

```
$ ls -l | cut -c -15
total 16
-rw-r--r-- 1 st
-rw-r--r-- 1 st
-rw-r--r-- 1 st
-rw-r--r-- 1 st
-rw-r--r-- 1 st
$
```

Если не указана начальная позиция, значит, данные берутся с начала строки. Если не указать последнюю позицию, то до конца строки:

```
$ ls -l | cut -c 15-
tavr users 110 2008-09-05 18:36 afile
tavr users 272 2008-09-05 20:31 nosort
tavr users 272 2008-09-05 20:31 sorted
tavr users  0 2008-09-04 21:50 test
tavr users 110 2008-09-05 18:34 testfile
$
```

В предыдущих примерах `cut` брала данные со стандартного ввода, но в качестве источника данных можно так же использовать файл. Например, необходимо выбрать из файла `/etc/passwd` информацию о том, какая программа у пользователей будет запускаться при его входе в систему.

Логин пользователя — первое поле, программа — седьмое. Символ разделитель полей в файле `passwd` — двоеточие. Чтобы ограничить количество строк при выводе, посмотрим данные только по последним четырем пользователям.

```
$ cut -f 1,7 -d: /etc/passwd | tail -4
haldaemon:/bin/false
pop:/bin/false
nobody:/bin/false
stavr:/bin/bash
$
```

Как видно из примера, для того, чтобы `cut` обрабатывала поля, ей был передан параметр `-f` с указанием номеров полей. Мы не использовали диапазон, как это было в примере с параметром `-с`. Если необходимо указать конкретные поля или символы, их следует перечислить через запятую.

Поскольку в файле символ разделитель полей — это двоеточие. При помощи параметра `-d` мы явно определили его при запуске программы.

Если в качестве символа разделителя необходимо использовать пробел, тогда параметр `-d` будет выглядеть следующим образом: `-d ' '`.

## paste

Программа `paste` объединяет файлы по строкам.

`paste [параметры] [файл]...`

При объединении файлов, программа `paste` разделяет данные в строке символом табуляции. Для того, что бы использовать другой символ разделитель существует параметр `-d`.

Например, существует файл с именем `test` содержащий следующие строки:

*Линия 1: один*

*Линия 2: два*

*Линия 3: три*

Предположим, что в этом файле необходимо поменять местами столбцы и в результате получить следующие строки:

*один:Линия 1*

*два:Линия 2*

*три:Линия 3*

Для этого придется выполнить несколько команд.

```
$ cut -f1 -d: test > tmp1
$ cut -f2 -d: test > tmp2
$ paste -d: tmp2 tmp1 > test
$ rm tmp*
$
```

Первая команда выбирает первое поле из файла `test` и помещает его во временный файл `tmp1`. Вторая команда выбирает второе поле и помещает его во временный файл `tmp2`. Затем командой `paste` эти файлы объединятся, разделяя строки символом `:`. Результат сохраняется в исходном файле `test`. Временные файлы удаляются.

## sort

Программа `sort` сортирует файлы по строкам.

`sort [параметры] [файл]...`

`Sort` при сортировке учитывает структуру файла. Если при вызове программы не указать ни одного параметра, строки сортируются сначала по первому полю. Если в нескольких строках содержимое первого поля одинаково, происходит сортировка этих строк по второму полю. Если второе поле одинаково — по третьему и т.д.

**Внимание!** По умолчанию в качестве символов разделителей полей используются пробелы и табуляции. Поля, в отличие от программы `cut`, нумеруются с нуля.

У программы много различных параметров, влияющих на ее работу. Ниже приведены только некоторые из них.

- `-n` — сортировать цифры как цифры, а не как символы. Если не указывать этот параметр, `sort` рассматривает цифры как символы, при этом может получиться, что 20 будет больше чем 100.
- `-r` — обратный порядок сортировки.
- `-b` — игнорировать начальные пробелы в строке.

При вызове программы можно указать поля, по которым будет происходить сортировка. Это делается при помощи символов `+`, `-` и следующей за ними цифры. Например, чтобы отсортировать вывод программы `ls` по четвертому полю, необходимо выполнить следующую команду, которая сортирует поля, начиная с четвертого поля, заканчивая пятым, не включая последнего. То есть сортировка происходит только по четвертому полю.

```
$ ls -l | sort +4 -5
total 16
-rw-r--r-- 1 stavr users 0 2008-09-04 21:50 test
-rw-r--r-- 1 stavr users 110 2008-09-05 18:34 testfile
-rw-r--r-- 1 stavr users 110 2008-09-05 18:36 afile
-rw-r--r-- 1 stavr users 272 2008-09-05 20:31 nosort
-rw-r--r-- 1 stavr users 272 2008-09-05 20:31 sorted
$
```

Четвертое поле — это размер файла.

## wc

Программа `wc` выдает количество строк, слов и символов.

`wc [параметры]... [файл]...`

Параметры программы `wc`:

- `-l` — показать количество строк.
- `-w` — показать количество слов.
- `-c` — показать количество символов. При подсчете учитываются все символы, в том числе пунктуации и пробелы.

Несмотря на то, что программа `wc` кажется бесполезной, ее очень часто применяют для подсчета строк в программах, написанных на языке встроенном в оболочку. Например, необходимо посчитать количество файлов в текущей директории. Это можно сделать следующим образом:

```
$ ls | wc -l
5
$
```

Программа `ls` видит, что результаты ее работы передаются по конвейеру команд, поэтому файлы выдаются по одному на строку. Посчитав количество строк, мы узнаем количество файлов.

Тоже самое можно сделать и с пользователями, работающими в системе. `Who` показывает, кто сейчас работает в системе — один пользователь одна строка. Посчитав количество строк, получаем количество пользователей.

```
$ who | wc -l
1
$
```

## tr

Программа `tr` заменяет или удаляет символы.

`tr [параметры] набор1 [набор2]`

Программу очень удобно использовать, когда требуется заменить одни символы другими. Например, символ табуляции заменить пробелом:

```
$ cat test
Next symbol is TAB      end.
$ cat test | tr "\t" " "
Next symbol is TAB end.
$
```

Можно определить несколько символов для замены. В следующем примере все символы `a` будут заменены символом `V`, а `d` — на `F`:

```
$ cat test
Symbols abcd.
$ cat test | tr "ad" "VF"
Symbols VbcF.
$
```

Обращу ваше внимание на тот факт, что преобразование происходит только с информацией, попадающей на

стандартный вывод. Оригинальный файл остается неизменным.

При помощи `tr` легко написать программу-перекодировщик из KOI8-R в Win-1251. Просто в первом наборе символов перечисляются все буквы русского алфавита в кодировке KOI8-R, а во втором наборе символов — все русские буквы, но уже в кодировке Win-1251.

## diff

Программа предназначена для сравнения файлов и каталогов.

`diff file1 file2`

Например, создадим два файла и сравним:

```
$ cat > wares.txt
table:34
car:24
apple:23
car:12
<Ctrl>+D
$ cat > wares2.txt
table:34
car:24
pen:32
apple:23
car:12
<Ctrl>+D
$ diff wares.txt wares2.txt
2a3
> pen:32
$
```

Мы получили результат сравнения, что во втором файле присутствует строка, отсутствующая в первом файле.

С помощью программы `diff` очень удобно рассмотреть применение патчей, например, к текстовым исходникам программ:

```
$ diff wares.txt wares2.txt > patch.txt
$ patch wares.txt patch.txt
patching file wares.txt
$ cat wares.txt
table:34
car:24
pen:32
apple:23
car:12
$
```

## Обработка текстовой информации

Тему обработки текстовых файлов надо понимать как редактирование не самого файла, а информации выводимой на стандартный вывод. И одной из форм редактирования вывода является поиск в текстовом файле соответствия заданному шаблону и вывод на экран результатов поиска. Поиск осуществляется с помощью программы `grep`.

## grep

Программа производит поиск фрагмента текста (шаблона) в файле.

`grep [опции] [-E | -f файл] шаблон [файл...]`

Параметры программы `grep`:

- `-i` — игнорировать регистр букв.
- `-s` — не выводить ошибки.

- `-r` — рекурсивный поиск.
- `-n` — выводить номер найденных строк.
- `-v` — выводить строки, не совпадающие с шаблоном.

При поиске есть возможность использования регулярных выражений, то есть использовать более слабые или более жесткие условия поиска.

### Примеры регулярных выражений

- `*` — любое количество любых символов, в том числе и их отсутствие.
- `.` — один символ.
- `^` — шаблон должен находиться в начале строки.
- `$` — шаблон должен находиться в конце строки.
- `[]` — перечисление символов, которые должны находиться в данной позиции.
- `[^]` — перечисление символов, которые не должны находиться в данной позиции.

### Примеры использования `grep`

```
grep -rsni pppd /usr/share/doc | less
grep -rsni pppd$ /usr/share/doc | less
grep -rsni "^..You" /usr/share/doc | less
```

## sed

И если `grep` позволяет только выбирать строки, содержащие искомый шаблон, то `sed` может еще и изменять найденную информацию в удобный для вас вид. Итак... `sed`.

Потоковый редактор `sed` предназначен для обработки текстовых файлов и информации поступающей на стандартный ввод, согласно команд, передаваемых в командной строке или во внешнем файле.

Еще раз обращаю ваше внимание на то, что `sed` редактирует информацию, в конечном итоге поступающую на терминал, а оригинальная информация (файл) остается неизменной. `Sed` еще называют однократным потоковым редактором, потому что он обрабатывает поток данных поступающий на стандартный ввод.

`sed [ адрес1 [,адрес2] | /regex/ ] команда [ опции команды ]`

Параметры программы `sed`:

- `-e` — определяет команду редактора.
- `-f` файл — определяет файл, в котором находятся команды редактора.

### Команда `a`

Команда `a` добавляет новую строку после указанной строки. Формат команды предусматривает, что добавляемая строка должна быть описана на новой строке.

Пример:

```
$ who | sed '/root/a\SUPER USER'
root      pts/0          Nov 24 10:34 (c1.unix.class)
SUPER USER
stavr     pts/1              Nov 24 10:46 (c1.unix.class)
$
```

### Команда `i`

Команда `i` ведет себя аналогично команде `a` с той лишь разницей, что вставка новой строки происходит перед найденным шаблоном, а не после.

## Команда *r*

Команда *r* предназначена для вывода на экран текста без изменений

Пример:

```
$ sed -n '1p' /etc/passwd
root:x:0:0::/root:/bin/bash
$
```

Была выведена первая строка файла `/etc/passwd`

## Команды *s* и *g*

*s*/что ищем/на что меняем/[*g*]

При помощи команды *s* можно произвести замену в указанных строках. По умолчанию заменяется только первое найденное совпадение в строке. Для замены всех искомых слов в строке необходимо использовать опцию *g*.

Пример:

```
$ sed 's/root/SUPERUSER/' /etc/passwd
SUPERUSER:x:0:0::/root:/bin/bash
...
operator:x:11:0:operator:/SUPERUSER:/bin/bash
...
$
$ sed 's/root/SUPERUSER/g' /etc/passwd
SUPERUSER:x:0:0::/SUPERUSER:/bin/bash
...
$
```

## Команда *q*

Команда *q* заставляет редактор *sed* завершить обработку информации

```
$ sed -e '4q' test
```

## Команда *d*

Команда *d* удаляет найденную строку.

Пример:

```
$ mcedit test
192.168.0.1
192.168.0.21
# not kill
# 192.168.12.114
# 192.168.12.254

$ sed -e '/#/ d' test
192.168.0.1
192.168.0.21
$
```

## Использование командного файла

```
$ mcedit sedcom
```

```
s/new/old/g
s/This/That/
```

```
$ echo "This is a new day in a new life" | sed -f sedcom
That is a old day in a old life
$
```

## Способы адресации

Адресация с помощью шаблона:

```
$ mcedit test
any big
other thin
me small
any big
other thin
me small
```

```
$ mcedit sedcom
/any/ s/big/biggest/
/me/ s/small/best boy/
```

```
$ sed -f sedcom test
any biggest
other thin
me best boy
any biggest
other thin
me best boy
$
```

Адресация указанием номеров строк:

```
$ mcedit sedcom
1,3 s/big/biggest/
4,6 s/small/best boy/
```

```
$ sed -f sedcom test
any biggest
other thin
me small
any big
other thin
me best boy
$
```

## Глава 5. Ссылки

В файловой системе Linux предусмотрена возможность создания и использования ссылок на объекты файловой системы. Существуют два типа ссылок:

- Жесткие (hard link).
- Символьные (symbolic link).

С жесткими ссылками мы уже встречались, когда рассматривали физическое устройство файловой системы. Жесткие ссылки — это еще одно имя файла.

У жестких ссылок есть два серьезных ограничения:

- Нельзя создавать ссылку на директорию (*точнее говоря, это может сделать только суперпользователь*).
- Ссылка работает только в пределах одной физической файловой системы.

Действительно жесткая ссылка — это дополнительное имя файла, ссылающееся на номер inode файла. Этот номер уникален в пределах одной физической файловой системы. Именно по этому ссылка работает только в пределах физической файловой системы.

Второй тип ссылок — символьные. Они лишены всех недостатков жестких ссылок. Могут ссылаться на любые объекты файловой системы: файлы, директории, символьные ссылки и т.д. Они не ограничены пределами физической файловой системы.

Символьная ссылка — это специальный тип файла, очень похожий на ярлыки Windows, но имеющий больше возможностей. В файле символьной ссылки хранится путь к объекту файловой системы, на которую она ссылается. При использовании ссылки в указании пути к файлу, драйвер файловой системы пересчитывает путь с учетом того пути, что хранится в ссылке. Если символьная ссылка переносится, переименовается или удаляется при помощи программ mv, cp и rm, эти действия применяются к файлу ссылки, а не к файлу на который она ссылается.

При создании символьной ссылки, в нее можно записать как абсолютный, так и относительный путь к объекту, на который она ссылается. Если был записан абсолютный путь, например: /путь/к/файлу, то такую символьную ссылку можно переносить в любое место файловой системы, и она всегда будет правильно ссылаться на указанный объект. Если же написать относительный путь, например: путь/к/файлу, то она будет ссылаться на объект, относительно той директории, в которой расположена ссылка.

### ln

Программа ln предназначена для создания жестких и символьных ссылок.

ln [-s] файл ссылка

При вызове программы необходимо указать файл, на который будет создаваться ссылка и имя самой ссылки. По умолчанию создаются жесткие ссылки. Для создания символьной ссылки необходимо указать параметр -s.

Рассмотрим пример работы со ссылками.

```
$ cat > test
The test file
<Ctrl>+D
$ ls -l
total 48
.....
-rw-r--r-- 1 stavr users 14 2008-09-05 22:44 test
.....
$ ln test hardlink
$ ln -s test slink
$ ls -il
total 52
.....
144 -rw-r--r-- 2 stavr users 14 2008-09-05 22:44 hardlink
145 lrwxrwxrwx 1 stavr users 4 2008-09-05 22:45 slink -> test
144 -rw-r--r-- 2 stavr users 14 2008-09-05 22:44 test
.....
```



```
$
```

При помощи программы `cat` создается файл `test`. (После ввода информации не забудьте нажать комбинацию клавиш «`Ctrl+D`».) Затем создаются: жесткая ссылка `hardlink` и символическая ссылка `slink`. Имена ссылок могут быть любыми.

При вызове программы `ls` был указан параметр `-i` — выводить номер `inode` файла. Обратите внимание на то, что у файлов `test` и `hardlink` абсолютно одинаковые параметры, включая одни и те же номера `inode`. Физически это одни и те же данные, но имеющие несколько имен. А вот файл `slink` — это совсем другой файл, об этом говорит уникальный номер его `inode`. Программа `ls` запускается по умолчанию с параметром `-F` и поэтому после имени символической ссылки показывает файл, на который она ссылается.

Обратите внимание на третье поле при выводе данных программой `ls`, в нем находится число, обозначающее количество жестких ссылок на файл. У `hardlink` и `test` там стоит число 2. Подробную информацию о файле можно посмотреть при помощи программы `stat`:

```
$ stat hardlink
  File: `hardlink'
  Size: 14          Blocks: 8          IO Block: 4096   regular file
Device: 303h/771d  Inode: 144          Links: 2
Access: (0644/-rw-r--r--)  Uid: ( 1000/   stavr)   Gid: (  100/   users)
Access: 2008-09-05 22:30:15.013095297 +0500
Modify: 2008-09-05 22:44:42.267508176 +0500
Change: 2008-09-05 22:45:38.213352500 +0500

$ stat slink
  File: `slink' -> `test'
  Size: 4          Blocks: 0          IO Block: 4096   symbolic link
Device: 303h/771d  Inode: 145          Links: 1
Access: (0777/lrwxrwxrwx)  Uid: ( 1000/   stavr)   Gid: (  100/   users)
Access: 2008-09-05 22:45:58.824549978 +0500
Modify: 2008-09-05 22:45:50.756865275 +0500
Change: 2008-09-05 22:45:50.756865275 +0500
$
```

При просмотре файлов `hardlink` и `slink` будет показано содержимое файла, на который они ссылаются:

```
$ cat hardlink
The test file
$
$ cat slink
The test file
$
```

Теперь удалим файл `test` и посмотрим, что получится.

```
$ rm test
$ ls -l
total 48
.....
-rw-r--r-- 1 stavr users 14 2008-09-05 22:44 hardlink
lrwxrwxrwx 1 stavr users  4 2008-09-05 22:45 slink -> test
.....
$ cat hardlink
The test file
$
$ cat slink
cat: slink: No such file or directory
$
```

При выводе списка файлов программа `ls` покрасила файл `slink` в красный цвет. Таким образом, намекая нам, что файлу `slink` стыдно за то, что он ссылается на несуществующий файл. При обращении к `hardlink` мы получаем содержимое файла. Напомню, что файл (его данные) существует до тех пор, пока на него есть хотя бы одна жесткая ссылка. При обращении к файлу `slink` получили сообщение об ошибке.

Теперь создадим файл `test`.

```
$ touch test
$ ls -il
total 48
.....
144 -rw-r--r-- 1 stavr users 14 2008-09-05 22:44 hardlink
145 lrwxrwxrwx 1 stavr users 4 2008-09-05 22:45 slink -> test
156 -rw-r--r-- 1 stavr users 0 2008-09-05 22:58 test
.....
$ cat slink
$ cat hardlink
The test file
$
```

Как видно по выводу программы `ls`, файлы `hardlink` и `test` — это два разных файла. Об этом свидетельствуют разные номера `inode` и число `1` в поле, определяющем количество жестких ссылок. Да и размеры у этих файлов разные. Но после создания файла `test` ссылка `slink` снова выводится нормальным цветом, ведь появился файл, на который она ссылается. Символьной ссылке все равно, какой `inode` у этого файла, в ней записано имя файла, а не его номер. При просмотре содержимого `slink` на экран ничего не выводилось, ведь размер `test` равен нулю, то есть в нем нет никаких данных. А `hardlink` содержит старые данные.

Еще одна замечательная особенность символьных ссылок заключается в том, что их можно создавать на файлы, которые в данный момент не существуют. Ссылка будет создана, но не будет работать до тех пор, пока не появится файл, на который она ссылается. Так можно создавать ссылки на файлы в сетевых файловых системах или CD-ROM.

Символьные ссылки — это очень удобный инструмент, которым Вы будете пользоваться постоянно.

## Глава 6. Основы системы безопасности

Система безопасности Linux базируется на ограничении прав доступа к файлам. Когда пользователь заходит в систему, он представляется, говорит свой логин и пароль. После этого все программы пользователя выполняются от его имени. Когда программа пытается обратиться к какому-либо файлу, проверяется, а имеет ли пользователь, с правами которого работает программа, права на доступ к файлу. Если имеет, то операция разрешается, если нет — запрещается.

Если необходимо ограничить доступ пользователей к какому-либо устройству, ограничивают доступ к файлу соответствующего устройства. Ведь для того, чтобы работать, например, с CD-ROM, необходимо обратиться к файлу устройства CD-ROM. Если Вы, как администратор, запретили доступ к этому файлу, пользователь не может использовать CD-ROM.

### Права доступа

Права доступа в Linux реализованы очень просто. У каждого файла в системе существуют свои собственные права доступа.

**Внимание!** Права доступа не наследуются так, как это принято в Windows и Novell NetWare.

В Windows можно определить права доступа на директорию, и они автоматически распространяются на все файлы и поддиректории. В Linux права доступа сохраняются в inode файла, и поскольку inode у каждого файла свой собственный, права доступа у каждого файла свои.

Если посмотреть на вывод программы `ls -l`, в первом столбце показаны права доступа файла.

```
$ ls -l
total 48
-rw-r--r-- 1 stavr users 110 2008-09-05 18:36 afile
-rw-r--r-- 1 stavr users  14 2008-09-05 22:44 hardlink
-rw-r--r-- 1 stavr users 272 2008-09-05 20:31 nosort
-rw-r--r-- 1 stavr users  13 2008-09-05 21:25 patch.txt
-rw-r--r-- 1 stavr users  41 2008-09-05 22:29 sedcom
lrwxrwxrwx 1 stavr users   4 2008-09-05 22:45 slink -> test
-rw-r--r-- 1 stavr users 272 2008-09-05 20:31 sorted
.....
$
```

В первом поле десять символов. Первый символ — это тип файла, остальные девять показывают права. Права доступа делятся на три группы:

r w x | r w x | r w x

user | group | other

- **user** — права хозяина файла.
- **group** — права группы, которой принадлежит файл.
- **other** — права всех остальных пользователей системы.

При обращении программы к файлу сначала проверяется, является ли пользователь, с правами которого выполняется программа, хозяином файла? Если да, тогда на программу распространяются права хозяина файла, все остальные права игнорируются. Если нет, тогда проверяется, принадлежит ли файл группе, с правами которой выполняется программа? Если да, тогда применяются права доступа для группы, а все остальные права игнорируются. Если файл не принадлежит ни пользователю, ни группе, с правами которых выполняется программа, тогда применяются права для всех остальных.

**Внимание!** Права доступа пользователя и группы не суммируются. Если программа выполняется с правами пользователя и группы, которым принадлежит файл — работают только права хозяина файла.

### Права доступа к файлам

Права доступа к файлам и директориям имеют различное значение. Для файлов:

- **r** — право на чтение данных из файла.
- **w** — право на изменение содержимого файла (запись).
- **x** — право на исполнение файла.

Все права достаточно просты в понимании, единственно, на что следует обратить внимание — **w** не дает права на удаление файла, только на изменение содержимого.

Теперь о праве на исполнение. Это право можно установить для любого файла. Получается, что потенциально любой файл в системе можно исполнить? Это действительно так. В Linux является ли файл исполняемым или нет, определяется не по его расширению (*понятие расширения файла отсутствует в файловой системе Linux*), а по правам доступа. Если у файла установлено право **x**, его можно запустить на выполнение.

Что происходит, когда мы пытаемся выполнить файл? Мы набираем имя файла в командной строке и нажимаем Enter. В первую очередь проверяется, а имеет ли пользователь права на исполнение этого файла? Если имеет, тогда система смотрит, а это исполняемый бинарный файл? В Linux все исполняемые бинарные файлы в начале файла имеют заголовок ELF. Если это исполняемый бинарный файл, тогда, согласно его заголовку, происходит распределение оперативной памяти, и управление передается программе.

Если файл не бинарный, тогда считается, что это текстовый файл. В начале первой строки файла ищется последовательность символов `#!`. После этой последовательности, в той же строке, указывается программа, которую необходимо запустить и передать ей в командной строке текущий файл. Например, для файлов, написанных на языке shell-scripts, первая строка будет выглядеть так:

```
#!/bin/sh
```

Для программ, написанных на perl, так:

```
#!/bin/perl
```

Во всех интерпретируемых языках программирования `#` — это символ комментария. То есть первая строка считается комментарием и программой не выполняется. При указании интерпретатора можно писать аргументы командной строки. Например:

```
#!/bin/sed -f command
```

Если в файле в первой строке нет этих символов, тогда все зависит от программы оболочки, в которой запускается программа. Если используется `bash`, то он считает, что файл содержит программу, написанную на языке shell-scripts, запускает копию себя любимого и передает этой копии файл на интерпретацию. Если в файле действительно находится программа, то он ее выполняет. Если в файле находится «Война и мир» графа Льва Николаевича Толстого, то на экране появляются сообщения об ошибках shell-scripts: «Я не знаю оператор Пьер Безухов. Наташа Ростова — это оператор или функция?»

## Права доступа к директориям

Права доступа к директориям интерпретируются по-другому:

- **r** — право на чтение директории. Прочитать содержимое директории — получить список файлов.
- **w** — право на изменение содержимого директории — создание и удаление файлов в этой директории.
- **x** — право на «вхождение» в директорию.

**Внимание!** Обратите внимание на то, что если Вы имеете право на запись в директории — значит, Вы можете удалить **любые** файлы в этой директории, даже те, которые Вам не принадлежат. *Правда, существуют дополнительные права доступа, которые позволяют ограничить право на запись в директорию, но их мы рассмотрим чуть позднее.*

Теперь о праве **x** для директории. Это право позволяет Вам войти в директорию: `cd dir`.

**Внимание!** Право **x** на директорию среди других прав на доступ к файлу всегда проверяется в первую очередь!

Предположим, что другой пользователь системы (`user2`) написал программу `/home/user2/bin/program`. Права доступа у этой программы `г-хг-хг-х`. Он попросил Вас ее протестировать. При попытке запуска этой программы пользователем `user1` было получено сообщение: Доступ запрещен. Почему могла возникнуть такая ситуация? Дело в том, что при обращении к файлу система сначала проверяет право **x** у всех директорий, стоящих в пути этого файла, и только затем права на сам файл. Если хотя бы у одной директории право **x**

отсутствует, доступ к этой директории и всему ее содержимому для Вас запрещается. Например:

```
/      home/  user2/  bin/      program
r-x    r-x    r--    r-x      r-x
```

Как видно из примера, у директории /home/user2 отсутствует право на исполнение. Поэтому доступ к директориям /home/user2, /home/user2/bin и файлу program запрещен.

## Права доступа к символическим ссылкам

Если посмотреть на права символических ссылок, то они всегда выглядят так: *rwxxrwxrwx*. Дело в том, что права на символическую ссылку не имеют особого значения. При использовании ссылки драйвер файловой системы пересчитывает реальный путь к файлу и применяет права доступа, определенные для реального пути уже без учета символической ссылки.

## chmod

Программа **chmod** предназначена для изменения прав доступа к файлам.

chmod [параметры] права файл...

**Внимание!** В Linux нет права на изменение права. Поэтому хозяин файла всегда может изменять права доступа на свои и только свои файлы. Суперпользователь может менять права доступа у любых файлов в системе.

При вызове программы необходимо всегда указывать права доступа, устанавливаемые на файл. Права доступа можно написать, используя два формата записи: числовой и символический.

## Числовой формат записи прав доступа

Числовой формат записи наиболее распространен, но он труден для понимания тем, кто начинает работу с Linux. Вам обязательно придется его изучить, так как он используется для указания прав доступа в описаниях к программам в литературе по Linux.

Права доступа — это биты в inode файла. Биты делят на три группы, каждую группу представляют в виде десятичного числа.

Возьмем одну группу бит и запишем таблицу преобразования из двоичного в десятичный форматы:

r	w	x	
0	0	1	1
0	1	0	2
1	0	0	4

Теперь, чтобы определить права доступа, достаточно воспользоваться приведенной таблицей. Например, правам доступа *r--* соответствует число 4, правам *r-x* соответствует число 4+1=5 и т.д. Поскольку мы имеем три группы бит, в результате получается три числа. Например:

*rx-rw----* — 660

*rwxr-x--x* — 751

*rwxrwxrwx* — 777

В первом примере первая 6 — это права доступа хозяина файла, вторая 6 — права доступа группы, которой принадлежит файл, а 0 — это права доступа всех остальных пользователей системы.

При вызове программы **chmod** можно указывать не все три числа. Тогда числа начинают считаться справа. Например:

```
$ ls -l test
-rw-r--r-- 1 stavr users 0 2008-09-05 22:58 test
```

```
$ chmod 6 test
$ ls -l test
-----rw- 1 stavr users 0 2008-09-05 22:58 test
$
```

установит права gw- для всех остальных пользователей системы. А следующая программа:

```
$ chmod 66 test
$ ls -l test
----rw-rw- 1 stavr users 0 2008-09-05 22:58 test
$
```

установит права gw- для группы и для всех остальных пользователей системы.

## Символьный формат записи прав доступа

Символьный формат прав доступа можно описать при помощи следующего шаблона:

[ugoa][[+=[rwxst]][...]

- **u** — права пользователя.
- **g** — права группы.
- **o** — права всех остальных.
- **a** — права пользователя, группы и всех остальных.
- **+** — установить бит в единицу.
- **-** — установить бит в ноль.
- **=** — установить три бита.
- **r** — бит r.
- **w** — бит w.
- **x** — бит x.
- **s** — SUID или SGID бит (*о них будет рассказано ниже*).
- **t** — stiky бит (*об этом праве будет рассказано ниже*).

В отличие от числового формата записи прав, при помощи символьного можно устанавливать или сбрасывать только один бит, а не все биты. Например, права доступа у файла были следующие: ---gw-gw-. После выполнения программы:

```
$ ls -l test
----rw-rw- 1 stavr users 0 2008-09-05 22:58 test
$ chmod u+r test
$ ls -l test
-r--rw-rw- 1 stavr users 0 2008-09-05 22:58 test
$
```

Права стали такими: r--gw-gw-. Как видно, был изменен всего лишь один бит. Таким же образом можно сбрасывать биты в ноль:

```
$ chmod g-w test
$ ls -l test
-r--r--rw- 1 stavr users 0 2008-09-05 22:58 test
$
```

Можно одновременно изменять права для, например, хозяина и группы:

```
$ chmod ug+x test
$ ls -l test
-r-xr-xrw- 1 stavr users 0 2008-09-05 22:58 test*
$
```

При использовании символа = изменяются сразу три бита. После = описывают, какие биты должны быть установлены в единицу. Неуказанные биты будут сброшены в ноль:

```
$ chmod u=rw test
$ ls -l test
-rw-r-xrw- 1 stavr users 0 2008-09-05 22:58 test*
$
```

Если не удастся записать изменения прав при помощи одной записи, их можно написать несколько. В качестве символа разделителя используют запятую.

```
$ chmod o=rw,g=r test
$ ls -l test
-rw---x--- 1 stavr users 0 2008-09-05 22:58 test*
$
```

## Копирование прав доступа

В новой версии программы `chmod` появился параметр `--reference`. При помощи `reference` можно указать файл, права доступа которого будут копироваться на другие файлы.

```
$ ls -l
total 48
-rw---x--- 1 stavr users 0 2008-09-05 22:58 test*
-rw-r--r-- 1 stavr users 110 2008-09-05 18:34 testfile
.....
$ chmod --reference=testfile test
$ ls -l
total 48
-rw-r--r-- 1 stavr users 0 2008-09-05 22:58 test
-rw-r--r-- 1 stavr users 110 2008-09-05 18:34 testfile
.....
$
```

## Специальные права

Кроме базовых прав доступа в `inode` файла находятся еще три бита, описывающие специальные права доступа.

- **SUID** (Set UID) — программа, у которой установлен этот бит, выполняется с правами хозяина файла программы. Имеет смысл только при установке на исполняемые файлы.
- **SGID** (Set GID) — назначение бита аналогично SUID, только программа выполняется с правами группы, которой принадлежит файл. Кроме исполняемых файлов этот бит может устанавливаться на директории.
- **Stiky** — бит имеет смысл, только если он установлен на директорию. Означает, что в этой директории файл может удалить только его хозяин. *(Суперпользователь может удалять любые файлы, в том числе и не принадлежащие ему.)*

## suid

Рассмотрим пример применения бита SUID. В системе любой пользователь может поменять себе пароль. Новый пароль записывается в файл `/etc/shadow`:

```
$ ls -l /etc/shadow
-rw-r----- 1 root shadow 570 2008-08-15 17:02 /etc/shadow
$
```

Как видно из прав доступа файла `shadow`, право на запись имеет только суперпользователь `root` (*Наличие права на чтение для группы — это особенность дистрибутива Slackware Linux, в других дистрибутивах права доступа обычно устанавливаются в 600*). То есть другие пользователи системы не имеют права изменять содержимое этого файла, поэтому они не имеют права на изменение пароля. Но ведь каким-то образом им удастся изменить свой пароль!

Для изменения пароля используется программа `passwd`.

```
$ ls -l /usr/bin/passwd
-rws--x--x 1 root root 34844 2008-03-25 00:11 /usr/bin/passwd*
```

```
$
```

Обратите внимание на символ **s**, который стоит вместо **x** в правах хозяина файла. Наличие этого символа говорит о том, что у этого файла установлен бит SUID. А это значит, что программа будет выполняться с правами пользователя, которому принадлежит файл — с правами пользователя **root**! Root имеет право на запись в файл **shadow**, и именно поэтому обыкновенный пользователь может изменить свой пароль.

При выполнении программы **chmod** для установки бита SUID можно пользоваться как числовым, так и символьным форматом. Например:

```
$ ls -l test
-rw-r--r-- 1 stavr users 0 2008-09-05 22:58 test
$ chmod 4751 test
$ ls -l test
-rwsr-x--x 1 stavr users 0 2008-09-05 22:58 test*
$
```

**4** — это обозначение бита SUID. При установке SUID-бита с использованием символьного формата записи, строка будет выглядеть так:

```
$ chmod u+s test
$ ls -l test
-rwsr-x--x 1 stavr users 0 2008-09-05 22:58 test*
$
```

Для сброса специальных бит при числовом формате записи прав желательно явно указывать ноль в соответствующей позиции:

```
$ chmod 0751 test
$ ls -l test
-rwxr-x--x 1 stavr users 0 2008-09-05 22:58 test*
$
```

**Внимание!** SUID-бит — это очень опасный механизм. Вы как администратор системы должны постоянно контролировать наличие файлов с установленным SUID-битом. Это можно сделать при помощи программы поиска файлов в файловой системе **find**.

```
$ find /usr/bin -perm +4000
/usr/bin/lppasswd
/usr/bin/crontab
/usr/bin/fdmount
/usr/bin/chage
/usr/bin/chfn
/usr/bin/expiry
/usr/bin/newgrp
/usr/bin/gpasswd
/usr/bin/chsh
/usr/bin/passwd
/usr/bin/at
/usr/bin/sudo
/usr/bin/kcheckpass
/usr/bin/start_kdeinit
/usr/bin/kgrantpty
/usr/bin/fileshareset
/usr/bin/kpac_dhcp_helper
/usr/bin/kppp
/usr/bin/traceroute6
/usr/bin/rcp
/usr/bin/rsh
/usr/bin/rlogin
/usr/bin/procmail
/usr/bin/traceroute
/usr/bin/cu
/usr/bin/uux
/usr/bin/uucp
/usr/bin/uuname
```



```
/usr/bin/uustat
/usr/bin/Xorg
$
```

С программой `find` более подробно мы познакомимся позднее.

## sgid

Бит SGID на исполняемые файлы устанавливается очень редко. Его основное назначение — организация работы группы пользователей над файлами одного проекта.

Когда пользователь создает файл, этот файл принадлежит пользователю и его основной группе. Изменять хозяина файла и группу, которой он принадлежит, пользователь не имеет права. Теперь представьте, что есть некоторый проект, с файлами которого должны работать несколько пользователей системы. То есть они должны иметь полный доступ к этим файлам.

Предположим, что для размещения файлов проекта выделена отдельная директория, например, `/home/project`. Для пользователей, работающих над проектом, создана специальная группа `pr1`, в которую добавлены пользователи `stavr` и `user1`. На директорию установлены следующие права доступа:

```
$ ls -ld /home/project
drwxrwx--- 2 root pr1 6 2008-09-08 13:49 /home/project/
$
```

Поскольку пользователи `stavr` и `user1` входят в группу `pr1`, они имеют право на создание файлов в этой директории. Каждый пользователь создал в директории новый файл.

```
$ ls -l /home/project/
total 0
-rw-rw---- 1 stavr users 0 2008-09-08 13:57 file_stavr
-rw-rw---- 1 user1 user1 0 2008-09-08 14:01 file_user1
$
```

Как видно из прав доступа к файлам, `stavr` ничего не может сделать с файлом пользователя `user1` и наоборот. Они могут только удалить эти файлы, но не изменить их содержимое. Почему так получилось? Дело в том, что файл принадлежит пользователю, его создавшему, и основной группе пользователя. У `stavr` основная группа — `users`, у `user1` — `user1`. Поэтому эти файлы принадлежат именно этим группам.

Для решения проблемы необходимо сделать так, что бы все вновь создаваемые файлы принадлежали той группе, в которую входят оба пользователя. Это можно сделать, установив SGID-бит на директорию `/home/project`. Тогда все создаваемые в этой директории файлы будут принадлежать не основным группам пользователей, а группе, которой принадлежит директория `project`.

```
# chmod g+s /home/project
# ls -ld /home/project
drwxrws--- 2 root pr1 40 2008-09-08 14:01 /home/project/
# rm /home/project/*
#
```

Изменение прав доступа к директории производил суперпользователь `root`. У `root` символ приглашения командной строки обычно заканчивается на `#`.

Обратите внимание на символ `s`, который стоит вместо `x` в правах группы. Таким образом обозначается бит SGID.

Теперь пользователь `stavr` создаст в директории `project` свой файл:

```
$ touch /home/project/stavr_file
$ ls -l /home/project/
total 0
-rw-rw---- 1 stavr pr1 0 2008-09-08 14:08 stavr_file
$
```

Новый файл принадлежит пользователю `stavr` и группе `pr1`, а не группе `users`. Если в этой директории создаст файл пользователь `user1`, файл тоже будет принадлежать группе `pr1`.

```
$ touch /home/project/file_user1
```

```
$ ls -l /home/project
total 0
-rw-rw---- 1 user1 pr1 0 2008-09-08 14:11 file_user1
-rw-rw---- 1 stavr pr1 0 2008-09-08 14:08 stavr_file
$
```

Теперь оба пользователя могут работать с файлами этого проекта.

**Внимание!** При создании новой директории в директории с уже установленным SGID-битом, у созданной директории SGID-бит устанавливается автоматически!

```
$ mkdir /home/project/dir
$ ls -l /home/project/
total 0
drwxrws--- 2 user1 pr1 6 2008-09-08 14:14 dir/
-rw-rw---- 1 user1 pr1 0 2008-09-08 14:11 file_user1
-rw-rw---- 1 stavr pr1 0 2008-09-08 14:08 stavr_file
$
```

## stiky

Sticky-бит применяется только к директориям. Если такой бит установлен на директорию, значит, в этой директории файл может удалить только хозяин файла или суперпользователь. Sticky-бит рекомендуется устанавливать на публичные директории, такие как: /tmp и /var/tmp.

```
$ ls -ld /tmp/
drwxrwxrwt 5 root root 53 2008-09-05 21:25 /tmp//
$ ls -ld /var/tmp/
drwxrwxrwt 2 root root 6 2008-09-08 14:01 /var/tmp//
$
```

На эти директории установлены права 777, то есть кто угодно может создавать файлы и удалять их. Но обратите внимание на символ **t** в правах доступа для всех остальных пользователей системы. Он означает, что у директории установлен sticky-бит.

## Обозначение

Поскольку специальные права используются крайне редко, при выводе программы `ls` символ, их обозначающий, закрывает символ стандартных прав доступа. И не понятно, `rwt` — это `rw-` или `rwX`? Определить, стоит ли символ стандартных прав доступа под символами **s** и **t** очень просто. Если **t** маленькое, значит **x** установлен. Если **T** большое, значит **x** не установлен. То же самое правило распространяется и на **s**.

## Права доступа по умолчанию

При создании новых файлов и директорий устанавливаются права доступа по умолчанию. У директорий это 777, а у файлов — 666. Если сейчас пользователь создаст файл или директорию, то их права не будут равны правам доступа по умолчанию.

```
$ touch newtest
$ mkdir newdir
$ ls -l
total 48
drwxr-xr-x 2 stavr users 6 2008-09-08 14:23 newdir/
-rw-r--r-- 1 stavr users 0 2008-09-08 14:23 newtest
.....
```

У директории права доступа 755, а у файла — 644 (*в различных реализациях Linux права доступа могут отличаться*). Почему так получилось? Дело в том, что при создании файла учитывается так называемая пользовательская маска (`umask`).

Пользовательская маска — это число, которое отнимается от прав доступа по умолчанию. Получившиеся в результате вычитания права доступа применяются к создаваемому файлу.

Для просмотра текущего значения и изменения маски используется встроенная в оболочку команда `umask`.

```
umask [маска]
```

В следующем примере показано, как получить текущее значение маски и как установить маску 0077.

```
$ umask
0022
$ umask 0077
$ touch new_file
$ ls -l new_file
-rw----- 1 stavr users 0 2008-09-08 14:26 new_file
$
```

При создании нового файла была учтена маска. Результирующие права доступа стали равны  $666-0077=0600$ , то есть `rw-----`.

При создании новой директории права доступа получатся такими:  $777-0077=0700$ , то есть `gwx-----`.

```
$ mkdir new_dir
$ ls -ld new_dir
drwx----- 2 stavr users 6 2008-09-08 14:28 new_dir/
$
```

После выхода пользователя из системы и входа обратно будет использоваться значение маски по умолчанию. Для того, чтобы новое значение маски не пропадало, добавьте команду `umask 0077` в любую строку файла `.bash_profile` (в *SuSE Linux* надо использовать файл `.profile`), находящегося в домашней директории пользователя.

## POSIX ACL

Тем, кто перед Linux работал в Windows или Novell NetWare, может показаться, что права доступа в Linux предоставляют очень мало возможностей. На самом деле все как в старом анекдоте:

— Не люблю я кошек.

— Вы просто не умеете их готовить!

Стандартных прав доступа хватает для решения почти всех задач. Но, как мне кажется, специально для тех, кто пришел в Linux из мира Windows, была добавлена поддержка POSIX ACL (списки контроля доступа). Если раньше можно было определить только права хозяина файла, группы и всех остальных пользователей, то ACL позволяют указать права доступа конкретных пользователей и групп.

ACL в Linux появились недавно, и не все файловые системы их поддерживали. Но если используется ядро 2.6.x, тогда проблем не предвидится. Системе необходимо явно указать, в каких файловых системах будут использоваться ACL; это делается при помощи параметра монтирования «acl».

ACL хранятся в inode файла и, по сути, являются дополнительными атрибутами.

Для управления ACL предназначены программы `getfacl` и `setfacl`.

## getfacl

При помощи программы **getfacl** можно посмотреть список текущих ACL.

**getfacl** [параметры] файлы

Поскольку мы еще не определяли ACL на файлы, то в примере использования программы будут показаны права по умолчанию.

```
$ getfacl test
# file: test
# owner: stavr
# group: users
user::rw-
group::r-x
other::--x
$
```

## setfacl

Для установки и удаления ACL используется программа **setfacl**.

**setfacl** [параметры] [{-m|-x} ACL] файлы

**setfacl** [параметры] [{-M|-X} файл] файлы

Для добавления и изменения ACL используется параметр **-m**. Например, чтобы добавить права доступа для пользователя `user1`, необходимо выполнить следующую команду:

```
$ setfacl -m u:user1:r test
```

После **-m** записывается ACL. Формат записи достаточно простой. Если первый символ **u**, то добавляются права для указанного после двоеточия пользователя. Если первый символ **g** — определяются права для группы. Если **m** — определяется маска. Сами права записываются в символьном виде.

Теперь можно посмотреть, какие ACL установлены у файла `test`:

```
$ getfacl test
# file: test
# owner: stavr
# group: users
user::rw-
user:user1:r--
group::r-x
mask::r-x
other::--x
$
```

Для того, чтобы изменить права пользователя `user1`, необходимо выполнить следующую команду:

```
$ setfacl -m u:user1:rx test
$ getfacl test
# file: test
# owner: stavr
# group: users
user::rw-
user:user1:r-x
group::r-x
mask::r-x
other::--x
$
```

Как видно из примера, у пользователя `user1` были изменены права доступа.

Теперь добавим в ACL группу `bin`:

```
$ setfacl -m g:bin:rx test
$ getfacl test
# file: test
# owner: stavr
# group: users
user::rw-
user:user1:r-x
group::r-x
group:bin:rx
mask::rx
other::--x
$
```

В списке ACL присутствует слово `mask`. Это не пользователь, а специальная возможность, ограничивающая права доступа. Маска является фильтром, определяющим максимально возможные права доступа. Например, группа `bin` имеет полные права доступа. Мы сами дали ей эти права. Но мы можем их ограничить, определив маску.

```
$ setfacl -m m::rw test
$ getfacl test
# file: test
```

```
# owner: stavr
# group: users
user::rw-
user:user1:r-x          #effective:r--
group::r-x              #effective:r--
group:bin:rw-           #effective:rw-
mask::rw-
other::--x
$
```

Обратите внимание на то, что после введения маски напротив пользователей и групп появилось поле, указывающее реальные права доступа. Если у группы bin при записи ее в ACL права были gwx, то теперь они стали gw-. С пользователем user1 случилось то же самое. До определения маски права были r-x, после — стали r--. При помощи маски gw- мы сказали, что право x учитываться не будет.

**Внимание!** На права хозяина файла маска не влияет.

```
$ chmod u+rw- test
$ getfacl test
# file: test
# owner: stavr
# group: users
user::rw-
user:user1:r-x          #effective:r--
group::r-x              #effective:r--
group:bin:rw-           #effective:rw-
mask::rw-
other::--x
$
```

**Внимание!** Для того, чтобы маска заработала, ее необходимо определить явно при помощи setfacl. Иначе она будет «плавающей» и не будет выполнять свои функции.

**Внимание!** Маску всегда определяют самой последней, после указания прав доступа всех пользователей и групп. Если этого не делать, она станет «плавающей» и не будет работать.

Если у файла установлены ACL, определение прав доступа происходит в том порядке, в котором они выводятся программой getfacl.

Программа setfacl предоставляет возможность скопировать ACL одного файла на другой файл. Для этих целей служит параметр --set-file. Например, необходимо скопировать списки доступа файла test в файл newtest. Тогда командная строка будет выглядеть следующим образом:

```
$ getfacl test | setfacl --set-file=- newtest
$ getfacl newtest
# file: newtest
# owner: stavr
# group: users
user::rw-
user:user1:r-x          #effective:r--
group::r-x              #effective:r--
group:bin:rw-           #effective:rw-
mask::rw-
other::--x
$
```

Удаление элементов из списка ACL происходит при помощи параметра -x.

```
$ setfacl -x u:user1 test
$ getfacl test
# file: test
# owner: stavr
# group: users
user::rw-
group::r-x
group:bin:rw-
```

```
mask::rwx
other::--x
$
```

При выполнении этой команды пропала маска. Но об этой особенности я предупреждал Вас раньше.

## Формат записи ACL

Рассмотрим возможные форматы записи ACL.

```
[d[efault:][u[ser]:][пользователь]:[+|^]права]
```

```
[d[efault:] g[roup]:[группа]:[+|^]права]
```

```
[d[efault:] m[ask]:[+|^]права]
```

```
[d[efault:] o[ther]:[+|^]права]
```

Почти все возможные варианты записи мы уже рассматривали. Осталось разобраться с default и специальными символами: + и ^.

Default — устанавливает ACL по умолчанию. Имеет смысл только для директории. Если создавать файлы в директории, у которой установлен ACL по умолчанию, то значения ACL копируются во вновь создаваемые файлы.

Если при написании ACL права доступа начинаются с символа +, то эти права добавляются к уже существующим. Если в начале описания прав стоит символ ^, то их значение отнимаются от существующих прав доступа.

У программы setfacl есть еще две интересные опции: -M и -X. Первая изменяет или добавляет ACL, вторая удаляет ACL. В отличие от -m и -x, список ACL передается не в командной строке, а в отдельном файле.

Формат файла соответствует выводу программы getfacl.

```
$ getfacl newtest > aclfile
$ cat aclfile
# file: newtest
# owner: stavr
# group: users
user::rwx
user:user1:r-x #effective:r--
group::r-x #effective:r--
group:bin:rwx #effective:rw-
mask::rw-
other::--x
$
```

Теперь установим ACL, записанные в файле aclfile, на файл test.

```
$ setfacl -M aclfile test
$ getfacl test
# file: test
# owner: stavr
# group: users
user::rwx
user:user1:r-x #effective:r--
group::r-x #effective:r--
group:bin:rwx #effective:rw-
mask::rw-
other::--x
$
```

К сожалению, многие программы не показывают наличие у файла установленных ACL. Новая технология, еще не все ее приняли к рассмотрению. Более того, стандартные программы работы с архивами в Linux не понимают ACL и не сохраняют их значение в архиве. То есть, у Вас могут возникнуть проблемы с резервным копированием файлов системы. Так что прежде чем использовать ACL хорошо подумайте, а так ли они Вам нужны?

## Изменение владельца и группы файлов

Иногда возникают ситуации, когда необходимо сменить владельца файла или директории. Например, при увольнении одного сотрудника, его файлы передаются другому. Для решения таких задач используются команды `chown` и `chgrp`.

### chown

Программа предназначена для смены владельца файлов и директорий.

`chown username файл`

`chown username:groupname файл`

Выполнять смену владельца может только суперпользователь.

```
# ls -l /home/project/
total 0
drwxrws--- 2 user1 prl 6 2008-09-08 14:14 dir/
-rw-rw---- 1 user1 prl 0 2008-09-08 14:11 file_user1
-rw-rw---- 1 stavr prl 0 2008-09-08 14:08 stavr_file
# chown stavr:users /home/project/file_user1
# ls -l /home/project/
total 0
drwxrws--- 2 user1 prl 6 2008-09-08 14:14 dir/
-rw-rw---- 1 stavr users 0 2008-09-08 14:11 file_user1
-rw-rw---- 1 stavr prl 0 2008-09-08 14:08 stavr_file
#
```

### chgrp

Программа предназначена для смены группы файлов и директорий.

`chgrp groupname файл`

Смену группы может выполнить сам владелец файла или суперпользователь. Для того, чтобы иметь право сменить группу, владелец должен дополнительно быть членом той группы, которой он хочет дать права на данный файл.

```
$ ls -l new_file
-rw-r-x--- 1 stavr users 0 2008-09-08 14:26 new_file
$ chgrp prl new_file
$ ls -l new_file
-rw-r-x--- 1 stavr prl 0 2008-09-08 14:26 new_file
$
```

## Глава 7. Процессы

Самое краткое определение процесса, которое я встречал, звучит так: Процесс — это экземпляр выполняемой программы. Действительно, работающую программу принято называть процессом. Но учтите, что некоторые программы порождают копии себя любимой, значит, может быть несколько процессов с одинаковыми именами программ.

Процесс в Linux имеет некоторое количество атрибутов описывающих его. В первую очередь — это его уникальный номер PID. При запуске процесса ему присваивается PID. После завершения работы, процесс освобождает PID и система может его использовать для другого процесса. PID присваиваются по порядку от меньшего значения к большему. Когда достигается верхний предел, система возвращается к началу и выдает PID начина с первого свободного младшего номера.

В Linux максимальное количество PID равно 32768 (имеется в виду ядро версии 2.6.x. В следующих версиях это значение может быть изменено). Этого значения должно хватить как для рабочей станции, так и для серверов. Я себе с трудом представляю, как на одном PC-совместимом компьютере одновременно будет работать 32 тысячи процессов. Компьютер умрет раньше, чем будет достигнуто такое количество процессов. В реально работающей системе количество процессов гораздо меньше.

### ps

Программа **ps** выводит список процессов системы.

#### ps [параметры]

Если программу запустить без параметров, будет показан список процессов выполняемых текущим пользователем на текущем терминале.

```
$ ps
  PID TTY          TIME CMD
 2568 pts/0    00:00:00 bash
 2666 pts/0    00:00:00 ps
$
```

Ниже описаны поля, которые вывела программа.

- **PID** — PID процесса.
- **TTY** — терминал, к которому подключен процесс. То есть тот терминал, к которому подключены стандартный ввод, вывод и вывод ошибки программы. Иногда эти терминалы называют управляющими терминалами.
- **TIME** — время выполнение программы. То есть то время, которое реально тратит на выполнение программы процессор.
- **CMD** — имя, с которым была запущена программа. В Linux одну программу можно запускать под разными именами (для этого используют ссылки: жесткие и символичные).

Любой пользователь может получить список всех процессов выполняемых в системе. Для этого можно использовать параметр **-e**.

```
$ ps -e
  PID TTY          TIME CMD
    1 ?            00:00:03 init
    2 ?            00:00:00 kthreadd
    3 ?            00:00:00 migration/0
    4 ?            00:00:00 ksoftirqd/0
    5 ?            00:00:02 events/0
    6 ?            00:00:00 khelper
   98 ?            00:00:00 kblockd/0
  101 ?            00:00:00 kacpid
  102 ?            00:00:00 kacpi_notify
  143 ?            00:00:00 ata/0
  144 ?            00:00:00 ata_aux
.....
2326 tty1        00:00:00 agetty
2327 tty2        00:00:00 agetty
```



```

2328 tty3      00:00:00 agetty
2329 tty4      00:00:00 agetty
2330 tty5      00:00:00 agetty
2331 tty6      00:00:00 agetty
2400 ?         00:00:01 sshd
2403 pts/1     00:00:00 bash
2563 ?         00:00:00 sshd
2566 ?         00:00:01 sshd
2568 pts/0     00:00:00 bash
2667 pts/0     00:00:00 ps
$

```

В примере показан не полный вывод программы `ps`, уж слишком большой он получился. Если Вы хотите посчитать количество процессов, это можно сделать, например, так:

```

$ ps -e | sed -e '1d' | wc -l
67
$

```

В этом примере `sed` удаляет первую строку, содержащую заголовок, остается только список процессов. А программа `wc` считает их количество.

Если необходимо посмотреть процессы определенного пользователя, используйте параметр `-u`. Ниже показано, как посмотреть процессы, выполняемые с правами пользователя `daemon`.

```

$ ps -u daemon
  PID TTY          TIME CMD
 2314 ?           00:00:00 atd
$

```

Параметр `-t` позволяет посмотреть процессы, подключенные к определенному терминалу. Например, так можно посмотреть процессы, выполняемые на терминале `pts/1`:

```

$ ps -t pts/1
  PID TTY          TIME CMD
 2403 pts/1     00:00:00 bash
 2672 pts/1     00:00:00 sh
 2673 pts/1     00:00:00 sh
 2674 pts/1     00:00:00 man
 2689 pts/1     00:00:00 sh
 2691 pts/1     00:00:00 less
$

```

Если Вы хотите решать какие поля должна выводить программа `ps`, используйте параметр `-o`. Параметр требует указание дополнительных опций. Ниже приведены некоторые из них.

- **args** — показывать имя программы и аргументы командной строки, переданные программе при ее запуске. Заголовок поля: `COMMAND`.
- **comm** — имя программы. Заголовок поля: `COMMAND`.
- **time** — время, которое тратит процессор на выполнение программы. Заголовок поля: `TIME`.
- **etime** — время, прошедшее с момента запуска программы. Заголовок поля: `ELAPSED`.
- **ni** — значение `nice`. Заголовок поля: `NI`.
- **pid** — PID процесса. Заголовок поля: `PID`.
- **ppid** — PID родительского процесса. Заголовок поля: `PPID`.
- **s** — состояние процесса. Заголовок поля: `S`.
- **start** — время, когда программа была запущена на выполнение. Заголовок поля: `STARTED`.
- **tty** — терминал, к которому подключен процесс. Заголовок поля: `TTY`.

Например, необходимо посмотреть список всех процессов системы с указанием PID, PPID, терминала и имени программы. Тогда `ps` должна быть запущена со следующими аргументами:

```

$ ps -eo pid,ppid,TTY,comm

```

```

PID  PPID  TT      COMMAND
  1      0  ?      init
  2      0  ?      kthreadd
  3      2  ?      migration/0
  4      2  ?      ksoftirqd/0
  5      2  ?      events/0
  6      2  ?      khelper
 98      2  ?      kblockd/0
101      2  ?      kacpid
102      2  ?      kacpi_notify
143      2  ?      ata/0
144      2  ?      ata_aux
.....
2326      1 tty1    agetty
2327      1 tty2    agetty
2328      1 tty3    agetty
2329      1 tty4    agetty
2330      1 tty5    agetty
2331      1 tty6    agetty
2400    2262 ?      sshd
2403    2400 pts/1   bash
2563    2262 ?      sshd
2566    2563 ?      sshd
2568    2566 pts/0   bash
2672    2403 pts/1   sh
2673    2672 pts/1   sh
2674    2673 pts/1   man
2689    2674 pts/1   sh
2691    2689 pts/1   less
2693    2568 pts/0   ps
$

```

Администраторы используют программу `ps` в основном для того, что бы узнать работает ли интересующий его процесс. И если он работает, то, с каким PID он выполняется. Например, необходимо узнать работает ли сервер безопасного подключения `sshd` и если он работает, то какие конкретные процессы и с каким PID сейчас находятся в системе. Для выяснения этой информации воспользуемся программами `ps` и `pgrep`:

```

$ ps -e | grep sshd
2262 ?      00:00:00 sshd
2400 ?      00:00:01 sshd
2563 ?      00:00:00 sshd
2566 ?      00:00:01 sshd
$

```

Еще одна программа, которая может помочь в решении этой задачи — это `pgrep`.

## pgrep

Программа **pgrep** просматривает список работающих процессов и выводит PID процессов удовлетворяющих критериям отбора, заданным при вызове программы.

**pgrep [параметры] [критерии отбора]...**

Если попытаться получить PID всех процессов `sshd` в данный момент работающих в системе, следует выполнить `pgrep` с указанием имени программы:

```

$ pgrep sshd
2262
2400
2563
2566
$

```

`Pgrep` позволяет использовать следующие критерии отбора:

- **-P pid** — вывести все процессы запущенные процессом с указанным PID.

- **-t term** — вывести все процессы, выполняющиеся на указанном терминале.
- **-u uid** — вывести все процессы, выполняемые с правами указанного пользователя.
- **-U uid** — вывести все процессы, запущенные на выполнение указанным пользователем.

Например, следующая команда покажет PID процессов выполняющихся с правами пользователя root на виртуальном терминале tty6:

```
$ pgrep -u root -t tty6
2331
$
```

При помощи параметра **-d** можно указать символ разделитель, используемый при выводе PID. По умолчанию используется символ перевод строки. Например, ниже показано, как выводить список через пробел.

```
$ pgrep -d ' ' sshd
2262 2400 2563 2566
$
```

## Потомок, родитель

Каждый процесс в системе имеет «родителя» — процесс, который его запустил на выполнение. Любой процесс может быть «родителем», то есть запускать другие программы. У процесса может быть один или несколько «потомков». Каждый процесс «знает» своего родителя, у него хранится его PID — PPID (Parent PID).

Посмотреть «дерево» процессов можно при помощи программы **ps** с параметром **-f**.

```
$ ps axf
  PID TTY          STAT TIME  COMMAND
    2 ?           S<    0:00 [kthreadd]
    3 ?           S<    0:00 \_ [migration/0]
    4 ?           S<    0:00 \_ [ksoftirqd/0]
    5 ?           S<    0:03 \_ [events/0]
    6 ?           S<    0:00 \_ [khelper]
   98 ?           S<    0:00 \_ [kblockd/0]
  101 ?           S<    0:00 \_ [kacpid]
  102 ?           S<    0:00 \_ [kacpi_notify]
  143 ?           S<    0:00 \_ [ata/0]
  144 ?           S<    0:00 \_ [ata_aux]
  145 ?           S<    0:00 \_ [ksuspend_usbd]
  151 ?           S<    0:00 \_ [khubd]
  154 ?           S<    0:00 \_ [kseriod]
  187 ?           S    0:00 \_ [pdflush]
  188 ?           S    0:00 \_ [pdflush]
  189 ?           S<    0:00 \_ [kswapd0]
  228 ?           S<    0:00 \_ [aio/0]
  250 ?           S<    0:00 \_ [jfsIO]
  251 ?           S<    0:00 \_ [jfsCommit]
  252 ?           S<    0:00 \_ [jfsSync]
  254 ?           S<    0:00 \_ [xfslogd/0]
  255 ?           S<    0:00 \_ [xfsdatad/0]
  259 ?           S<    0:00 \_ [xfs_mru_cache]
  264 ?           S<    0:04 \_ [gfs2_scand]
  265 ?           S<    0:00 \_ [glock_workqueue]
  944 ?           S<    0:00 \_ [scsi_tgt/0]
 1021 ?           S<    0:00 \_ [exec-osm/0]
 1027 ?           S<    0:00 \_ [block-osm/0]
 1034 ?           S<    0:00 \_ [khpsbpkt]
 1062 ?           S<    0:00 \_ [ksnapd]
 1067 ?           S<    0:00 \_ [rpciod/0]
 1069 ?           S<    0:01 \_ [xfsbufd]
 1070 ?           S<    0:00 \_ [xfssyncd]
 1396 ?           S<    0:00 \_ [kpsmoused]
 2152 ?           S<    0:01 \_ [xfsbufd]
 2153 ?           S<    0:00 \_ [xfssyncd]
 2154 ?           S<    0:01 \_ [xfsbufd]
```

```

2155 ?      S<      0:00  \_ [xfssyncd]
  1 ?      Ss      0:03  init [3]
1124 ?      S<S     0:01  /sbin/udevd --daemon
2197 ?      Ss      0:00  /usr/sbin/syslogd
2201 ?      Ss      0:00  /usr/sbin/klogd -c 3 -x
2262 ?      Ss      0:00  /usr/sbin/sshd
2400 ?      Ss      0:01  \_ sshd: root@pts/1
2403 pts/1   Ss      0:00  |      \_ -bash
2672 pts/1   S       0:00  |          \_ sh
2673 pts/1   S       0:00  |              \_ sh
2674 pts/1   S+      0:00  |                  \_ man hier
2689 pts/1   S+      0:00  |                      \_ sh -c /bin/bzip2 -c -d /usr/m
2691 pts/1   S+      0:00  |                          \_ /usr/bin/less -is
2563 ?      Ss      0:00  \_ sshd: stavr [priv]
2566 ?      S       0:01  \_ sshd: stavr@pts/0
2568 pts/0    Ss      0:00  \_ -bash
2712 pts/0    R+      0:00  \_ ps axf
2269 ?      Ss      0:00  /usr/sbin/acpid
2280 ?      Ss      0:00  /usr/bin/dbus-daemon --system
2286 ?      Ss      0:01  /usr/sbin/hald --daemon=yes
2287 ?      S       0:00  \_ hald-runner
2293 ?      S       0:00  \_ hald-addon-input: Listening on /dev/input/eve
2299 ?      S       0:00  \_ hald-addon-acpi: listening on acpid socket /v
2305 ?      S       0:23  \_ hald-addon-storage: polling /dev/hdc (every 2
2312 ?      S       0:00  /usr/sbin/crond -l10
2314 ?      Ss      0:00  /usr/sbin/atd -b 15 -l 1
2324 ?      Ss      0:00  /usr/sbin/gpm -m /dev/mouse -t imps2
2326 tty1     Ss+     0:00  /sbin/agetty 38400 tty1 linux
2327 tty2     Ss+     0:00  /sbin/agetty 38400 tty2 linux
2328 tty3     Ss+     0:00  /sbin/agetty 38400 tty3 linux
2329 tty4     Ss+     0:00  /sbin/agetty 38400 tty4 linux
2330 tty5     Ss+     0:00  /sbin/agetty 38400 tty5 linux
2331 tty6     Ss+     0:00  -bash
$

```

Как видно из примера, взаимоотношения родитель-потомок обозначаются при помощи символов псевдографики.

## pstree

Еще одна программа, которая позволяет посмотреть дерево процессов, — это **ps**tree.

**ps**tree [параметры]...

Ниже показан пример работы программы:

```

$ pstree
init--+-acpid
      |-5*[agetty]
      |-atd
      |-bash
      |-crond
      |-dbus-daemon
      |-gpm
      |-hald---hald-runner--+-hald-addon-acpi
      |                     |-hald-addon-inpu
      |                     `--hald-addon-stor
      |-klogd
      |-sshd--+-sshd---bash---sh---sh---man---sh---less
      |      `--sshd---sshd---bash---pstree
      |-syslogd
      `--udevd
$

```

## nohup

Если «родитель» завершает свою работу раньше потомка, у потомка меняется значение PPID на 1, то есть потомок продолжает работу, но уже с новым родителем. Процесс с PID 1 — это программа init. Init запускается самым ядром операционной системы, при ее старте. До тех пор пока работает init, работает Linux.

Правда, существуют исключения из этого правила. Если завершает работу программа-оболочка, в которой работал пользователь, завершают работу все программы, запущенные из этого экземпляра shell.

При необходимости продолжения выполнения программы после выхода пользователя из системы используют программу **nohup**.

nohup программа [параметры программы]...

При завершении работы управляющей оболочки все программы получают уведомление об отключении от терминала и должны завершить свою работу. Nohup заставляет программу игнорировать получаемое уведомление, и она (программа) продолжает свою работу даже после завершения работы оболочки.

## Демоны

В дальнейшем Вы увидите, что на терминологию, применяемую в системе, сильно повлиял англо-саксонский фольклор: демоны, зомби и т.д. Что тут говорить — это особенность системы, и все термины являются «официальными». Стоит отметить, что авторы UNIX не почивали на лаврах после ее разработки. Они создали экспериментальную систему Plan 9, а затем операционную систему Inferno. Так что демоны — это цветочки, то ли еще будет.

Если говорить серьезно, демоны — это процессы, которые при помощи специальной функции отключены от контрольного терминала. То есть после завершения работы оболочки пользователя эта программа продолжит свое выполнение. Это похоже на применение программы nohup, но отличие состоит в том, что отключение от терминала предусмотрено в коде самой программы (*если вы программист, обратите внимание на функцию daemon(3)*).

Обычно процессы-демоны сравнивают со службами Windows. Они запускаются при старте системы и выполняют функции какого-либо сервиса: почта, WEB-сервер и т.д.

## Сигналы

Предположим, что есть две программы. Одна программа должна передать другой большое количество данных. Передача данных будет происходить через файл типа FIFO. То есть одна программа открывает этот файл на запись и передает данные, другая должна открыть этот файл на чтение и получить данные. Проблема заключается в том, что вторая программа должна узнать, что пришло время открывать файл и читать данные.

Чтобы уведомить вторую программу, первая может послать ей сигнал. Сигнал — это число. Сигнал можно представить в виде программного прерывания.

Можно вспомнить очень известный анекдот про Петьку и Василия Ивановича:

Летят Петька и Василий Иванович на самолете.

— Петька, прибор!

— 28.

— Что 28?

— А что прибор?

Реакция программы на получаемый сигнал зависит от программиста, написавшего эту программу. Программист, написавший вторую программу из нашего примера, в документации к ней укажет, что если программе послать сигнал, например, 10, то она откроет указанный FIFO-файл на чтение.

Если говорить более точно, то реакция программы на получаемый сигнал зависит от системы. Но программисты могут переопределить стандартную реакцию программы.

Чтобы получить список всех сигналов, которые поддерживаются системой, используйте программу kill с параметром -l:

```
$ kill -l
1) SIGHUP      2) SIGINT      3) SIGQUIT     4) SIGILL
5) SIGTRAP     6) SIGABRT     7) SIGBUS      8) SIGFPE
```

```

 9) SIGKILL      10) SIGUSR1      11) SIGSEGV      12) SIGUSR2
13) SIGPIPE     14) SIGALRM      15) SIGTERM      16) SIGSTKFLT
17) SIGCHLD     18) SIGCONT      19) SIGSTOP      20) SIGTSTP
21) SIGTTIN     22) SIGTTOU      23) SIGURG       24) SIGXCPU
25) SIGXFSZ     26) SIGVTALRM    27) SIGPROF      28) SIGWINCH
29) SIGIO       30) SIGPWR      31) SIGSYS       34) SIGRTMIN
35) SIGRTMIN+1  36) SIGRTMIN+2  37) SIGRTMIN+3  38) SIGRTMIN+4
39) SIGRTMIN+5  40) SIGRTMIN+6  41) SIGRTMIN+7  42) SIGRTMIN+8
43) SIGRTMIN+9  44) SIGRTMIN+10 45) SIGRTMIN+11 46) SIGRTMIN+12
47) SIGRTMIN+13 48) SIGRTMIN+14 49) SIGRTMIN+15 50) SIGRTMAX-14
51) SIGRTMAX-13 52) SIGRTMAX-12 53) SIGRTMAX-11 54) SIGRTMAX-10
55) SIGRTMAX-9  56) SIGRTMAX-8  57) SIGRTMAX-7  58) SIGRTMAX-6
59) SIGRTMAX-5  60) SIGRTMAX-4  61) SIGRTMAX-3  62) SIGRTMAX-2
63) SIGRTMAX-1  64) SIGRTMAX
$

```

Количество поддерживаемых сигналов зависит от типа системы. Даже в разных версиях Linux (имеются в виду версии ядра) может применяться разное количество сигналов. Но их никогда не бывает меньше 32.

Каждый сигнал имеет номер, полное и краткое имена. Например, сигнал 1 имеет полное имя SIGHUP, краткое имя сигнала — HUP.

**Внимание!** В различных типах UNIX номера сигналов могут не совпадать, поэтому, если Вы регулярно работаете в разных UNIX, используйте имена, а не номера сигналов. Тогда Вы никогда не ошибетесь при указании сигнала.

Как уже говорилось выше, у программ есть стандартная реакция на получаемые сигналы. Давайте рассмотрим некоторые из них.

Сигнал	Описание сигнала	Стандартная реакция программы
HUP(1)	Сброс.	Завершение работы. Для демонов — перечитать конфигурационный файл.
INT(2)	Посылается, если нажата комбинация клавиш Ctrl+C.	Завершение работы.
KILL(9)	Безусловное завершение работы программы.	Завершение работы.
TERM(15)	Завершение работы программы.	Завершение работы.
CONT(18)	Продолжение выполнения приостановленной программы.	Игнорируется.
STOP(19)	Приостановление выполнения программы. Приостановление выполнения программы.	

Сигнал TERM(15) посылается программе, если необходимо завершить ее выполнение. При получении этого сигнала программа выполняет все необходимые действия для завершения работы: закрывает соединения, открытые файлы, сохраняет данные. В общем, корректно завершает свою работу.

Теперь представьте, что каким-то чудом (это действительно будет чудо, при условии, что вы правильно управляете Linux-машиной) хакер поставил и запустил программу, которая, например, форматирует ваш диск с проверкой на сбойные блоки в режиме записи (*очень длительная процедура*). Вы обнаружили эту программу и решили завершить ее работу. При послышке 15-го сигнала программа не завершит работу, т.к. программист, ее написавший, поставил свой собственный обработчик сигналов. Он, конечно, предусмотрел возможность получения 15-го сигнала, и в ответ на сигнал программа либо просто продолжает свою работу, либо начинает форматировать диск в ускоренном режиме.

В системе предусмотрены сигналы, которые не могут быть переопределены программистом. И это в первую очередь сигнал KILL(9). Но этим сигналом следует пользоваться с особой осторожностью.

**Внимание!** Посылка сигнала KILL программе аналогична нажатию на кнопку reset, то есть программа завершает работу не корректно! Применять сигнал надо только в том случае, если программа не реагирует на

посылку сигнала TERM.

Еще один сигнал, который обрабатывает система, а не программа, — STOP(19). При посылке этого сигнала программе ее выполнение будет приостановлено. То есть оперативная память, выделенная под программу, не освобождается, просто программа не будет выполняться до тех пор, пока не будет послан сигнал CONT(18).

Сигнал INT(2) посылается программе, если нажата комбинация клавиш Ctrl+C.

Сигнал HUP(1) посылается программе, если произошло отключение терминала. Действительно, в далекие времена рабочие места подключались через модемы или COM-порты, и при отключении связи программам посылался сигнал HUP. В современных системах этот сигнал говорит программе, что необходимо перечитать свой конфигурационный файл. Дело в том, что программы после запуска «забывают» о своем конфигурационном файле и не следят за вносимыми в него изменениями. Поэтому после редактирования содержимого конфигурационного файла программы, ей (программе) рекомендуется послать сигнал HUP. Это в первую очередь относится к процессам-демонам.

## kill

Несмотря на страшное название программы (*сразу видно, что систему писали в стране, которая очень любит воевать. Если бы UNIX писали в России, то программу скорее всего назвали бы "послать".*), она всего лишь позволяет пользователю послать сигнал программе.

kill [-сигнал] [PID]...

При вызове программы следует указать PID процесса. В случае если необходимо послать сигнал нескольким процессам одновременно, введите несколько PID через пробел.

Параметр -l показывает список всех поддерживаемых сигналов.

Если при вызове программы явно не указывается номер посылаемого сигнала, по умолчанию посылается сигнал TERM(15).

В следующем примере показано, как можно послать сигнал при помощи программы kill.

```
$ ps -t tty6
  PID TTY          TIME CMD
 2761 tty6      00:00:00 bash
$ kill 2761
-bash: kill: (2761) - Operation not permitted
```

Сначала при помощи программы ps был получен список процессов, выполняющихся на терминале tty6. Там работает только оболочка пользователя. Ее PID — 2761. Затем выполняется программа kill, которой передается PID процесса bash. И здесь мы получаем от системы ответ о невозможности запрошенного действия. Дело в том, что bash на виртуальной консоли tty6 запущен пользователем root. Обычный пользователь может управлять только своими процессами. Управлять любыми процессами в системе может только пользователь root.

Попробуем все сначала, только от имени пользователя root.

```
# ps -t tty6
  PID TTY          TIME CMD
 2761 tty6      00:00:00 bash
# kill 2761
# ps -t tty6
  PID TTY          TIME CMD
 2761 tty6      00:00:00 bash
#
```

Как видно из вывода программы ps, bash не завершил своей работы. Дело в том, что если не указать номер посылаемого сигнала, по умолчанию посылается сигнал TERM(15). Bash, работающий в качестве основной оболочки пользователя, игнорирует сигнал TERM. Для завершения работы программы пошлем ей сигнал KILL(9).

```
# kill -9 2761
# ps -t tty6
  PID TTY          TIME CMD
 2787 tty6      00:00:00 agetty
#
```

После получения сигнала KILL программа bash завершила свою работу.

## killall

Программа посылает сигнал процессам с указанным именем.

killall [-сигнал] [имя]...

Программа killall полностью повторяет возможности kill. Единственное, что их отличает друг от друга, — killall необходимо указывать не PID, а имя процесса.

## pgrep

Программа посылает сигнал процессам, соответствующим атрибутам поиска, указанным при вызове программы.

pgrep [параметры] [критерии отбора]...

Параметры pgrep полностью совпадают с параметрами программы pkill.

## Мониторинг процессов

Администратору системы достаточно часто приходится смотреть список процессов. Программа ps позволяет получить моментальный «срез» системы — процессы, исполняемые на момент вызова ps. Иногда необходимо в реальном времени отслеживать, какие процессы выполняются в системе. Одна из наиболее популярных программ, которая позволяет посмотреть список процессов в реальном (или почти реальном) времени, — это программа top.

## top

### Описание

Программа **top** позволяет увидеть динамически обновляющийся список процессов. Также, она показывает суммарную информацию о загрузке системы.

top -hv

top -bcisS -d задержка -n повтор -p PID [,PID ...]

Топ — интерактивная программа, и после запуска полностью занимает терминал. Во время работы программы доступны различные команды, но в первую очередь следует запомнить две из них:

- **h** — выводит экран помощи,
- **q** — выход из программы.

По умолчанию top отображает на экране следующую информацию:

```
top - 17:31:29 up 9:22, 2 users, load average: 0.00, 0.00, 0.00
Tasks: 65 total, 1 running, 64 sleeping, 0 stopped, 0 zombie
Cpu(s): 0.0%us, 0.3%sy, 0.0%ni, 99.3%id, 0.0%wa, 0.0%hi, 0.3%si, 0.0%st
Mem: 250756k total, 139188k used, 111568k free, 280k buffers
Swap: 250448k total, 0k used, 250448k free, 107160k cached
```

PID	USER	PR	NI	VIRT	RES	SHR	S	%CPU	%MEM	TIME+	COMMAND
1	root	20	0	772	304	260	S	0.0	0.1	0:03.50	init
2	root	15	-5	0	0	0	S	0.0	0.0	0:00.00	kthreadd
3	root	RT	-5	0	0	0	S	0.0	0.0	0:00.00	migration/0
4	root	15	-5	0	0	0	S	0.0	0.0	0:00.00	ksoftirqd/0
5	root	15	-5	0	0	0	S	0.0	0.0	0:04.02	events/0
6	root	15	-5	0	0	0	S	0.0	0.0	0:00.14	khelper
98	root	15	-5	0	0	0	S	0.0	0.0	0:00.18	kblockd/0
101	root	15	-5	0	0	0	S	0.0	0.0	0:00.00	kacpid
102	root	15	-5	0	0	0	S	0.0	0.0	0:00.00	kacpi_notify
143	root	15	-5	0	0	0	S	0.0	0.0	0:00.00	ata/0
144	root	15	-5	0	0	0	S	0.0	0.0	0:00.00	ata_aux
145	root	15	-5	0	0	0	S	0.0	0.0	0:00.00	ksuspend_usbd



```

151 root      15   -5      0      0      0 S  0.0  0.0  0:00.00 khubd
154 root      15   -5      0      0      0 S  0.0  0.0  0:00.02 kseriod
187 root      20    0      0      0      0 S  0.0  0.0  0:00.00 pdflush
188 root      20    0      0      0      0 S  0.0  0.0  0:00.72 pdflush
189 root      15   -5      0      0      0 S  0.0  0.0  0:00.00 kswapd0

```

Как видно из примера, вывод программы делится на три части:

- заголовок с суммарной информацией о загрузке системы,
- строка ввода (в данный момент пустая),
- отсортированный список процессов.

В заголовке отображается следующая информация:

Первая строка — загрузка системы.

- Текущее время.
- Время работы системы после включения питания (up time).
- Количество пользователей, которые в данный момент работают в системе.
- Средняя загрузка системы (load average) минуту, пять минут и пятнадцать минут назад.

Вторая строка — процессы.

- **total** — общее количество процессов в системе.
- **running** — количество процессов, выполняемых процессором или стоящих в очереди на выполнение.
- **sleeping** — количество процессов, ожидающих какого-либо события ввода-вывода.
- **stopped** — количество приостановленных процессов.
- **zombie** — количество процессов, находящихся в состоянии «зомби» (подробнее о зомби будет рассказано ниже).

Третья строка — состояние процессора.

- **us** — процент использования процессорного времени программами пользователей.
- **sy** — процент использования процессорного времени процессами ядра Linux.
- **ni** — процент использования процессорного времени программами с измененным приоритетом.
- **id** — простой процессора.

Четвертая строка — использование оперативной памяти.

- **total** — общее количество оперативной памяти.
- **used** — количество использованной оперативной памяти.
- **free** — количество свободной оперативной памяти.

Пятая строка — использование swap-пространства.

- **total** — общее количество swap-пространства.
- **used** — количество использованного swap-пространства.
- **free** — количество свободного swap-пространства.

В списке процессов по умолчанию показаны следующие столбцы:

- **PID** — PID процесса.
- **USER** — пользователь, с правами которого выполняется процесс.
- **PR** — приоритет процесса.
- **NI** — на сколько больше или меньше процесс «нравится» (от слова nice) системе. То есть на сколько был изменен приоритет процесса.
- **VIRT** — общее количество виртуальной памяти, используемой программой. Значение в килобайтах.
- **RES** — количество резидентной (не перемещаемой в swap) памяти в килобайтах.

- **SHR** — количество разделяемой (shared) памяти программы в килобайтах.
- **S** — состояние процесса:
  - **D** — uninterruptible sleep.
  - **R** — процесс выполняется процессором или стоит в очереди на выполнение.
  - **S** — процесс ожидает событие ввода-вывода.
  - **T** — выполнение процесса приостановлено.
  - **Z** — состояние «зомби».

## Команды

Теперь рассмотрим, какие команды поддерживает `top`.

Команда	Описание
A	Переключает количество одновременно выводимых на экран окон. По умолчанию показано одно окно.
d или s	Устанавливает значение задержки перерисовки экрана. Значение по умолчанию 3 секунды.
l	Включение/выключение поля «средняя загрузка системы» в заголовке.
t	Включение/выключение полей описания процессов и загрузки процессора в заголовке.
m	Включение/выключение полей, описывающих использование оперативной памяти и файла подкачки.
b	Выделение процессов, выполняемых процессором (состояние Run).
c	Включает/выключает отображение имени программы/полной командной строки в поле COMMAND.
i	Включает отображение только процессов, выполняемых процессором, или всех процессов системы.
x	Включает/выключает подсветку колонки, по которой происходит сортировка процессов.
z	Включает/выключает цвет.
Пробел	Заставляет программу пересчитать список процессов.
u	Показывает процессы, выполняемые определенным пользователем.
k	Послать сигнал процессу.
r	Изменить значение поля nice (приоритет процесса).

Использовать `top` в интерактивном режиме достаточно просто. Например, необходимо посмотреть процессы только одного пользователя. Для этого во время работы программы достаточно нажать на клавишу `u`:

```
top - 17:58:51 up 9:49, 2 users, load average: 0.00, 0.00, 0.00
Tasks: 65 total, 1 running, 64 sleeping, 0 stopped, 0 zombie
Cpu(s): 0.0%us, 0.3%sy, 0.0%ni, 99.7%id, 0.0%wa, 0.0%hi, 0.0%si, 0.0%st
Mem: 250756k total, 139368k used, 111388k free, 280k buffers
Swap: 250448k total, 0k used, 250448k free, 107160k cached
```

**Which user (blank for all):**

```
PID USER PR NI VIRT RES SHR S %CPU %MEM TIME+ COMMAND
264 root 15 -5 0 0 0 S 0.7 0.0 0:04.68 gfs2_scand
2796 stavr 20 0 2344 1056 840 R 0.7 0.4 0:06.40 top
1 root 20 0 772 304 260 S 0.0 0.1 0:03.58 init
2 root 15 -5 0 0 0 S 0.0 0.0 0:00.00 kthreadd
```

В появившейся строке ввести учетную запись интересующего пользователя и нажать Enter.

```
top - 17:58:51 up 9:49, 2 users, load average: 0.00, 0.00, 0.00
Tasks: 65 total, 1 running, 64 sleeping, 0 stopped, 0 zombie
Cpu(s): 0.0%us, 0.3%sy, 0.0%ni, 99.7%id, 0.0%wa, 0.0%hi, 0.0%si, 0.0%st
Mem: 250756k total, 139368k used, 111388k free, 280k buffers
Swap: 250448k total, 0k used, 250448k free, 107160k cached
```

**Which user (blank for all): stavr**

PID	USER	PR	NI	VIRT	RES	SHR	S	%CPU	%MEM	TIME+	COMMAND
264	root	15	-5	0	0	0	S	0.7	0.0	0:04.68	gfs2_scand
2796	stavr	20	0	2344	1056	840	R	0.7	0.4	0:06.40	top
1	root	20	0	772	304	260	S	0.0	0.1	0:03.58	init
2	root	15	-5	0	0	0	S	0.0	0.0	0:00.00	kthreadd

В результате, будут показаны процессы, выполняемые с правами пользователя stavr:

```
top - 18:08:48 up 9:59, 3 users, load average: 0.00, 0.00, 0.00
Tasks: 69 total, 1 running, 68 sleeping, 0 stopped, 0 zombie
Cpu(s): 0.0%us, 0.3%sy, 0.0%ni, 99.7%id, 0.0%wa, 0.0%hi, 0.0%si, 0.0%st
Mem: 250756k total, 144104k used, 106652k free, 280k buffers
Swap: 250448k total, 0k used, 250448k free, 108556k cached
```

PID	USER	PR	NI	VIRT	RES	SHR	S	%CPU	%MEM	TIME+	COMMAND
2566	stavr	20	0	6648	1320	940	S	0.0	0.5	0:03.20	sshd
2568	stavr	20	0	3600	2036	1288	S	0.0	0.8	0:00.62	bash
2821	stavr	20	0	4908	1928	1436	S	0.0	0.8	0:00.02	mc
2823	stavr	20	0	3548	1888	1172	S	0.0	0.8	0:00.02	bash

Для того, чтобы увидеть процессы всех пользователей, необходимо опять ввести команду u и нажать на Enter, не вводя имени учетной записи пользователя.

Находясь в top, Вы можете послать сигнал интересующему Вас процессу. Например, необходимо завершить работу программы mc, работающей с правами пользователя stavr. В top вводим команду k, предварительно получив все процессы пользователя stavr:

```
top - 18:09:30 up 10:00, 3 users, load average: 0.00, 0.00, 0.00
Tasks: 69 total, 1 running, 68 sleeping, 0 stopped, 0 zombie
Cpu(s): 0.0%us, 0.3%sy, 0.0%ni, 99.7%id, 0.0%wa, 0.0%hi, 0.0%si, 0.0%st
Mem: 250756k total, 144104k used, 106652k free, 280k buffers
Swap: 250448k total, 0k used, 250448k free, 108556k cached
```

**PID to kill:**

PID	USER	PR	NI	VIRT	RES	SHR	S	%CPU	%MEM	TIME+	COMMAND
2566	stavr	20	0	6648	1320	940	S	0.0	0.5	0:03.20	sshd
2568	stavr	20	0	3600	2036	1288	S	0.0	0.8	0:00.62	bash
2821	stavr	20	0	4908	1928	1436	S	0.0	0.8	0:00.02	mc
2823	stavr	20	0	3548	1888	1172	S	0.0	0.8	0:00.02	bash

В появившемся приглашении вводим PID процесса.

```
top - 18:09:30 up 10:00, 3 users, load average: 0.00, 0.00, 0.00
Tasks: 69 total, 1 running, 68 sleeping, 0 stopped, 0 zombie
Cpu(s): 0.0%us, 0.3%sy, 0.0%ni, 99.7%id, 0.0%wa, 0.0%hi, 0.0%si, 0.0%st
Mem: 250756k total, 144104k used, 106652k free, 280k buffers
Swap: 250448k total, 0k used, 250448k free, 108556k cached
```

**PID to kill: 2821**

PID	USER	PR	NI	VIRT	RES	SHR	S	%CPU	%MEM	TIME+	COMMAND
2566	stavr	20	0	6648	1320	940	S	0.0	0.5	0:03.20	sshd
2568	stavr	20	0	3600	2036	1288	S	0.0	0.8	0:00.62	bash
2821	stavr	20	0	4908	1928	1436	S	0.0	0.8	0:00.02	mc
2823	stavr	20	0	3548	1888	1172	S	0.0	0.8	0:00.02	bash

И нажимаем Enter. Появляется предложение указать номер сигнала.

```
top - 18:09:30 up 10:00, 3 users, load average: 0.00, 0.00, 0.00
Tasks: 69 total, 1 running, 68 sleeping, 0 stopped, 0 zombie
```

```
Cpu(s):  0.0%us,  0.3%sy,  0.0%ni, 99.7%id,  0.0%wa,  0.0%hi,  0.0%si,  0.0%st
Mem:    250756k total,  144104k used,  106652k free,    280k buffers
Swap:   250448k total,    0k used,  250448k free,  108556k cached
Kill PID 2821 with signal [15]:
  PID USER      PR  NI  VIRT  RES  SHR S %CPU %MEM    TIME+  COMMAND
 2566 stavr    20   0  6648 1320  940 S  0.0  0.5    0:03.20 sshd
 2568 stavr    20   0  3600 2036 1288 S  0.0  0.8    0:00.62 bash
 2821 stavr    20   0  4908 1928 1436 S  0.0  0.8    0:00.02 mc
 2823 stavr    20   0  3548 1888 1172 S  0.0  0.8    0:00.02 bash
```

Если явно не указать номер, будет послан 15-й сигнал. Поскольку этот сигнал будет проигнорирован программой, пошлем ей сигнал 9. То есть введем число 9 и нажмем Enter.

```
top - 18:12:59 up 10:03,  3 users,  load average: 0.00, 0.00, 0.00
Tasks:  67 total,   2 running,  65 sleeping,   0 stopped,   0 zombie
Cpu(s):  0.0%us,  0.0%sy,  0.0%ni,100.0%id,  0.0%wa,  0.0%hi,  0.0%si,  0.0%st
Mem:    250756k total,  143024k used,  107732k free,    280k buffers
Swap:   250448k total,    0k used,  250448k free,  108556k cached

  PID USER      PR  NI  VIRT  RES  SHR S %CPU %MEM    TIME+  COMMAND
 2566 stavr    20   0  6648 1320  940 S  0.0  0.5    0:03.20 sshd
 2568 stavr    20   0  3600 2036 1288 S  0.0  0.8    0:00.62 bash
```

Посылать сигнал программе может только пользователь, с правами которого выполняется программа, или пользователь root.

## Batch-режим

Кроме интерактивного режима, в котором top выводит данные на экран, можно использовать так называемый командный режим (Batch mode). Он применяется, когда результаты работы программы необходимо передать другим программам или сохранить в файле.

Для запуска программы в командном режиме используют параметр `-b`. В этом случае список отсортированных процессов (по умолчанию, процессы сортируются по проценту использования процессорного времени) будет с определенной задержкой (по умолчанию три секунды) выводиться на стандартный вывод. Количество повторов не ограничено, поэтому необходимо явно завершать работу программы, например, при помощи комбинации клавиш `Ctrl+C`.

Рассмотрим некоторые параметры, которые можно применять в командном режиме программы top.

- `-n` — количество повторов.
- `-d` — задержка между повторами в секундах.
- `-u` — определяет пользователя, с правами которого выполняются процессы.
- `-p` — определяет PID процессов, за которыми должна следить программа.

Например, необходимо с задержкой в четыре секунды два раза получить список процессов, выполняемых с правами пользователя daemon. Для этого программу запускают со следующими параметрами:

```
$ top -b -d 4 -n 2 -u daemon
top - 18:19:24 up 10:10,  3 users,  load average: 0.01, 0.01, 0.00
Tasks:  67 total,   1 running,  66 sleeping,   0 stopped,   0 zombie
Cpu(s):  0.0%us,  0.1%sy,  0.0%ni, 99.8%id,  0.0%wa,  0.0%hi,  0.0%si,  0.0%st
Mem:    250756k total,  142788k used,  107968k free,    280k buffers
Swap:   250448k total,    0k used,  250448k free,  108584k cached

  PID USER      PR  NI  VIRT  RES  SHR S %CPU %MEM    TIME+  COMMAND
 2314 daemon    20   0  1912  404  304 S  0.0  0.2    0:00.00 atd
```

```
top - 18:19:28 up 10:10,  3 users,  load average: 0.01, 0.01, 0.00
Tasks:  67 total,   1 running,  66 sleeping,   0 stopped,   0 zombie
Cpu(s):  0.0%us,  0.5%sy,  0.0%ni, 99.5%id,  0.0%wa,  0.0%hi,  0.0%si,  0.0%st
Mem:    250756k total,  142796k used,  107960k free,    280k buffers
Swap:   250448k total,    0k used,  250448k free,  108584k cached
```

```
PID USER      PR  NI  VIRT  RES  SHR  S  %CPU  %MEM    TIME+  COMMAND
2314 daemon    20   0  1912  404  304  S   0.0   0.2   0:00.00  atd
```

```
$
```

## Приоритет процессов

Приоритет процесса определяет, как часто данный процесс, по сравнению с другими процессами, стоящими в очереди на выполнение процессора, будет исполняться процессором.

Вы должны понять, что приоритет — понятие относительное. Ведь в данный момент в очереди на выполнение могут стоять двадцать процессов, а в следующий — два. Поэтому приоритет — это «плавающее» значение, которое определяет сама система в зависимости от текущей ситуации.

В Linux значение приоритета может быть равно числу в диапазоне сорока значений. Как же я туманно выразился. На самом деле, все зависит от версии ядра Linux. Значение приоритета может быть в диапазоне от -20 до 19 или от 0 до 39. В обоих случаях возможное значение ограничено сорока числами. В ядрах версии 2.6.x значение приоритета находится в диапазоне от 0 до 39.

Чем меньше число — тем выше приоритет процесса. То есть процесс с приоритетом 5 имеет больший приоритет, чем процесс с приоритетом 16.

Вы не можете непосредственно изменять приоритет процесса, это прерогатива операционной системы. Но Вы можете определить, на сколько больше или меньше процесс будет «нравиться» системе. Для этого следует использовать поле `nice`. То есть для изменения приоритета администратор должен изменять значение поля `nice`. Если значение `nice` равно нулю — значит, у программы приоритет не изменен. Если в поле `nice` записано отрицательное число — приоритет процесса увеличен, положительное — уменьшен.

Давайте посмотрим на соответствующие поля, показанные программой `ps`:

```
$ ps -t pts/0 -o pid,pri,ni,comm
  PID PRI  NI COMMAND
 2836  19   0  bash
 2849  19   0   ps
$
```

Как видно из примера, у программ, выполняющихся на терминале `pts/1`, значение поля `nice` равно нулю. Следовательно, эти программы выполняются со стандартным приоритетом, установленным самой системой.

В следующем примере будут показаны программы с увеличенным приоритетом, которые в данный момент работают в системе (В качестве примера я хотел использовать программу `ps` `ps -e -o pid,pri,ni,comm`», но почему-то значение поля приоритет выводилось неправильно. Нет в мире совершенства. Поэтому пришлось использовать `top` в командном режиме.)

```
$ top -b -nl | grep "\-" | sed -e '1d'
  2 root      15  -5    0    0    0 S   0.0   0.0   0:00.00  kthreadd
  3 root      RT  -5    0    0    0 S   0.0   0.0   0:00.00  migration/0
  4 root      15  -5    0    0    0 S   0.0   0.0   0:00.00  ksoftirqd/0
  5 root      15  -5    0    0    0 S   0.0   0.0   0:04.64  events/0
  6 root      15  -5    0    0    0 S   0.0   0.0   0:00.14  khelper
 98 root      15  -5    0    0    0 S   0.0   0.0   0:00.18  kblockd/0
101 root      15  -5    0    0    0 S   0.0   0.0   0:00.00  kacpid
102 root      15  -5    0    0    0 S   0.0   0.0   0:00.00  kacpi_notify
143 root      15  -5    0    0    0 S   0.0   0.0   0:00.00  ata/0
144 root      15  -5    0    0    0 S   0.0   0.0   0:00.00  ata_aux
145 root      15  -5    0    0    0 S   0.0   0.0   0:00.00  ksuspend_usbd
151 root      15  -5    0    0    0 S   0.0   0.0   0:00.00  khubd
154 root      15  -5    0    0    0 S   0.0   0.0   0:00.02  kseriod
189 root      15  -5    0    0    0 S   0.0   0.0   0:00.00  kswapd0
228 root      15  -5    0    0    0 S   0.0   0.0   0:00.00  aio/0
250 root      15  -5    0    0    0 S   0.0   0.0   0:00.00  jfsIO
.....
$
```

Как видите, таких программ достаточно много.

Любой пользователь системы может изменить значение поля `nice`. Но существуют два ограничения:

- Пользователь может изменить приоритет только у программ, выполняющихся с его правами.
- Пользователь может только понизить приоритет (записать положительное число в поле `nice`).

Как обычно, суперпользователь может все! Он может изменить приоритет любому процессу системы, он может уменьшать и увеличивать приоритеты.

Для изменения приоритета можно воспользоваться программой `top`. Сначала получим список всех процессов выполняемых пользователем `stavr`. Для краткости, заголовок `top` я буду удалять.

```
PID USER      PR  NI  VIRT  RES  SHR S %CPU %MEM    TIME+  COMMAND
2833 stavr    20   0  6516 1188  832 S   0.0   0.5   0:00.16 sshd
2836 stavr    20   0  3592 1968 1228 S   0.0   0.8   0:00.10 bash
2859 stavr    20   0  2344 1052  840 R   0.0   0.4   0:00.04 top
```

Для изменения приоритета процесса используйте команду `r`.

`PID to renice:`

```
PID USER      PR  NI  VIRT  RES  SHR S %CPU %MEM    TIME+  COMMAND
2833 stavr    20   0  6516 1188  832 S   0.0   0.5   0:00.24 sshd
2836 stavr    20   0  3592 1968 1228 S   0.0   0.8   0:00.10 bash
2859 stavr    20   0  2344 1052  840 R   0.0   0.4   0:00.66 top
```

Программа попросит ввести PID процесса, которому следует изменить приоритет. Введите PID и нажмите Enter.

`Renice PID 2859 to value:`

```
PID USER      PR  NI  VIRT  RES  SHR S %CPU %MEM    TIME+  COMMAND
2833 stavr    20   0  6516 1188  832 S   0.0   0.5   0:00.24 sshd
2836 stavr    20   0  3592 1968 1228 S   0.0   0.8   0:00.10 bash
2859 stavr    20   0  2344 1052  840 R   0.0   0.4   0:00.66 top
```

Теперь программа просит ввести значение, которое будет записано в поле `nice`. Введем число 10 и нажмем Enter.

```
PID USER      PR  NI  VIRT  RES  SHR S %CPU %MEM    TIME+  COMMAND
2833 stavr    20   0  6516 1188  832 S   0.0   0.5   0:00.26 sshd
2836 stavr    20   0  3592 1968 1228 S   0.0   0.8   0:00.10 bash
2859 stavr    30  10  2344 1052  840 R   0.0   0.4   0:00.68 top
```

Значение поля `nice` изменилось на 10. Соответственно изменилось и значение поля приоритета процесса.

Существуют еще две программы, которые можно использовать для изменения приоритета: `nice` и `renice`.

## nice

**Nice** запускает программу, сразу изменяя ее приоритет.

`nice [параметры] [программа [аргументы программы]...]`

При помощи параметра `-n` определяют значение поля `nice`, которое будет установлено у программы. Можно использовать числа от -20 до 19. Если значение явно не указано, по умолчанию используется 10.

Например, необходимо запустить `top` с уменьшением приоритета. В поле `nice` попытаемся записать число -5.

```
$ nice -n-5 top -b -n1
nice: cannot set niceness: Permission denied
$
```

Мы, работая простым пользователем, попытались увеличить приоритет программы. Этого делать нельзя, поэтому программа выдала сообщение об ошибке.

Теперь попробуем уменьшить приоритет программы.

```
$ nice -n5 top -b -n1 | grep " 5 "
2863 stavr    25   5  2340  960  752 R   0.0   0.4   0:00.02 top
$
```

Как видите, в поле `nice` было записано число 5, и соответственно приоритет программы был уменьшен.

## renice

Программа **renice** изменяет приоритет уже работающего процесса.

renice приоритет [[-p] PID ...] [[-g] группа ...] [[-u] пользователь ...]

Программе необходимо указать значение поля nice.

При помощи следующих параметров можно определить, каким процессам будем менять приоритет:

- **-p PID** — процессам с указанным PID. Можно написать несколько значений PID, разделяя их пробелами.
- **-g группа** — всем процессам, выполняемым с правами указанной группы.
- **-u пользователь** — всем процессам, выполняемым с правами указанного пользователя

Например, необходимо изменить значение поля nice у уже работающей программы top (я запустил программу top в другой виртуальной консоли).

```
$ top -b -nl -p 2868
  PID USER      PR  NI  VIRT  RES  SHR S %CPU  %MEM    TIME+  COMMAND
 2868 stavr      20   0 2344 1056  840 S   0.0   0.4   0:00.26 top
$
```

Воспользуемся программой renice.

```
$ renice 5 -p 2868
2868: old priority 0, new priority 5
$ top -b -nl -p 2868
  PID USER      PR  NI  VIRT  RES  SHR S %CPU  %MEM    TIME+  COMMAND
 2868 stavr      25   5 2344 1056  840 S   0.0   0.4   0:01.58 top
$
```

Как видите, в поле nice было записано значение 5.

## Временная остановка выполнения процесса

В Linux есть возможность временной остановки выполнения процесса. После остановки процесс остается в оперативной памяти, его данные не теряются. Просто он не будет попадать в очередь на выполнения процессора. В любой момент можно продолжить его выполнение.

Для того, чтобы временно остановить выполнение процесса, ему следует послать сигнал STOP(19).

Например, чтобы остановить работающую программу top, ей посылается сигнал.

```
$ kill -STOP 2868
$ ps -t pts/0 -o pid,s,comm
  PID S COMMAND
 2836 S bash
 2868 T top
$
```

Обратите внимание на поле состояние у процесса top, там стоит буква T. Это значит, что выполнение процесса приостановлено.

В этом месте я должен был написать следующее: «Чтобы продолжить выполнение процесса, пошлите ему сигнал CONT(18)». Так вот, даже если Вы пошлете процессу сигнал CONT, он все равно остается в приостановленном состоянии. Почему так происходит? Дело в том, что данный процесс «привязан» к терминалу, поэтому он не будет реагировать на 18-й сигнал. И возобновлять его работу можно только при помощи средств, встроенных в shell bash. На сигнал CONT будут реагировать только процессы, не «привязанные» к терминалу, например, процессы-демоны.

```
# ps -eo pid,s,comm | grep syslogd
2197 S syslogd
# kill -STOP 2197
# ps -eo pid,s,comm | grep syslogd
2197 T syslogd
# kill -CONT 2197
```

```
# ps -eo pid,s,comm | grep syslogd
 2197 S syslogd
#
```

Программа `syslogd` — это демон, не «привязанный» к терминалу. Сначала ему был послан сигнал `STOP`, и он перешел в состояние `T`. Затем ему послали сигнал `CONT`, и он вернулся в рабочее состояние (*состояние S означает, что процесс ожидает какой-либо операции ввода-вывода, а значит, он находится в рабочем состоянии.*).

В `shell bash` есть интересная встроенная возможность — `Job control` (управление заданиями). Пользователь может останавливать и продолжать выполнение программ. Оболочка «знает» обо всех программах, запущенных при ее помощи, и может выдать полный список всех работающих программ. Для получения списка используют команду `jobs`.

`jobs [-lnprs] [jobspec ...]`

Но прежде, чем мы рассмотрим использование управления заданиями, разберемся с `foreground` (интерактивный) и `background` (фоновый) режимами работы программ.

Когда Вы обычным образом запускаете на выполнение программу, она занимает терминал, точнее, стандартные ввод, вывод и вывод ошибки, связанные с этим терминалом. Пока эта программа не завершит свою работу, Вы не сможете в нем выполнять новые программы, так как не будет доступна командная строка. Такой режим работы называют `foreground`.

`Bash` позволяет запускать программы в фоновом режиме. В этом случае, программа начинает работать параллельно оболочке, на экране появляется приглашение командной строки, и Вы сможете запускать другие программы.

Для того, чтобы запустить программу в фоновом режиме, в командной строке после указания всех параметров программы напишите символ `&`.

**Внимание!** Если программа, запущенная в фоновом режиме, попытается что-либо прочитать со стандартного ввода, ее выполнение будет временно прекращено.

`Jobs` показывает только временно приостановленные и работающие в фоновом режиме программы. Также следует учитывать, что `shell` (а `jobs` — это встроенная команда `shell`) «видит» только те программы, которые были запущены из него. Если какая-то программа была запущена в фоновом режиме в другом терминале или другой программой, `jobs` текущего `shell` не покажет эту программу в своем списке.

Для того, чтобы временно остановить выполнение программы, выполняющейся в `foreground`-режиме, достаточно нажать комбинацию клавиш `Ctrl+Z`. Процесс тут же будет приостановлен, и на экране появится приглашение командной строки. Например, во время работы программы `top` была нажата указанная выше комбинация клавиш. На экран будет выведено сообщение о приостановке выполнения программы:

```
[1]+  Stopped                  top
$ ps -o pid,s,comm
    PID S COMMAND
  2836 R  bash
  2868 T  top
  2904 R  ps
$
```

Если теперь запустить `jobs`, он покажет список приостановленных или выполняющихся в фоновом режиме программ.

```
$ jobs
[1]+  Stopped                  top
$
```

`1` — это не `PID`, а внутренний номер программы. Если необходимо увидеть `PID` программ, тогда используйте параметр `-p`:

```
$ jobs -p
2868
$
```

Еще больше информации показывает `jobs` с параметром `-l`:



```
$ jobs -l
[1]+ 2868 Stopped (signal)          top
$
```

Символ плюс означает последнюю остановленную программу, минус — предпоследнюю.

Чтобы продолжить выполнение программы в foreground-режиме, можно использовать команду `fg`. Для продолжения выполнения в фоновом режиме — `bg`. Обеим программам в качестве параметра можно указывать номер задания — число, которое выводит `jobs`. Если номер не указан, команда работает с последним процессом.

```
$ fg
top
top - 20:12:45 up 12:03,  4 users,  load average: 0.00, 0.00, 0.00
Tasks:  70 total,   1 running,  69 sleeping,   0 stopped,   0 zombie
Cpu(s):  0.0%us,  0.3%sy,  0.0%ni, 99.7%id,  0.0%wa,  0.0%hi,  0.0%si,  0.0%st
Mem:    250756k total,  144440k used,  106316k free,    280k buffers
Swap:   250448k total,    0k used,  250448k free,  108612k cached

  PID USER      PR  NI  VIRT  RES  SHR S %CPU %MEM    TIME+  COMMAND
 2833 stavr    20   0  6516 1308  940 S   0.0   0.5   0:02.18 sshd
 2836 stavr    20   0  3592 2008 1268 S   0.0   0.8   0:00.18 bash
 2868 stavr    25   5  2344 1056  840 R   0.0   0.4   0:11.14 top
 2872 stavr    20   0  6516 1308  940 S   0.0   0.5   0:00.26 sshd
 2874 stavr    20   0  3592 1972 1228 S   0.0   0.8   0:00.08 bash
```

## Глава 8. Система помощи

Поскольку UNIX изначально разрабатывался как среда для разработки программного обеспечения, в нем сразу была предусмотрена мощная система подсказок. С дальнейшим развитием UNIX развивалась и система помощи.

### **man**

Основной системой помощи в Linux является набор «страниц руководства» (manual pages). Каждая «страница» выполнена в виде отдельного текстового файла, в котором для форматирования текста используются специальные управляющие последовательности (теги). Страницы руководства можно сравнить с HTML-файлами, но вместо тегов HTML используются свой собственный вариант разметки.

Для просмотра HTML-страниц используются специальные программы-браузеры. Точно так же, для просмотра страниц руководства используется специальный браузер — программа `man`.

`man [параметры] [секция] файл`

При вызове программы всегда указывается имя страницы руководства. Например, для получения справки по программе `man` в командной строке надо набрать: `man man`. В результате на экране будет показана соответствующая страница:

```
man (1) man (1)
```

#### NAME

*man - format and display the on-line manual pages*

#### SYNOPSIS

```
man [-acdfFhkKtwW] [--path] [-m system] [-p string] [-C config_file]
[-M pathlist] [-P pager] [-B browser] [-H htmlpager] [-S section_list]
[section] name ...
```

#### DESCRIPTION

*man formats and displays the on-line manual pages. If you specify section, man only looks in that section of the manual. name is normally the name of the manual page, which is typically the name of a command, function, or file. However, if name contains a slash (/) then man interprets it as a file specification, so that you can do man ./foo.5 or even man /cd/foo/bar.1.gz.*

*See below for a description of where man looks for the manual page files.*

#### MANUAL SECTIONS

*The standard sections of the manual include:*

.....

На самом деле `man` — это программа-координатор. Она сама не показывает содержимое страниц, она запускает целую цепочку действий, в результате которых на экране появляется содержимое страницы. В первую очередь `man` вызывает программу-парсер, задача которой состоит в превращении непонятных тегов в понятный пользователю текст. Ниже показано, как на самом деле выглядит страница руководства по программе `man`, точнее, только маленькая часть страницы.

```
.TH man 1 "September 2, 1995"
.LO 1
.SH NAME
man \- format and display the on-line manual pages
.SH SYNOPSIS
.B man
.RB [ \-acdfFhkKtwW ]
.RB [ --path ]
.RB [ \-m
.IR system ]
.RB [ \-p
```

```
.IR string ]
.RB [ \-C
.IR config_file ]
```

## Программа просмотра

В качестве парсера обычно используется одна из разновидностей программы `goff`. В Linux обычно используется `nroff` (в Linux `nroff` — это скрипт, который эмулирует работу одноименной программы. Для эмуляции он использует другую программу — `groff`).

После подготовки текста, он (текст) передается программе, в которой пользователь будет его смотреть. В Linux для просмотра используется программа `less`. Соответственно, при просмотре страницы руководства можно использовать все команды программы `less`.

Если, по каким-либо причинам, `less` вас не устраивает, для просмотра страниц руководства вы можете использовать другую программу. Чтобы указать, какая программа будет использована, ее имя можно передать `man` при помощи параметра `-P`. Например:

```
man -P /usr/bin/most man
```

Если вы предполагаете постоянно пользоваться другой программой просмотра, определите переменную среды окружения **PAGER**.

```
export PAGER=/usr/bin/most
```

Эту строку необходимо добавить в файл `/etc/profile` или `~/.bash_profile` (при условии, что вы пользуетесь `shell bash`).

## Директории

На любую программу в Linux есть своя собственная страница руководства. Программ много, значит, и страниц не меньше. Как уже говорилось выше, страницы руководства — это отдельные файлы, которые обычно находятся в директории `/usr/share/man`. Но, поскольку файлов много, их не хранят в одной директории. Если посмотреть содержимое директории `/usr/share/man`, в ней вы увидите директории `man1`, `man2` и так по `man9`. Каждая из перечисленных директорий предназначена для хранения определенного типа информации.

<b>man1</b>	Программы, доступные всем пользователям системы.
<b>man2</b>	Функции системы. Имеются в виду функции, которые программисты могут использовать в своих программах.
<b>man3</b>	Функции различных библиотек. Если в системе установлены отличные от C языки программирования, например Perl, описание функций этих языков тоже будут располагаться в этой директории.
<b>man4</b>	Описание устройств. К сожалению, в этой директории находится документации далеко не по всем устройствам, которые могут присутствовать в Linux. За более подробной информацией обращайтесь к документации, поставляемой с исходными кодами ядра Linux.
<b>man5</b>	Описание конфигурационных файлов. Формат многих конфигурационных файлов описывается в страницах руководства программы.
<b>man6</b>	Описание игр.
<b>man7</b>	Разное.
<b>man8</b>	Программы, доступные администратору системы.
<b>man9</b>	В Linux не используется.

## MANPATH

Когда в систему устанавливается новая программа, вместе с ней устанавливаются страницы руководства этой программы. Иногда эти страницы помещаются не в стандартные директории. Программе `man` можно указать, в каких директориях, кроме `/usr/share/man`, она должна искать страницы. Для этих целей используется переменная среды окружения **MANPATH**, в которой через двоеточие перечисляются дополнительные директории. Например, на моем рабочем компьютере переменная содержит следующие директории.

```
$ echo $MANPATH
/usr/local/man:/usr/man:/usr/lib/java/man
$
```

## Поиск в страницах руководства

Для поиска информации в страницах руководства предназначены программы `whatis` и `apropos`.

`whatis` слово

`apropos` слово

`Whatis` ищет искомое слово в специальной базе данных — `/usr/share/man/whatis`. Базу данных можно создать при помощи программы `makewhatis`. Обычно база создается автоматически, и вам навряд ли придется запускать `makewhatis` (на моей памяти, я ни разу ее не использовал).

Например, необходимо найти информацию о программе `kill`:

```
$ whatis kill
kill                (1)  - terminate a process
kill                (2)  - send signal to a process
kill [builtins]     (1)  - bash built-in commands, see bash(1)
$
```

Обратите внимание на число в скобках — это номер директории `man`, в которой находится страница руководства. То есть описание программы `kill` надо искать в `man1`, а функции `kill` — в `man2`. Если просто набрать `man kill`, вы всегда будете получать описание программы. Дело в том, что `man` ищет страницу руководства по порядку начиная с директории `man1` и заканчивая `man9`. Поскольку описание программы `kill` находится в первой директории, именно эта страница руководства будет показана `man`. Для того, чтобы увидеть описание функции `kill`, при вызове `man` следует явно указать номер директории, в которой необходимо искать страницу руководства.

```
$ man 2 kill
```

```
KILL(2)                                Linux Programmer's Manual                                KILL(2)
```

NAME

*kill - send signal to a process*

SYNOPSIS

```
#include <sys/types.h>
#include <signal.h>
```

```
int kill(pid_t pid, int sig);
```

*Feature Test Macro Requirements for glibc (see `feature_test_macros(7)`):*

```
kill(): _POSIX_C_SOURCE || _XOPEN_SOURCE
```

DESCRIPTION

*The `kill()` system call can be used to send any signal to any process group or process.*

.....

Программа `apropos` почти полностью повторяет функциональность программы `whatis`. Но ищет не слово целиком, а часть слова. Например, при попытке найти информацию о `kill`, Вы получите все слова, в которых встречается последовательность символов `kill`.

```

$ apropos kill
SDL_KillThread      (3) - Gracelessly terminates the thread
XKillClient [XSetCloseDownMode] (3) - control clients
kill                (1) - terminate a process
kill                (2) - send signal to a process
kill [builtins]     (1) - bash built-in commands, see bash(1)
killall             (1) - kill processes by name
killchar [curs_termattrs] (3x) - curses environment query routines
killpg              (2) - send signal to a process group
killwchar [curs_termattrs] (3x) - curses environment query routines
mysql_waitpid       (1) - kill process and wait for its termination
mysql_zap           (1) - kill processes that match a pattern
pkill [pgrep]       (1) - look up or signal processes based on name and other
attributes
pthread_kill        (3p) - send a signal to a thread
skill               (1) - send a signal or report process status
snice [skill]       (1) - send a signal or report process status
tgkill [tkill]      (2) - send a signal to a thread
timelimit           (1) - spawn a subprocess and if the child does not finish
within the time limit either kill it, or exit, leaving the child in the
background
tkill               (2) - send a signal to a thread
xkill               (1x) - kill a client by its X resource
yes                 (1) - output a string repeatedly until killed
$

```

## Русские man pages

В вашей системе могут быть установлены русские страницы руководства. Они обычно располагаются в директории `/usr/share/man/ru`. Увидеть их можно только в том случае, если в системе определены переменные локализации (LANG и LC\_\*), причем они должны указывать на русскую локаль. В дистрибутиве Slackware Linux русских страниц руководства крайне мало, это американский дистрибутив. Да они и не нужны. Если говорить о переводах страниц руководства, он (перевод) иногда запаздывает и может быть выполнен не качественно. Кроме того, существует большое количество версий одной и той же программы. Вы уверены, что русский перевод будет именно по установленной у вас версии?

## info

Вы, конечно, обратили внимание на то, что у страниц руководства есть один серьезный недостаток — у них нет гиперссылок. После просмотра информации приходится закрывать программу и запускать ее снова. В свое время сообществом GNU была разработана альтернативная страницам руководства система с поддержкой гиперссылок — `info`.

Но `info` не получила такого большого распространения, как страницы руководства, потому, что появился формат HTML. Большинство документации к программам доступно на сайтах в Интернете в виде HTML-страничек. Хотя, сообщество GNU все равно пытается продвигать `info`, и много информации, связанной с программированием (описание компиляторов, утилит сборки, базовых библиотек и т.д.), доступно в `info`.

## --help

Почти все программы, написанные сообществом GNU, поддерживают параметр `--help`. Он заставляет программу выводить краткую справку о программе.

```

$ /bin/pwd --help
Usage: /bin/pwd [OPTION]
Print the full filename of the current working directory.

```

```

--help      display this help and exit
--version   output version information and exit

```

NOTE: your shell may have its own version of `pwd`, which usually supersedes the version described here. Please refer to your shell's documentation for details about the options it supports.

```
Report bugs to <bug-coreutils@gnu.org>.  
$
```

## HOWTO

Существует большой проект по разработке документации для Linux — [Linux Documentation Project](http://www.tldp.org/) (<http://www.tldp.org/>) Один из разделов этого проекта — так называемые HOWTO (Как сделать). Это документация, написанная в свободном стиле, в обыкновенных текстовых файлах. В этих файлах написано как настроить сеть ([NET3-4-HOWTO](#)), как использовать модемы ([Modem-HOWTO](#)), как настроить сервер LDAP ([LDAP-HOWTO](#)) и так далее. Документов HOWTO очень много.

Существует переводы HOWTO на русский язык. К сожалению, переведены не все документы, многие из них устарели или имеют плохое качество перевода. Найти их можно на русскоязычных сайтах, посвященных Linux:

- <http://www.linux.ru/docs/>
- <http://www.linux.org.ru/books/>
- <http://www.opennet.ru/docs/5.shtml>

И других.

## Документация к программам

В состав пакета кроме страниц руководства может входить дополнительная документация. Она находится в директории `/usr/share/doc/имя_программы` (в *SuSE Linux документация к программам находится в директории `/usr/share/doc/packages`*). Поскольку большинство документации выполнено в виде страниц руководства, в этих директориях будет минимум файлов. Например, содержимое директории `apache-1.3.33`:

```
$ ls /usr/share/doc/apache-1.3.33  
ABOUT_APACHE  INSTALL  README          README.EAPI  
Announcement  LICENSE  README.configure  
$
```

Как видите, в ней очень мало файлов. Дело в том, что вся документация сервера Apache выполнена в виде HTML-страниц и находится в той директории, в которой располагаются HTML-файлы, раздаваемые сервером. В случае Slackware Linux это директория `/var/www/htdocs/manual`.

Но все же, в некоторых директориях вы найдете полноценную документацию. Например, с пакетом TeTeX поставляется большое количество файлов документации, выполненной как в виде HTML-файлов, так и в виде dvi-файлов (*Dvi — это формат документов, который получается в результате работы издательской системы TeX. Для его просмотра и конвертации в другие форматы существует большое количество программ.*).

## Глава 9. Текстовые редакторы

Учитывая то, что настройка Linux в подавляющем большинстве случаев предполагает редактирование конфигурационных файлов, любой администратор или пользователь системы просто обязан знать как минимум один текстовый редактор. Как бы ни старались производители некоторых дистрибутивов создавать различные мастер-программы по настройке системы, все равно при помощи мастера не получится полностью настроить Linux. Обязательно останутся какие-нибудь нюансы, которые приходится править руками.

Текстовых редакторов очень много. Они могут быть очень простыми, такими как `рiсo`. Они могут быть очень разносторонними, например, `emacs`. Они могут быть непривычными и непонятными, как `vi`. Существует большое количество редакторов, работающих только в графической оболочке, но кто сказал, что на сервере она (оболочка) будет установлена?

Несмотря на то, что редакторов много, настоящий Линуксоид обязательно должен знать один из них — `vi`. Несмотря на кажущуюся непонятность и странность этого редактора у него есть одно неоспоримое достоинство — он поставляется с любым UNIX. Он всегда будет у вас под рукой, работаете ли вы в полноценной системе или только с загрузочным диском.

Если вам кажется, что этот редактор вам не нужен, поверьте старому Юниксоиду — вы очень сильно заблуждаетесь.

### Редактор `vi`

Текстовый редактор `vi` — это один из самых распространенных редакторов в UNIX.

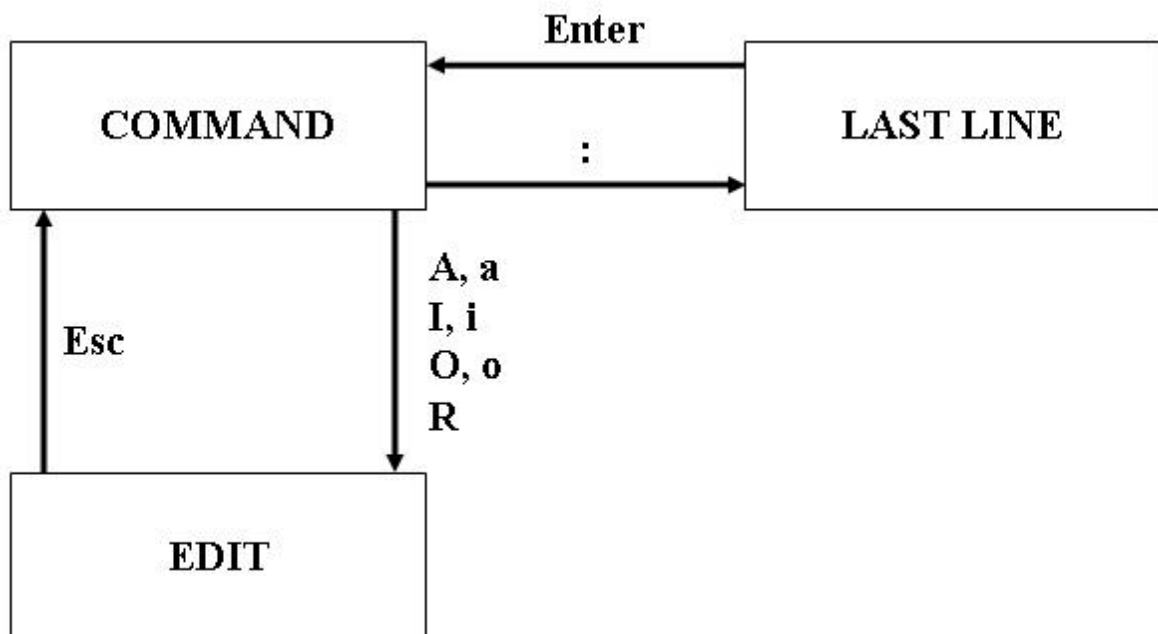
Существует большое количество клонов редактора. В Linux наиболее распространен редактор `vim`.

`vi` [опции] [файл]

Опции:

- `-г` или `-R` — открыть файл в режиме только для чтения.

### Режимы работы редактора `vi`



После включения, редактор находится в командном режиме.

В командном режиме любое нажатие на клавиши воспринимается как команда редактора.

Для входа в режим редактирования необходимо выполнить одну из команд.

Для возвращения в командный режим используют клавишу *ESC*.

Для входа в режим последней строки следует нажать клавишу ":".

## Команды редактора vi

Все команды вводятся только в командном режиме или в режиме последней линии.

Перед выполнением команды можно указать, сколько раз эта команда должна выполняться. Например, для выполнения команды *k* пять раз следует нажать на клавишу *5*, а затем на *k*.

- *h* — стрелка влево.
- *j* — стрелка вниз.
- *k* — стрелка вверх.
- *l* — стрелка вправо.
- Все ниже перечисленные команды переводят редактор в режим вставки текста.
  - *a* — перемещает курсор на одну позицию вправо.
  - *A* — перемещает курсор в конец строки.
  - *i* — не перемещает курсор.
  - *I* — перемещает курсор в начало строки.
  - *o* — добавляет новую строку под курсором и переводит редактор в режим редактирования.
  - *O* — добавляет новую строку перед курсором и переводит редактор в режим редактирования.
- Команды удаления
  - *x* — удаляет символ под курсором.
  - *dw* — удаляет слово под курсором.
  - *dd* — удаляет строку под курсором.
- Все команды отмены работают только в пределах одной строки.
  - *u* — отмена последнего действия.
  - *U* — отмена всех действий в текущей строке.
- *:q* — выход из редактора. Если файл не был сохранен, то выход не возможен.
- *:wq* — выход с сохранением.
- *ZZ* — выход с сохранением.
- *:q!* — выход без сохранения.
- *:w* — запись изменений.
- *:w file\_name* — сохранить файл под другим именем.
- *:1,100 w file\_name* — сохранить строки с первой по сотую в файле с другим именем.
- */ слово* — поиск слова “слово” в тексте.
- *:1,100 s/что/на что/g* — поиск и замена слова «что» на «на что» в строках с первой по сотую.
- *:%s/что/на что/g* — поиск и замена слова “что” на “на что” во всем тексте.

## Примеры использования vi

```
$ vi nosort
```

*выполните команды h j k l*

*выполните команду o и введите строку*

*нажмите Esc*

*выполните команду 5x*



*выйдите без сохранения - :q!*

*\$ vi nosort*

*удалите строку – dd*

*выйдите с сохранением – ZZ*

*\$ vimtutor*

## **Вместо заключения**

Я в своей работе использую текстовый редактор *joe*. Это довольно простая программа-оболочка с встроенной системой помощи. У нее несколько непривычный интерфейс, так как для управления используются наборы сочетаний клавиш. Однако к ней быстро привыкаешь и всегда можно получить помощь. Клавиши управления курсором, а также Backspace и Enter работают как и обычно.

В реальной жизни мне приходилось сталкиваться с самыми различными unix-подобными системами и часто, того редактора к которому привык, в используемой системе нет. Так например во FreeBSD моим любимым редактором является *ee*. В HP-UX, помучившись некоторое время, я нашел бинарный пакет редактора *ee* и теперь доволен. Однако, до тех пор, пока вы не настроите систему под себя, вам придется довольствоваться тем что есть. И знание работы редактора *vi* не раз выручало меня.

## Глава 10. Основы shell script

В shell встроен язык программирования, называемый shell script. Поскольку, существуют две основные разновидности интерпретаторов shell: Bourne shell и C shell, имеются две разновидности языков программирования, встроенных в эти оболочки.

Язык C shell, по своему синтаксису напоминает язык программирования C. Именно по этому оболочка и получила своё название. Язык Bourne shell — это самостоятельный язык со своим синтаксисом и особенностями.

Для чего администратору Linux необходимо знать shell script? Дело в том, что этот язык разрабатывался как вспомогательный инструмент администратора.

Одна из замечательных особенностей UNIX — это большой набор небольших утилит, каждая из которых выполняет свою задачу. При помощи конвейерной обработки команд эти утилиты можно объединять для выполнения более сложных задач. К сожалению, не все можно решить путем составления конвейеров, иногда требуется нечто большее. Можно написать программу на каком-либо языке программирования, но для этого мало того, что надо изучить этот язык, в системе должен присутствовать компилятор. Программы на shell script не надо компилировать, так как они выполняются интерпретатором, встроенным в оболочку. То есть shell script — это интерпретируемый язык.

Кроме того, вся система инициализации Linux (загрузка системы) построена с использованием набора файлов на языке shell script и администратор Linux может по своему усмотрению изменять эти файлы, но для этого необходимы хотя бы базовые знания языка.

Файл с программой на shell script — это набор команд, которые пользователь может вводить в командной строке. И если файл запустить на выполнение, будут выполнены все команды, описанные в этом файле. Но кроме простого выполнения команд в shell имеются дополнительные возможности, присущие многим языкам программирования: переменные, массивы, условные операторы, циклы и другое.

В данном курсе будут изучаться основы языка, встроенного в Bourne shell, т.к. именно эта разновидность используется в файлах системы инициализации Linux.

### Запуск приложений

Для того, чтобы файлы shell script можно было запускать, их следует сделать исполняемыми. В первой строке указать путь к интерпретатору, который будет вызван для выполнения программы. Поскольку в Linux наибольшее распространение получила разновидность Bourne shell — bash, в начале файла необходимо писать:

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

Существует еще один вариант запуска программ, написанных на shell script, причем файл с программой может быть не исполняемым. В этом случае необходимо быть уверенным в том, что в файле используются команды оболочки, в которой Вы сейчас работаете. Для запуска на выполнение программы в командной строке необходимо набрать:

```
. program
```

Тогда указанная программа будет выполнена экземпляром оболочки, в которой Вы сейчас работаете. То есть, для выполнения программы не будет порожден новый процесс. Правда, если в коде программы будет выполнен оператор exit — Вы выйдете из системы.

### Переменные

В shell script, как и в любом языке программирования можно использовать переменные. Переменные shell script — это переменные среды окружения программы интерпретатора (bash).

В первую очередь следует отметить, что переменные не типизированы. Все значения в переменных считаются строками. И только если переменная будет использоваться в математическом выражении, будет происходить проверка типа переменной.

В shell script нет такого понятия как «область видимости» переменных. Переменные доступны в любом месте кода, в том числе и в функциях. Если переменная будет определена в функции, она все равно будет глобальной переменной, и к ней можно будет обращаться из любого места программы.

**Внимание!** В bash версии 2 появилась возможность определять локальные переменные при помощи оператора `local`. Но эта возможность не является стандартной для других разновидностей Bourne shell.

Если обратиться к неопределенной переменной, интерпретатор не будет выдавать сообщение об ошибке. Просто будет подставлена пустая строка.

Ниже приводится содержимое файла `sample01` с пронумерованными строками.

```
1 #!/bin/bash
2 CAR="porsh the best"
3 echo "CAR:"
4 echo CAR
5 echo '$CAR: '
6 echo $CAR
```

В строке 1 указывается тип интерпретатора, который будет запущен для выполнения этой программы.

Для определения переменной необходимо написать имя переменной (большие и маленькие буквы отличаются), затем символ «равно» и значение переменной (строка 2). Если в значении переменной встречаются пробелы, значение необходимо поместить в двойные или одинарные кавычки.

```
CAR="porsh the best"
```

Для получения значения переменной, перед ее именем следует писать символ `$` (строка 6).

```
echo $CAR
```

В строке 3, на экран будет выведено — **CAR:**

В строке 4, на экран будет выведено — **CAR**

В строке 5 применяются одинарные кавычки. В shell script они имеют особое значение — все специальные символы, находящиеся внутри одинарных кавычек экранируются. То есть, символ `$` не будет интерпретироваться, как специальный символ и в результате на экран будет выведено **\$CAR:**

```
$ mcedit sample01
#!/bin/bash
CAR="porsh the best"
echo "CAR:"
echo CAR
echo '$CAR: '
echo $CAR
$ chmod 700 sample01
$ ./sample01
CAR:
CAR
$CAR:
porsh the best
$
```

## Массивы переменных

В shell script можно использовать массивы переменных. В классическом bourne shell существует одно ограничение интерпретатора — в массиве не может быть больше, чем 1024 элемента. В bash это ограничение снято.

Ниже приводится содержимое файла `sample02` с пронумерованными строками.

```
1 #!/bin/bash
2 #Массивы
3 #CAR[0]=porsh
4 #CAR[1]=bmw
5 #CAR[2]=mers
6 #CAR[3]=zaporozets
```

```

7 #CAR[10]=LADA
8 #CAR=([0]=porsh [1]=bmw [2]=mers [5]=zaporozets [10]=LADA)
9 CAR=(porsh bmw mers zaporozets LADA)
10 echo "*****"
11 echo "CAR[0]={CAR[0]}"
12 echo "CAR[1]={CAR[1]}"
13 echo "CAR[2]={CAR[2]}"
14 echo "CAR[3]={CAR[3]}"
15 echo "CAR[4]={CAR[4]}"
16 echo "*****"
17 echo "ALL - ${CAR[*]}"
18 echo "UHO - ${CAR[@]}"
19 echo "*****"

```

В bash существует несколько способов определения массива. В файле примера первые два способа закомментированы (символ # в начале строки).

В первом примере (строки с 3 по 7) определение элементов происходит путем указания имени массива. Затем в квадратных скобках указывается номер элемента в массиве и дальше ему присваивается значение как обычной переменной.

```
CAR[1]=bmw
```

Элементы массива определяются не по порядку: 0, 1, 2, 3, и 10. Если обратиться к несуществующему элементу массива, интерпретатор не будет выдавать ошибку.

Другой способ определения массива описан на строке 8.

```
CAR=([0]=porsh [1]=bmw [2]=mers [5]=zaporozets [10]=LADA)
```

В этом примере сначала пишется имя массива, а затем внутри круглых скобок элементам массива присваиваются значения.

Третий пример определения массива (строка 9) похож на предыдущий, но в нем не указываются номера элементов массива. Это значит, что значения будут присваиваться по порядку: сначала нулевому элементу, затем первому и т.д.

```
CAR=(porsh bmw mers zaporozets LADA)
```

Для получения значения элемента массива следует использовать следующую конструкцию:

```
${CAR[0]}
```

**Внимание!** Обратите внимание на фигурные скобки. В случае массивов их использование обязательно.

В примере есть две строки: 17 и 18, в которых выводится все содержимое массива. Для этого, вместо номера элемента массива необходимо указать либо символ @, либо \*.

```
${CAR[*]}
```

```
${CAR[@]}
```

```

$ mcedit sample02
#!/bin/bash
#Массивы
#CAR[0]=porsh
#CAR[1]=bmw
#CAR[2]=mers
#CAR[3]=zaporozets
#CAR[10]=LADA
#CAR=([0]=porsh [1]=bmw [2]=mers [5]=zaporozets [10]=LADA)
CAR=(porsh bmw mers zaporozets LADA)
echo "*****"
echo "CAR[0]={CAR[0]}"

```

```

echo "CAR[1]={CAR[1]}"
echo "CAR[2]={CAR[2]}"
echo "CAR[3]={CAR[3]}"
echo "CAR[4]={CAR[4]}"
echo "*****"
echo "ALL - ${CAR[*]}"
echo "ELE - ${CAR[@]}"
echo "*****"
$ chmod 700 sample02
$ ./sample02
*****
CAR[0]=porsh
CAR[1]=bmw
CAR[2]=mers
CAR[3]=zaporozets
CAR[4]=LADA
*****
ALL - porsh bmw mers zaporozets LADA
ELE - porsh bmw mers zaporozets LADA
*****
$

```

## Переменные окружения

Программа, написанная на языке shell script, имеет доступ к переменным окружения, может изменять их значение, а так же определять новые переменные окружения.

Ниже приводится содержимое файла sample03 с пронумерованными строками.

```

1 #!/bin/bash
2 #PATH=$PATH:~/bin; export PATH
3 #export PATH=$PATH:~/bin
4 echo "Workdir=$PWD"
5 echo "Этo UID=$UID"
6 echo "Bash level= $SHLVL"
7 echo "Random=$RANDOM"

```

В классическом варианте Bourne shell для создания новой переменной окружения сначала создается переменная оболочки, а затем она экспортируется. Для экспорта переменной используется оператор export (строка 2).

```

PATH=$PATH:~/bin
export PATH

```

В bash операции определения и экспортирования переменной могут происходить одновременно (строка 3).

```

export PATH=$PATH:~/bin

```

Переменные окружения будут доступны в текущем процессе, а также во всех порожденных этой программой процессах. В других процессах системы эти переменные не будут видны.

Для просмотра всех переменных окружения и функций можно воспользоваться командой set. Программа env покажет только переменные, а export — только переменные помеченные как экспортируемые.

Удаление переменных происходит при помощи оператора unset. Например:

```

unset CAR

```

```

$ mcedit sample03
#!/bin/bash
#PATH=$PATH:~/bin; export PATH
#export PATH=$PATH:~/bin
echo "Workdir=$PWD"
echo "Этo UID=$UID"
echo "Bash level= $SHLVL"
echo "Random=$RANDOM"

```

```
$ chmod 700 sample03
$ ./sample03
Workdir=/home/stavr
Это UID=1000
Bash level= 2
Random=4626
$
```

В shell имеется большое количество встроенных переменных. В таблице перечислены только некоторые из них. Описание всех встроенных переменных можно найти в справочном руководстве интерпретатора.

<i><b>Переменная</b></i>	<i><b>Описание</b></i>
HOME	Домашняя директория пользователя
LONGNAME	Регистрационное имя пользователя
MAILCHECK	Количество секунд, через которое будет происходить проверка наличия новых писем в почтовом ящике пользователя.
PATH	Содержит список директорий, разделенных двоеточием, в которых интерпретатор будет искать программу, если пользователь при запуске последней явно не указал путь к ней.
PS1	Внешний вид приглашения командной строки.
PS2	Внешний вид дополнительного приглашения командной строки.
SHELL	Содержит интерпретатор по умолчанию текущего пользователя.
TERM	Определяет тип терминала.
PWD	Содержит текущую директорию.
UID	UID пользователя, выполняющего программу.
SHLVL	Эта переменная увеличивается на 1, при следующем запуске shell. Учитываются только порожденные процессы.
RANDOM	При каждом чтении из переменной пользователь получает псевдослучайное число.

## Взаимодействие с пользователем

Для того, чтобы программа на shell script могла получить данные, вводимые пользователем с клавиатуры, используется специальный оператор read.

**read [переменная ...]**

При выполнении оператора read, на экране терминала появится курсор и пользователю дается возможность ввести данные. Ввод завершается нажатием на кнопку Enter.

Ниже приводится содержимое файла sample04 с пронумерованными строками.

```
1 #!/bin/bash
2 #REPLY test
3 #echo "Write a car name and press \"Enter\" :\"
4 echo -n 'Write a car name and press "Enter" :'
5 read
6 echo "Вы выбрали - $REPLY"
```

Обычно перед применением оператора read на экран выводится вопрос. Делается это при помощи программы echo. Если программе echo не указать опцию -n, она автоматически добавляет символ перевода строки после вывода данных. Поэтому строка 3 в примере закомментирована.

В 4-й строке приглашение выводится на экран. Выводимая строка взята в одинарные кавычки, которые используются для того, чтобы отменить специальное значение символов двойные кавычки, окружающие слово **Enter**.

Строка 5. Если оператор read вызывать без указания переменной, он все данные, введенные пользователем, поместит в переменную по умолчанию — **REPLY**.

В 6-й строке выводится содержимое этой переменной.

```
$ mcedit sample04
#!/bin/bash
#REPLY test
#echo "Write a car name and press \"Enter\" :\"
echo -n 'Write a car name and press "Enter" :\'
read
echo "Вы выбрали - $REPLY"
$ chmod 700 sample04
$ ./sample04
Write a car name and press "Enter" :Oka
Вы выбрали - Oka
$
```

## Подстановка

В shell script встроено очень мощное средство — подстановка данных, выводимых программой на стандартный вывод.

Для того, чтобы воспользоваться подстановкой необходимо взять программу в обратные одинарные кавычки. Например, так:

```
`date`
```

или в круглые скобки со знаком \$:

```
$(date)
```

В том месте кода, где используется подстановка, будет подставляться то, что программа вывела бы на стандартный вывод. Это значение динамическое, то есть, подставляются данные на момент выполнения скрипта.

Ниже приводится содержимое файла sample05 с пронумерованными строками.

```
1 #!/bin/bash
2 #Подстановка
3 echo "*****"
4 DATE=`date`
5 echo "DATE=$DATE"
6 echo "*****"
7 USERS=`who | wc -l`
8 echo "USERS v sisteme=$USERS"
9 echo "*****"
10 UP=$(date; uptime)
11 echo "Sostavnie=$UP"
12 echo "*****"
13 exit 0
```

В строке 4 переменной **DATE** присваивается то, что программа **date** вывела бы на стандартный вывод. В 5-й строке выводится содержимое этой переменной.

В подстановке можно использовать конвейер команд (строка 7). В результате будет использоваться то, что программа **wc** вывела бы на стандартный вывод.

В строке 10 в подстановке выполняются сразу две программы. В результате переменной UP будет присвоено то, что программы **date** и **uptime** вывели бы на стандартный вывод.

```
$ mcedit sample05
#!/bin/bash
#Подстановка
echo "*****"
```

```
DATE=`date`
echo "DATE=$DATE"
echo "*****"
USERS=`who | wc -l`
echo "USERS v sisteme=$USERS"
echo "*****"
UP=$(date; uptime)
echo "Sostavnie=$UP"
echo "*****"
exit 0
$ chmod 700 sample05
$ ./sample05
*****
DATE=Tue Sep  9 15:54:21 SAMST 2008
*****
USERS v sisteme=1
*****
Sostavnie=Tue Sep  9 15:54:21 SAMST 2008
15:54:21 up 2:41, 1 user, load average: 0.00, 0.00, 0.00
*****
$
```

## Арифметические выражения

Язык, встроенный в оболочку, в основном предназначен для операций с объектами файловой системы. Поэтому, хотя в нем и есть возможность использования арифметических выражений, можно пользоваться только целочисленной арифметикой и минимальным набором арифметических операций.

В арифметических выражениях можно использовать операторы: +, -, \*, / и круглые скобки. Так же можно использовать унарные операторы: ++ и --.

Для подстановки значения арифметического выражения его необходимо поместить в две круглые скобки, начинающиеся символом \$.

**`$(( 2*2 ))`**

В арифметическом выражении можно использовать переменные оболочки и окружения. При этом происходит проверка наличия в них целого числа. Если переменная не содержит целое число, ее значение в выражении принимается равным нулю.

Ниже приводится содержимое файла sample06 с пронумерованными строками.

```
1 #!/bin/bash
2 # Подстановка арифметических выражений
3 PERM=2
4 echo "2*2=$(( 2*$PERM ))"
5 echo "((2*3+5)-4)/2=$(( ((2*3+5)-4)/2 ))"
```

Если в примере переменной PERM присвоить строку, например, test, то в результате программа echo (строка 4) выведет на экран 0.

```
$ mcedit sample06
#!/bin/bash
# Подстановка арифметических выражений
PERM=2
echo "2*2=$(( 2*$PERM ))"
echo "((2*3+5)-4)/2=$(( ((2*3+5)-4)/2 ))"
$ chmod 700 sample06
$ ./sample06
2*2=4
((2*3+5)-4)/2=3
$
```



## Условный оператор *if*

В операторе *if* в качестве условия проверяется число — код возврата программы. Если программа была выполнена успешно — она возвращает ноль. Если во время выполнения программы возникли ошибки, она возвращает число, отличное от нуля.

Таким образом, в shell script истиной считается ноль. Все что не ноль — это ложь.

Оператор *if* всегда должен завершаться оператором *fi*.

Если проверяемое условие истина, тогда выполняется набор операторов, находящихся между ключевыми словами *then* и *fi*.

Используя оператор *else*, можно определить набор команд, которые будут выполняться в случае ложного значения.

Ниже приводится содержимое файла *sample07* с пронумерованными строками.

```

1 #!/bin/bash
2 # Пример if then else
3 if rm test 2> /dev/null
4 then
5     echo "Deleted"
6 else
7     echo "Not deleted"
8 fi
```

В строке 3 проверяется код возврата программы *rm*. Тут указываются все опции, которые будут переданы программе *rm* при ее запуске. Если программа смогла удалить файл *test*, ее код возврата будет равен нулю. Если по какой-то причине программа не сможет удалить файл, она вернет отличный от нуля код. При этом сообщение об ошибке выводиться не будет, т.к. стандартный вывод ошибки был перенаправлен в */dev/null*.

При коде возврата 0 будет выполнена строка 5. При коде возврата отличном от нуля — 7.

```

$ mcedit sample07
#!/bin/bash
# Пример if then else
if rm test 2> /dev/null
then
    echo "Deleted"
else
    echo "Not deleted"
fi
$ chmod 700 sample07
$ ./sample07
Deleted
$ ls -l test
/bin/ls: cannot access test: No such file or directory
$ ./sample07
Not deleted
$
```

Если в качестве условия оператора *if* используется выражение, помещенное в квадратные скобки, для разрешения этого условия будет вызвана программа *test*. Если будет проверять код возврата программы *test*.

Программа в файле *sample08* выполняет те же действия, что и в предыдущем примере, но для проверки условия существования файла вызывается программа *test* (строка 3).

```

1 #!/bin/bash
2 # Пример if then else с использованием test
3 if [ -w $HOME/bin -a -f $HOME/bin/test ]
4 then
5     rm $HOME/bin/test
```

```
6      echo "test deleted"
7 else
8      echo "test not deleted"
9 fi
```

**Внимание!** Обратите внимание на наличие пробелов после символа “[” и перед “]”. Они обязательны.

```
$ mcedit sample08
#!/bin/bash
# Пример if then else с использованием test
if [ -w $HOME/bin -a -f $HOME/bin/test ]
then
    rm $HOME/bin/test
    echo "test deleted"
else
    echo "test not deleted"
fi
$ chmod 700 sample08
$ ./sample08
test not deleted
$
```

## Проверка условий с помощью оператора test

Программа может проверить два типа логических условий И (AND) и ИЛИ (OR).

**Выражение1 -а Выражение2** — возвращает истину, если истинно и Выражение1 и Выражение2.

**Выражение1 -о Выражение2** — возвращает истину, если истинно или Выражение1 или Выражение2.

Оператор **!** инвертирует значение логического выражения.

Сравнение чисел происходит при помощи следующих операторов:

**число1 -eq число2** — истина, если числа равны.  
**число1 -ne число2** — истина, если числа не равны.  
**число1 -gt число2** — истина, если первое число больше второго.  
**число1 -ge число2** — истина, если первое число больше или равно второму.  
**число1 -lt число2** — истина, если первое число меньше второго.  
**число1 -le число2** — истина, если первое число меньше или равно второму.

Сравнение строк:

**-n строка** — истина, если строка имеет не нулевую длину.  
**-z строка** — истина, если строка имеет нулевую длину  
**строка1 = строка2** — истина, если строка1 идентична строке2. При сравнении строк учитывается регистр символов.

Проверка существования и типов файлов:

**-e /путь/к/файлу** — истина, если файл существует.  
**-f /путь/к/файлу** — истина, если файл существует и является обыкновенным файлом.  
**-d /путь/к/файлу** — истина, если файл существует и является директорией.  
**-L /путь/к/файлу** — истина, если файл существует и является символьной ссылкой.  
**-r /путь/к/файлу** — истина, если файл существует и доступен для чтения.  
**-w /путь/к/файлу** — истина, если файл существует и доступен на запись.  
**-x /путь/к/файлу** — истина, если файл существует и доступен на выполнение.  
**-s /путь/к/файлу** — истина, если файл существует и имеет не нулевую длину.

## Операторы && и ||

В shell script есть два оператора, предназначенные для проверки условий логическое И — &&, и логическое ИЛИ — ||.

Наиболее часто их применяют, когда необходимо проверить условие и, если оно истинно, выполнить одну команду или наоборот — не выполнять. Например:

```
[ -f file ] && rm file
```

Поскольку проверяется логическое И, необходимо чтобы оба условия были истинной. Поэтому, если первое условие истина, то будет проверяться второе условие, то есть будет вызвана программа `rm`.

```
[ -f file ] || touch file
```

В этом примере проверяется наличие файла `file`. Если его не существует (первое условие ЛОЖЬ), вызывается программа `touch`, которая его создает. Для того, что бы получилась ИСТИНА, хотя бы одно из условий должно вернуть значение ИСТИНА. Поэтому, если файл существует (ИСТИНА), программа `touch` не будет вызываться, так как нет необходимости в проверки второго условия. Если файл не существует (ЛОЖЬ), необходимо проверить второе условие — будет выполнена программа `touch`.

## Оператор case

Оператор `case` поочередно сравнивает строку с шаблонами. Если шаблон совпадает, то выполняется группа операторов, находящихся между шаблоном и специальными символами `;;`. После выполнения всех строк управление передается операторам, находящимся за ключевым словом `esac`.

Оператор `case` всегда завершается ключевым словом `esac`.

Ниже приводится содержимое файла `sample09` с пронумерованными строками.

```
1 #!/bin/bash
2 # Пример case esac
3 case $TERM in
4     *term)
5         echo "May be xterm?!"
6     ;;
7     unknown|vt[0-9]*)
8         echo "May be vt100 ?"
9     ;;
10    linux)
11        echo " This is a LINUX terminal!!!"
12    ;;
13    *)
14        echo "I don't know a this terminal :("
15 esac
16 exit 0
```

В строке 3 оператору `case` для проверки передается строка, находящаяся в переменной `TERM`.

В строке 4 происходит сравнение с шаблоном `*term)`. При написании шаблона можно использовать символы подстановки, такие же, как и в именах файлов. В данном шаблоне имеется в виду любая строка, заканчивающаяся на `term`. Если шаблон сработает, то будет выполнена команда в строке 5. Если шаблон не сработает, то произойдет проверка следующего шаблона.

В 7-й строке показано применение условного оператора ИЛИ — символ `|`. Таким образом, проверяется, соответствует ли строка слову `unknown` или начинается ли она на `vt` и цифру. Если шаблон срабатывает, то выполняется команда в строке 8. Если не срабатывает, то проверяется следующий шаблон.

В строке 10 проверяется, соответствует ли строка слову `linux`. Если соответствует, то выполняется команда в

строке 11. Если нет, то проверяется следующий шаблон.

Если ни один из перечисленных выше шаблонов не сработал (строка 13), можно в качестве шаблона использовать символ \*, что аналогично применению ключевого слова default языка программирования C.

```
$ mcedit sample09
#!/bin/bash
# Пример case esac
case $TERM in
    *term)
        echo "May be xterm?!"
        ;;
    unknown|vt[0-9]*)
        echo "May be vt100 ?"
        ;;
    linux)
        echo " This is a LINUX terminal!!!"
        ;;
    *)
        echo "I don't know a this terminal :("
esac
exit 0
$ chmod 700 sample09
$ ./sample09
May be xterm?!
$ echo $TERM
xterm
$
```

Ниже приводится код из файла sample10, в котором демонстрируется еще одна возможность применения оператора case для разбора ответа пользователя.

```
1 #!/bin/bash
2 # Пример case esac
3 echo -n "Please enter [Y|yes] : "
4 read YN
5 case $YN in
6     [yY]|[yY][eE][sS])
7         echo "You enter $YN"
8     ;;
9     *)
10        echo "You don't enter [Y|yes]"
11 esac
12 exit 0
```

В этом примере пользователя просят ввести Y или yes (строка 3). Ответ пользователя помещается в переменную YN (строка 4). При помощи оператора case проверяется, что он ввел. Поскольку регистр букв нас не интересует, шаблон на строке 6 проверяет различные варианты написания слова. Если было введено то значение, которое просили, выполняется команда на строке 7. Если было введено любое другое значение, срабатывает шаблон по умолчанию (строка 9) и выводится сообщение об ошибке (строка 10).

```
$ mcedit sample10
#!/bin/bash
# Пример case esac
echo -n "Please enter [Y|yes] : "
read YN
case $YN in
    [yY]|[yY][eE][sS])
        echo "You enter $YN"
    ;;

```

```

*)
    echo "You don't enter [Y|yes]"
esac
exit 0
$ chmod 700 sample10
$ ./sample10
Please enter [Y|yes] : Yes
You enter Yes
$

```

## Оператор for

При каждой итерации в операторе for, переменной присваивается следующее значение из списка, и выполняются все операторы, находящиеся между do и done.

Оператор работает до тех пор, пока не будет обработан весь список или в теле цикла не встретится оператор break.

Ниже приведен код из файла sample11, в котором показан пример простейшего цикла for.

```

1 #!/bin/bash
2 # Пример for
3 for I in 1 2 3 4 5 6 7 8 9 0
4 do
5     echo "--> $I <--"
6 done
7 exit 0

```

В этом примере переменной I подставляются значения из списка: 1, 2, 3, 4, 5, 6, 7, 8, 9 и 0. В результате на экран будут выведены 10 строк.

Элементы в списке могут разделяться символами пробела или табуляции. Если список не помещается на одну строку, его можно продолжить на следующей, но перед тем как нажать на Enter, поставьте символ \ для экранирования значения символа перевода строки. Например:

```

for I in list1 list2 list3 \
list5 list6

$ mcedit sample11
#!/bin/bash
# Пример for
for I in 1 2 3 4 5 6 7 8 9 0
do
    echo "--> $I <--"
done
exit 0
$ chmod 700 sample11
$ ./sample11
--> 1 <--
--> 2 <--
--> 3 <--
--> 4 <--
--> 5 <--
--> 6 <--
--> 7 <--
--> 8 <--
--> 9 <--
--> 0 <--
$

```

В цикле for в качестве списка можно указывать шаблон файловой системы. В этом случае в каждой итерации, переменной будет присваиваться путь к файлу, удовлетворяющий шаблону. Использование этого механизма показано в sample12.

```

1 #!/bin/bash
2 # Пример for
3 # Создание html файлов
4 #for FILES in `ls ~/wares*`
5 for FILES in ~/wares*
6 do
7     echo "<HTML>" > ${FILES}.html
8     echo "<HEAD><TITLE>${FILES}</TITLE></HEAD>" >> ${FILES}.html
9     echo "<BODY><PRE>" >> ${FILES}.html
10    cat ${FILES} >> ${FILES}.html
11    echo "</PRE></BODY></HTML>" >> ${FILES}.html
12    chmod a+r ${FILES}.html
13 done
14 exit 0

```

В строке 5 в качестве списка используется шаблон `~/wares*`. Это означает, что переменной `FILES` будет присваиваться файл, находящийся в домашней директории пользователя, имя которого начинается с `wares`. Этому условию соответствуют ранее созданные файлы: `wares.txt` и `wares2.txt`. То есть, будет две итерации, в каждой из которых переменная `FILES` будет иметь одно из перечисленных значений.

В строках с 7-й по 11-ю создается файл с таким же именем как и у исходного, но с расширением `.html`. В создаваемый файл помещается содержимое исходного файла.

После выполнения этого скрипта в домашней директории пользователя должны появиться файлы с расширением `.html`.

```

$ mcedit sample12
#!/bin/bash
# Пример for
# Создание html файлов
#for FILES in `ls ~/wares*`
for FILES in ~/wares*
do
    echo "<HTML>" > ${FILES}.html
    echo "<HEAD><TITLE>${FILES}</TITLE></HEAD>" >> ${FILES}.html
    echo "<BODY><PRE>" >> ${FILES}.html
    cat ${FILES} >> ${FILES}.html
    echo "</PRE></BODY></HTML>" >> ${FILES}.html
    chmod a+r ${FILES}.html
done
exit 0
$ chmod 700 sample12
$ ./sample12
$ ls -l wares*
-rw-r--r-- 1 stavr users  39 2008-09-05 21:25 wares.txt
-rw-r--r-- 1 stavr users 129 2008-09-09 17:02 wares.txt.html
-rw-r--r-- 1 stavr users  39 2008-09-05 21:20 wares2.txt
-rw-r--r-- 1 stavr users 130 2008-09-09 17:02 wares2.txt.html
$ cat wares.txt.html
<HTML>
<HEAD><TITLE>/home/stavr/wares.txt</TITLE></HEAD>
<BODY><PRE>
table:34
car:24
pen:32
apple:23
car:12
</PRE></BODY></HTML>

```

\$

## Получение данных из внешних файлов

Иногда возникает необходимость помещать конфигурационные данные программы shell script во внешние файлы. Существует несколько способов получения данных из внешних файлов. Один из них показан в sample13. Другие будут рассматриваться в следующих примерах.

Предположим, что в скрипте необходимо выполнить много одинаковых команд, в которых изменяется содержимое только одного параметра. Например, необходимо в правилах firewall описать несколько IP адресов машин, с которых можно получать доступ к определенным ресурсам. Строка, описывающая эти разрешения, может выглядеть следующим образом:

```
iptables -A FORWARD -s IP_ADDRESS -j ACCEPT
```

Наша задача вместо IP\_ADDRESS подставить значения IP адресов, находящихся во внешнем файле. В качестве такого файла возьмем sample13-data:

```
1 192.168.0.1
2 192.168.0.2
3 #192.168.0.3
4 192.168.0.4 # not pay
5 192.168.0.5
```

В этом файле приходится комментировать строку или часть строки (строки 3 и 4). Как это принято в Linux, для комментариев используется символ #.

В sample13 показано, как можно получить данные из файла:

```
1 #!/bin/bash
2 # Пример получения данных из внешнего файла
3 for I in `cat ./sample13-data`
4 do
5     echo "--> $I"
6 done
7 exit 0
```

Для этого применяется цикл for. В качестве списка берутся данные, которые будут выданы в результате подстановки: cat ./sample13-data. В итоге, на экран будут выведены не только IP адреса, но и все остальные значения, которые были в этом файле. Дело в том, что shell script — язык вспомогательный и в нем нет операторов работы со строками.

```
$ mcedit sample13-data
192.168.0.1
192.168.0.2
#192.168.0.3
192.168.0.4 # not pay
192.168.0.5

$ mcedit sample13
#!/bin/bash
# Пример получения данных из внешнего файла
for I in `cat ./sample13-data`
do
    echo "--> $I"
done
exit 0
$ chmod 700 sample13
$ ./sample13
--> 192.168.0.1
--> 192.168.0.2
--> #192.168.0.3
```

```
--> 192.168.0.4
--> #
--> not
--> pay
--> 192.168.0.5
$
```

Для получения правильного результата необходимо сначала осуществить фильтрацию данных при помощи других программ. Один из возможных примеров показан в sample13-2:

```
1 #!/bin/bash
2 # Пример получения данных из внешнего файла
3 for I in `cat ./sample13-data | cut -f1 -d ' ' | sed -e '/#/ d`
4 do
5     echo "--> $I"
6 done
7 exit 0
```

В нем сначала отбираются первые поля файла, а затем удаляются строки, содержащие символ #.

```
$ mcedit sample13-2
#!/bin/bash
# Пример получения данных из внешнего файла
for I in `cat ./sample13-data | cut -f1 -d ' ' | sed -e '/#/ d`
do
    echo "--> $I"
done
exit 0
$ chmod 700 sample13-2
$ ./sample13-2
--> 192.168.0.1
--> 192.168.0.2
--> 192.168.0.4
--> 192.168.0.5
$
```

## Оператор while

В цикле while выполняются строки, расположенные между do и done, до тех пор, пока условие истинно или пока не встретится оператор break или exit.

Самый простой пример использования оператора while показан в файле sample14:

```
1 #!/bin/bash
2 # Primer while
3 X=1
4 while [ $X -lt 10 ]
5 do
6     echo "--> $X <--"
7     X=$(( $X+1 ))
8 done
9 exit 0
```

Сначала переменной X присваивается значение 1 (строка 3). Затем проверяется: X меньше 10? Если условие истинно, то выполняются строки 6 и 7. В 7-й строке значение X увеличивается на 1 и снова проверяется условие. В результате работы скрипта на экран будет выведено девять строк.

```
$ mcedit sample14
#!/bin/bash
```



```
# Primer while
X=1
while [ $X -lt 10 ]
do
    echo "--> $X <--"
    X=$(( $X+1 ))
done
exit 0
$ chmod 700 sample14
$ ./sample14
--> 1 <--
--> 2 <--
--> 3 <--
--> 4 <--
--> 5 <--
--> 6 <--
--> 7 <--
--> 8 <--
--> 9 <--
$
```

## Оператор select

Оператор select выводит пронумерованный список на стандартный вывод и строку приглашения, в которой пользователь должен ввести номер элемента списка и нажать Enter. Затем значение выбранного элемента присваивается переменной и выполняются строки, расположенные между do и done. После этого опять выводится список, либо выводится приглашение на ввод номера элемента (зависит от версии shell).

Выход из цикла возможен путем явного вызова операторов break или exit.

Пример использования оператора select находится в файле sample15:

```
1 #!/bin/bash
2 # Пример оператора select
3 select FILE in ~/w* QUIT
4 do
5     if [ -e $FILE ]
6     then
7         ls -l $FILE
8     else
9         break
10    fi
11 done
12 exit 0
```

Оператор select выводит пронумерованные значения списка. В список попадут все файлы, находящиеся в домашней директории пользователя, начинающиеся на w и слово QUIT. Когда на экране появится приглашение ввода, следует ввести номер элемента и нажать Enter. После этого переменной FILE будет присвоено значение, соответствующее номеру и будут выполнены строки, находящиеся между do и done.

В строке 5 проверяется условие: «а существует ли такой файл?». Причем тип файла не имеет значения. Если он существует, то выполняется программа ls (строка 7). Затем снова появляется либо список, либо приглашение ввода. Если файл не существует, а это условие может быть выполнено, только если был выбран номер, соответствующий слову QUIT, будет выполнен оператор break (строка 9) и программа выйдет из цикла select.

```
$ mcedit sample15
#!/bin/bash
# Пример оператора select
select FILE in ~/w* QUIT
do
```

```

        if [ -e $FILE ]
        then
            ls -l $FILE
        else
            break
        fi
done
exit 0
$ chmod 700 sample15
$ ./sample15
1) /home/stavr/wares.txt           4) /home/stavr/wares2.txt.html
2) /home/stavr/wares.txt.html     5) QUIT
3) /home/stavr/wares2.txt
#? 1
-rw-r--r-- 1 stavr users 39 2008-09-05 21:25 /home/stavr/wares.txt
#? 5
$

```

## Оператор «точка». Функции.

Оператор точка позволяет в текущем интерпретаторе выполнить код shell script, находящийся в другом файле.

По своей сути оператор точка похож на инструкцию include языка программирования C. Но при включении другого shell script файла ему можно передавать параметры командной строки, как обычной программе при ее выполнении.

Оператор точка очень часто используют для включения конфигурационных параметров, находящихся во внешних конфигурационных файлах. Например, существует файл со следующим содержанием:

**PARAM=value**

**PARAM2=value2**

Для того, чтобы воспользоваться этими параметрами в другом файле, необходимо подключить текст первого файла. Подключаемый файл может быть не исполняемым.

**. file**

**echo \$PARAM**

**echo \$PARAM2**

В примере sample16 показано использование оператора точка.

**1 #!/bin/bash**

**2 # Пример использования функций и оператора "."**

**3 if [ ! -f \$HOME/sample16-2 ]; then**

**4 exit 1**

**5 fi**

**6 . \$HOME/sample16-2**

**7 select FILE in ~/.\* QUIT**

**8 do**

**9 if [ -f \$FILE ]**

**10 then**

**11 any**

**12 else**

**13 break**

**14 fi**

**15 done**

**16 exit 0**

По своей функциональности данный пример выполняет те же действия, что и sample15, но в нем используются две интересные особенности.

В строке 3 проверяется наличие дополнительного файла sample16-2, того файла, который будет подключаться в строке 6. Если его нет, то работа скрипта завершается оператором exit, с кодом возврата 1, свидетельствующем об ошибке, возникшей при выполнении программы.

В sample16-2 показан пример определения функции:

```

1 #!/bin/bash
2 # Определение функции
3 # function
4 any()
5 {
6     ls -l $FILE
7 }
```

Функцию в shell script можно определить двумя способами: при помощи оператора function или после имени функции написать открывающую и закрывающую круглые скобки. Тело функции располагается между фигурными скобками.

Поскольку все переменные в shell script являются глобальными, внутри тела функции можно пользоваться переменными, определенными в любом месте shell script. А также определять новые переменные, которые можно использовать в любом месте программы. Единственным исключением являются позиционные переменные, о которых будет рассказано далее.

Если в функцию необходимо передать какие-либо параметры, они пишутся после имени функции так же, как аргументы командной строки. Получить эти параметры внутри функции можно при помощи позиционных переменных.

```

$ mcedit sample16-2
#!/bin/bash
# Определение функции
# function
any()
{
    ls -l $FILE
}

$ mcedit sample16
#!/bin/bash
# Пример использования функций и оператора "."
if [ ! -f $HOME/sample16-2 ]; then
    exit 1
fi
. $HOME/sample16-2
select FILE in ~/.* QUIT
do
    if [ -f $FILE ]
    then
        any
    else
        break
    fi
done
exit 0
$ chmod 700 sample16
$ ./sample16
1) /home/stavr/.                6) /home/stavr/.mc
2) /home/stavr/..              7) /home/stavr/.screenrc
3) /home/stavr/.bash_history   8) /home/stavr/.xsession
4) /home/stavr/.joe_state      9) QUIT
```

```
5) /home/stavr/.lessht
#? 3
-rw----- 1 stavr users 3902 2008-09-08 21:29 /home/stavr/.bash_history
#? 9
$
```

## Специальные переменные

В shell script встроены специальные переменные. В первую очередь — это позиционные переменные. При их помощи можно получить значение параметров, переданных при вызове программы в командной строке.

**\$0 \$1 ... \$9** — позиционные переменные.

- **\$0** — имя программы.
- **\$1** — первый параметр командной строки.
- **\$2** — второй параметр командной строки и т.д.
- **\$#** — количество параметров командной строки, переданных программе. Имя самой программы не учитывается.
- **\$\*** и **\$@** — все параметры командной строки.

Есть существенное различие между переменными **\$\*** и **\$@**. Предположим, что программа была запущена следующим образом:

```
program -v -f «The file»
```

Если внутри этой программы попытаться получить список всех переменных, например, в цикле `for`, можно написать одну из перечисленных ниже строк:

```
for I in $* # четыре итерации.
```

```
for I in $@ # четыре итерации.
```

```
for I in «$@» # три итерации.
```

Как видно из примера, удовлетворительный результат можно получить только используя «**\$@**», обязательно поместив переменную в двойные кавычки.

Другие специальные переменные.

- **\$?** — код возврата последней выполненной программы.
- **\$!** — PID последней программы, запущенной в background режиме.
- **\$\$** — PID процесса shell, исполняющего данный shell script.

Пример использования позиционных переменных показан в sample17:

```
1 #!/bin/bash
2 # Primer ispolzovanija $0
3 case $0 in
4     *listtar)
5         echo "List archive $1 ..."
6         TARGS="-tvf $1"
7     ;;
8     *maketar)
9         echo "Create archive $1.tar ..."
10        TARGS="-cvf $1.tar $1"
11    ;;
12    *) echo "Usage: listtar file | maketar dir"
13    exit 88
14 esac
```

**15 tar \$TARGS****16 exit 0**

В Linux довольно часто программам необходимо передавать большое количество аргументов командной строки. Причем, все время программе передаются одни и те же аргументы командной строки. Чтобы для каждого конкретного случая не писать свой скрипт, запускающий программу, используют следующий способ запуска программ. На нее делают несколько символьных ссылок. И запускают программу, пользуясь этими ссылками. Программа видит, с каким именем ее запускают (по имени ссылки) и в зависимости от имени автоматически включаются необходимые функции.

В shell script тоже можно проследить, с каким именем запускают файл сценария. Для этого используют позиционную переменную \$0. В приведенном примере на строке 3 оператор case контролирует имя, с которым вызвана программа. При помощи шаблонов отслеживается два возможных имени: listtar (строка 4) и maketar (строка 8). Поскольку мы не знаем, какой путь будет указан при запуске программы (а в \$0 попадает и имя и путь), в шаблоне перед именем стоит символ \*.

Скрипт предназначен для просмотра содержимого и создания tar архивов. Поэтому, в случае listtar, программе необходимо передать имя просматриваемого архива. Это имя попадает в переменную \$1. На строке 5, на экран выводится сообщение «List archive \$1 ...», где вместо \$1 подставляется имя архива. А на строке 6 переменной TARGS присваиваются параметры, которые будут переданы программе tar. После этого управление переходит на строку 15.

В случае maketar, программа будет создавать архив. В качестве аргумента следует передать имя директории. Это имя попадет в переменную \$1. На строке 9 на экран выводится сообщение об имени создаваемого архива. А на 10-й строке формируются параметры, которые будут переданы программе tar. После этого управление передается на 15-ю строку.

Если программу вызвать под каким-либо другим именем, то сработает шаблон по умолчанию (строка 12). При этом будет выведено сообщение об ошибке и мы выйдем из программы с кодом ошибки 88 (строка 13).

Для того, чтобы скрипт заработал, необходимо создать две символьных ссылки: listtar и maketar. И вызывать его только по этим именам. Если скрипт вызвать по его имени — sample17, будет выдано сообщение об ошибке.

```
$ mcedit sample17
#!/bin/bash
# Primer ispolzovaniya $0
case $0 in
    *listtar)
        echo "List archive $1 ..."
        TARGS="-tvf $1"
    ;;
    *maketar)
        echo "Create archive $1.tar ..."
        TARGS="-cvf $1.tar $1"
    ;;
    *) echo "Usage: listtar file | maketar dir"
    exit 88
esac
tar $TARGS
exit 0
$ chmod 700 sample17
$ ./sample17
Usage: listtar file | maketar dir
$ ln -s sample17 maketar
$ ./maketar new_file
Create archive new_file.tar ...
new_file
$ ln -s sample17 listtar
$ ./listtar new_file.tar
List archive new_file.tar ...
-rw----- stavr/prl          0 2008-09-08 14:26 new_file
$
```

**Внимание!** Какой код ошибки возвращает программа решает программист.

## Использование программы *getopts*

*getopts* — это встроенная в shell команда, позволяющая разобрать командную строку, передаваемую программе.

Она понимает только параметры, написанные в стиле POSIX, то есть, параметр должен состоять из одной буквы, перед которой необходимо поставить тире. Например: -v, -t, -f file и т.п. При определении параметра символ ":" означает, что параметр должен иметь дополнительное значение.

Пример использования программы показан в *sample18*.

```
1 #!/bin/bash
2 while getopts f:o:v OPTION
3 do
4     case $OPTION in
5         f) echo "Option f - argument $OPTARG" ;;
6         o) echo "Option o - argument $OPTARG" ;;
7         v) echo "Option v - no argument" ;;
8         \?) echo "Usage: `basename $0` -f infile [-o outfile] [-v]"
9     esac
10 done
11 exit 0
```

Программу *getopts* вызывают как условие цикла *while*. Она пытается найти аргумент командной строки и, если такой аргумент есть, программа помещает его в переменную *OPTION* и возвращает истину.

После завершения итерации цикла *while*, снова вызывается программа *getopts*. Она ищет следующий аргумент командной строки и если находит, то все повторяется как и в предыдущем случае. Если аргумент не находится, то программа возвращает ложь, и мы выходим из цикла *while*.

Если у аргумента командной строки присутствует дополнительный параметр, *getopts* помещает этот параметр в специальную переменную *OPTARG*.

Во время выполнения программы *getopts* могут возникать следующие ошибки:

- Указана не определенная в параметрах программы опция.
- У аргумента командной строки не указан обязательный дополнительный параметр.

Если при работе программы возникает ошибка, она выводит сообщение об ошибке на *stderr*, а в переменную *OPTION* помещает символ ?.

```
$ mcedit sample18
#!/bin/bash
while getopts f:o:v OPTION
do
    case $OPTION in
        f) echo "Option f - argument $OPTARG" ;;
        o) echo "Option o - argument $OPTARG" ;;
        v) echo "Option v - no argument" ;;
        \?) echo "Usage: `basename $0` -f infile [-o outfile] [-v]"
    esac
done
exit 0
$ chmod 700 sample18
$ ./sample18 -a
./sample18: illegal option -- a
Usage: sample18 -f infile [-o outfile] [-v]
$ ./sample18 -f testfile -o newtest -v
Option f - argument testfile
Option o - argument newtest
Option v - no argument
$
```

## Оператор *trap*

Оператор *trap* позволяет переопределить стандартную реакцию программы на получаемые сигналы.

В качестве первой опции необходимо указать команду, которая будет выполнена при получении сигнала. В качестве команды можно использовать функцию. Затем указать список сигналов, разделенных пробелами.

Пример использования оператора *trap* находится в файле *sample19*:

```

1 #!/bin/bash
2 trap clean 2
3 clean() {
4     X=1
5     echo "Start formatting /dev/hda3:"
6     while [ $X -lt 10 ]
7     do
8         echo -n ".."
9         sleep 2
10        X=$(( $X+1 ))
11    done
12    echo "Done"
13    exit 0
14 }
15 while [ 0 ]
16 do
17     :
18 done
19 exit 0

```

На строке 2 вызывается оператор *trap*, определяющий, что при получении программой сигнала 2 будет выполнена функция *clean*.

Основное тело программы представляет из себя бесконечный цикл *while* (строки 15-18), где условие всегда будет истина. И программа никогда не завершится. Поскольку в цикле *while* между *do* и *done* необходимо написать какие-либо операторы, используется пустой оператор — *:*.

Функция *clean* выведет на экран сообщение «Start formatting /dev/hda3:», а затем с задержкой в две секунды (оператор *sleep*, строка 9) будет выводить две точки. В конце функции выполняется оператор *exit* (строка 13), который завершает работу программы.

```

$ mcedit sample19
#!/bin/bash
trap clean 2
clean() {
    X=1
    echo "Start formatting /dev/hda3:"
    while [ $X -lt 10 ]
    do
        echo -n ".."
        sleep 2
        X=$(( $X+1 ))
    done
    echo "Done"
    exit 0
}
while [ 0 ]
do

```

```
        :  
        done  
exit 0  
$ chmod 700 sample19  
$ ./sample19
```

Программа «зависает» - терминал занят программой, но на стандартный вывод и вывод ошибки не поступает никакой информации. Обычное действие, которое в таком случае выполняет пользователь — посылает программе сигнал 2, то есть нажимает комбинацию клавиш <Ctrl>+C.

```
<Ctrl>+C  
Start formatting /dev/hda3:  
.....Done  
$
```



## Глава 11. Установка дистрибутива

Установка любого дистрибутива Linux состоит из нескольких этапов:

- Определение требований для компьютера
- Подготовка (разбиение) диска
- Инсталляция системы

### Требования к компьютеру

Судя по последним статьям в Интернет, ОС Linux в наши дни умеет работать на любом устройстве, в составе которого имеется микропроцессор, то есть практически "на всем, что движется". Правда, для этого требуются нестандартные дистрибутивы. Если же взять любую из последних версий стандартных дистрибутивов, то требования к аппаратному обеспечению предъявляются достаточно высокие. И объемы занимаемого такими дистрибутивами дискового пространства тоже впечатляют - не менее 5 ГБайт.

Вместе с тем, Linux можно установить и на очень "слабые" по сегодняшним понятиям компьютеры. Существуют специальные версии Linux, которые работают на системах следующей конфигурации:

CPU: 386

RAM: 4 MB

HDD или даже FDD

то есть загружаются с дискеты (смотри список дистрибутивов на сервере <http://distrowatch.com/>).

Для комфортной работы с графическим интерфейсом X Window нужно уже 64 MB памяти (на 8 MB тоже можно запуститься, но ожидание, пока проходит подкачка из swap-памяти, просто изматывает). Занимаемое место на жестком диске зависит от Ваших потребностей - полная установка некоторых дистрибутивов может занять больше 5 гигабайт, а для минимального старта и простой работы в shell достаточно 100 MB.

Таким образом, если Вы собираетесь серьезно работать с Linux - Вам потребуется как минимум Pentium/128 MB RAM/10 GB HDD. Как видите, это примерно те же самые требования, которые предъявляются к компьютеру, на который устанавливается Microsoft Windows. Однако Linux предоставляет гораздо более широкие возможности. А если у Вас есть старенький компьютер, на котором никакая Windows не запускается, то Вы с успехом можете использовать его для освоения Linux и, возможно, будете удивлены его возможностями.

### Загрузка

Обычно загрузка системы производится с CD/DVD носителя. В нашем случае мы используем DVD диск содержащий дистрибутив Slackware Linux 12.1.

В процессе загрузки нам будет предложено выбрать ранее скомпилированное ядро системы. Если вы используете многопроцессорную систему — просто нажмите клавишу «Enter» в ответ на приглашение системы

*boot:*

Если вы используете относительно старую или однопроцессорную систему, то вам рекомендуется выбрать ядро `huge.s`

```
ISOLINUX 3.52 2007-09-25 Copyright (C) 1994-2007 H. Peter Anvin

Welcome to Slackware version 12.1 (Linux kernel 2.6.24.5)!

If you need to pass extra parameters to the kernel, enter them at the prompt
below after the name of the kernel to boot (huge.s etc). NOTE: If your machine
is not at least a Pentium-Pro, you *must* boot and install with the huge.s
kernel, not the hugesmp.s kernel! For older machines, use "huge.s" at the
boot prompt.

In a pinch, you can boot your system from here with a command like:

boot: hugesmp.s root=/dev/hda1 rdinit= ro

In the example above, /dev/hda1 is the / Linux partition.

This prompt is just for entering extra parameters. If you don't need to enter
any parameters, hit ENTER to boot the default kernel "hugesmp.s" or press [F2]
for a listing of more kernel choices.

boot: huge.s_
```

Далее просто согласитесь с предложением использовать английскую раскладку клавиатуры и нажмите клавишу «Enter» в ответ на приглашение ввода логина (имени учетной записи). По умолчанию используется логин root.

## Подготовка (разметка) жесткого диска

После загрузки Вам будет необходимо разметить жёсткий диск. Надо создать разделы, куда вы будете устанавливать операционную систему. Как минимум, рекомендуется создать два раздела; один для корневой файловой системы (/) и один для подкачки (swap space).

После того, как произойдет загрузка с диска вам предоставят приглашение для входа в систему (login). Войдите как root (вам не нужен пароль для этого). В приглашении командной строки наберите fdisk. Ниже мы кратко рассмотрим программу fdisk.

Начнём с запуска fdisk для выбранного вами жёсткого диска. В Linux жёстким дискам не присваиваются буквенные обозначения, но каждому диску соответствует определённый файл. Первый IDE диск (primary master) называется /dev/hda; primary slave - /dev/hdb, и так далее. SCSI диски определяются по такой же системе, но в виде /dev/sda.

Что бы получить информацию об имеющихся на вашем компьютере жестких дисках необходимо выполнить команду

```
fdisk -l
```

```
root@slackware:~# fdisk -l

Disk /dev/hda: 17.1 GB, 17179869184 bytes
255 heads, 63 sectors/track, 2088 cylinders
Units = cylinders of 16065 * 512 = 8225280 bytes

   Device Boot      Start         End      Blocks    Id System
/dev/hda1   *           1         653     5245191    7  HPFS/NTFS

root@slackware:~# _
```

Вам необходимо запустить fdisk в применении к выбранному жёсткому диску (я использую для примера IDE primary master):

```
fdisk /dev/hda
```

```

root@slackware:~# fdisk /dev/hda

The number of cylinders for this disk is set to 2088.
There is nothing wrong with that, but this is larger than 1024,
and could in certain setups cause problems with:
 1) software that runs at boot time (e.g., old versions of LILO)
 2) booting and partitioning software from other OSs
    (e.g., DOS FDISK, OS/2 FDISK)

Command (m for help): _

```

Как и все хорошие Unix программы, fdisk выдаст вам приглашение командной строки (а вы думали, что получите меню, не так ли?). Первое, что вам необходимо сделать — познакомиться с возможностями программы. Fdisk предлагает использовать клавишу m для получения справки:

```

Command (m for help): m
Command action
  a   toggle a bootable flag
  b   edit bsd disklabel
  c   toggle the dos compatibility flag
  d   delete a partition
  l   list known partition types
  m   print this menu
  n   add a new partition
  o   create a new empty DOS partition table
  p   print the partition table
  q   quit without saving changes
  s   create a new empty Sun disklabel
  t   change a partition's system id
  u   change display/entry units
  v   verify the partition table
  w   write table to disk and exit
  x   extra functionality (experts only)

Command (m for help): _

```

Далее необходимо проверить уже существующие разделы. Мы сделаем это, напечатав p в командной строке программы:

*Command (m for help): p*

```

Command (m for help): p

Disk /dev/hda: 17.1 GB, 17179869184 bytes
255 heads, 63 sectors/track, 2088 cylinders
Units = cylinders of 16065 * 512 = 8225280 bytes

   Device Boot      Start         End      Blocks    Id  System
/dev/hda1   *           1         653       5245191    7  HPFS/NTFS

Command (m for help): _

```

Программа выведет на экран всю информацию о существующих метках и разделах. Обычно выбирают свободный диск для установки системы, в противном случае удаляют все существующие на нём метки и разделы.

**Внимание!** ОЧЕНЬ ВАЖНО СОЗДАТЬ РЕЗЕРВНЫЕ КОПИИ ВСЕЙ ИНФОРМАЦИИ, КОТОРУЮ ВЫ ЖЕЛАЕТЕ СОХРАНИТЬ ДО ТОГО, КАК УДАЛИТЬ РАЗДЕЛ НА КОТОРОМ ОНА НАХОДИТСЯ.

Не существует простого способа восстановления данных после удаления раздела жёсткого диска, так что создайте резервную копию до того, как играть с разделами.

В таблице разделов вы увидите номер раздела, его размер и тип. Вы найдёте больше информации на экране, но пока она вам не понадобится. Для удаления раздела используется команда d:

```
Command (m for help): d
```

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

Для нашего случая мы оставим существующий раздел MS Windows и создадим новые разделы. Следует отметить, что схемы разбиения диска для Unix систем являются предметом многочисленных споров, и большинство пользователей расскажет вам лучший способ сделать это. Со временем вы узнаете больше о схеме разбиения жёсткого диска, подходящей для вашей системы.

Для создания раздела используется команда n:

```
Command (m for help): n
```

```
Command (m for help): n
Command action
  e   extended
  p   primary partition (1-4)
p
Partition number (1-4): 2
First cylinder (654-2088, default 654):
Using default value 654
Last cylinder or +size or +sizeM or +sizeK (654-2088, default 2088): +200M
Command (m for help): _
```

Вам следует убедиться, что вы создали раздел.

```
Command (m for help): p

Disk /dev/hda: 17.1 GB, 17179869184 bytes
255 heads, 63 sectors/track, 2088 cylinders
Units = cylinders of 16065 * 512 = 8225280 bytes

   Device Boot      Start         End      Blocks   Id  System
/dev/hda1    *           1         653     5245191    7  HPFS/NTFS
/dev/hda2             654         678       200812+   83  Linux

Command (m for help): _
```

Второй раздел будет разделом подкачки. Таким образом мы получили раздел размером в 200 мегабайт для подкачки. (Размер необходимый для раздела подкачки, зависит от того, сколько ОЗУ есть у вас в системе. Существует соглашение, что размер раздела подкачки должен быть в два раза больше, чем объём ОЗУ.)

Теперь нам необходимо сменить тип файловой системы для swap-раздела:

```
Command (m for help): t
Partition number (1-4): 2
Hex code (type L to list codes): _
```

Используйте клавишу L для того, что бы посмотреть список доступных файловых систем:

```

0 Empty          1e Hidden W95 FAT1 80 Old Minix       be Solaris boot
1 FAT12          24 NEC DOS      81 Minix / old Lin bf Solaris
2 XENIX root     39 Plan 9        82 Linux swap    c1 DRDOS/sec (FAT-
3 XENIX usr      3c PartitionMagic 83 Linux         c4 DRDOS/sec (FAT-
4 FAT16 <32M     40 Venix 80286   84 OS/2 hidden C: c6 DRDOS/sec (FAT-
5 Extended      41 PPC PReP Boot 85 Linux extended c7 Syrix
6 FAT16          42 SFS           86 NTFS volume set da Non-FS data
7 HPFS/NTFS     4d QNX4.x        87 NTFS volume set db CP/M / CTOS / .
8 AIX           4e QNX4.x 2nd part 88 Linux plaintext de Dell Utility
9 AIX bootable  4f QNX4.x 3rd part 8e Linux LVM      df BootIt
a OS/2 Boot Manag 50 OnTrack DM     93 Amoeba        e1 DOS access
b W95 FAT32      51 OnTrack DM6 Aux 94 Amoeba BBT     e3 DOS R/O
c W95 FAT32 (LBA) 52 CP/M         9f BSD/OS        e4 SpeedStor
e W95 FAT16 (LBA) 53 OnTrack DM6 Aux a0 IBM Thinkpad hi eb BeOS fs
f W95 Ext'd (LBA) 54 OnTrackDM6    a5 FreeBSD      ee EFI GPT
10 OPUS          55 EZ-Drive     a6 OpenBSD      ef EFI (FAT-12/16/
11 Hidden FAT12   56 Golden Bow   a7 NeXTSTEP     f0 Linux/PA-RISC b
12 Compaq diagnost 5c Priam Edisk  a8 Darwin UFS   f1 SpeedStor
14 Hidden FAT16 <3 61 SpeedStor    a9 NetBSD       f4 SpeedStor
16 Hidden FAT16   63 GNU HURD or Sys ab Darwin boot  f2 DOS secondary
17 Hidden HPFS/NTF 64 Novell Netware b7 BSDI fs       fd Linux raid auto
18 AST SmartSleep 65 Novell Netware b8 BSDI swap     fe LANstep
1b Hidden W95 FAT3 70 DiskSecure Mult bb Boot Wizard hid ff BBT
1c Hidden W95 FAT3 75 PC/IX
Hex code (type L to list codes): _

```

Используйте код 82 для смены типа файловой системы на Linux swap:

Hex code (type L to list codes): 82

```

Hex code (type L to list codes): 82
Changed system type of partition 2 to 82 (Linux swap)

Command (m for help): p

Disk /dev/hda: 17.1 GB, 17179869184 bytes
255 heads, 63 sectors/track, 2088 cylinders
Units = cylinders of 16065 * 512 = 8225280 bytes

   Device Boot      Start         End      Blocks   Id  System
/dev/hda1   *           1         653     5245191    7  HPFS/NTFS
/dev/hda2             654        678      200812+   82  Linux swap

Command (m for help):

```

Затем мы определим раздел номер 3, начинающийся первым доступным цилиндром и заканчивающийся также +200M. Этот раздел мы будем использовать позднее.

```

Command (m for help): n
Command action
  e   extended
  p   primary partition (1-4)
p
Partition number (1-4): 3
First cylinder (679-2088, default 679):
Using default value 679
Last cylinder or +size or +sizeM or +sizeK (679-2088, default 2088): +200M
Command (m for help): _

```

Далее нам необходимо создать дополнительный раздел на оставшемся участке жесткого диска и разместить в нем еще три раздела: для корня файловой системы, и для разделов home и var:



```
Command (m for help): n
Command action
  e   extended
  p   primary partition (1-4)
e
Selected partition 4
First cylinder (704-2088, default 704):
Using default value 704
Last cylinder or +size or +sizeM or +sizeK (704-2088, default 2088):
Using default value 2088

Command (m for help): n
First cylinder (704-2088, default 704):
Using default value 704
Last cylinder or +size or +sizeM or +sizeK (704-2088, default 2088): +8G

Command (m for help): _
```

```
Command (m for help): n
First cylinder (1678-2088, default 1678):
Using default value 1678
Last cylinder or +size or +sizeM or +sizeK (1678-2088, default 2088): +1G

Command (m for help): n
First cylinder (1801-2088, default 1801):
Using default value 1801
Last cylinder or +size or +sizeM or +sizeK (1801-2088, default 2088):
Using default value 2088

Command (m for help): _
```

Ну вот, разбиение почти завершено.

**Внимание!** Все действия, выполняемые вами в программе fdisk (в отличие от его аналога от MS) не вносят никаких изменений до тех пор, пока вы не выполнили команду w для применения сделанных изменений. То есть вы фактически создаете проект будущего. Вы можете безнаказанно удалять и создавать любые разделы, а выйдя из программы fdisk используя команду q — выход без сохранения внесенных изменений, вы увидите таблицу разделов жесткого диска без изменений.

До того как записывать изменения на диск, вам следует ещё раз посмотреть на таблицу разделов. Воспользуйтесь для этого командой p.

```
Command (m for help): p

Disk /dev/hda: 17.1 GB, 17179869184 bytes
255 heads, 63 sectors/track, 2088 cylinders
Units = cylinders of 16065 * 512 = 8225280 bytes

   Device Boot      Start         End      Blocks   Id  System
/dev/hda1   *           1           653     5245191    7  HPFS/NTFS
/dev/hda2             654           678       200812+   82  Linux swap
/dev/hda3             679           703       200812+   83  Linux
/dev/hda4             704          2088    11125012+    5  Extended
/dev/hda5             704          1677     7823623+   83  Linux
/dev/hda6          1678          1800       987966    83  Linux
/dev/hda7          1801          2088     2313328+   83  Linux

Command (m for help): _
```

Если всё хорошо, то наберите w, чтобы сохранить изменения на диск и выйти из fdisk.

```
Command (m for help): w
The partition table has been altered!

Calling ioctl() to re-read partition table.
Syncing disks.
root@slackware:~#
```

Для проверки, вновь используйте команду `fdisk -l`

```
root@slackware:~# fdisk -l

Disk /dev/hda: 17.1 GB, 17179869184 bytes
255 heads, 63 sectors/track, 2088 cylinders
Units = cylinders of 16065 * 512 = 8225280 bytes

   Device Boot      Start         End      Blocks   Id  System
/dev/hda1  *           1           653     5245191    7  HPFS/NTFS
/dev/hda2             654           678      200812+   82  Linux swap
/dev/hda3             679           703      200812+   83  Linux
/dev/hda4             704          2088    11125012+    5  Extended
/dev/hda5             704          1677     7823623+   83  Linux
/dev/hda6          1678          1800      987966    83  Linux
/dev/hda7          1801          2088    2313328+   83  Linux
root@slackware:~#
```

## Типы файловых систем

Операционная система Linux поддерживает огромное количество разнообразных типов файловых систем. С точки зрения Linux файловые системы можно условно разделить на четыре группы:

- «Родные» файловые системы. Имеется в виду, что файловая система поддерживает все атрибуты, свойственные Linux: права доступа, временные метки, информацию о владельце файла и т.д.:
  - ext2
  - ext3
  - reiserfs
  - JFX
  - XFS
- «Неродные» файловые системы. То есть файловые системы, не поддерживающие атрибуты Linux;
- Виртуальные. Это файловые системы, которые не имеют физического носителя;
- Сетевые файловые системы.

## Файловая система ext2

Ext2 — это одна из первых файловых систем, используемых в Linux (Если говорить более точно, то первая файловая система Linux — это minix. Но возможности этой fs весьма ограничены, и она применялась только на начальном этапе развития Linux.). Она была создана в 1993 году. Эта файловая система считается очень надёжной и проверенной временем. Но, поскольку ext2 разрабатывалась в те времена, когда жёсткий диск размером 300 Мбайт считался очень большим, ей присущи некоторые ограничения. Применять эту fs для больших разделов не имеет смысла, она начнёт «тормозить», когда в разделе будет большое количество файлов. То есть ext2 считается медленной (Понятие «медленная» очень относительное. Ext2 считается медленной в Linux. Но если сравнить её со стандартной файловой системой FreeBSD, окажется, что ext2 очень даже быстрая.). Конечно, с увеличением размеров дисков, с появлением новых веяний, в файловую систему вносились изменения, улучшающие её работу и функциональность. Например, поддержка POSIX ACL. Но всё же её не коснулись глобальные изменения, позволяющие говорить:

— Да, это та самая, единственная файловая система, которая меня полностью устраивает.

Кроме того, ext2 имеет серьёзные ограничения:

- Максимальный размер файла — 2048 Гбайт.
- Максимальный размер файловой системы — 32768 Гбайт.
- Максимальное количество поддиректорий в одной директории — 32768.

## Журналируемые файловые системы

Сейчас файловую систему ext2 уже практически не используют. И дело даже не в её ограничениях, ext2 достаточно надёжная файловая система. Всё дело в скорости загрузки Linux-серверов. Необходимо, чтобы сервер работал постоянно. Но чудес не бывает, сервера иногда приходится перегружать. Ваша задача — сделать так, чтобы после падения системы они перегружались как можно быстрее. При включении сервера происходит проверка дисков. Процедура проверки файловых систем, особенно больших, — достаточно длительная процедура. Если таких файловых систем несколько, то их проверка может занять очень много времени. А сервер должен работать!

Для уменьшения времени, используемого на проверку, и для увеличения надёжности были разработаны журналируемые файловые системы. Если вы работали с базами данных, вам наверняка известно такое понятие как транзакция. В транзакцию объединяют несколько SQL-операторов. Система должна выполнить все операторы. Если хотя бы один из них не срабатывает, то система откатывается на начало транзакции. Если система была отключена во время выполнения транзакции, при включении, если это возможно, она пытается выполнить оставшиеся операторы или вернуться на начало транзакции.

В современные файловые системы была добавлена поддержка журнала транзакций. С точки зрения работы файловой системы все операции с файлом выглядят как одна транзакция. Если посмотреть подробнее на файловые операции в Linux, запись или изменение файла — это довольно сложная процедура, состоящая из многих действий с данными на диске. При использовании журнала транзакций, прежде чем какие-либо физические изменения будут произведены на диске, в журнале открывается новая транзакция, в которой будут записаны все действия, которые будут производиться с файловой системой. И только после того, как транзакция будет сохранена на диске, будут производиться изменения в файловой системе.

Если файловая система будет отключена некорректно, программа проверки сначала смотрит журнал транзакций и на основании данных, находящихся в нём, попытается либо вернуть (откатить) систему на момент начала транзакции, или, если это возможно, завершить действия, описанные в транзакции. Учитывая то, что журнал имеет небольшой размер (в файловой системе ext3 он равен 32 Мбайт), процесс восстановления файловой системы значительно ускоряется.

## Файловая система ext3

Когда возникла необходимость внедрения журналируемых файловых систем в Linux, компания RedHat разработала файловую систему ext3. В RedHat пошли путём наименьшего сопротивления — за основу взяли хорошо известную ext2 и добавили поддержку журнала.

По своему физическому устройству ext2 идентична ext3. Эта особенность позволила применять для работы с ext3 такие же утилиты (создание, проверка и настройка файловых систем), как и для работы с ext2.

Несмотря на добавление журнала, ext3 работает быстрее, чем ext2. К достоинствам ext3 следует также отнести возможность журналирования не только необходимых действий, но и данных, что не позволяют делать другие журналируемые системы. Благодаря этой особенности ext3 считается очень надёжной.

Ext3 поддерживает три режима работы:

- Writeback — в этом режиме не происходит журналирования данных. В журнал сначала помещаются так называемые метаданные (inode файла, ссылки на блоки). Только после того, как они попали в журнал, происходит запись данных в файловую систему.
- Ordered (режим по умолчанию) — этот режим похож на описанный выше. Единственным отличием является то, что в режиме writeback в журнал сначала помещаются все метаданные, и только после этого происходят изменения в файловой системе. А в режиме ordered при помещении информации о блоке в журнал этот блок сразу же изменяется в файловой системе. Затем в журнал помещается информация о следующем блоке, и блок записывается, и так далее. То есть данные изменяются параллельно с изменением в журнале.
- Journal — режим полного журналирования. В журнал попадают метаданные и данные. И только после этого происходит изменение в файловой системе.



## Файловая система ReiserFS

ReiserFS разрабатывается Хансом Рейзером (Hans Reiser) и его компанией Namesys (<http://www.namesys.com>). Это очень быстрая файловая система, хорошо приспособленная для хранения большого количества маленьких файлов.

В ней удалось решить проблему размещения на диске маленьких файлов. Например, в ext2/3 для размещения файла, содержащего единственный символ, на диске будет занят целый блок. Блок ext2/3 может иметь размер от 1 до 8 Кбайт (размер зависит от объема файловой системы). А в ReiserFS в одном блоке могут быть размещены данные нескольких файлов. Более того, если размер файла очень мал, данные могут быть размещены в inode, то есть непосредственно в метаданных.

Файловая система базируется на оптимизированных деревьях (B-tree). Это увеличивает скорость поиска в файловой системе и снимает вопрос ограничения количества файлов и директорий в директории.

С файлами большого размера данная файловая система тоже справляется весьма уверенно.

Для файловой системы ReiserFS версии 3.6 существуют следующие ограничения:

- Максимальный размер файла — 8 Тбайт (для 32-битных компьютеров);
- Максимальный размер файловой системы — 16 Тбайт.

Сейчас разрабатывается следующая версия ReiserFS — четвёртая.

## Файловая система JFS

Эта файловая система разрабатывается компанией IBM и распространяется под лицензией GNU GPL. Описание JFS можно найти в Интернете на сайте <http://jfs.sourceforge.net>. JFS используется не только в Linux, но и в других операционных системах, например, в AIX и OS/2.

JFS — журналируемая файловая система. Основной её конёк — использование совместно с LVM (Logical Volume Manager). LVM позволяет объединять несколько физических разделов жёстких дисков в один логический, который затем можно разбивать на разделы как обыкновенный жёсткий диск. При этом некоторые типы LVM позволяют на лету подключать новое дисковое пространство. И если на увеличивающихся разделах использовать файловую систему ext3, в один прекрасный момент вы получите сообщение о невозможности создания нового файла. Дело в том, что при форматировании раздела в ext3 в нём заранее, в зависимости от размера, резервируется конечное количество inodes. То есть заранее известно максимальное количество файлов. Если размер файловой системы не будет увеличиваться, то этого количества inodes вполне хватает для нормальной работы. В JFS есть возможность динамического увеличения файловой системы и количества inodes. Благодаря этому свойству, при увеличении размера файловой системы не возникает ограничение на количество создаваемых файлов.

JFS имеет интересное внутренне устройство. Например, при создании файловой системы ext3 в разделе выделяется суперблок, в котором описаны параметры и корневая директория файловой системы. При создании файла выделяется дисковое пространство для inode и данных файла. В JFS раздел разбивается на так называемые агрегаты. В каждом агрегате создается свой суперблок и свой журнал. Фактически мы получаем набор файловых систем. Причём эти файловые системы при желании можно использовать отдельно! При увеличении размера файловой системы просто создаётся новый агрегат, в котором распределяется свое количество inodes (Подробно об устройстве файловой системы JFS написано тут: <http://www-128.ibm.com/developerworks/library/l-jfslayout>).

Для файловой системы JFS существуют следующие ограничения:

- Максимальный размер файла ограничивается разрядностью операционной системы.
- Максимальный размер файловой системы — 512 Тбайт.

## Файловая система XFS

Файловая система XFS разрабатывалась в компании SGI (бывшая Silicon Graphics, Inc.). XFS появилась на свет в 1994 году и изначально поставлялась с операционной системой IRIX. Компания SGI славится своими рабочими станциями для производства видео, а также серверами для хранения данных. Поэтому файловая система оптимизирована для обслуживания большого количества огромных файлов и для поддержки больших директорий. Благодаря своей структуре, она так же хорошо поддерживает большое количество маленьких файлов. По своему быстродействию она сопоставима с файловой системой ReiserFS, а по надёжности превосходит файловую систему Ганса (Сколько данных было мной потеряно в файловой системе ReiserFS!

Спасало только резервное копирование. Поэтому сейчас я ReiserFS на серверах не использую.).

Поддержка больших файлов возможна благодаря тому, что XFS — это 64-битная файловая система. А скорость работы файловой системы достигается благодаря использованию B+ деревьев для поиска и описания внутренних структур.

Внутренне устройство файловой системы достаточно сложное, и я не вижу необходимости в кратком описании её структуры. Тем более, что в Интернете есть хорошие статьи, подробно описывающие XFS:

[http://www.opennet.ru/docs/RUS/xfs\\_arch](http://www.opennet.ru/docs/RUS/xfs_arch)

<http://www.lugr.ru/articles/XFS-layout>

## Файловые системы компании Microsoft

Если говорить о файловых системах компании Microsoft, в Linux поддерживаются FAT и NTFS. С FAT всё очень просто, структура файловой системы известна, поэтому в Linux она поддерживается полностью. Единственное, что необходимо учесть при использовании FAT, в Linux существует две её разновидности:

- msdos — FAT12/16.
- vfat — FAT32.

Поддержку FAT следует включать в том случае, если вы предполагаете использовать гибкие диски и различные USB-накопители: флеш-карты, жёсткие диски и т.д. Дело в том, что все они обычно отформатированы в FAT.

С NTFS немного сложнее. Эта файловая система нормально поддерживается в режиме только для чтения. В режиме записи её не рекомендуется использовать. Хотя режим записи поддерживается, но если почитать документацию к драйверам NTFS, вы увидите, что там большими буквами написано: в режиме записи можно только изменять содержимое существующих файлов, ни в коем случае нельзя создавать новые файлы, удалять или изменять размер существующих — это может разрушить файловую систему.

## Файловые системы iso9660 и udf

Эти файловые системы используются для хранения информации на CD- и DVD-дисках.

Изначально iso9660 была очень простой файловой системой с большим количеством ограничений. Например, имена файлов как в MS DOS, ограничение на количество вложений директорий. Поэтому для iso9660 было написано несколько дополнений, расширяющих её возможности. В том числе, дополнения, позволяющие сохранять атрибуты файлов UNIX. Все дополнения поддерживаются драйвером файловой системы, и никаких затруднений при работе быть не должно. Более того, драйвер iso9660 поддерживает, как это ни странно звучит, режим записи. Он применяется при создании образов CD-ROM.

С udf тоже не замечено особых проблем. Таким образом, работа с CD- и DVD-дисками поддерживается в Linux без каких-либо ограничений.

## Виртуальные файловые системы

Под виртуальными файловыми системами понимаются файловые системы, появляющиеся при загрузке операционной системы и не имеющие реального размещения на жестких дисках. Эти виртуальные файловые системы размещаются в оперативной памяти компьютера на RAM дисках.

## Файловая система proc

Это очень полезная файловая система. В своей работе администратора вы очень часто будете обращаться к её возможностям. В одной из первых глав, рассказывающих об организации файловой системы Linux, я вкратце рассказывал о предназначении этой файловой системы. Просто напомним, что файлы, которые находятся в директории /proc — это отображение области данных ядра на файловую систему. То есть, если вы просматриваете содержимое какого-либо файла, на самом деле вы видите определённую часть области данных ядра.

Ниже я опишу некоторые интересные файлы, которые вы можете встретить в директории /proc. Содержимое файлов в вашей системе будет отличаться от содержимого файлов, показанных в качестве примеров.

**/proc/cmdline**

Содержит командную строку, переданную ядру при его запуске.

```
# cat cmdline
BOOT_IMAGE=Linux ro root=305 vt.default_utf8=0
#
```

### **/proc/cpuinfo**

Информация о процессоре или процессорах.

```
# cat cpuinfo
processor       : 0
vendor_id      : GenuineIntel
cpu family     : 15
model          : 4
model name     : Intel(R) Pentium(R) 4 CPU 3.00GHz
stepping      : 3
cpu MHz        : 2991.964
cache size     : 2048 KB
fdiv_bug      : no
hlt_bug       : no
f00f_bug      : no
coma_bug      : no
fpu           : yes
fpu_exception  : yes
cpuid level    : 2
wp            : yes
flags          : fpu vme de pse tsc msr mce cx8 apic pge cmov clflush mmx fxsr
sse sse2 constant_tsc up sync_rdtsc monitor
bogomips      : 6020.72
clflush size   : 64
#
```

### **/proc/devices**

Список устройств.

```
# cat devices
Character devices:
 1 mem
 2 pty
 3 tty
 4 /dev/vc/0
 4 tty
 4 ttyS
 5 /dev/tty
 5 /dev/console
 5 /dev/ptmx
 6 lp
 7 vcs
 9 st
10 misc
13 input
29 fb
128 ptm
136 pts
171 ieee1394
180 usb
189 usb_device
249 hidraw
250 usb_endpoint
251 megaraid_sas_ioctl
252 megadev
253 megadev_legacy
254 aac

Block devices:
 1 ramdisk
```

```
2 fd
3 ide0
7 loop
8 sd
9 md
11 sr
22 ide1
65 sd
66 sd
67 sd
68 sd
69 sd
70 sd
71 sd
80 i2o_block
128 sd
129 sd
130 sd
131 sd
132 sd
133 sd
134 sd
135 sd
253 device-mapper
254 mdp
#
```

### **/proc/dma**

Использование каналов DMA.

```
# cat dma
2: floppy
4: cascade
#
```

### **/proc/filesystems**

Список поддерживаемых файловых систем.

```
# cat filesystems
nodev sysfs
nodev rootfs
nodev bdev
nodev proc
nodev securityfs
nodev sockfs
nodev usbfs
nodev pipefs
nodev anon_inodefs
nodev futexfs
nodev tmpfs
nodev inotifyfs
nodev configfs
nodev devpts
nodev reiserfs
nodev ext3
nodev ext4dev
nodev ext2
nodev ramfs
nodev minix
nodev msdos
nodev vfat
nodev iso9660
nodev nfs
nodev ntfs
```

```

romfs
udf
jfs
xfs
gfs2
gfs2meta
nodev mqueue
nodev rpc_pipefs
#

```

### **/proc/interrupts**

Распределение прерываний.

```

# cat interrupts
CPU0
0: 2001961 XT-PIC-timer
1: 9 XT-PIC-i8042
2: 0 XT-PIC-cascade
5: 522 XT-PIC-eth2
6: 3 XT-PIC-floppy
8: 1 XT-PIC-rtc
9: 0 XT-PIC-acpi, ohci_hcd:usb2
10: 7 XT-PIC-eth1
11: 7 XT-PIC-ehci_hcd:usb1, eth0
12: 36 XT-PIC-i8042
14: 7591 XT-PIC-ide0
15: 24 XT-PIC-ide1
NMI: 0 Non-maskable interrupts
LOC: 2002013 Local timer interrupts
RES: 0 Rescheduling interrupts
CAL: 0 function call interrupts
TLB: 0 TLB shootdowns
TRM: 0 Thermal event interrupts
SPU: 0 Spurious interrupts
ERR: 0
MIS: 0
#

```

### **/proc/modules**

Список загруженных модулей.

```

# cat modules
ip6 234724 10 - Live 0xd0c48000
nls_koi8_r 8960 1 - Live 0xd0bcb000
agpgart 30664 0 - Live 0xd0b83000
lp 13348 0 - Live 0xd097b000
parport_pc 27556 0 - Live 0xd0b7b000
parport 34632 2 lp,parport_pc, Live 0xd0b71000
psmouse 40336 0 - Live 0xd0b4c000
serio_raw 9092 0 - Live 0xd0b3a000
evdev 12672 0 - Live 0xd0b2a000
i2c_piix4 11020 0 - Live 0xd0b15000
ac 8068 0 - Live 0xd0b12000
thermal 16540 0 - Live 0xd0b24000
processor 32680 1 thermal, Live 0xd0b2f000
button 10000 0 - Live 0xd0b0e000
pcnet32 34564 0 - Live 0xd0b1a000
mii 8448 1 pcnet32, Live 0xd0b0a000
#

```

### **/proc/mounts**

Содержит список подключенных файловых систем.

```
# cat mounts
rootfs / rootfs rw 0 0
/dev/root / xfs rw,ikloop,noquota 0 0
proc /proc proc rw 0 0
sysfs /sys sysfs rw 0 0
tmpfs /dev tmpfs rw 0 0
devpts /dev/pts devpts rw 0 0
usbfs /proc/bus/usb usbfs rw 0 0
/dev/hda6 /home xfs rw,ikloop,noquota 0 0
/dev/hda7 /var xfs rw,ikloop,noquota 0 0
/dev/hda1 /mnt/win ntfs ro,uid=0,gid=0,fmask=0177,dmask=077,nls=koi8-
r,errors=continue,mft_zone_multiplier=1 0 0
/dev/hda3 /mnt/free ext3 rw,data=ordered 0 0
tmpfs /dev/shm tmpfs rw 0 0
#
```

### /proc/partitions

Содержит список разделов всех подключенных накопителей.

```
# cat partitions
major minor #blocks name
3 0 16777216 hda
3 1 5245191 hda1
3 2 200812 hda2
3 3 200812 hda3
3 4 1 hda4
3 5 7823623 hda5
3 6 987966 hda6
3 7 2313328 hda7
#
```

### /proc/swaps

Содержит список подключенных swap файлов и разделов.

```
# cat swaps
Filename Type Size Used Priority
/dev/hda2 partition 200804 0 -1
#
```

### /proc/version

Содержит информацию о версии операционной системы и ядра Linux.

```
# cat version
Linux version 2.6.24.5-smp (root@midas) (gcc version 4.2.3) #2 SMP Wed Apr 30
13:41:38 CDT 2008
#
```

### Информация о процессах

Кроме файлов в /proc находятся директории, имеющие в качестве имени число. Каждая директория описывает процесс, PID которого соответствует имени директории. Файлы в этой директории описывают параметры процесса. Содержимое одной из директорий приведено ниже. Эта директория соответствует pid моего текущего bash

```
# ls 2384
attr/ coredump_filter fd/ mem oom_score statm
auxv cwd@ fdinfo/ mounts root@ status
clear_refs environ limits mountstats smaps task/
cmdline exe@ maps oom_adj stat wchan
#
```

Только несколько из приведенных в примере файлов содержат информацию, которая была бы понятна без предварительной обработки.

**cmdline**

Содержит аргументы командной строки.

```
# cat 2384/cmdline
-bash
#
```

**environ**

Содержит значения переменных среды окружения процесса.

```
# cat 2384/environ
USER=rootLOGNAME=rootHOME=/rootPATH=/usr/local/sbin:/usr/sbin:/sbin:/usr/local/bin:/usr/bin:/binMAIL=/var/mail/rootSHELL=/bin/bashSSH_CLIENT=192.168.102.2 1224 22SSH_CONNECTION=192.168.102.2 1224 192.168.102.1 22SSH_TTY=/dev/pts/0TERM=xterm
#
```

**status**

Содержит информацию о состоянии процесса в формате понятном человеку.

```
# cat 2384/status
Name:  bash
State:  S (sleeping)
Tgid:  2384
Pid:   2384
PPid:  2381
TracerPid:  0
Uid:  0      0      0      0
Gid:  0      0      0      0
FDSize: 256
Groups: 0 1 2 3 4 6 10 11 17 18 19 26 83
VmPeak:  3112 kB
VmSize:  3112 kB
VmLck:    0 kB
VmHWM:   1776 kB
VmRSS:   1776 kB
VmData:   400 kB
VmStk:    84 kB
VmExe:    640 kB
VmLib:   1620 kB
VmPTE:    12 kB
Threads:  1
SigQ:  0/2047
SigPnd: 0000000000000000
ShdPnd: 0000000000000000
SigBlk: 0000000000010000
SigIgn: 0000000000384004
SigCgt: 000000004b813efb
CapInh: 0000000000000000
CapPrm: 00000000fffffeff
CapEff: 00000000fffffeff
voluntary_ctxt_switches: 369
nonvoluntary_ctxt_switches: 391
#
```

**Другие директории**

Коме директорий, описывающих процессы системы, в /proc могут находиться и другие директории. Ниже приведу назначение некоторых из них:

- **ide** — информация об устройствах, подключенных к ide интерфейсу.
- **irq** — информация о распределении прерываний.
- **net** — информация о сети. Содержимое таблицы **agr** и таблицы маршрутизации. Статистика по сетевым интерфейсам и протоколом. И так далее.

- `scsi` — информация о SCSI устройствах.
- `sys` — содержит изменяемые параметры системы.

### **/proc/sys**

Файловая система `/proc/sys` — это отдельная большая тема. При помощи файлов, находящихся в этой директории можно «на лету» изменять параметры системы. Достаточно записать нужное значение в определенный файл. Описывать `/proc/sys` я не буду, слишком много информации и слишком много надо знать, что бы понять для чего используются файлы. Поэтому я расскажу, где найти документацию и описание по этой файловой системе:

В первую очередь — это документация к ядру. Она поставляется с исходными кодами ядра. Описание `/proc/sys` можно найти в файле `Documentation/filesystems/proc.txt`. Ей посвящена отдельная (и далеко не маленькая) глава.

В Интернет по адресу <http://ipsysctl-tutorial.frozentux.net/ipsysctl-tutorial.html> можно найти документ «`Ipsysctl tutorial`». И его перевод на русский язык: [http://www.citforum.ru/operating\\_systems/linux/ipsysctl](http://www.citforum.ru/operating_systems/linux/ipsysctl).

## **Файловая система sysfs**

Виртуальная файловая система `sysfs` — это особенность ядра Linux версии 2.6. В файловой системе отображаются некоторые структуры данных ядра, в основном описывающие устройства, которые в данный момент находятся в системе.

`sysfs` используется программой `udev` для динамического создания файлов устройств.

### **Файловая система для устройств /dev**

*Устройства:* В Linux устройство является специальным «оборудованием» (или кодом, эмулирующим его), которое представляет методы для ввода или вывода информации (IO — Input/Output). Например, клавиатура — устройство для ввода. Жесткий диск — устройства для ввода (запись) и вывода (чтение). Большинство устройств в Linux представлено как файлы в особой файловой системе (за исключением сетевых карт). Эти файлы хранятся в каталоге `/dev`, куда к ним обращается система для выполнения задач, связанных с вводом/выводом.

Грубо говоря, устройства можно разделить на две категории: символьные и блочные. Символьные устройства вводят/выводят по символам. Наиболее показательным примером служит клавиатура, у которой нажатие каждой клавиши формирует символ, передаваемый компьютеру. Мышь работает немного по-другому. Каждое движение или нажатие на кнопку отправляет символ на `/dev/mouse`.

Блочные устройства считывают данные большими объемами. Примерами служат устройства для хранения данных, такие как IDE жесткие диски (`/dev/hd`), SCSI жесткие диски (`/dev/sd`) и CD-ROM'ы (например, `/dev/cdrom0` — символическая ссылка на первый CD-ROM). Операции ввода/вывода блочные устройства проводят с определенными блоками данных, что позволяет работать с большими объемами информации более эффективно.

*Названия устройств:* Устройства часто называются путем сокращения имен представляемого ими оборудования. Устройства с именами `/dev/fb` представляют буферы фреймов (frame buffers) для графики. Устройства `/dev/hd` представляют IDE жесткие диски (hard disks). В некоторых случаях для пояснения того, чем является устройство, используются символические ссылки: например, `/dev/mouse`, устройство, представляющее мышь, может быть прилинковано к последовательному, USB или PS/2 устройству, в зависимости от используемого «железа». Символическая ссылка помогает и человеку, и машине разобраться, какое из устройств — мышь.

Иногда бывает несколько устройств одного типа. Например, у машины два ATAPI CD-ROM'а. Каждому CD-приводу нужен файл в `/dev`. В таком случае, возможен вариант, что `/dev/cdrom0` будет первым CD-ROM'ом, а `/dev/cdrom1` — вторым.

С именами жестких дисков немного сложнее. Название устройства жесткого диска зависит от типа диска, его позиции и раздела (partition'a). Первый жесткий диск может быть назван `/dev/hda`, где часть «`hd`» означает, что это IDE жесткий диск, а «`a`» показывает, что это первый жесткий диск. Тогда `/dev/hdb` будет ссылаться на второй жесткий диск. Каждый жесткий диск разбит на разделы. Первый раздел первого жесткого диска получит название `/dev/hda1`, где единица в конце обозначает номер раздела. Обратите внимание на то, что, если индексы некоторых устройств (например, `/dev/cdrom0`) могут начинаться с нуля, то индекс устройств с разделами обычно начинается с единицы. Вот примерный список файлов в `/dev` для двух IDE жестких дисков:



```

/dev/hda
/dev/hda1
/dev/hda2
/dev/hda3
/dev/hda4
/dev/hdb
/dev/hdb1
/dev/hdb2
/dev/hdb3

```

SCSI жесткие диски используют `/dev/sd` вместо `/dev/hd`, но все остальное выглядит также. `/dev/sda1` ссылается на первый раздел первого SCSI жесткого диска.

*Специальные устройства:* Существует несколько специальных устройств, которые порой бывают очень полезны: `/dev/null`, `/dev/zero`, `/dev/full` и `/dev/random`.

Нулевое устройство, `/dev/null` представляет собой что-то типа «мусорной корзины». Часто некоторые программы выводят множество ненужной информации. Shell-скрипты обычно используют `/dev/null` для того, чтобы пользователь не видел ненужных ему сообщений от вызываемых утилит. Вот пример вызова модуля ядра с выводом всех сообщений в `/dev/null`.

```
$ modprobe cipher-twofish > /dev/null
```

`/dev/zero` близок к `/dev/null`. Как и `/dev/null`, устройство может быть использовано для блокирования вывода ненужной информации, но чтение `/dev/zero` возвращает 0 символы (чтение `/dev/null` возвращает символы end-of-file — конец файла). Поэтому `/dev/zero` обычно используется для создания пустых файлов.

```
$ dd if=/dev/zero of=/my-file bs=1k count=100
```

Такая команда создаст файл размером в 100кб, наполненный null-символами.

`/dev/full` служит для имитации «полного» устройства. Запись в `/dev/full` сопровождается ошибкой. «Полное» устройство полезно для того, чтобы посмотреть, как тестируемое приложение будет себя вести при попытке доступа к заполненному устройству (т.е. например, к жесткому диску, на котором не осталось места).

```
$ cp test-file /dev/full
```

```
cp: writing `/dev/full': No space left on device
$
```

Устройства `/dev/random` и `/dev/urandom` создают «случайные» данные. Хотя вывод обоих может показаться абсолютно случайным, `/dev/random` более случаен чем `/dev/urandom`. `/dev/random` создает случайные символы, основываясь на «окружающем шуме». Так как количество этого случайного шума ограничено, `/dev/random` работает медленно и может временно останавливаться для дальнейшего сбора данных. `/dev/urandom` использует тот же шум, что и `/dev/random`, но если случайных данных больше нет, оно создает псевдо-случайные данные. Таким образом увеличивается его скорость, но уменьшается безопасность.

*Старая файловая система /dev:* Раньше файловая система `/dev` была частью обычной файловой системы. Она состояла из специальных файлов, созданных однажды (обычно при установке системы) и сохраненных на жестком диске.

В старых системах файловая система `/dev` должна содержать информацию обо всех устройствах, которые могут быть подключены к компьютеру. Из-за этого `/dev` была слишком большой — приходилось хранить сведения о множестве жестких дисков, дисководов и т.п. Ранее мы рассматривали список разделов жесткого диска `hdb`. В старой файловой системе `/dev` приходилось содержать файлы с `/dev/hdb1` до `/dev/hdb11`. Чтобы выяснить, какие устройства действительно привязаны к разделам жесткого диска (если помните, у меня всего три раздела на

hdb), нужно вызвать специальную утилиту. Команда "file -s hdb\*" поможет в этом разобраться:

```
$ file -s /dev/hdb?  
/dev/hdb1: Linux/i386 ext2 file system  
/dev/hdb2: Linux/i386 ext2 file system  
/dev/hdb3: Linux/i386 ext2 file system  
/dev/hdb4: empty  
/dev/hdb5: empty  
/dev/hdb6: empty  
/dev/hdb7: empty  
/dev/hdb8: empty  
/dev/hdb9: empty
```

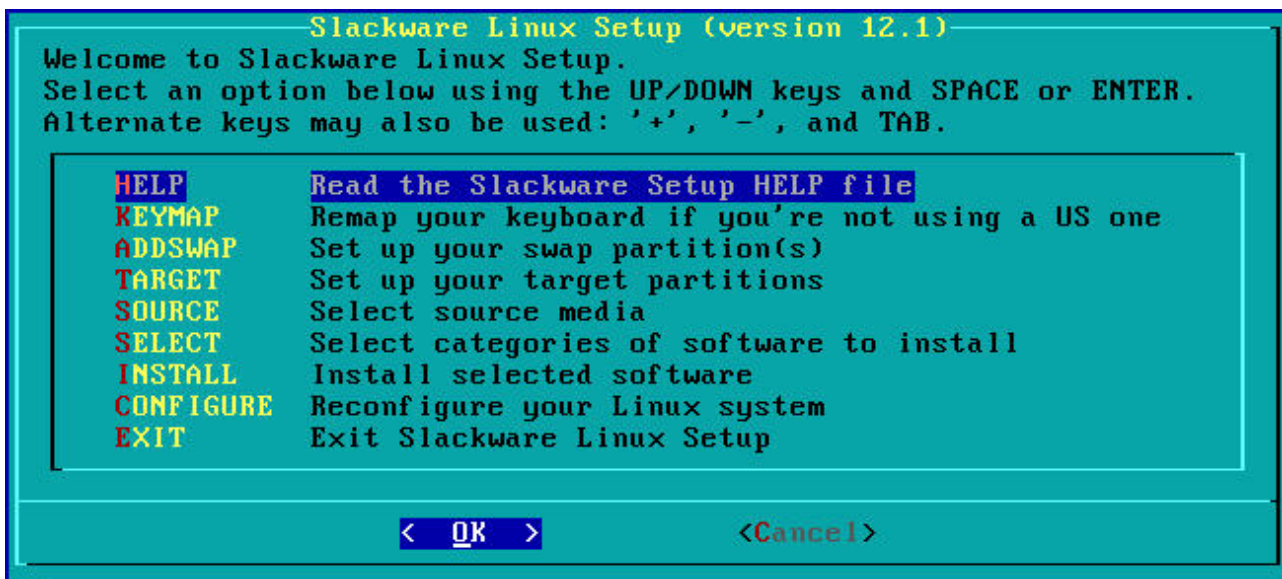
Если указанного файла устройства не было, приходилось его создавать с помощью mknod или другой программы (типа MAKEDEV). Хотя «старый способ» работал, он был сложен и неудобен.

## Программа установки

После того, как вы создали разделы, вы готовы к установке Slackware. Следующий шаг в процессе установки - это запуск программы setup. Чтобы запустить её, просто наберите setup в приглашении командной строки оболочки.

```
root@slackware:/# setup_
```

setup - меню управляемая система для фактической установки Slackware пакетов и настройки вашей системы.



Процесс установки происходит по следующему сценарию: Вы проходите через каждую опцию программы установки в том порядке, в котором они перечислены. (Конечно, вы можете проделать всё это в практически любом желаемом порядке, но шансы на то, что это не сработает достаточно высоки.) Выбор пункта меню производится при помощи кнопок-стрелок вверх и вниз, а выбор кнопок "Ok" или "Cancel" производится при помощи стрелок вправо и влево. В добавок к этому, каждому пункту меню соответствует определённая кнопка, которая подсвечена на экране в имени опции. Опции-флаги или, иначе говоря, переключатели (те которые отмечены [X]) помечаются при помощи клавиши пробел.

Разумеется, всё это вы можете найти в разделе "help" программы установки, но мы пользуемся принципом - пользователю всё самое лучшее за его деньги.

Ниже приведено достаточно подробное описание пунктов меню программы setup. Хочу лишь добавить, что начав с пункта ADDSWAP программа setup сама проведет вас через весь процесс инсталляции linux.

### HELP

Если вы устанавливаете Slackware впервые, вы скорее всего захотите заглянуть в этот раздел. Там дано описание каждого раздела setup (очень похожее на то, что вы читаете сейчас, но менее предвзятое) и инструкции по навигации через процесс установки.

## KEYMAP

Если вам необходима раскладка клавиатуры, отличная от United States "qwerty", вы возможно захотите заглянуть в этот раздел. Там вы найдёте большой список альтернативных раскладок, для получения наслаждения от использования вашей клавиатурой.

## ADDSWAP

Если вы создали раздел подкачки swap (см. раздел "Разбиение жёсткого диска"), то этот пункт поможет вам активизировать его. Он автоматически обнаружит разделы подкачки, выведет на экран информацию о существующих разделах подкачки, позволяя вам выбрать один, который будет отформатирован и включён.

## TARGET

Пункт target (цель) определяет, какие из других (не swap) разделов должны быть отформатированы и подключены к точкам монтирования вашей файловой системы. На экран выводится список разделов вашего жёсткого диска. Для каждого раздела вам будет предоставлена возможность отформатировать его (а так же, проверить на наличие bad-блоков) и список для выбора размера инод. Для обычного использования, значение размера инод можно выбрать предлагаемое по умолчанию.

Первая опция в пункте target - выбор раздела, на который установить корневую (\) файловую систему. После этого вы сможете подключить другие разделы к файловой системе, на ваше усмотрение. (Например, вы можете захотеть, чтобы ваш третий раздел, скажем /dev/hda3, был каталогом домашних файловых систем пользователей. Это лишь пример; подключайте ваши разделы, как считаете нужным.)

## SOURCE

Пункт source (источник) позволяет вам выбрать, носитель информации, с которого вы будете устанавливать Slackware. На сегодняшний день есть пять вариантов решения этого вопроса. CD-ROM, раздел жесткого диска, NFS, FTP или заранее под-монтированный каталог.

Выбор пункта CD-ROM активизирует установку с CD/DVD. Этот пункт предложит вам на выбор, либо автоматический поиск вашего CD-ROM привода, либо выбор устройства, соответствующего приводу из списка. Убедитесь в том, что Slackware CD/DVD вставлен в привод компакт дисков, до того, как начнёте сканирование.

Установка с существующего раздела жесткого диска предполагает, что файлы дистрибутива размещены там ранее.

Вариант установки через NFS попросит вас ответить на вопросы о вашей сети и сетевой информации вашего NFS сервера. NFS сервер должен быть настроен заранее. Так же следует отметить, что вы не можете пользоваться сетевыми именами, вы должны указывать IP адрес и для вашего компьютера, и для NFS сервера (на установочном диске нет преобразователя имён).

Использование FTP сервера позволяет загрузить весь дистрибутив с удаленного сервера по протоколу ftp.

Установка из ранее под-монтированного каталога является наиболее гибким пунктом. Вы можете воспользоваться этим методом для установки с таких носителей, как Jaz диск, NFS подключённый через PLIP и с файловых систем FAT11. Под-монтируйте файловую систему к выбранной вами точке монтирования до запуска программы установки, затем укажите эту точку здесь.

## SELECT

Этот пункт позволяет вам выбрать, какие разделы программ вы желаете установить. Пожалуйста, обратите внимание на то, что вы должны установить раздел A, чтобы получить минимальную работающую систему. Все другие разделы не являются обязательными.

С целью упрощения, Slackware исторически разделён на разделы программ. Когда-то названных "дисковыми разделами", потому что они были ориентированы на установку с флоппи дисков. Сейчас дистрибутив разбит на разделы в основном с целью структурирования программ, поставляемых с дистрибутивом.

Раздел	Содержание
A	Основная система. Содержит необходимый минимум системных программ, текстовый редактор и основные коммуникационные программы.
AP	Различные приложения, которые работают без X Window системы.
D	Инструменты для разработки программ. Компиляторы, дэбаггеры, интерпретаторы и man странички для них.
E	GNU emacs.
F	FAQи, HOWTO, и другая дополнительная документация.

GNOME	Рабочая среда GNOME , GTK widget библиотека, и GIMP.
K	Исходный код ядра Linux.
KDE	Рабочая среда KDE. Qt библиотека, необходимая для KDE, так же находится здесь.
KDEI	Модули для национальной локализации KDE
L	Системные библиотеки
N	Сетевые программы. Демоны, почтовые программы, telnet, программы чтения новостей, и так далее.
T	Система форматирования документов teTeX.
TCL	Tool Command Language. Tk, TclX, и TkDesk.
X	Основа X Window Системы.
XAP	Приложения X которые не являются частью основной окружающей среды рабочего стола. (Например, Ghostscript и Netscape).
Y	Игры

## INSTALL

В случае, если вы уже прошли через пункты "target", "source" и "select", этот пункт позволит вам выбрать, какие пакеты программ из выбранных вами разделов вы желаете установить. Иначе вам будет предложено вернуться назад и завершить всё в других пунктах программы установки. Этот пункт позволяет вам выбрать один из шести различных методов установки: full (полный), newbie (новичок), menu (меню), expert (эксперт), custom (выборочный) и tag path.

Подпункт full установит все пакеты из выбранных вами в пункте "select" разделов программ. Никаких больше вопросов. Это самый простой метод установки, так как вам не надо принимать никаких решений по поводу того, какие пакеты устанавливать, а какие - нет. Конечно, этот подпункт наиболее требователен к дисковому пространству.

Следующий доступный подпункт - newbie. Этот подпункт устанавливает все действительно необходимые пакеты в выбранных вами разделах. Для каждого из остальных пакетов вам будет предложено выбрать "Yes", "No" или "Skip". Yes или No очевидны, а Skip пропустит все остальные необязательные пакеты из данного раздела программ, и перейдёт к следующему. Дополнительно вам будет выведено описание и требования к дисковому пространству для каждого из пакетов с целью помочь выбрать то, что вам действительно необходимо. Этот вариант рекомендуется для новых пользователей, так как он гарантирует, что все необходимые пакеты будут установлены. Тем не менее, этот метод немного медленен, из за постоянных опросов.

Гораздо более быстрый и расширенный метод - menu. Для каждого раздела программ вы увидите меню, в котором вы можете выбрать, какие из пакетов (не обязательных для этого раздела), должны быть установлены. Пакеты, установка которых необходима не показываются в этом меню.

Для более опытных пользователей, программа установки предлагает подпункт expert. Этот метод позволяет вам получить абсолютный контроль над тем, какие из пакетов должны быть установлены. Вы можете установить то, что вы желаете. Это может привести к неработающей системе в том случае, если вы не установите некоторые из абсолютно необходимых пакетов. С другой стороны, вы полностью контролируете, что должно быть установлено в вашей системе. Мы настоятельно не рекомендуем пользоваться этим способом установки новым пользователям. Так как вы довольно легко можете "прострелить себе ногу".

custom и path tag варианты установки так же рекомендуются для использования только опытным пользователям. Эти методы позволяют вам произвести установку на основе пользовательских tag файлов, созданных вами в дереве каталогов дистрибутива. Это очень удобно, если вам необходимо установить систему на большое количество компьютеров, сравнительно быстро.

После того, как вы выбрали, каким из предложенных способов воспользоваться, возможны различные варианты продолжения. Если вы выбрали menu, то появится экран с меню, в котором вы можете выбрать пакеты для установки. Если вы выбрали full, то программа установки немедленно перейдёт к процессу копирования пакетов программ на выбранный вами ранее раздел жёсткого диска. Если вы выбрали newbie, то пакеты начнут копироваться до тех пор, пока не дойдёт очередь одного из дополнительных пакетов.

Пожалуйста, помните, что если вы выбрали слишком много пакетов программ для установки, по сравнению с тем, сколько свободного пространства имеется на жёстком диске, выбранном в пункте target, то место на диске

может закончиться. Наиболее безопасным решением будет, не спешить с установкой некоторых из программ, а установить их позднее. Это можно проделать весьма легко, при помощи инструментов Slackware для работы с пакетами программ.

## CONFIGURE

Пункт configure (настройка) позволяет вам выполнить основные настройки системы. То что вы увидите здесь, во многом зависит от того, какие пакеты программ вы установили. Но всегда вы увидите следующее:

### Kernel selection

- выбор ядра. Здесь вы должны выбрать, какое ядро будет использоваться. Вы можете установить ядро с загрузочного диска, использованного вами в процессе установки, с CD диска Slackware или с другого источника, который вы приготовили заранее. Так же вы можете пропустить выбор ядра. В этом случае будет использовано ядро по умолчанию.

### Make a boot disk

- создание загрузочного диска. Создание загрузочного диска для использования в будущем, вероятно является хорошей идеей. Вы сможете отформатировать флоппи диск и затем создать один из двух видов загрузочного диска. Первый тип, simple - просто записывает ядро на флоппи. Более гибкий вариант - создать загрузочный диск lilo. С загрузчиком lilo мы познакомимся позднее. Так же вы можете продолжить без создания загрузочного флоппи диска.

### Modem

Вам будут заданы вопросы о настройках вашего модема. Точнее, вы должны будете выбрать, есть ли у вас модем и если он у вас есть, то к какому последовательному порту он подключён.

Следующие пункты могут появиться, а могут и нет, в зависимости от того, были или не были установлены соответствующие им пакеты программ.

### Timezone

- часовой пояс. Всё довольно понятно: вас спросят, в каком часовом поясе вы находитесь. Если вы работаете по времени Зулу (Zulu) то мы приносим вам свои извинения, так как ваш часовой пояс находится в самом конце списка.

### Mouse

- мышь. Этот пункт спросит, какого типа мышь установлена в вашей системе, а так же, хотите ли вы, чтобы gpm(8) (поддержка мыши в режиме командной строки) была запущена при загрузке.

### Hardware clock

- аппаратные часы. Этот раздел спрашивает, идут ли аппаратные часы вашего компьютера в соответствии с Координированным Универсальным Временем (UTC или GMT). Для большинства компьютеров ответ будет нет.

### Font

- шрифт. Подраздел font позволяет вам выбрать из списка подходящий шрифт для режима командной строки.

## LILO

Здесь вас спросят об установке LILO (Linux LOader - загрузчик Linux. Если Slackware является единственной системой на вашем компьютере, то опция simple должна замечательно работать с вашей системой. Если у вас есть несколько операционных систем, то вам необходимо выбрать раздел expert. Третий пункт - не устанавливая, не рекомендован, до тех пор, пока у вас нет весьма серьёзных оснований поступить так. Если вы выполняете expert install, вам будет предоставлен выбор, куда разместить LILO. Вы можете разместить его в MBR - (Master Boot Record - главный загрузочный сектор) вашего жёсткого диска. В суперблок корневого каталога Linux, или на флоппи диск. Пожалуйста, обратите внимание на то, что если вы пользуетесь другой операционной системой, которая имеет свой загрузчик, то вам необходимо знать, что в случае размещения LILO в MBR прежний загрузчик будет замещен.

### Network

- сеть. Раздел настройки сети, на самом деле выполнение отдельной программы netconfig.

### CD-ROM

Здесь вас спросят, хотите ли вы, чтобы система автоматически проверяла, есть ли диск в CD-ROM и монтировала его, если таковой имеется при загрузке.

### X Window Manager

- Менеджер X Window. Здесь вы можете выбрать, какой менеджер окон использовать по умолчанию.

В независимости от того, какие пакеты были установлены, программа установки спросит вас, хотите ли вы установить пароль суперпользователя. Настоятельно рекомендуется сделать это, по крайней мере из соображений безопасности; тем не менее, почти как и всё в Slackware, вы можете и не делать этого.

EXIT    Выход из программы setup

## Глава 12. Управление накопителями и файловыми системами

С каждой файловой системой поставляются не только драйвера этой системы, но и программы, которые позволяют работать с ней: создавать, настраивать, проверять файловую систему. Поскольку количество поддерживаемых в Linux файловых систем очень большое, не удалось (да и не имело большого смысла) писать универсальные программы, которые поддерживали бы особенности всех файловых систем. И если завтра какой-нибудь супер гениальный Вася Пупкин придумает свою мега файловую систему, ему кроме драйверов придётся поставлять программы для работы с ней.

Ранее мы увидели как создает файловые системы программа `setup`. Теперь необходимо разобраться что же происходит на самом деле.

Мы научились создавать разделы на жестком диске компьютера, познакомившись с программой `fdisk`. Теперь же нам необходимо научиться создавать в выбранном разделе файловую систему и управлять ей.

### Программы для работы с накопителями

#### **badblocks**

Программа предназначена для проверки дискового пространства на наличие сбойных блоков.

`badblocks [параметры] устройство`

Программе необходимо указывать файл устройства, которое она будет проверять.

Рассмотрим некоторые параметры `badblocks`.

- **-b**      Указывает размер блока в байтах. Этот параметр желательно определять всегда. Размер используемого блока зависит от типа создаваемой файловой системы и её размера.
- **-o**      Определяет имя файла, куда программа будет помещать номера сбойных блоков. Затем этот файл можно передать программе, создающей файловую систему, чтобы она не использовала отмеченные блоки.
- **-i**      Определяет имя файла, в котором находятся номера сбойных блоков. Его можно использовать для того, чтобы программа лишней раз не обращалась к этим блокам.
- **-p**      Параметр определяет количество проходов.
- **-n**      Использовать для проверки режим только для чтения. Параметр по умолчанию.
- **-s**      Показывать номера проверяемых блоков.
- **-v**      Выдавать подробную информацию во время работы программы.
- **-w**      Использовать для проверки режим записи. Гарантированная потеря данных!

Например, необходимо проверить устройство `/dev/hda1` на наличие сбойных блоков. Размер раздела около 5 Гбайт. Предполагается, что программа создания файловой системы будет использовать блоки размером 4096. Результаты работы программы следует сохранить в файле `bad_list`.

```
# badblocks -b 4096 -v -o bad_list /dev/hda1
Checking blocks 0 to 1311296
Checking for bad blocks (read-only test): done
Pass completed, 0 bad blocks found.
# cat bad_list
#
```

#### **mkfs**

Mkfs создает файловые системы.

`mkfs [-V] [-t тип] [параметры] файл [блоки]`

При вызове программы, необходимо обязательно указать файл, в котором будет создаваться файловая система. Это может быть не только файл блочного устройства, но и простой файл файловой системы.

При помощи параметра `-t` определяется тип создаваемой файловой системы. Программа `mkfs` является оболочкой, в этом можно убедиться, набрав в командной строке `mkfs` и два раза нажав на символ табуляции.

```
# mkfs
mkfs          mkfs.cramfs    mkfs.ext3      mkfs.minix     mkfs.reiserfs
mkfs.bfs      mkfs.ext2     mkfs.jfs       mkfs.ntfs      mkfs.xfs
#
```

Как Вы видите, существует много различных программ для создания конкретных файловых систем.

После указания типа создаваемой системы, можно указать параметры файловой системы. Параметры зависят от типа системы.

Следующий пример показывает, как создать файловую систему типа `ext2` в разделе `/dev/hda3`:

```
# mkfs -t ext2 /dev/hda3
mke2fs 1.40.8 (13-Mar-2008)
Filesystem label=
OS type: Linux
Block size=1024 (log=0)
Fragment size=1024 (log=0)
50400 inodes, 200812 blocks
10040 blocks (5.00%) reserved for the super user
First data block=1
Maximum filesystem blocks=67371008
25 block groups
8192 blocks per group, 8192 fragments per group
2016 inodes per group
Superblock backups stored on blocks:
    8193, 24577, 40961, 57345, 73729

Writing inode tables: done
Writing superblocks and filesystem accounting information: done

This filesystem will be automatically checked every 26 mounts or
180 days, whichever comes first.  Use tune2fs -c or -i to override.
#
```

При создании файловой системы можно провести проверку диска на наличие сбойных блоков. Для это необходимо использовать параметр `-c`. В этом случае произойдет проверка в режиме чтения.

```
# mkfs -t ext2 -c /dev/hda3
mke2fs 1.40.8 (13-Mar-2008)
Filesystem label=
OS type: Linux
Block size=1024 (log=0)
Fragment size=1024 (log=0)
50400 inodes, 200812 blocks
10040 blocks (5.00%) reserved for the super user
First data block=1
Maximum filesystem blocks=67371008
25 block groups
8192 blocks per group, 8192 fragments per group
2016 inodes per group
Superblock backups stored on blocks:
    8193, 24577, 40961, 57345, 73729

Checking for bad blocks (read-only test): done
Writing inode tables: done
Writing superblocks and filesystem accounting information: done

This filesystem will be automatically checked every 21 mounts or
180 days, whichever comes first.  Use tune2fs -c or -i to override.
#
```

Если необходимо провести проверку в режиме записи, параметр `-c` следует указать два раза. Например так:



```
# mkfs -t ext2 -c -c /dev/hda3
mke2fs 1.40.8 (13-Mar-2008)
Filesystem label=
OS type: Linux
Block size=1024 (log=0)
Fragment size=1024 (log=0)
50400 inodes, 200812 blocks
10040 blocks (5.00%) reserved for the super user
First data block=1
Maximum filesystem blocks=67371008
25 block groups
8192 blocks per group, 8192 fragments per group
2016 inodes per group
Superblock backups stored on blocks:
    8193, 24577, 40961, 57345, 73729

Testing with pattern 0xaa: done
Reading and comparing: done
Testing with pattern 0x55: done
Reading and comparing: done
Testing with pattern 0xff: done
Reading and comparing: done
Testing with pattern 0x00: done
Reading and comparing: done
Writing inode tables: done
Writing superblocks and filesystem accounting information: done

This filesystem will be automatically checked every 37 mounts or
180 days, whichever comes first. Use tune2fs -c or -i to override.
#
```

**Внимание!** Форматирование файловых систем ext2 и ext3 с проверкой на сбойные блоки — это очень долгая процедура.

Файловая система создается очень быстро. Дело в том, что при ее создании не происходит форматирование как это принято в файловых системах FAT и NTFS. При создании выделяется суперблок в котором хранится информация о файловой системе, корневая директория и некоторая служебная информация. Затем копия суперблока равномерно распределяется по дисковому пространству, об этом говорит строка Superblock backups stored on blocks:. Все остальное дисковое пространство не форматируется. А зачем? Ведь жесткий диск уже имеет низкоуровневое форматирование, он разделен на блоки. Вот этими блоками в дальнейшем и будет пользоваться драйвер файловой системы. То есть дисковое пространство будет использоваться по мере необходимости. Блоки, используемые файлами, будут описываться в метаданных (inode) этих файлов.

Так же обратите внимание на то, что в файловой системе ext2 (да и в ext3) заранее выделяется количество inodes. То есть заранее известно максимально возможное количество файлов. В нашем примере на файловую систему объемом 200Мбайт выделено: 50400 inodes.

Следующая строка говорит о том, что за суперпользователем резервируется 5% дискового пространства. Это значение по умолчанию. Если оно кажется Вам большим, его можно определить при помощи параметра -m, указав число процентов. Например так:

```
# mkfs -t ext2 -m 2 /dev/hda3
mke2fs 1.40.8 (13-Mar-2008)
Filesystem label=
OS type: Linux
Block size=1024 (log=0)
Fragment size=1024 (log=0)
50400 inodes, 200812 blocks
4016 blocks (2.00%) reserved for the super user
First data block=1
Maximum filesystem blocks=67371008
25 block groups
8192 blocks per group, 8192 fragments per group
2016 inodes per group
Superblock backups stored on blocks:
```

```
8193, 24577, 40961, 57345, 73729
```

```
Writing inode tables: done
Writing superblocks and filesystem accounting information: done
```

```
This filesystem will be automatically checked every 33 mounts or
180 days, whichever comes first. Use tune2fs -c or -i to override.
#
```

В приведенном примере, при создании файловой системы за суперпользователем резервируется 2%.

**Внимание!** Следует обязательно запомнить, что зарезервированное место будет доступно только суперпользователю. Оно будет использоваться для различных административных задач, например, проверки файловой системы. Остальные пользователи не имеют доступа к этому пространству.

```
This filesystem will be automatically checked every 33 mounts or
180 days, whichever comes first. Use tune2fs -c or -i to override.
```

«Эта файловая система будет автоматически проверяться каждые 33 подключений или через 180 дней, в зависимости от того, что случится первым. Используйте tune2fs с параметрами `-c` или `-i` для изменения параметров». О чем идет речь? В суперблоке файловой системы есть поле состояние файловой системы. После корректного отключения файловой системы в это поле прописывается состояние `clean`. Если отключение было не правильным, например, при помощи кнопки `Reset`, в этом поле записи `clean` не будет. При старте системы запускается программа проверки файловой системы. Она (программа) смотрит значение поля состояния и если там не `clean`, выполняет проверку файловой системы. В противном случае, проверка не производится. Но чудес не бывает, в любой программе есть ошибки (если в программе нет ошибок, значит программа выводит на экран «Hello word». И то, все зависит от компилятора, некоторые компиляторы умудрялись и в таких программах делать ошибки). Так и в файловой системе они могут накапливаться. Поэтому, несмотря на состояние файловой системы, ее периодически необходимо проверять. В нашем случае, проверка будет производиться после 33 подключений.

Число, которое определяет после какого количества подключений, будет производиться принудительная проверка — псевдослучайное. В этом можно легко убедиться запустив создание файловой системы `ext2` на одном и том же устройстве (или файле) несколько раз подряд. Данная мера введена для ускорения загрузки системы: представим себе, что у нас имеется десяток файловых систем, при создании каждой из которых было выбрано, что проверку необходимо производить после 33 подключений. Если эти файловые системы подключались и отключались равное количество раз (что вполне возможно), то после 33-го подключения их все придется проверить (а сделано это будет при старте системы), и если файловых систем много, и каждая из них имеет значительный размер, то загрузка может затянуться. Назначение же каждой файловой системе псевдослучайного числа подключений, после которого будет производиться проверка, позволит разнести «плановые» проверки на разные загрузки.

Если Ваш сервер очень долго работал без выключения, проверка файловой системы будет обязательно производиться после 180 дней (полгода).

Как уже говорилось выше, `mkfs` — это программа оболочка. В зависимости от параметра `-t` она вызывает программу, предназначенную для создания конкретной файловой системы.

Для создания файловых систем типа `ext2` и `ext3` будут вызываться соответственно программы `mkfs.ext2` и `mkfs.ext3`. Но если внимательно к ним присмотреться, то мы увидим, что это просто символичные ссылки на программу `mke2fs`. Именно последняя программа будет создавать перечисленные выше файловые системы.

```
# file /sbin/mkfs.ext2
/sbin/mkfs.ext2: symbolic link to `mke2fs'
#
```

```
# file /sbin/mkfs.ext3
/sbin/mkfs.ext3: symbolic link to `mke2fs'
#
```

Аналогичная ситуация с `mkfs.reiserfs` и `mkfs.jfs`.

```
# file /sbin/mkfs.reiserfs
/sbin/mkfs.reiserfs: symbolic link to `mkreiserfs'
#
```

```
# file /sbin/mkfs.jfs
/sbin/mkfs.jfs: symbolic link to `jfs_mkfs'
#
```

А вот mkfs.xfs — это реальная программа, создающая файловую систему XFS.

```
# file /sbin/mkfs.xfs
/sbin/mkfs.xfs: ELF 32-bit LSB executable, Intel 80386, version 1 (SYSV),
dynamically linked (uses shared libs), stripped
#
```

Давайте рассмотрим эти программы подробнее. Начнем с mke2fs.

## **mke2fs**

Программа предназначена для создания файловых систем типа ext2 и ext3.

**mke2fs [параметры] файл**

Если указать только файл устройства, то будет создаваться файловая система ext2. Для создания файловой системы ext3 необходимо явно указать параметр `-j`. В этом случае будет создаваться файл журнала.

По умолчанию размер файла журнала равен 32 Мбайт и он располагается в создаваемой файловой системе. Но Вы можете повлиять на размер журнала и его размещение. Для этих целей следует использовать параметр `-J` и дополнительные опции:

- `-J size=4M` — определяет размер файла журнала.
- `-J device=file` — определяет устройство, на котором будет находиться файл журнала.

Журнал должен быть предварительно создан:

**mke2fs -O устройство файл.**

При помощи параметра `-L` можно установить метку (label) на файловую систему.

Параметр `-m` позволяет указать, сколько процентов от объема файловой системы будет зарезервировано за суперпользователем. Значение параметра указывается в процентах.

## **mkreiserfs**

Программа предназначена для создания файловой системы типа reiserfs.

**mkreiserfs [параметры] файл**

Из параметров программы следует выделить:

- `-s` — определяет размер журнала. Значение в блоках.
- `-t` — определяет максимальный размер транзакции. Значение в блоках. Размер не должен превышать половину размера журнала.
- `-l` — определяет метку файловой системы.

У программы нет опции, заставляющей ее производить проверку на сбойные блоки при форматировании файловой системы. Но есть возможность, при помощи параметра `-B` указать файл, в котором находится информация о сбойных блоках раздела. Этот файл создается программой badblocks.

Вот пример:

```
# badblocks -b 4096 -o bad_list /dev/hda3
# cat bad_list
2369728
2369762
2369763
2369764
2369765
2369766
2369767
# mkreiserfs -B bad_list /dev/hda3
mkreiserfs 3.6.19 (2003 www.namesys.com)
```

*A pair of credits:*

*Edward Shushkin wrote the encryption and compression file plugins, and the V3 journal relocation code.*

*Yury Umanets (aka Umka) developed libreiser4, userspace plugins, and all userspace tools (reiser4progs) except of fsck.*

*Guessing about desired format.. Kernel 2.6.24.5 is running.*

*Format 3.6 with standard journal*

*Count of blocks on the device: 50192*

*Number of blocks consumed by mkreiserfs formatting process: 8213*

*Blocksize: 4096*

*Hash function used to sort names: "r5"*

*Journal Size 8193 blocks (first block 18)*

*Journal Max transaction length 1024*

*inode generation number: 0*

*UUID: 9e1e771a-3c12-447a-84e4-bfc3b3ef9dba*

*ATTENTION: YOU SHOULD REBOOT AFTER FDISK!*

*ALL DATA WILL BE LOST ON '/dev/hda3'!*

*Continue (y/n):y*

*Initializing journal - 0%....20%....40%....60%....80%....100%*

*Syncing..ok*

*Tell your friends to use a kernel based on 2.4.18 or later, and especially not a kernel based on 2.4.9, when you use reiserFS. Have fun.*

*ReiserFS is successfully created on /dev/hda3.*

*#*

Какая же она многословная! Обратите внимание на то, кто разрабатывает эту файловую систему — сплошные соотечественники.

### ***jfs\_mkfs***

Программа предназначена для создания файловых систем типа JFS.

**jfs\_mkfs** [параметры] файл

У программы не много опций, две из них показаны ниже:

- **-c** - Производить проверку на сбойные блоки перед созданием файловой системы.
- **-L** - Определяет метку файловой системы.

*# jfs\_mkfs -c /dev/hda3*

*jfs\_mkfs version 1.1.12, 24-Aug-2007*

*Warning! All data on device /dev/hda3 will be lost!*

*Continue? (Y/N) y*

*100 percent of the disk has been formatted.*

*Format completed successfully.*

*200812 kilobytes total disk space.*

*#*

### ***mkfs.xfs***

Программа предназначена для создания файловой системы типа XFS

**mkfs.xfs** [параметры] файл

Параметров много, очень много. Некоторые из параметров имеют большое количество дополнительных опций. Рассмотрим некоторые из них.

- **-b** - Определяет размер блока. При его указании требуется наличие дополнительной опции: **-b size=4096**.
- **-i** - Определяет параметры inode. Требуется указание дополнительных параметров.
- **-L** - Определяет метку файловой системы.
- **-f** - Принудительное (force) создание файловой системы.

```
# mkfs.xfs /dev/hda3
mkfs.xfs: /dev/hda3 appears to contain an existing filesystem (jfs).
mkfs.xfs: Use the -f option to force overwrite.
#
# mkfs.xfs -f /dev/hda3
meta-data=/dev/hda3          isize=256    agcount=4, agsize=12551 blks
                        =               sectsz=512   attr=2
data      =                  bsize=4096   blocks=50203, imaxpct=25
                        =               sunit=0      swidth=0 blks
naming    =version 2          bsize=4096
log        =internal log      bsize=4096   blocks=1200, version=2
                        =               sectsz=512   sunit=0 blks, lazy-count=0
realtime  =none              extsz=4096   blocks=0, rtextents=0
#
```

При первой попытке создания файловой системы, программа выдала предупреждение о том, что в этом разделе уже есть файловая система. Пришлось явно указывать параметр **-f** для того, что бы файловая системы была создана.

## tune2fs

При помощи этой программы можно настраивать параметры уже существующих файловых систем типа ext2 и ext3.

### tune2fs параметры файл

Рассмотрим некоторые параметры программы.

- **-j** добавляет файл журнала. При помощи этого параметра можно превратить файловую систему ext2 в ext3. Иногда, при повреждении файловой системы ext3, программа проверки удаляет файл журнала. Tune2s позволяет создать файл по новой.
- **-c max-mount-counts** количество подключений файловой системы перед принудительной проверкой
- **-i interval-between-checks [d|m|w]** количество дней до принудительной проверки файловой системы
- **-J** позволяет настраивать параметры журнала. При определении требуется наличие дополнительных параметров.
- **size** определяет размер файла журнала.
- **device** определяет устройство, на котором будет расположен файл журнала.
- **-l** позволяет получить информацию, хранящуюся в суперблоке.
- **-L** позволяет изменить метку файловой системы.
- **-m** позволяет изменить процент дискового пространства резервируемого за суперпользователем.

Посмотрим содержимое суперблока файловой системы типа ext2.

```
# tune2fs -l /dev/hda3
tune2fs 1.40.8 (13-Mar-2008)
Filesystem volume name:   <none>
Last mounted on:          <not available>
Filesystem UUID:          ba9359bc-4672-44a4-8a02-e381d4e35bc9
Filesystem magic number:  0xEF53
Filesystem revision #:    1 (dynamic)
Filesystem features:      ext_attr resize_inode dir_index filetype sparse_super
Filesystem flags:         signed_directory_hash
Default mount options:    (none)
```

```
Filesystem state:      clean
Errors behavior:      Continue
Filesystem OS type:   Linux
Inode count:          50400
Block count:          200812
Reserved block count: 10040
Free blocks:          192899
Free inodes:          50389
First block:          1
Block size:           1024
Fragment size:        1024
Reserved GDT blocks:  256
Blocks per group:      8192
Fragments per group:   8192
Inodes per group:      2016
Inode blocks per group: 252
Filesystem created:    Wed Sep 10 20:33:28 2008
Last mount time:       n/a
Last write time:       Wed Sep 10 20:33:28 2008
Mount count:           0
Maximum mount count:   33
Last checked:          Wed Sep 10 20:33:28 2008
Check interval:        15552000 (6 months)
Next check after:      Mon Mar 9 19:33:28 2009
Reserved blocks uid:   0 (user root)
Reserved blocks gid:   0 (group root)
First inode:           11
Inode size:            128
Default directory hash: tea
Directory Hash Seed:   a122c7b5-7c9c-48a0-82fb-64d5b495c10f
#
```

Теперь добавим файл журнала.

```
# tune2fs -j /dev/hda3
tune2fs 1.40.8 (13-Mar-2008)
Creating journal inode: done
This filesystem will be automatically checked every 33 mounts or
180 days, whichever comes first. Use tune2fs -c or -i to override.
#
```

После добавления файла журнала файловую систему можно использовать как ext3.

И напоследок, уменьшим процент зарезервированного за суперпользователем дискового пространства до двух процентов. Ему этого объема хватит.

```
# tune2fs -m 2 /dev/hda3
tune2fs 1.40.8 (13-Mar-2008)
Setting reserved blocks percentage to 2% (4016 blocks)
#
```

Для настройки файловых систем других типов используются собственные программы:

Reiserfs — **reiserfstune**.

JFS — **jfs\_tune**.

XFS — **xfs\_admin**.

## fsck

Программа предназначена для проверки файловых систем.

**fsck** [-t тип] [параметры] файл

Наиболее часто используемые параметры:

- **-a**       автоматически исправлять ошибки
- **-A**       проверять все файловые системы, описанные в файле `/etc/fstab`
- **-f**       принудительная проверка файловой системы

По аналогии с программой `mkfs`, `fsck` — это программа оболочка, которая в зависимости от типа файловой системы вызывает программу, предназначенную для проверки этой файловой системы.

Ext2/Ext3 — **e2fsck**.

Reiserfs — **reiserfsck**.

JFS — **jfs\_fsck**.

XFS — **fsck.xfs**.

**Внимание!** Настоятельно рекомендуется производить проверку не подключенных или работающих в режиме только для чтения файловых систем.

Проверять файловые системы, работающие в режиме чтения/записи можно, но Вы рискуете потерять данные. Как уже говорилось выше, файловые системы в Linux не нуждаются в дефрагментации. Механизм, при помощи которого файловые системы не фрагментируются, основан на том, что при создании или изменении файла его данные располагаются на новом месте. Если при проверке файловой системы, будет изменяться файл, программа проверки может этого не понять. Я лично наблюдал, как один из слушателей на курсах решил проверить это утверждение и начал проверку корневой файловой системы, работающей в режиме полного доступа. Ну что сказать? Систему пришлось устанавливать заново.

У нас есть одна файловая система, которую нельзя отключить или перевести в режим работы только для чтения. Это корневая файловая система. Если ее отключить, система перестанет работать. Ее, в принципе, можно перевести в режим только для чтения, но для этого нужно закрыть все программы, которые держат открытыми файлы в этой файловой системе. А это означает, что проще систему перезагрузить. Именно по этому корневую файловую систему рекомендуется делать как можно меньше. А директории, в которых предполагается большое количество операций с файлами, выносить на отдельные разделы. Проверка корневой файловой системы производится при старте компьютера, когда она еще подключена в режиме только для чтения.

В суперблоке файловых систем есть поле, описывающее состояние файловой системы. В момент выключения машины происходит отключение файловой системы и в это поле записывается состояние: `clean`. Ниже приведена часть вывода программы `tune2fs -l`:

```
Filesystem state:      clean
```

При проверке файловой системы, программа `fsck` считывает значение этого поля. И если его значение `clean`, проверка не производится.

```
# fsck /dev/hda3
fsck 1.40.8 (13-Mar-2008)
e2fsck 1.40.8 (13-Mar-2008)
/dev/hda3: clean, 11/50400 files, 12027/200812 blocks
#
```

Если Вы хотите произвести принудительную проверку файловой системы, воспользуйтесь параметром `-f` (`force`):

```
# fsck -f /dev/hda3
fsck 1.40.8 (13-Mar-2008)
e2fsck 1.40.8 (13-Mar-2008)
Pass 1: Checking inodes, blocks, and sizes
Pass 2: Checking directory structure
Pass 3: Checking directory connectivity
Pass 4: Checking reference counts
Pass 5: Checking group summary information
/dev/hda3: 11/50400 files (9.1% non-contiguous), 12027/200812 blocks
#
```

Как видно из примера, была произведена полная проверка файловой системы.

## Использование файловых систем

### mount

Программа предназначена для подключения файловых систем, просмотра списка подключенных систем и изменения параметров подключения.

`mount [-t тип] [параметры] [файл директория]`

`mount [-t тип] [параметры] [файл | директория]`

Для получения списка подключенных файловых систем, программу следует запускать без указания каких либо параметров.

```
# mount
/dev/hda5 on / type xfs (rw)
proc on /proc type proc (rw)
sysfs on /sys type sysfs (rw)
usbfs on /proc/bus/usb type usbfs (rw)
/dev/hda6 on /home type xfs (rw)
/dev/hda7 on /var type xfs (rw)
/dev/hda1 on /mnt/win type fuseblk
(rw,nosuid,nodev,noatime,allow_other,default_permissions,blksize=4096)
tmpfs on /dev/shm type tmpfs (rw)
#
```

Программа берет данные из файла `/proc/mounts`.

```
# cat /proc/mounts
rootfs / rootfs rw 0 0
/dev/root / xfs rw,ikeep,noquota 0 0
proc /proc proc rw 0 0
sysfs /sys sysfs rw 0 0
tmpfs /dev tmpfs rw 0 0
devpts /dev/pts devpts rw 0 0
usbfs /proc/bus/usb usbfs rw 0 0
/dev/hda6 /home xfs rw,ikeep,noquota 0 0
/dev/hda7 /var xfs rw,ikeep,noquota 0 0
/dev/hda1 /mnt/win fuseblk
rw,nosuid,nodev,noatime,user_id=0,group_id=0,default_permissions,allow_other 0 0
tmpfs /dev/shm tmpfs rw 0 0
#
```

У программы `mount` много параметров, рассмотрим некоторые из них:

- **-a** Смонтировать все файловый системы, описанные в файле `/etc/fstab`. При указания типа файловой системы будут подключены все файловые системы этого типа.
- **-n** Подключить все файловые системы. Но не записывать информацию о них в файл `/etc/mtab` (О файлах `/etc/fstab` и `/etc/mtab` будет рассказано ниже).
- **-p** В случае использования устройства `loop` для шифрования файловой системы, задает пароль.
- **-r** Подключать файловую систему в режиме только для чтения.
- **-w** Подключать файловую систему в режиме полного доступа. Параметр по умолчанию.
- **-t** Определяет тип подключаемой файловой системы. Обычно этот параметр требуется указывать при подключении файловых систем, находящихся в файлах. При монтировании устройств, в большинстве случаев, `mount` самостоятельно определяет тип файловой системы.
- **-o** Позволяет указать параметры монтирования файловых систем. Если параметров много, они должны перечисляться через запятую, без использования пробелов.
- **--move** Позволяет подключить смонтированную файловую систему к другой точке монтирования.



## Параметры монтирования файловых систем

При подключении файловой системы, при помощи параметра `-o` можно указать параметры монтирования, влияющие на ее дальнейшую работу. У разных файловых систем существуют различные параметры. Какие параметры можно использовать написано в странице руководства по программе `mount`. Но! Обновления ядра Linux выходят чаще, чем обновления страницы руководства. Поэтому лучше смотреть документацию к ядру, находящуюся в его исходных кодах. Обычно документация находится в директории `/usr/src/linux/Documentation`. Информация о файловых системах находится в директории `filesystems`.

```
# ls /usr/src/linux/Documentation/filesystems/
00-INDEX                ext2.txt                proc.txt
9p.txt                  ext3.txt                quota.txt
Exporting               ext4.txt                ramfs-rootfs-initramfs.txt
Locking                 files.txt               relay.txt
adfs.txt                fuse.txt                romfs.txt
affs.txt                gfs2.txt                smbfs.txt
afs.txt                 hfs.txt                 spufs.txt
automount-support.txt   hfsplus.txt             sysfs-pci.txt
befs.txt                hpfs.txt                sysfs.txt
bfs.txt                 inotify.txt              sysv-fs.txt
cifs.txt                isofs.txt                tmpfs.txt
coda.txt                jfs.txt                 udf.txt
configfs/               locks.txt                ufs.txt
cramfs.txt              mandatory-locking.txt    vfat.txt
dentry-locking.txt      ncpfs.txt                vfs.txt
directory-locking       ntfs.txt                 xfs.txt
dlmfs.txt                ocfs2.txt                xip.txt
ecryptfs.txt            porting
#
```

## Параметры монтирования, общие для всех файловых систем

- **rw** Работа в режиме полного доступа. Параметр по умолчанию.
- **ro** Работа в режиме только для чтения.
- **defaults** Включаются перечисленные параметры монтирования: `rw`, `suid`, `dev`, `exec`, `auto`, `nouser`, `async`.
- **exec** Позволяет исполнять файлы в этой файловой системе.
- **auto** Файловая система будет подключаться программой `mount`, запущенной с параметром `-a`.
- **noatime** Не изменять значение поля время последнего доступа (`access time`).
- **noauto** Файловая система НЕ будет подключаться программой `mount`, запущенной с параметром `-a`.
- **nodev** Игнорировать файлы устройств в этой файловой системе.
- **nosuid** Игнорировать специальные биты доступа (`SUID` и `SGID`) в этой файловой системе.
- **owner** Позволяет отключать файловую систему только тому пользователю, который ее подключил.
- **remount** Говорит программе `mount`, что данную файловую систему следует только переподключить. Применяется тогда, когда необходимо передать новые параметры монтирования уже подключенной файловой системе. Например, при переводе файловой системы из режима полного доступа в режим только для чтения.
- **sync** Включает синхронный режим работы. В этом случае не используются буфера файловой системы, и данные сразу попадают на диск. Это режим удобно использовать в работе со съемными накопителями, такими как гибкий диск.
- **user** Позволяет обыкновенному пользователю системы подключать и отключать файловую систему. По умолчанию эту операцию может производить только суперпользователь.

## Параметры монтирования файловой системы ext3

- **data** Определяет, какой режим работы с журналом будет использоваться файловой системой. Возможные значения: `writeback`, `ordered` (режим по умолчанию) и `journal`. Значение режимов было описано выше в разделе Файловая система `ext3`.

- **commit** Определяет количество секунд, через которое данные, находящиеся в буфере будут сбрасываться на жесткий диск. Значение по умолчанию — 5 секунд. На ноутбуках, это значение следует увеличить, с целью сбережения энергии. Увеличение значения увеличивает быстродействие файловой системы.
- **user\_xattr** Разрешает применение дополнительных файловых атрибутов. Требует соответствующей поддержки в ядре Linux.
- **acl** Разрешает использование POSIX ACL. Требует соответствующей поддержки в ядре Linux.
- **errors** Определяет, что необходимо сделать если во время работы файловой системы возникают ошибки.
  - Возможные значения: **remount-ro** — перевести файловую систему в режим только для чтения и продолжить работу. Значение по умолчанию. **continue** — ничего не делать, продолжать работу. **panic** — отключить машину.
- **quota, usrquota, grpquota, uqnoenforce** Позволяет включить квотирование дискового пространства.

### **Параметры монтирования файловой системы JFS**

- **resize** Позволяет изменить размер файловой системы при ее повторном подключении. Если значение у параметра не указано, будет использован весь раздел, в котором эта файловая система находится.
- **nointegrity** Позволяет отключить использование журнала. Применяется для ускорения проверки файловой системы.
- **integrity** Включает поддержку записи метаданных в журнал. Значение по умолчанию.
- **errors** Имеет такие же значение и параметры как аналогичный параметр файловой системы ext3.

### **Параметры монтирования файловой системы XFS**

- **noatime** Не изменять значение поля время последнего доступа (access time). Увеличивает скорость работы файловой системы.
- **quota, usrquota, grpquota, uqnoenforce** Позволяет включить квотирование дискового пространства.

### **Параметры монтирования файловой системы iso9660**

- **gid** Определяет группу, которой будут принадлежать все файлы, находящиеся в этой файловой системе.
- **uid** Определяет пользователя, которому будут принадлежать все файлы, находящиеся в этой файловой системе. Если CD-ROM был создан в Windows, в файловой системе не будет использовано дополнительное расширение, позволяющее сохранять атрибуты Linux файловых систем. Поэтому, по умолчанию, все файлы, находящиеся в этой файловой системе, принадлежат тому пользователю, который ее подключил.
- **umask** Определяет маску пользователя, изменяющую права доступа.
- **iocharset** Определяет кодировку, которая используется для отображения символов на Linux машине. Параметр предназначен для того, что бы драйвер на лету осуществлял перекодировку символов в именах файлов.
- **nojoliet** Отключение Joliet extensions.
- **norock** Отключение Rock Ridge extensions.

### **Параметры монтирования файловой системы vfat**

- **umask** Определяет маску пользователя, изменяющую права доступа.
- **dmask** Определяет маску пользователя, изменяющую права доступа к директориям.
- **fmask** Определяет маску пользователя, изменяющую права доступа к файлам.
- **codepage** Определяет кодировку, которая используется в именах файлов в этой файловой системе. Для русских букв следует явно указывать кодировку 866.

- **iocharset**      Определяет кодировку, которая используется для отображения символов на Linux машине.

### Параметры монтирования файловой системы *ntfs*

- **umask**      Определяет маску пользователя, изменяющую права доступа.
- **dmask**      Определяет маску пользователя, изменяющую права доступа к директориям.
- **fmask**      Определяет маску пользователя, изменяющую права доступа к файлам.
- **iocharset**      Устаревший параметр. Рекомендуется использовать параметр **nl**.
- **nl**      Определяет кодировку, которая используется для отображения на Linux машине.
- **gid**      Определяет группу, которой будут принадлежать все файлы в этой файловой системе.
- **uid**      Определяет пользователя, которому будут принадлежать все файлы в этой файловой системе.

## umount

Программа предназначена для отключения файловых систем.

`umount [параметры] директория | устройство`

**Внимание!** Отключить можно только неиспользуемую файловую систему. То есть ту систему, в которой ни одна программа не держит файлы открытыми, и нет занятых (текущих) директорий

В качестве аргумента можно указывать либо файл устройства, либо точку монтирования.

При указании параметра `-a` будут отключены все файловые системы, описанные в файле `/etc/mtab`, кроме виртуальной файловой системы `/proc`. Некоторые версии `umount` используют файл `/proc/mounts`.

## Работа с файловыми системами

Для подключения файловой системы необходимо использовать точку монтирования — любую пустую директорию. Можно использовать директорию, в которой есть файлы и директории, но после подключения к ней файловой системы, старые файлы и директории видны не будут. Сами файлы никуда не пропадают, они становятся доступными после отключения файловой системы.

Сначала создадим директорию, к которой будем подключать файловую систему. И только потом ее смонтируем.

```
# mkdir /mnt/free
# mount -t ext3 /dev/hda3 /mnt/free
# mount
/dev/hda5 on / type xfs (rw)
proc on /proc type proc (rw)
sysfs on /sys type sysfs (rw)
usbfs on /proc/bus/usb type usbfs (rw)
/dev/hda6 on /home type xfs (rw)
/dev/hda7 on /var type xfs (rw)
/dev/hda1 on /mnt/win type fuseblk
(rw,nosuid,nodev,noatime,allow_other,default_permissions,blksize=4096)
tmpfs on /dev/shm type tmpfs (rw)
/dev/hda3 on /mnt/free type ext3 (rw)
#
```

Теперь посмотрим содержимое директории `/mnt/free`. В ней мы увидим содержимое подключенной файловой системы.

```
# ls /mnt/free
lost+found/
#
```

В подавляющем большинстве случаев, наличие директории `lost+found` говорит, что перед нами корень физической файловой системы. Эта директория используется программой `fsck`. При обнаружении потерянных блоков, она восстанавливает их в виде файлов в директории `lost+found`. Предполагается, что затем Вы сможете восстановить данные.

Файловая система автоматически подключилась в режиме полного доступа, поэтому проверять ее нельзя. Что необходимо сделать, что бы ее проверить? Перевести в режим только для чтения. Для этого воспользуемся программой `mount` и параметрами монтирования `ro` и `remount`.

```
# mount -o ro,remount /mnt/free
# mount
/dev/hda5 on / type xfs (rw)
proc on /proc type proc (rw)
sysfs on /sys type sysfs (rw)
usbfs on /proc/bus/usb type usbfs (rw)
/dev/hda6 on /home type xfs (rw)
/dev/hda7 on /var type xfs (rw)
/dev/hda1 on /mnt/win type fuseblk
(rw,nosuid,nodev,noatime,allow_other,default_permissions,blksize=4096)
tmpfs on /dev/shm type tmpfs (rw)
/dev/hda3 on /mnt/free type ext3 (ro)
#
```

Теперь попробуем создать файл в этой файловой системе.

```
# touch /mnt/free/test
touch: cannot touch `/mnt/free/test': Read-only file system
#
```

Проверим файловую систему на наличие ошибок. `Fsck` предупредит о том, что мы пытаемся проверить используемую файловую систему. Но мы знаем, что она подключена в режиме только для чтения и ее можно проверять.

```
# fsck -f /dev/hda3
fsck 1.40.8 (13-Mar-2008)
e2fsck 1.40.8 (13-Mar-2008)
/dev/hda3 is mounted.
```

```
WARNING!!! Running e2fsck on a mounted filesystem may cause
SEVERE filesystem damage.
```

```
Do you really want to continue (y/n)? yes
```

```
Pass 1: Checking inodes, blocks, and sizes
Pass 2: Checking directory structure
Pass 3: Checking directory connectivity
Pass 4: Checking reference counts
Pass 5: Checking group summary information
/dev/hda3: 11/50400 files (9.1% non-contiguous), 12027/200812 blocks
#
```

После проверки, переводим файловую систему в режим полного доступа.

```
# mount -o rw,remount /mnt/free
# mount
/dev/hda5 on / type xfs (rw)
proc on /proc type proc (rw)
sysfs on /sys type sysfs (rw)
usbfs on /proc/bus/usb type usbfs (rw)
/dev/hda6 on /home type xfs (rw)
/dev/hda7 on /var type xfs (rw)
/dev/hda1 on /mnt/win type fuseblk
(rw,nosuid,nodev,noatime,allow_other,default_permissions,blksize=4096)
tmpfs on /dev/shm type tmpfs (rw)
/dev/hda3 on /mnt/free type ext3 (rw)
#
```

Еще одна форма записи этой же команды:

```
# mount -w -o remount /mnt/free
```

Для отключения файловой системы воспользуемся программой `umount`.

```
# umount /dev/hda3
umount: /mnt/free: device is busy
umount: /mnt/free: device is busy
#
```

Упс! Не получилось. Файловая система кем то используется и ее отключить нельзя. Что делать? Необходимо узнать какая программа использует файловую систему и либо завершить ее работу, либо сделать так, что бы программа перестала использовать директорию в этой файловой системе. Если программа держит открытым файл в этой файловой системе и не производит операции чтения или записи, то система может быть отключена. Попробуем определить, кто нам мешает. Получить список открытых файлов можно при помощи программы `lsof`.

```
# lsof /mnt/free
COMMAND  PID  USER   FD   TYPE DEVICE SIZE NODE NAME
bash     2417 root    cwd   DIR   3,3 1024    2 /mnt/free
#
```

Итак, мы видим, что программа `bash`, запущенная пользователем `root` в данный момент находится в директории `/mnt/free`. Достаточно перейти в другую директорию и повторить попытку отключения файловой системы.

Другой способ получения информации об использовании файловой системы — это программа `fuser`.

```
# fuser -v /mnt/free
USER          PID ACCESS COMMAND
/mnt/free:    root      2417 ..c.. bash
#
```

В отличие от `lsof`, `fuser` может не только показать какие программы используют файловую систему, но и послать этим программам сигнал.

```
:~# fuser -km /mnt/free
/mnt/free:    2417c
#
```

При помощи параметра `-k` мы говорим, что бы `fuser` послал сигнал `KILL(9)` всем программам, использующим указанный файл. А параметр `-m` говорит, что файл — это точка монтирования либо файл блочного устройства и сигнал посылается всем программам, которые используют файлы в этой файловой системе.

Теперь файловую систему можно отключать.

```
# umount /mnt/free
# mount
/dev/hda5 on / type xfs (rw)
proc on /proc type proc (rw)
sysfs on /sys type sysfs (rw)
usbfs on /proc/bus/usb type usbfs (rw)
/dev/hda6 on /home type xfs (rw)
/dev/hda7 on /var type xfs (rw)
/dev/hda1 on /mnt/win type fuseblk
(rw,nosuid,nodev,noatime,allow_other,default_permissions,blksize=4096)
tmpfs on /dev/shm type tmpfs (rw)
#
```

## Файлы `fstab` и `mtab`

Это конфигурационные файлы программ `mount`, `umount` и `fsck`.

Начнем с файла `/etc/mtab`. Когда программа `mount` подключает файловую систему, она дописывает соответствующую строку в `/etc/mtab`. Когда `umount` отключает файловую систему, из этого файла соответствующая строка удаляется.

```
# cat /etc/mtab
/dev/hda5 / xfs rw 0 0
proc /proc proc rw 0 0
sysfs /sys sysfs rw 0 0
usbfs /proc/bus/usb usbfs rw 0 0
```

```
/dev/hda6 /home xfs rw 0 0
/dev/hda7 /var xfs rw 0 0
/dev/hda1 /mnt/win fuseblk
rw,nosuid,nodev,noatime,allow_other,default_permissions,blksize=4096 0 0
tmpfs /dev/shm tmpfs rw 0 0
#
```

Поскольку это обыкновенный текстовый файл, Вы можете при помощи редактора добавлять или удалять информацию. Поэтому верить содержимому этого файла нельзя.

Файл `/etc/fstab` имеет большое значение для системы. В нем описывают все файловые системы, которые Вы собираетесь использовать.

```
# cat /etc/fstab
/dev/hda2      swap          swap          defaults      0      0
/dev/hda5      /             xfs           defaults      1      1
/dev/hda6      /home        xfs           defaults      1      2
/dev/hda7      /var         xfs           defaults      1      2
/dev/hda1      /mnt/win     ntfs-3g       umask=077     1      0
#/dev/cdrom    /mnt/cdrom   auto          noauto,owner,ro 0      0
/dev/fd0       /mnt/floppy  auto          noauto,owner   0      0
devpts        /dev/pts     devpts        gid=5,mode=620 0      0
proc          /proc        proc          defaults      0      0
tmpfs         /dev/shm     tmpfs         defaults      0      0
#
```

Формат файла `/etc/fstab` очень простой. Символ комментария — `#`. Одна запись — одна строка. В строке есть шесть полей:

1. Файл устройства.
2. Точка монтирования.
3. Тип файловой системы. Возможно перечисление нескольких файловых систем через запятую, или использование ключевого слова `auto`, для автоматического определения файловой системы.
4. Параметры монтирования файловой системы разделяемые запятой. Пробелы ни в коем случае использовать нельзя!
5. Флаг программы `dump`. Поскольку эта программа в Linux практически не используется, этот флаг оставлен для совместимости.
6. Флаг программы `fsck`. Если он равен нулю, значит эта файловая система при запуске программы `fsck` с параметром `-A` не будет автоматически проверяться, в том числе и при старте машины. Если равен единице, значит проверка будет производиться. Файловые системы проверяются в том порядке, в котором они описаны в этом файле. Если Вы хотите изменить порядок проверки, вместо единицы используйте двойку. Тогда сначала проверяются все файловые системы с единицей, а затем с двойкой.

**Внимание!** Последняя строка в файле `/etc/fstab` должна быть пустой.

Вернемся к файловой системе `/dev/hda3`. Если ничего не сделать, она не будет автоматически монтироваться при старте машины. Чудес не бывает. Все чудеса в Linux описаны в каком либо конфигурационном файле. Поэтому мы добавим строку, описывающую файловую систему `/dev/hda3` в файл `/etc/fstab`. Она будет выглядеть следующим образом:

```
/dev/hda3 /mnt/free ext3 defaults 0 1
```

Устройство, точка монтирования, тип файловой системы, параметры монтирования, флаг программы `dump`, флаг программы `fsck`.

Вопрос заключается в следующем: в какой строке файла `fstab` следует добавить эту строку? Файловые системы монтируются в том порядке, в котором они описаны в этом файле (Правда из этого правила существуют исключения, например корневая директория и некоторые другие файловые системы. Мы рассматривали эти случаи, когда изучали систему инициализации Linux.). Принцип очень простой — на момент монтирования файловой системы должна существовать точка монтирования (директория). А директория будет существовать, если файловая система, в которой она физически размещена, уже смонтирована. В нашем случае, директория `/mnt/free` находится в той же файловой системе, что и корневая файловая система Linux, в `/dev/hda3`. Таким образом, мы можем эту строку написать в любой строке файла, но после того места, где описано монтирование корневой файловой системы. Например так:

```
# cat /etc/fstab
/dev/hda2      swap          swap          defaults      0      0
/dev/hda5      /             xfs           defaults      1      1
/dev/hda6      /home         xfs           defaults      1      2
/dev/hda7      /var          xfs           defaults      1      2
/dev/hda1      /mnt/win      ntfs-3g       umask=077     1      0
/dev/hda3      /mnt/free     ext3          defaults      0      1
/dev/cdrom     /mnt/cdrom    auto          noauto,owner,ro 0      0
/dev/fd0       /mnt/floppy   auto          noauto,owner   0      0
devpts         /dev/pts      devpts        gid=5,mode=620 0      0
proc           /proc         proc          defaults      0      0
tmpfs          /dev/shm      tmpfs         defaults      0      0
#
```

Теперь смонтируем файловую систему.

```
# mount /mnt/free
# mount
/dev/hda5 on / type xfs (rw)
proc on /proc type proc (rw)
sysfs on /sys type sysfs (rw)
usbfs on /proc/bus/usb type usbfs (rw)
/dev/hda6 on /home type xfs (rw)
/dev/hda7 on /var type xfs (rw)
/dev/hda1 on /mnt/win type fuseblk
(rw,nosuid,nodev,noatime,allow_other,default_permissions,blksize=4096)
tmpfs on /dev/shm type tmpfs (rw)
/dev/hda3 on /mnt/free type ext3 (rw)
#
```

Обратите внимание на то, что при вызове программы `mount` я указал только точку монтирования. Дело в том, что файл `/etc/fstab` — это конфигурационный файл программы `mount`. И все недостающие параметры программа берет из него. Вместо точки монтирования можно написать файловую систему — `/dev/hda3`, результат будет таким же.

Теперь внесем кое-какие изменения, связанные со строками, где описываются параметры монтирования CD-ROM и гибкого диска.

В строке, описывающей параметры монтирования CD-ROM необходимо добавить еще один тип файловой системы — `udf`. Если это рабочая станция, то простым пользователям системы необходимо разрешить монтировать CD-ROM, за это отвечает параметр монтирования `user`. Я обычно не пишу параметры `iocharset` и `codepage`, отвечающие за преобразование русских букв. Так как использую CD записанные не только в Windows. В результате строка будет выглядеть следующим образом:

```
/dev/cdrom /mnt/cdrom iso9660,udf noauto,owner,user,ro 0 0
```

Строка, описывающая параметры монтирования гибкого диска, тоже не идеальна. Поэтому мы над ней немного пошаманим. Во-первых, Вы никогда не будете форматировать гибкие диски с использованием Linux файловых систем. Как Вы наверное помните, слишком много места на диске отводится под служебную информацию. Следовательно, не удастся использовать весь объем гибкого диска. Поэтому мы будем использовать файловую систему `fat`.

В Linux поддерживаются две разновидности (с точки зрения Linux) файловых систем: `msdos` и `vfat`. Если в поле, определяющем тип файловой системы, оставить параметр `auto` (автоматическое определение файловой системы), Вас ожидает маленькая неприятность — после монтирования диска все имена файлов будут выглядеть так как это было в MS-DOS (ктонибудь еще помнит эту чудо OS?): имя состоит из восьми символов, расширение из трех. Это происходит потому, что при автоматическом определении файловой системы выбирается `msdos`. Для нормального отображения имен файлов необходимо использовать файловую систему `vfat`. Поэтому вместо `auto` мы напишем `vfat`.

Если это рабочая станция, простым пользователям системы можно позволить монтировать файловую систему, используя параметр `user`. Ну и напоследок, можно указать параметры, отвечающие за преобразование русских кодировок в именах файлов: `iocharset` и `codepage`. В результате строка будет выглядеть так:

```
/dev/fd0 /mnt/floppy vfat noauto,user,owner,iocharset=koi8-r,codepage=866 0 0
```

## Использование USB накопителей

**Внимание!** Все USB накопители видны в системе как SCSI устройства.

Как узнать какое SCSI устройство использовать для подключения USB накопителя? Это очень легко. Сначала подключаете накопитель. Затем, при помощи программы fdisk с параметром -l определяете, какие накопители подключены к системе. Программу fdisk может запускать только суперпользователь.

```
# fdisk -l
```

```
Disk /dev/hda: 17.1 GB, 17179869184 bytes
255 heads, 63 sectors/track, 2088 cylinders
Units = cylinders of 16065 * 512 = 8225280 bytes
Disk identifier: 0x40627a92
```

Device	Boot	Start	End	Blocks	Id	System
/dev/hda1	*	1	653	5245191	7	HPFS/NTFS
/dev/hda2		654	678	200812+	82	Linux swap
/dev/hda3		679	703	200812+	83	Linux
/dev/hda4		704	2088	11125012+	5	Extended
/dev/hda5		704	1677	7823623+	83	Linux
/dev/hda6		1678	1800	987966	83	Linux
/dev/hda7		1801	2088	2313328+	83	Linux

```
Disk /dev/sda: 8204 MB, 8204057600 bytes
33 heads, 63 sectors/track, 7707 cylinders
Units = cylinders of 2079 * 512 = 1064448 bytes
Disk identifier: 0x7c6dd3c8
```

Device	Boot	Start	End	Blocks	Id	System
<b>/dev/sda1</b>		<b>1</b>	<b>7708</b>	<b>8011759</b>	<b>b</b>	<b>W95 FAT32</b>

```
#
```

Как видите, этот накопитель (Flash) был определен как устройство /dev/sda1. Дальнейшие действия такие же, как и с любыми другими накопителями: монтируем, работаем, отключаем.

```
# mkdir /mnt/flash
# mount -t vfat /dev/sda1 /mnt/flash -o iocharset=koi8-r,codepage=866
# ls /mnt/flash/
AutoRun.exe* AutoRun.inf*
# umount /mnt/flash
#
```

Иногда, при работе с флеш накопителями могут возникать проблемы. Дело в том, что некоторые флеш не разбиваются на разделы, а форматируются целиком. В этом случае fdisk будет выдавать на экран полную ерунду. К сожалению (а может быть и к счастью) у меня под рукой не нашлось такого устройства, поэтому показать как это выглядит на экране не могу. Просто учтите, что в этом случае для монтирования следует использовать не раздел, а устройство, например, /dev/sda.

Если флеш накопитель не определяется, надо смотреть, поддерживает ли Ваша система USB как таковой и USB накопители в частности? Для этого, после подключения накопителя, в первую очередь следует посмотреть содержимое директории /proc, на предмет наличия директории /proc/bus/usb. Если эта директория есть, значит USB поддерживается. После этого загляните в файл devices, который должен находиться в указанной директории. В этом файле видны все найденные USB устройства. Если в списке присутствует Ваш накопитель, значит устройство было определено.

Ниже я покажу только часть файла devices, уж очень он большой.

```
T: Bus=01 Lev=01 Prnt=01 Port=00 Cnt=01 Dev#= 3 Spd=480 MxCh= 0
D: Ver= 2.00 Cls=00(>ifc ) Sub=00 Prot=00 MxPS=64 #Cfgs= 1
P: Vendor=058f ProdID=6387 Rev= 1.41
S: Manufacturer=JetFlash
S: Product=Mass Storage Device
S: SerialNumber=BBLT3LOP
C:* #Ifs= 1 Cfg#= 1 Atr=80 MxPwr=100mA
I:* If#= 0 Alt= 0 #EPs= 2 Cls=08(stor.) Sub=06 Prot=50 Driver=usb-storage
```



```
E: Ad=01 (O) Atr=02 (Bulk) MxPS= 512 Iv1=0ms
E: Ad=82 (I) Atr=02 (Bulk) MxPS= 512 Iv1=0ms
```

Вот такие строки описывают флеш накопитель в данный момент подключенный к моей машине. На что следует обратить внимание? На *Driver=usb-storage*. То есть устройство было определено как накопитель. Если в списке для Вашего устройства не был определен драйвер, значит Ваше устройство не поддерживается или не загружен соответствующий драйвер.

Если при наличии драйвера *fdisk* все равно не видит устройство. Посмотрите содержимое директории */dev*, на наличие файлов устройств SCSI: */dev/sda*, */dev/sda1*, */dev/sdb* и т.д. Если их нет, значит у Вас барахлит *udev* (система, применяемая с ядрами Linux версии 2.6, предназначенная для автоматического создания файлов устройств.). Тут два варианта, либо настроить *udev*, либо создавать файлы устройств самому.

## df

Программа *df* (disk free) показывает степень использования дисковых накопителей.

*df* [параметры] [файл...]

Программа показывает информацию только о подключенных файловых системах.

```
# df
Filesystem      1K-blocks      Used Available Use% Mounted on
/dev/hda5        7813380    3650336    4163044   47% /
/dev/hda6         983164         8352     974812    1% /home
/dev/hda7        2303088     32948     2270140    2% /var
/dev/hda1        5245188    2536724     2708464   49% /mnt/win
/dev/hda3        194449         5664     184769    3% /mnt/free
tmpfs            125588           0     125588    0% /dev/shm
#
```

Информация выдается в блоках. Один блок равен 1 Кбайт. Если Вы хотите получать информацию в более привычном виде, используйте параметр *-h*.

```
# df -h
Filesystem      Size  Used Avail Use% Mounted on
/dev/hda5       7.5G  3.5G  4.0G   47% /
/dev/hda6       961M   8.2M  952M    1% /home
/dev/hda7       2.2G   33M   2.2G    2% /var
/dev/hda1       5.1G  2.5G  2.6G   49% /mnt/win
/dev/hda3       190M   5.6M  181M    3% /mnt/free
tmpfs           123M     0   123M    0% /dev/shm
#
```

Если необходимо получить информацию о количестве используемых *inodes*, применяют параметр *-i*.

```
# df -i
Filesystem      Inodes   IUsed   IFree IUse% Mounted on
/dev/hda5      7823616 232272 7591344    3% /
/dev/hda6      987904     4 987900    1% /home
/dev/hda7     2313280   2051 2311229    1% /var
/dev/hda1     2741232  18103 2723129    1% /mnt/win
/dev/hda3       50400     11  50389    1% /mnt/free
tmpfs          31397     1  31396    1% /dev/shm
#
```

Программа позволяет получить информацию о конкретной файловой системе. Для этого необходимо явно указать интересующую файловую систему.

```
# df /dev/hda3
Filesystem      1K-blocks      Used Available Use% Mounted on
/dev/hda3        194449         5664     184769    3% /mnt/free
#
```

## du

Программа du (disk usage) показывает, сколько места занимают файлы в директории.

du [параметры] [файл...]

Если при вызове программы не была указана директория, программа получает информацию о текущей директории.

```
# du
0      ./log/setup/tmp
84     ./log/setup
12884  ./log/packages
0      ./log/removed_packages
0      ./log/removed_scripts
3552   ./log/scripts
0      ./log/cups
0      ./log/sa
0      ./log/httpd
0      ./log/iptraf
0      ./log/nfsd
0      ./log/samba
0      ./log/uucp
16720  ./log
.....
160    ./www/cgi-bin
12     ./www/error/include
204    ./www/error
260    ./www/icons/small
948    ./www/icons
6808   ./www
24     ./yp
24052  .
#
```

Как Вы видите, программа выдает информацию о каждой директории и только в самом конце выводит итоговое значение. Размеры выдаются в блоках. Один блок равен 1 Кбайт.

Если Вы хотите получить только итоговое значение, используйте параметр `-s`.

```
# du -sh
24M    .
#
```

Вот здесь хочу привести вам пример, как администратор может, используя знание shell script облегчить себе жизнь.

Наверняка ранее вам приходилось встречаться с ситуацией, когда место на жестком диске катастрофически сокращается и нужно выяснить какой из подкаталогов сколько «весит». В среде MS Windows это сделать не так легко и наглядно как в Linux. Вот пример скрипта, который я использую в своей повседневной жизни.

```
# cat /sbin/bdu
#!/bin/bash
for I in `ls`
do
    if [ -d $I ]
    then
        cd $I
        echo -n "`pwd` "; du -sh
        cd - > /dev/null
    fi
done
echo -n "`pwd` "; du -sh
#
```

У программы du есть один «минус». При ее использовании, она выдает размеры всего дерева подкаталогов. А если это дерево очень велико? Вы будете сидеть и смотреть на бегущие строки по экрану, мало что успевая

запечатлеть. Можно, конечно, вывод программы `du` передать, например, программе `more` или `grep`, но ведь задача перед вами наверняка иная.

Вот вывод программы `bdu` (название вы можете использовать любое, лишь бы не совпадало с системными):

```
# bdu
/var/X11 1.9M .
/var/X11R6 1.9M .
/var/adm 17M .
/var/cache 280K .
/var/db 0 .
/var/empty 0 .
/var/lib 80K .
/var/lock 0 .
/var/log 17M .
/var/mail 20K .
/var/man 0 .
/var/named 12K .
/var/run 92K .
/var/rwho 0 .
/var/spool 32K .
/var/state 0 .
/var/tmp 0 .
/var/www 6.7M .
/var/yp 24K .
/var 24M .
#
```

Данный скрипт обрабатывает содержимое только текущей директории. Если у вас есть желание поэкспериментировать – напишите скрипт, который будет использовать в качестве аргумента путь к интересующему каталогу.

## Глава 13. Работа с пользователями

Если бы пользователей не было, для администраторов настали бы райские дни. Но, к сожалению, приходится констатировать, что администраторы, в подавляющем большинстве случаев — это обслуживающий персонал. Обслуживающий, в том числе, и пользователей. Хотя меня очень радует, что администратору не приходится готовить кофе и убирать за пользователями их рабочие места.

Вы должны понимать, что в Linux тоже есть понятие пользователь сервера и пользователь домена. Информация о пользователях сервера хранится в файлах на самом сервере. Информация о пользователях домена хранится в специальных программах: сервера NIS или LDAP. В этом разделе мы будем рассматривать только пользователей сервера.

### Файл *passwd*

Файл `/etc/passwd` — это текстовая база данных, в которой описываются учетные записи пользователей сервера.

Формат файла очень простой и уважающий себя администратор должен знать его наизусть. На одного пользователя в файле отводится одна строка. Поля в строке разделяются символами двоеточия. В файле семь полей:

1. Имя учетной записи пользователя (login).
2. Хеш пароля пользователя.
3. Уникальный номер пользователя в системе (UID).
4. Уникальный номер основной группы пользователя (GID).
5. Дополнительная информация.
6. Домашняя директория пользователя.
7. Программа, которая запускается при входе пользователя в систему.

Обязательными полями считаются только: login, UID и GID основной группы. Все остальные поля можно не определять.

Имя учетной записи пользователя — это обязательное поле. Имя должно быть уникальным. Максимальная длина 256 символов (Если в Вашей компании Вы с кем-то не очень дружите, можете такому «другу» сделать учетную запись с именем длиной в 256 символов. Ему это будет приятно. Особенно по утрам, когда он будет входить в систему.). В имени можно использовать только английские буквы, большие или маленькие (они отличаются) и цифры.

В современных системах хеш пароля пользователя не хранится в файле `passwd`, он вынесен в отдельный файл `/etc/shadow`. Сделано это потому, что файл `passwd` обязательно должен быть доступен на чтение всем пользователям системы. Несмотря на то, что в файле хранился хеш пароля (это значит, что пароль пользователя нельзя получить методом обратного преобразования), его можно подобрать. То есть, из пароля получают хеш и сравнивают с хешем, хранящимся в файле. Если в поле пароль в файле `passwd` стоит символ `x` — это значит, что хеш пароля находится в файле `/etc/shadow`.

Уникальный номер пользователя в системе — это обязательное поле. Существует два типа пользователей: суперпользователь (root) и все остальные. UID суперпользователя равен нулю. Иногда у администраторов возникает не здоровое желание поменять имя суперпользователя. Аргументируют это тем, что все пытаются получить пароль пользователя root. Если мы поменяем имя суперпользователя, допустим на `admin`, то никто не сможет подобрать его пароль. Так вот — это не правильное решение. Файл `/etc/passwd` обязательно должен быть доступен для чтения всем пользователям системы. Если кто угодно может прочесть его содержимое, значит они могут узнать новый логин суперпользователя.

Уникальный номер группы, к которой принадлежит пользователь. Особенностью Linux (как и любого другого UNIX) является то, что пользователь обязательно является членом хотя бы одной группы. Эту группу называют основной группой пользователя (primary group). Все файлы, которые создает пользователь, принадлежат этому пользователю и его основной группе.

Дополнительная информация о пользователе — это не обязательный параметр. В этом поле можно описать пользователя. Написать его имя и фамилию, домашний и рабочий телефон и т.д. Существует специальный формат — GECOS, описывающий какие поля и для чего применяются в этом поле. Если Вы хотите сделать

адресную книгу, для этого лучше использовать какойнибудь сервер LDAP: OpenLDAP или eDirectory.

Домашняя директория пользователя — это домашняя директория пользователя.

Последнее, седьмое поле очень интересное. Там записана программа, которая будет выполняться при входе пользователя в систему. Пользователь работает в системе до тех пор, пока работает эта программа. Обычно в этом поле написан shell, например: /bin/bash.

Теперь представьте себе, что у Вас есть удаленный офис, в котором работают три кассира: Баба Маша, Баба Глаша и Баба Оля. Это женщины весьма преклонного возраста. Они начали работать кассирами еще во времена Царя Гороха, когда самым мощным вычислительным устройством были счеты. Поэтому научить их работать на компьютере крайне сложно. Постоянные звонки в службу поддержки:

— Сынок, у меня куда то пропала программа!

— Баба Глаша, может быть ты ее минимизировала?

— Чего Я сделала?

Знакомая ситуация?

Программа для работы с кассой — это клиент к какойнибудь базе данных. Можно попросить программистов написать такой клиент для Linux (Если программистам заплатить деньги они могут написать программу, работающую в любой операционной системе.), причем работать он будет в текстовом режиме. А кто сказал, что графический режим для таких программ самый удобный? Не факт.

Поставьте на рабочие места этих женщин компьютеры с Linux. Добавьте учетные записи. В качестве программы оболочки укажите программу касса. Теперь осталось научить наших клиентов пользоваться Linux.

— Баба Глаша, смотри. Компьютер включается вот этой большой кнопкой. Теперь по экрану побежали буковки. Не пугайтесь, так надо. Вот на экране появилось иностранное слово Login. Введите Ваш логин. Это то слово, которое я написал Вам на бумажке.

— Ага, вот это слово. А можно я этот стикер приклею на экран?

— Можно. Теперь на экране появилось иностранное слово Password. Сейчас Баба Глаша я вам скажу Ваш пароль. Вы его запомните, только не пугайтесь.

Тут Вы говорите пароль Бабе Глаше. Она этот пароль ОБЯЗАТЕЛЬНО запомнит. Почему? Объясню чуть позже. Баба Глаша слушает пароль, краснеет, но запоминает его.

Поскольку в качестве оболочки Вы установили программу касса, значит именно эта программа будет запущена при входе пользователя в систему. Для Бабы Глаши касса известна вдоль и поперек. Работать в этой программе она умеет. Я надеюсь, что программисты не предусмотрят выход в систему из программы, потому что внешний вид командной строки Бабу Глашу сразит наповал. После завершения работы программы, на экране будет знакомое для Бабы Глаши приглашение входа в систему.

Для выключения компьютера достаточно нажать на кнопку выключения питания. Останется настроить систему энергосбережения так, что бы система правильно реагировала на нажатие на кнопку выключения питания.

Внедряя описанную Выше систему, Вы облегчаете жизнь себе и Вашим клиентам. Более того, клиенты смогут работать только в тех программах, в которых им разрешено работать. Один раз в Сбербанке я наблюдал очень интересную картину. Сидит кассир и обслуживает клиентов. На соседней машине сидит, как я понимаю, дочь кассира и набирает реферат! И все это происходит в Банке!

## Файл shadow

В файле /etc/shadow хранится информация о паролях и учетных записях пользователей.

**Внимание!** Файл должен быть доступен на чтение и запись только суперпользователю.

В файле на одного пользователя отводится одна строка. В строке есть девять полей, отделяемых друг от друга двоеточием:

1. Логин пользователя.
2. Хеш пароля пользователя.
3. Дата последнего изменения пароля пользователя.
4. Сколько дней после изменения пароля, пользователь не имеет право изменять свой пароль.
5. Сколько дней действителен пароль пользователя.

6. За сколько дней до устаревания пароля, при входе пользователя в систему, выводится предупреждение о смене пароля.
7. Сколько дней после окончания срока действия пароля, пользователь может входить в систему. При входе сразу будет запускаться программа смены пароля.
8. Дата окончания действия учетной записи пользователя.
9. Девятое поле не используется и должно быть пустым.

Первое поле необходимо для связи двух файлов: `/etc/passwd` и `/etc/shadow`.

Во втором поле хранится хеш пароля пользователя. Например, там может быть такая строка:

`$2a$10$ziHh/qOFqWRcLk3HN60rwe1ptT3TX0tBoZAgoilLZGz0C.12eX5bu`

Алгоритм хеширования не предусматривает обратного преобразования. То есть из последовательности символов нельзя получить пароль. Но, его можно подобрать. Из предполагаемого пароля получают хеш и сравнивают с хеш, который хранится в файле `shadow`. Поэтому файл `shadow` должен быть доступен только суперпользователю. Для подбора пароля можно воспользоваться программами `Crack` и `John the ripper`. Администратор сам должен регулярно использовать эти программы для обнаружения слабых паролей пользователей.

Подобрать пароль можно и не имея его хеш. Это можно сделать "методом тыка", просто перебирая все возможные пароли при входе в систему. Поэтому пароли должны быть сложными, длиной не менее восьми символов и не содержать имена жен, детей, собак и т.д. Желательно, что бы пароль состоял из какой либо абракадабры, иначе не смотря на длину, его легко могут подобрать по словарю. Причем не надо надеяться на то, что нет словарей русских слов. Кроме того, существуют словари русского транслита и словари русских слов набранных на английской клавиатуре.

С другой стороны, сложные пароли пользователями не запоминаются и очень часто мониторы оклеены стикерами с паролями. Существует замечательный метод, который позволяет делать сложные пароли легко запоминаемые пользователями. При этом ни одного стикера с паролем на мониторе Вы гарантировано не увидите. Это так называемый метод шокирующих паролей. Всем известно, что кроме основного русского языка существует второй русский язык. На втором русском в приличном обществе не общаются, но все прекрасно его знают. Богатство этого языка просто впечатляет. Теперь вспомните пример с кассой. Баба Глаша гарантировано забудет любой сложный пароль. Но если вы спросите ее: "Баба Глаша, скажи на втором русском все что ты думаешь о своем начальстве". Она Вам такое скажет! Так завернет! Установите эту фразу в качестве пароля. Для невозможности подбора по словарю (и такие словари встречаются в интернет), разделяйте слова в фразе каким либо символом: плюсом, пробелом и т.д. Что Вы в результате получите: этот пароль пользователь никогда не забудет и ни одного стикера на мониторе!

Если в поле пароля Вы видите любой символ, например, `*` или `!`. Это значит, что перед нами так называемый фиктивный пользователь. Он никогда не сможет войти в систему. Ведь какой бы пароль он ни ввел, из этого пароля после хеширования никогда не получится один символ. Такие пользователи предназначены для того, чтобы с их правами выполнялись программы.

Если вам необходимо временно заблокировать учетную запись пользователя, так что бы этот пользователь не мог входить в систему. Достаточно перед хешем пароля поставить любой символ. Например, восклицательный знак:

`!$2a$10$ziHh/qOFqWRcLk3HN60rwe1ptT3TX0tBoZAgoilLZGz0C.12eX5bu`

Теперь, какой бы пароль пользователь не ввел, в результате хеширования пароля никогда не получится строка, начинающаяся с символа `!`. Для разблокирования пароля достаточно удалит восклицательный знак, у пользователя восстанавливается его старый пароль. При этом ни до блокировки, ни после разблокирования мы не знали пароль пользователя. Кстати, такой механизм блокировки использует программа `passwd`, но о ней мы поговорим позже.

Дата последнего изменения пароля — это количество дней, прошедших после 1 Января 1970 года. Теперь попробуйте вычислить сколько дней прошло после этой замечательной даты. Слава Богу, что никто не заставляет этот файл редактировать руками. Все изменения в файл обычно вносятся при помощи специальных программ.

Назначение остальных полей должно быть понятно из их названия.

## Файл group

В файле `/etc/group` описаны группы пользователей. Формат файла тоже очень простой. Одна строка на одну группу. В строке четыре поля разделенные символом двоеточия:

1. Имя группы.
2. Хеш пароля группы.
3. Уникальный номер группы — GID.
4. Список членов группы, разделенных запятой.

**Внимание!** В Linux, в отличие от Windows, в группу могут входить только пользователи. Группа в группу входить не может!

Поскольку группа в группу входить не может, встречаются ситуации когда в системе присутствуют пользователи и группы с одинаковыми именами. Например, пользователь root и группа root.

**Внимание!** Добавление пользователя в группу root не дает ему прав пользователя root! Пользователь получает определенные права доступа к файлам, принадлежащим группе root и не более.

Добавить пользователя в группу очень просто, достаточно дописать его в конце списка пользователей необходимой группы.

Теперь предположим, что у Вас в системе 500 или 1000 пользователей. В одну из групп входит 250 пользователей. Представляете себе длину строки, описывающей эту группу в файле /etc/group? Поэтому, если у вас предполагается большое количество пользователей, не рекомендуется использование локальных файлов /etc/passwd, /etc/shadow и /etc/group. Используйте специальное программное обеспечение — сервера LDAP такие как OpenLDAP, Novell eDirectory или Red Hat Directory Server.

## Программы для работы с учетными записями пользователей

Создание, изменение и удаление строк в файлах паролей и групп зависит от версии операционной системы, которой Вы пользуетесь. Обычно файлы паролей редактируются непосредственно из командной строки, либо используются утилиты с графическим интерфейсом. Эти утилиты позволяют очень легко и удобно редактировать учетные записи пользователей.

Основные команды для работы с учетными записями в Linux - **useradd**, **adduser**, **userdel**, и **usermod**, а также средство редактирования файлов паролей **vipw**. Интерфейс команд следующий:

```
useradd      [-u uid [-o]] [-g group] [-G group,...]
              [-d home] [-s shell] [-c comment] [-m [-k template]]
              [-f inactive] [-e expire ] [-p passwd] name
userdel      [-r] name
usermod      [-u uid [-o]] [-g group] [-G group,...]
              [-d home [-m]] [-s shell] [-c comment] [-l new_name]
              [-f inactive] [-e expire ] [-p passwd] [-L|-U] name
```

Основные параметры имеют следующие значения:

- **name** - регистрационное имя пользователя. Это единственный обязательный параметр во всех командах.
- **uid** - является уникальным идентификатором пользователя, связанным с этим именем.
- **-o** - позволяет повторяться одному и тому же идентификатору пользователя.
- **-g** - выбирает главную группу для регистрационного имени. Параметр group нужно указать в виде номера основной группы пользователя (gid)
- **-G** - выбирает дополнительные группы. Здесь дополнительные группы можно указать в символьном виде. Например, root
- **home** - указывает на домашний каталог пользователя.
- **-m** - указывает на необходимость создания нового домашнего каталога или переноса текущего в новое место.
- **shell** - определяет оболочку интерпретатора команд для данного пользователя.
- **comment** - это дополнительный комментарий о пользователе.
- **template** — путь к каталогу, содержащему файлы, которые должны быть скопированы в новый

домашний каталог пользователя.

- **new\_name** — новое имя пользователя, для случаев замены имени login пользователя.
- **inactive** - указывает непрерывное число дней без регистрации в системе до того, как данная запись будет блокирована.
- **expire** - указывает точную дату, до которой действует регистрационная запись.
- **-r** — удаляется домашний каталог пользователя.

**Внимание!** При использовании команды `usermod` необходимо указывать все параметры учетной записи пользователя. При использовании этой программы фактически сначала происходит удаление соответствующих строк в конфигурационных файлах, а потом повторная запись с указанными параметрами. И если вы решите сменить лишь основную группу пользователя, то вам все равно нужно будет указать и все остальные параметры, иначе они будут удалены.

Программа `adduser` выполняет те же действия по созданию учетной записи пользователя, что и `useradd`, но является интерактивной. В процессе создания пользователя, программа `adduser` будет задавать вам вопросы, на которые вы должны будете ответить. Фактически, все вопросы которые будут заданы, соответствуют приведенным выше параметрам программы `useradd`.

Программа `adduser` в большинстве типов unix-like систем реализована в виде shell-scripts. В RedHat Linux это символическая ссылка на программу `useradd`. В Suse Linux такой программы нет.

## skel каталог

При создании учетной записи пользователя создается и его домашний каталог. В домашний каталог копируется некоторое количество файлов, определяющих первоначальную рабочую среду пользователя.

Администратор системы может изменять содержимое директории `/etc/skel`. Либо создать собственную директорию. Однако в этом случае нужно указать путь к каталогу с файлами пользовательской среды при создании учетной записи пользователя с параметром

`-k /путь/к/вашему/каталогу`

## vipw

Программа для прямого редактирования файлов `passwd` либо `master.passwd`. В некоторых unix-like системах файл `passwd` сохранился лишь для удовлетворения исторической справедливости. А файл `master.passwd` не доступен для прямого редактирования обычным способом. Точнее, при попытке сохранения изменений ничего не произойдет. А для успешного изменения этих файлов вручную нужно использовать программу `vipw`. Программа `vipw` открывает нужный файл в редакторе, который описан в переменной `$EDITOR`. Если эта переменная не определена, то в качестве редактора по умолчанию используется `vi`.

В Slackware Linux использование программы `vipw` необязательно. Для редактирования файла `passwd` можете использовать любой редактор, например, `joe`, однако команда `vipw` короче.

## Программа для смены пароля пользователя passwd

Эта команда используется для изменения паролей пользователей, блокирования и разблокирования учетных записей. Суперпользователь может менять пароли у любых пользователей системы. Обыкновенный пользователь — только свой пароль.

Формат ее вызова:

`passwd [name]`

`passwd [-x max] [-n min] [-w warn] [-i inact] login`

`passwd {-l|-u|-d|-S|-e} login`

Значение опций данной команды:

- **name (login)** — имя пользователя (используется только суперпользователем).
- **-x** устанавливает максимальный срок в днях до смены пароля (доступна только суперпользователю).
- **-n** устанавливает минимальный срок в днях до смены пароля (доступна только



суперпользователю).

- **-w** устанавливает срок в днях, когда пользователь начнет получать сообщения о необходимости смены пароля (доступна только суперпользователю).
- **-i** устанавливает срок в днях до момента, когда старый пароль перестанет быть активным и регистрационная запись блокируется (доступна только суперпользователю).
- **-l** блокирует указанного пользователя (доступна только суперпользователю). Блокировка осуществляется добавлением к паролю префикса **!**.
- **-u** разблокирование пользователя и удаление префикса **!** (доступна только суперпользователю).
- **-d** отмена пароля для пользователя (доступна только суперпользователю). Позволяет пользователю войти в систему без пароля и сменить его самостоятельно.
- **-S** выводит краткую информацию о состоянии пароля (о сроке его действия). Доступна только суперпользователю.
- **-e** изменяет срок до смена пароля до нуля дней.

## Управление группами

Управление группами (объединениями пользователей) ничуть не сложнее управления пользователями. В Linux предусмотрены команды для автоматизации создания и модификации групп, аналогичные тем, которые применяются для управления пользователями.

Все действия, осуществляемые программами `groupadd`, `groupdel` и `groupmod`, отражаются в файле `group`.

Новая группа создается командой `groupadd` (в некоторых дистрибутивах - `addgroup`). Чтобы создать группу, введите эту команду с именем создаваемой группы в качестве аргумента.

`groupadd [-g gid [-o]] group`

Созданной группе присваивается идентификационный номер, равный минимальному значению, которое больше 500 и больше значений идентификационных номеров остальных групп. Некоторые другие дистрибутивы Linux по умолчанию включают новых пользователей в группу (100).

Можно присвоить создаваемой группе идентификационный номер по своему усмотрению, введя команду с параметром `-g`.

**Внимание!** Если вы забыли номер требуемой группы, его можно найти в списке идентификационных номеров существующих групп, хранящемся в файле `/etc/group`.

Для изменения параметров существующей группы, то есть для смены имени группы либо ее `gid` необходимо воспользоваться программой `groupmod`:

`groupmod [-g gid [-o]] [-n name] group`

Здесь `name` – новое имя группы.

Для удаления группы:

`groupdel group`

При всей простоте команды процесс удаления группы связан с некоторыми тонкостями.

1. Файлы, принадлежавшие удаленной группе, не удаляются и не передаются другой группе.
2. Если для некоторого пользователя группа первична (то есть, именно она указана как группа этого пользователя в файле паролей), ее нельзя удалить.

Первая проблема решается точно так же, как и при удалении учетной записи. Сначала записывается идентификационный номер группы (его можно найти в файле `/etc/group`), затем, после удаления группы, при помощи команды `find` меняется принадлежность всех файлов этой группы. С программой `find` мы познакомимся несколько позднее.

## Программа `su`

`su` (сокр. от англ. *Substitute User*) — команда `unix-like` систем, позволяющая пользователю войти в систему под другим именем, не завершая текущий сеанс. Обычно используется для временного входа суперпользователем для выполнения административных работ.

### su [-] [имя\_пользователя [аргумент ... ]]

Команда `su` позволяет пользователю выполнять команды от имени другого пользователя, не завершая текущий сеанс, или получить роль. По умолчанию предполагается работа от имени пользователя `root` (суперпользователя).

Для использования `su` необходимо ввести соответствующий пароль (если только команду не вызывает пользователь `root`). Если введен правильный пароль, `su` создает новый процесс командного интерпретатора, с такими же реальными и эффективными идентификаторами пользователя и группы, а также списком дополнительных групп, что и у указанного пользователя. В качестве нового командного интерпретатора используется указанный в поле начальной программы записи файла `passwd` для соответствующего пользователя. Если командный интерпретатор там не указан, используется `sh`. Чтобы вернуться в прежний сеанс, введите символ конца файла, EOF (Ctrl-D) для завершения работы нового командного интерпретатора, либо команду `exit`.

Любые дополнительные аргументы, заданные в командной строке, передаются новому командному интерпретатору. При использовании программ типа `sh`, если аргумент имеет вид `-c` строка, то строка выполняется интерпретатором как команда, а аргумент `-r` приводит к запуску ограниченного командного интерпретатора.

Следующие утверждения верны, если в качестве начальной программы в учетной записи пользователя задана `/bin/sh` или пустая строка (что означает стандартный командный интерпретатор `/bin/sh`). Если первый аргумент `su` — дефис (`-`), среда будет установлена такой же, как при регистрации заданного пользователя. Иначе передается текущая среда, за исключением значения `$PATH`.

Все попытки стать другим пользователем с помощью команды `su` регистрируются в журнальном файле.

Чтобы стать пользователем `bin`, сохранив прежнюю среду, выполните команду:

```
$ su bin
```

Чтобы стать пользователем `bin`, изменив среду так, как если бы он действительно зарегистрировался в системе, выполните команду:

```
$ su - bin
```

Чтобы выполнить команду во временной среде и с правами пользователя `bin`, введите:

```
$ su - bin -c "command args"
```

## Настройка пользовательской среды окружения

Как вы уже знаете, при входе любого пользователя в систему для него запускается особый экземпляр оболочки — `login shell`. В процессе запуска в качестве `login shell` `bash` ищет следующие файлы:

`/etc/profile`

`~/.bash_profile`

`~/.bash_login`

`~/.profile`

(в указанном порядке) и выполняет содержащиеся в них команды. Если `bash` запускается повторно из командной строки в интерактивном режиме (т. е. не для выполнения какой-то одиночной команды), то он находит файл `~/.bashrc` и выполняет содержащиеся в нем команды. Впрочем, в дистрибутиве Mandrake файл `~/.bashrc` вызывается и для `login shell`, а из него вызывается еще и общесистемный файл `/etc/bashrc`. Так что, как видите, тут возможны варианты.

Файл `~/.profile` считывается только в том случае, если не найден файл `~/.bash_profile`. В противном случае файл `~/.profile` игнорируется.

Но какова бы ни была последовательность вызова этих скриптов, с их помощью для каждого сеанса работы пользователя создается так называемая "пользовательская среда" или окружение, представляющее собой набор переменных с установленными для них значениями. Эти значения считываются некоторыми программами и утилитами, и в соответствии с их значениями изменяется поведение системы в тех или иных ситуациях.

Файлы `/etc/profile` и `/etc/bashrc` определяют общесистемные настройки пользовательской среды, а остальные перечисленные файлы определяют индивидуальную среду конкретного пользователя. Сравнительно небольшие добавления или исправления в индивидуальных файлах настройки, изменяющие значения, заданные по умолчанию, могут сделать значительно более приятной для вас работу в системе (о вкусах, как известно, не

спорят, и вряд ли люди, которые определяли настройки по умолчанию, угадали ваши предпочтения). Поэтому давайте кратко рассмотрим основные переменные пользовательской среды и то, каким образом вы сами можете их изменять.

Вначале посмотрите переменные окружения, которые заданы по умолчанию. Как мы уже говорили ранее, это можно сделать с помощью команд `set` или `env`. Значение, присвоенное отдельной переменной, можно просмотреть с помощью команды `echo $name`, где `name` — имя переменной.

Из всех переменных, которые вы увидите по команде `set`, обычно меняют вид приглашения `PS1` и перечень путей поиска `PATH`. Так что осталось только решить, в каком именно скрипте задать этим переменным новые значения. Рассмотрим этот вопрос на примере переменной `PATH`.

Переменная `PATH` формируется в двух скриптах: `/etc/profile` (пути, общие для всех пользователей) и в одном из пользовательских скриптов (например, в `~/bash_profile`), где к ранее сформированному перечню пользователь может добавить пути по своему желанию. Только не стоит делать это в файле `~/bashrc`, так как последний перезапускается каждый раз при запуске второго, третьего и т. д. экземпляра оболочки. Для добавления пути в переменную `PATH` надо вписать в выбранный скрипт строку следующего вида (в этом примере в перечень добавляется путь `/home/user/bin`):

```
PATH=$PATH:/home/user/bin
```

Обратите внимание на то, что двоеточия в конце нет. И имейте в виду, что каталоги просматриваются в поисках нужного файла в том порядке, как они перечислены в переменной `PATH`.

В отличие от MS-DOS Linux не ищет исполняемый файл в текущем каталоге. Поэтому, если вы хотите, чтобы поиск производился и в текущем каталоге, надо добавить и этот каталог (напомним, что он имеет имя, состоящее из одной точки) в переменную `PATH`. Но имейте в виду, что с точки зрения безопасности добавлять текущий каталог в перечень путей поиска недопустимо, так как злоумышленник может поместить в один из доступных ему по записи каталогов вредоносную программу, названную именем одной из часто используемых системных утилит. И, когда вы запустите эту программу, считая, что запускаете системную утилиту, она может нанести большой вред вашей системе, тем более, если вы запустили ее от имени суперпользователя.

## Приглашения интерпретатора

Одна из очень важных переменных имеет имя `PS1`. Эта переменная задает вид приглашения, которое `bash` выводит, когда ожидает ввода очередной команды пользователем. По умолчанию этой переменной присвоено значение `"s-\v\$ "`. Вообще-то в `bash` существует четыре приглашения, которые используются в разных ситуациях. Переменная `PS1` задает вид строки приглашения, которая выдается тогда, когда оболочка ждет ввода команды. Вторичное приглашение, задаваемое переменной `PS2`, появляется тогда, когда оболочка ожидает от пользователя ввода еще каких-то данных, необходимых для продолжения работы запущенной команды или программы. По умолчанию переменная `PS2` имеет значение `">"`. Вы уже имели возможность видеть это приглашение, когда запускали команду `cat` для ввода данных с клавиатуры в файл.

Приглашение, задаваемое переменной `PS3`, используется в команде `select`. Приглашение, задаваемое переменной `PS4`, выводится перед каждой командой, в то время, когда `bash` отслеживает процесс выполнения. Значение по умолчанию — `"+"`.

Если у вас есть такое желание, вы можете изменить вид переменных `PS1` и `PS2`. При этом можно использовать как любые символы, вводимые с клавиатуры, так и некоторое число специальных символов, которые при формировании строки приглашения декодируются (приводим только некоторые из них, для примера; полный список см. в `man`-странице по утилите `bash`):

- `\a`      Звуковой сигнал (ASCII-код 07)
- `\d`      Дата в формате "День, месяц, число", например, Срд, Окт, 17.
- `\h`      Имя хоста (hostname) до первой точки
- `\H`      Полное имя хоста
- `\t`      Текущее время в 24-часовом формате: HH:MM:SS (часы:минуты:секунды)
- `\T`      Текущее время в 12-часовом формате: HH:MM:SS
- `\@`      Текущее время в 12-часовом формате am/pm
- `\u`      Имя пользователя, запустившего оболочку
- `\w`      Полное имя текущего рабочего каталога (начиная с корня)
- `\W`      Текущий рабочий каталог (без указания пути)

- `\$` Символ `#`, если оболочка запущена суперпользователем, и символ `$`, если оболочка запущена обычным пользователем.
- `\nnn` Символ, имеющий восьмеричный код `nnn`
- `\n` Новая строка (перевод строки)
- `\s` Имя оболочки
- `\#` Текущий номер команды
- `\\` Обратный слэш (a backslash)
- `\` Начало последовательности не печатаемых символов (этот символ может быть использован для того, чтобы включить в текст подсказки последовательность управляющих символов терминала)
- `\]` Конец последовательности не печатаемых символов
- `!\` Порядковый номер данной команды в истории команд

Текущий номер команды (порядковый номер выполняемой команды в рамках текущей сессии) может отличаться от номера данной команды в списке истории команд, поскольку последний включает в себя команды, которые были сохранены в файле истории команд.

После того, как значение переменной, определяющей подсказку, прочитано оболочкой, в нем могут быть произведены подстановки в соответствии с правилами расширения параметров, подстановок в именах команд и арифметических выражениях, а также разбиения слов (word splitting).

Например, после выполнения команды (поскольку в строке имеется пробел, кавычки обязательны)

```
root@master:~# PS1="[ \u@\h\W] \$"
[root@master~]$
```

## Файл `~/.bash_logout`

Судя по имени файла `.bash_logout` понятно, что он используется интерпретатором `bash` при его завершении. Вероятно, вы уже заметили, что по завершении работы, когда вы уже вышли из системы на экране все равно сохранились последние строки, раскрывающие ваши последние действия. С точки зрения безопасности это плохо. Фактически, любой оказавшийся у экрана может увидеть ваши последние действия и возможно использовать эту информацию во вред вам или вашей системе. Каким образом заставить систему очищать экран при выходе пользователя?

Для интерпретатора `bash` необходимо лишь добавить команду `clear` в файл, обрабатываемый интерпретатором по своему завершению - `.bash_logout`.

```
# cat ~/.bash_logout
clear
#
```

## Глава 14. Русификация консоли и консольных приложений (локализация)

Может возникнуть вопрос: а надо ли подробно разбирать вопрос русификации, не лучше ли просто сразу установить русифицированный дистрибутив? Тем более, что в последних версиях дистрибутивов Red Hat Cyrillic Edition, ASPLinux и AltLinux русификация выполнена на вполне приемлемом уровне. Однако, даже в случае установки русифицированного дистрибутива вы имеете шанс столкнуться с проблемой русификации на последующих этапах. Может получиться так, что новая версия нерусифицированного дистрибутива появится раньше, чем соответствующий русифицированный вариант, и вы захотите его установить. Не всегда хочется дожидаться пока выйдет русская версия. Русификация может нарушиться при обновлении отдельных программных пакетов. Таким образом, задача русификации может встать перед любым пользователем ОС Linux.

Когда я начинал работать с ОС Linux, самым лучшим материалом по русификации был "The Linux Cyrillic HOWTO" Александра Беликова (Версия 4.2 b2, Декабрь 11, 1998) в переводе Е.М. Балдина. Кроме этого HOWTO были доступны только материалы со странички Леонида Кантера. Однако оба этих источника уже в то время существенно устарели, так как в Red Hat версии 6 изменились даже команды выбора шрифта. Переводчик "The Linux Cyrillic HOWTO" Е. Балдин в настоящее время создает свой вариант HOWTO по кириллизации. Очень полезен также RU.LINUX.FAQ. Настоящая глава во многом следует этим двум основным источникам.

Начать надо с двух замечаний. Во-первых, поскольку способы вывода информации на экран в графическом и текстовом режимах принципиально различны, придется отдельно рассмотреть вопрос о русификации текстового и графического режима. Во-вторых, в системе Linux существуют два конкурирующих пакета управления консольными шрифтами и клавиатурой:

kbd (<ftp://ftp.win.tue.nl/pub/linux/utils/kbd/> или <ftp://ftp.kernel.org/pub/linux/utils/kbd/>)

и

consoletools (<http://lcr.sourceforge.net>).

В разных дистрибутивах применяются или один, или другой. Например, в Suse Linux и в Slackware для русификации консоли применяется пакет kbd. В Red Hat 6.x применяется уже другой пакет - consoletools.

### Таблицы кодировки символов

В человеческом мире информация представляется последовательностями символов. Каждый символ имеет каноническое изображение, которое позволяет однозначно идентифицировать данный символ. Шрифты задают разные варианты начертания символов.

В вычислительных машинах для представления информации используются цепочки байтов. Поэтому для перевода информации из машинного представления в человеческий необходимы таблицы кодировки символов — таблицы соответствия между символами определенного языка и кодами символов.

Самой известной таблицей кодировки является код ASCII (Американский стандартный код для обмена информацией), который был разработан для передачи текстов по телеграфу задолго до появления компьютеров. Этот код является 7 битовым, т. е. для кодирования символов английского языка, служебных и управляющих символов используются только 128 7-битовых комбинаций. При этом первые 32 комбинации (кода) служат для кодирования управляющих сигналов (начало текста, конец строки, перевод каретки, звонок, конец текста и т. д.).

При разработке первых компьютеров фирмы IBM этот код был использован для представления символов в компьютере. Поскольку в исходном коде ASCII было всего 128 символов, для их кодирования хватило тех однобайтовых кодов, у которых 8-й бит равен 0. Во второй половине кодовой таблицы (значения байта с 8-м битом равным 1) фирма IBM разместила символы псевдографики, математические знаки и некоторые символы из языков, отличных от английского (немецкие умляути, французские диакритические знаки, символы греческого алфавита и т.п.). Эту кодовую таблицу стали называть кодировкой IBM.

Когда IBM-совместимые персональные компьютеры стали использовать в других странах, потребовалось обеспечить обработку информации на языках, отличных от английского. Для того, чтобы полноценно поддерживать другие языки, фирма IBM ввела в употребление несколько кодовых таблиц, ориентированных на конкретные страны. Так для скандинавских стран была предложена таблица 865 (Nordic), для арабских стран — таблица 864 (Arabic), для Израиля — таблица 862 (Israel) и так далее. В этих таблицах часть кодов из второй половины кодовой таблицы использовалась для представления символов национальных алфавитов (за счет исключения некоторых символов псевдографики). Для представления символов кириллицы была введена кодировка IBM-866.

Однако с русским языком ситуация развивалась особым образом. Очевидно, что замену символов во второй половине кодовой таблицы можно произвести разными способами. В других европейских странах сумели найти единое решение, а для русского языка появилось несколько разных таблиц кодировки символов кириллицы: IBM-866, CP-1251, KOI8-R, ISO-8859-5. Все они одинаково изображают символы первой половины таблицы (от 0 до 127) и различаются представлением символов русского алфавита и псевдографики во второй половине.

Одна из самых известных кодовых таблиц для кириллицы получила название альтернативной (по отношению к кодировке IBM-866, наверное). Она была разработана фирмой Microsoft для MS-DOS. При ее разработке постарались сделать так, чтобы результирующая таблица была настолько это возможно совместима с кодировкой IBM. Поэтому альтернативная кодировка — это кодировка IBM, в которой все специфические европейские символы в верхней половине были заменены на кириллицу, оставляя псевдографические символы нетронутыми. Следовательно, это не портило вид программ, использующих для работы текстовые окна, что было очень существенным фактором для работы в среде MS-DOS, основой которой был именно текстовый режим.

Кодировка KOI-8 была разработана изначально с ориентировкой на UNIX. Так как UNIX в своей основе сетевая ОС, то основной идеей при создании KOI-8 была идея об обеспечении перемещения кириллической информации по сети. Но для передачи-то использовался 7-битный стандарт ASCII. Разработчики поместили кириллические символы в верхней части таблицы таким образом, что позиции кириллических символов соответствуют их фонетическим аналогам в английском алфавите в нижней части таблицы. Это означает, что, если в тексте, написанном в KOI-8, мы убираем восьмой бит каждого символа, то мы все еще имеем "читабельный" текст, хотя он и написан английскими символами! Не удивительно, что KOI8-R быстро стал фактически стандартом для кириллицы в Интернет, что и нашло отражение в RFC 1489 ("Registration of a Cyrillic Character Set"). Автором этого документа является Андрей А. Чернов, который проделал огромный объем работы, чтобы превратить KOI-8 в стандарт Интернет.

Международная организация по стандартизации (ISO) внесла свою лепту в создание различных кодировок кириллицы, когда ввела семейство стандартов, известных как ISO 8859-X. Это семейство есть совокупность 8-битных кодировок, где младшая половина каждой кодировки (символы с кодами 0—127) соответствует ASCII, а старшая половина определяет символы для различных языков. Например:

8859-0 — новый европейский стандарт (так называемый Latin 0);

8859-1 — Европа, Латинская Америка (также известный как Latin 1);

8859-2 — Восточная Европа;

8859-5 — кириллица;

8859-8 — идиш.

Фирма Microsoft еще больше запутала ситуацию с кодировками для русского языка, когда при разработке Windows ввела кодировку CP-1251.

Таблицы кодировок, содержащие 256 символов, стали называть расширенными кодами ASCII (потому что в основе любой из них лежит 128-символьный код ASCII), кодовыми страницами или английским термином *character set* (который часто сокращают до *charset*).

Но в мире есть языки, такие как китайский или японский, для которых 256 символов в принципе недостаточно. Кроме того, всегда существует проблема вывода или сохранения в одном файле одновременно текстов на разных языках (например, при цитировании). Поэтому была разработана универсальная кодовая таблица UNICODE, содержащая символы, применяемые в языках всех народов мира, а также различные служебные и вспомогательные символы (знаки препинания, математические и технические символы, стрелки, диакритические знаки и т. д.). Очевидно, что одного байта недостаточно для кодирования такого большого множества символов. Поэтому в UNICODE используются 16-битовые (2-байтовые) коды, что позволяет представить 65 536 символов. К настоящему времени задействовано около 49 000 кодов (последнее значительное изменение — введение символа валюты EURO в сентябре 1998 г.). Для совместимости с предыдущими кодировками первые 128 кодов совпадают со стандартом ASCII.

В стандарте UNICODE кроме определенного двоичного кода (эти коды принято обозначать буквой U, после которой следуют знак + и собственно код в шестнадцатеричном представлении) каждому символу присвоено определенное имя.

Еще одним компонентом стандарта UNICODE являются алгоритмы для взаимно-однозначного преобразования кодов UNICODE в последовательности байтов переменной длины. Необходимость таких алгоритмов обусловлена тем, что не все приложения умеют работать с UNICODE. Некоторые приложения понимают только 7-битовые ASCII-коды, другие приложения — 8-битовые (расширенные) ASCII-коды. Для представления символов, не поместившихся, соответственно, в 128-символьный или 256-символьный набор, такие приложения используют цепочки байтов переменной длины. Алгоритм UTF-7 служит для обратимого преобразования кодов

UNICODE в цепочки 7-битовых ASCII-кодов, а UTF-8 — для обратимого преобразования кодов UNICODE в цепочки из расширенных 8-битовых ASCII-кодов.

Отметим, что и ASCII, и UNICODE, и другие стандарты кодировки символов не определяют изображения символов, а только состав набора символов и способ его представления в компьютере. Кроме того (что, может быть, не сразу очевидно) они еще задают порядок перечисления символов в наборе, который очень важен, так как он влияет самым существенным образом на алгоритмы сортировки. Именно таблицу соответствия символов из какого-то определенного набора (скажем, символов, применяемых для представления информации на английском языке, или на разных языках, как в случае с UNICODE) и обозначают термином таблица кодировки символов или charset. Каждая стандартная кодировка имеет имя, например, KOI8-R, ISO\_8859-1, ASCII. К сожалению, стандарта на имена кодировок не существует.

## Локализация

Только организовать ввод и вывод символов национального языка еще недостаточно для того, чтобы можно было считать решенным проблему применения компьютеров в той или иной стране. В разных странах в силу сложившихся традиций имеются различия не только в используемом алфавите, но и в способах представления некоторых конкретных данных. Это касается, например, символов, используемых для обозначения валюты, форматов представления даты и времени, обычаев записи (чтения) текстов слева направо или справа налево и т.д. При создании ПО, рассчитанного на применение в разных странах, приходится учитывать такие местные особенности. Кроме того, большие трудности вызывают такие вопросы как проверка орфографии на конкретном языке, автоматическая расстановка переносов при вводе текста или перевод на данный язык всех меню и служебных сообщений в программном обеспечении.

Способ проектирования ПО (включая ОС), при котором возможность многоязыковой поддержки закладывается с самого начала, принято называть интернационализацией (кстати, загадочное `i18n` - это просто сокращение для слова `internationalization`: `i` - потом еще 18 букв - `n`, аналогично, `l10n` = `localization`). При интернационализации программного обеспечения КОД не зависит от национальных особенностей. Все языково-зависимые данные сосредотачиваются в особых "объектах локализации", которые разбиты на функциональные группы: категории локализации. При таком подходе локализация — это процесс настройки программной системы на особенности конкретной страны.

В стандарте POSIX (Portable Operating System) были определены средства локализации, которые состоят из следующих компонент:

- набор библиотечных (libc) вызовов (locale API): `setlocale()`, `isalpha()`, `toupper()`, и т.д;
- исходные тексты описания средств локализации, в том числе файл(ы) описания кодировки (Character Set Definition File);
- наборы данных для локализации, которые в Linux размещаются в каталогах `/usr/share/i18n/*` и `/usr/share/locale/*` ;
- утилита для получения информации о средствах локализации: `locale`;
- утилита для изготовления (компиляции) объектов локализации: `localedef`;
- переменные окружения, для управления средствами локализации: `LANG`, `LC_ALL`, `LC_CTYPE`, `LC_TIME`, `LC_COLLATE`, `LC_NUMERIC` и `LC_MONETARY` (они соответствуют категориям локализации).

Ниже кратко перечислено, на что именно влияет та или иная из этих переменных (или категорий локализации):

Категория	Описание
LC_CTYPE	Определяет правила классификации и преобразования одиночных символов. Позволяет правильно определять вид символа: цифра, буква, значок, заглавная буква или прописная и т.д. Другими словами, включает правильную работу системных вызовов <code>isalnum()</code> , <code>isalpha()</code> , <code>isctrl()</code> , <code>isdigit()</code> , ... и т.п. для местного алфавита. Вдобавок, включает правильный перевод строчных — прописных букв: <code>toupper()</code> и <code>tolower()</code>
LC_COLLATE	Определяет правила сравнения и преобразования строк. Позволяет определять лексикографический порядок символов (порядок сортировки) в местном алфавите. Включает правильную работу <code>strcoll()</code> и <code>strxfrm()</code> . Оказывает непосредственное влияние на работу утилит типа <code>sort</code> и т.д.
LC_TIME	Определяет правила национального представления времени и даты. Задаёт именованное

дней недели, месяцев и т.п. а также задает способ написания даты и времени (12/24). Непосредственно влияет на `strftime()`, а через нее на утилиты `date` и т.д.

LC_NUMERIC	Определяет правила национального представления чисел с плавающей точкой. Влияет на <code>strtod()</code> и форматы <code>%f</code> и <code>%g</code> <code>printf()</code> , <code>scanf()</code>
LC_MONETARY	Определяет правила национального представления денежных величин

Особая переменная `LC_ALL` служит для обращения одновременно ко всем категориям.

Заметим, что включение средств локализации изменяет также некоторые пути поиска, в частности, пути поиска к файлам помощи (man-страницам), так что вначале команда `man` ищет переведенные man-страницы и только при их отсутствии выдает информацию на английском.

## Настройка системных средств локализации

### Проверка наличия средств локализации

Современные дистрибутивы Linux (а тем более русифицированные) по умолчанию содержат системные средства локализации, перечисленные в предыдущем разделе.

Чтобы убедиться в этом, проверьте, что у вас имеются каталоги `/usr/share/locale/*` и `/usr/share/i18n/*`.

```
# ls -ld /usr/share/locale
drwxr-xr-x 141 root root 4096 2008-04-14 10:28 /usr/share/locale/
# ls -ld /usr/share/i18n
drwxr-xr-x 4 root root 35 2008-04-20 11:05 /usr/share/i18n/
#
```

Кроме того, дайте команду

```
# locale -a | grep ru
ru_RU
ru_RU.cp1251
ru_RU.koi8r
ru_RU.utf8
ru_UA
ru_UA.utf8
#
```

Для определения текущей локали дайте команду:

```
# locale
```

В ответ вы должны получить значения, присвоенные переменным окружения для управления средствами локализации. Я, например, увидел следующее:

```
# locale
LANG=en_US
LC_CTYPE="en_US"
LC_NUMERIC="en_US"
LC_TIME="en_US"
LC_COLLATE=C
LC_MONETARY="en_US"
LC_MESSAGES="en_US"
LC_PAPER="en_US"
LC_NAME="en_US"
LC_ADDRESS="en_US"
LC_TELEPHONE="en_US"
LC_MEASUREMENT="en_US"
LC_IDENTIFICATION="en_US"
LC_ALL=
```

Можно также дать команду `locale` с параметром, совпадающим с именем одной из переменных окружения, например:

```
# locale LC_TIME
```



(но я не берусь объяснить все то, что вы увидите, хотя много и будет очевидным).

Если результат этих проверок отрицательный (что маловероятно), то установите пакет локализации для русского языка. Найти его можно в коллекции средств локализации по адресу <http://www.ping.be/linux/locales/> или на [www.kiarchive.ru](http://www.kiarchive.ru).

После этого надо сказать Linux, какой язык вы хотите использовать, для чего необходимо задать значения переменных окружения для управления средствами локализации: LC\_CTYPE, LC\_TIME, LC\_COLLATE, LC\_NUMERIC, и LC\_MONETARY. Вообще говоря, достаточно задать всего одну переменную LC\_ALL, задание которой одновременно определяет значения всех перечисленных выше переменные.

Есть еще две переменных окружения, которые имеют отношение к локализации: LANG и LINGUAS. Они действуют примерно так же, как и LC\_ALL, в том смысле, что они определяют значения по умолчанию всех других переменных локализации, но, в отличие от LC\_ALL, они не переопределяют значений других переменных LC\_\*, для которых значения заданы отдельно.

Переменная LINGUAS является GNU-расширением переменной LANG. Не все программы знают о существовании этой переменной (хотя не знают очень немногие, и таких все меньше), но эта переменная обладает тем преимуществом, что она позволяет задать несколько вариантов локализации, в порядке предпочтения. В большинстве случаев достаточно задать только одну переменную локализации - LANG. Если вы хотите использовать несколько вариантов локализации, то надо задать LANG и LINGUAS. Остальные переменные необходимо задавать только в особых случаях, которые мы здесь не рассматриваем.

## Формат задания значений переменных локализации

В качестве значений переменных локализации используются строки формата

```
ll[_CC[.EEEE]][@dddd]
```

где:

- ll - это двухбуквенный код языка в соответствии со стандартом ISO для названий языков (ISO 639), записываемый в нижнем регистре (строчными латинскими буквами);
- CC - это двухбуквенный код страны в соответствии со стандартом ISO для названий стран (ISO 3166), записываемый в верхнем регистре;
- EEEE - это имя (название) таблицы кодировки, записываемое в верхнем регистре;
- dddd - название диалекта языка, который задается в том случае, если названия кодировки недостаточно для однозначного определения варианта локализации.

Для переменной LINGUAS задаваемые значения разделяются двоеточиями. Пример строки задания значений переменной вы уже видели: *ru\_RU.KOI8-R*.

Как уже говорилось, стандарта на названия кодовых таблиц нет, тем более на название диалектов. То, что в шаблоне дано только 4 символа, ничего не означает - символов может быть и больше, и меньше. Квадратные скобки, как всегда, выделяют необязательные элементы. Обязательным в этом шаблоне является только название языка, но в некоторых источниках рекомендуется для русского языка указывать все три части. Напомню еще, что аббревиатура SU теперь означает Судан, а не нашу страну.

## Включение средств локализации

Включение системных средств локализации в Red Hat Linux (а, следовательно, и в других дистрибутивах, основанных на Red Hat) и Slackware осуществляется из файла /etc/profile.d/lang.sh.

Как известно, при старте любого shell-а сначала выполняется /etc/profile. В файле /etc/profile прописаны команды, благодаря которым на исполнение вызываются также все файлы /etc/profile.d/\*.sh

Ниже приведены значения переменных локализации в файлах lang.sh. В файлах документации приведены краткие рекомендации относительно того, как задавать значения переменных, в частности, предлагается для LANG указать в качестве значения просто двухбуквенный код ISO для желаемого языка; для переменной LINGUAS - список кодов языков, разделенных двоеточием, а переменной LC\_ALL присвоить имя в соответствии с приведенным выше описанием формата для переменных локализации. Таким образом, в файле lang.sh рекомендуется прописать три строки следующего вида:

```
export LANG="ru_RU.KOI8-R"
export LINGUAS="ru_RU:en"
```

```
export LC_ALL="ru_RU.KOI8-R"
```

```
# cat /etc/profile.d/lang.sh
#!/bin/sh
export LANG=ru_RU.KOI8-R
export LINGUAS=ru_RU:en
export LC_ALL=ru_RU.KOI8-R
# End of /etc/profile.d/lang.sh
#
```

Заметим, что в RU.LINUX.FAQ написано следующее: "Хотя в принципе допустимо задавать короткое именование, вроде LANG=ru\_RU или даже LANG=ru, лучше использовать полное имя: LANG=ru\_RU.KOI8-R. Совершенно недопустимо задавать LANG=ru\_SU, такой страны больше нет :-)".

Если все, что было перечислено, сделано, можно считать, что локализация включена.

Правда, это верно только для случая, когда вы имеете права суперпользователя root. Но даже если вы простой пользователь Linux-системы и не можете редактировать системные файлы, то вы все же можете включить локализацию для себя, но несколько иным способом. А именно, поместите в свой файл \$HOME/.profile (или в любой файл, который выполняется в процессе логирования пользователя: \$HOME/.Xclients, \$HOME/.xinitrc или другой) следующие строки:

```
export LANG=ru_RU.KOI8-R
export LINGUAS=ru_RU:en
export LC_ALL="ru_RU.KOI8-R"
```

Вот и все! Теперь выходим из системы — exit. И вновь выполняем login.

Проверяем:

```
# locale
LANG=ru_RU.KOI8-R
LC_CTYPE="ru_RU.KOI8-R"
LC_NUMERIC="ru_RU.KOI8-R"
LC_TIME="ru_RU.KOI8-R"
LC_COLLATE="ru_RU.KOI8-R"
LC_MONETARY="ru_RU.KOI8-R"
LC_MESSAGES="ru_RU.KOI8-R"
LC_PAPER="ru_RU.KOI8-R"
LC_NAME="ru_RU.KOI8-R"
LC_ADDRESS="ru_RU.KOI8-R"
LC_TELEPHONE="ru_RU.KOI8-R"
LC_MEASUREMENT="ru_RU.KOI8-R"
LC_IDENTIFICATION="ru_RU.KOI8-R"
LC_ALL=ru_RU.KOI8-R
#
# cal -m
      Сентябрь 2008
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
#
```

## Локализация консоли

Теперь вы представляете, что для русификации консоли (а консоль - это, грубо говоря, совокупность клавиатуры и дисплея) требуется загрузить в драйвер консоли три таблицы:

- таблицу раскладки клавиатуры;
- таблицу экранного шрифта, в которой хранятся изображения символов;

- таблицу перекодировки символов.

Ранее мы уже узнали что существуют два пакета для управления клавиатурой – kbd и consoletools. Давайте их рассмотрим поочередно.

## consoletools

Таблицы раскладки клавиатуры находятся в каталоге /usr/lib/kbd/keymaps/i386/qwerty. Выбор конкретного файла раскладки задается файлом /etc/sysconfig/keyboard. Этот файл можно отрегулировать вручную, а можно с помощью программы kbdconfig. В последнем случае нужная таблица с раскладкой клавиатуры загружается в память. При ручном изменении файла /etc/sysconfig/keyboard перезагрузка таблицы произойдет только после перезапуска компьютера или после выполнения команды (в примере загружается раскладка из файла ru-win.map):

```
# loadkeys /usr/lib/kbd/keymaps/i386/qwerty/ru-win.map
```

Второй шаг русификации состоит в задании и загрузке таблицы экранного шрифта (SFM) в драйвер дисплея. Эти таблицы хранятся в виде файлов в каталоге /usr/lib/kbd/consolefonts. Загрузка шрифтов осуществляется немного по-разному в пакетах kbd и consoletools (соответственно, в версиях 5.2 и 6.x Red Hat Linux). В версии 5.2 загрузка шрифта осуществлялась с помощью команды setfont. Например, чтобы загрузить кодовую страницу из файла Cyr\_a8x16, нужно дать команду

```
# setfont /usr/lib/kbd/consolefonts/Cyr_a8x16
```

В версии 6.0 и последующих надо использовать команду consolechars с опцией -f:

```
# consolechars -f /usr/lib/kbd/consolefonts/Cyr_a8x16
```

(Отметим, что команда setfont тоже работает, только выдаст предупреждение о том, что надо пользоваться командой consolechars.)

Файлы шрифтов являются бинарными файлами размером 256\*H байт, содержащими битовые образы для каждого из 256 символов, по одному байту на каждую линию образа и по H байт на символ (0 < H ≤ 32). В этом случае о размере шрифта по вертикали можно судить по длине файла. В качестве файлов шрифтов могут использоваться файлы .psf; они имеют тот же самый формат и, кроме того, заголовок размером 4 байта. Некоторые файлы шрифтов содержат сразу три шрифта разного размера (например, 8x8, 8x14, 8x16), тогда в команде consolechars надо добавить опцию -H, например: -H 16, для выбора одного из размеров. Поскольку ядро Linux не поддерживает пока переключение режимов работы экрана, consolechars (как и setfont) не может изменить текущий режим EGA/VGA. Таким образом, пользователь полностью ответствен за выбор шрифта, соответствующего текущему режиму экрана.

Соответствие между символами кода ASCII и образами (или изображениями символов) из файла шрифта можно изменить, используя таблицу перекодировки (ACM - Application Charset Map). Если эту таблицу не загрузить, то, например, в программе Midnight Commander вы можете вместо красивых рамок увидеть столбцы и строки из непонятных символов. Некоторые файлы шрифтов включают таблицу перекодировки шрифта, и тогда consolechars загрузит эту таблицу. По умолчанию файлы шрифтов находятся в каталоге /usr/lib/kbd/consolefonts, а таблицы перекодировки - в каталоге /usr/lib/kbd/consoletrans.

Таблицу перекодировки в 6-ой версии Red Hat (т. е. в пакете consoletools) можно загрузить отдельной командой consolechars с опцией -m file:

```
# consolechars -m /usr/lib/kbd/consoletrans/koi2alt
```

Если таблица перекодировки не включена в файл шрифта и не указана в опции -m, то используется "тривиальная" таблица.

В версии 5.2 для загрузки таблицы перекодировки используется команда mapscrn:

```
# mapscrn /usr/lib/kbd/consoletrans/koi2alt.
```

В этом случае драйвер консоли должен быть дополнительно переведен в режим перекодировки, задаваемый таблицей, путем вывода на консоль специальной escape-последовательности. Эта последовательность есть <esc>(K для набора символов G0 (G0 character set) и <esc>J для набора символов G1 (G1 character set). Заметим, что активизировать эту таблицу необходимо в каждой консоли. При этом команда loadkeys действует одновременно во всех виртуальных консолях, а вот команда mapscrn действует только в той виртуальной консоли, в которой выполнена команда echo -ne '\033(K'.

**Внимание!** Esc(K требуется, когда загружается альтернативная кодировка и активизируется таблица перекодировки псевдографики командой mapscrn koi2alt. Если шрифт koi-8, то никаких Esc(K не надо.

**Внимание!** Все это не действует из-под Midnight Commander!

Существует еще таблица перекодировки для символов UNICODE. Некоторые файлы шрифтов включают эту таблицу, и она будет загружена командой `consolechars`, если только не задана опция `--force-no-sfm`. Отдельно загрузить таблицу перекодировки символов Unicode можно командой `consolechars` с опцией `-u` (см. руководство `man`).

Итак, для того, чтобы русифицировать консоль, нужно выполнить следующую последовательность команд:

– для версии 5.2 Red Hat:

```
loadkeys /usr/lib/kbd/keytables/i386/qwerty/ru.map
setfont /usr/lib/kbd/consolefonts/Cyr_a8x16
mapscrn /usr/lib/kbd/consoletrans/koi2alt
echo -ne '\033(K'
```

– для версии 6.0 Red Hat (и последующих):

```
loadkeys /usr/lib/kbd/keytables/i386/qwerty/ru.map
consolechars -f /usr/lib/kbd/consolefonts/Cyr_a8x16
consolechars -m /usr/lib/kbd/consoletrans/koi2alt
```

Но выполнять эту последовательность команд после каждого перезапуска компьютера, да еще в каждой виртуальной консоли, слишком обременительно. Поэтому рассмотрим вкратце, как русификация выполняется в дистрибутиве Black Cat Linux.

### Как это сделано в дистрибутиве Black Cat

Во-первых, в файле `/etc/sysconfig/i18n` вводится новая переменная: в версии 5.2 это переменная `SCRNMAP`, а в версии 6.02 - `SYSFONTACM`. Этой переменной по умолчанию присваивается значение `"koi2alt"`. Вот стандартный файл `i18n` из Black Cat Linux версии 6.02:

```
LANG=ru
LINGUAS=ru
LC_ALL=ru_RU.KOI8-R
SYSFONT=RUSCII_8x16
SYSFONTACM=koi2alt
```

Вызов файла `i18n` для задания значений переменных осуществляется из скрипта `/sbin/setsysfont`, из которого вызываются также команды `setfont` и `mapscrn` (в версии 5.2) или `consolechars` (в версии 6.0). Вот этот скрипт из Black Cat Linux версии 5.2:

```
#!/bin/sh
if [ -f /etc/sysconfig/i18n ]; then
    . /etc/sysconfig/i18n
fi
if [ -x /usr/bin/setfont ]; then
    if [ -n "$SYSFONT" ]; then
        /usr/bin/setfont $SYSFONT
    fi
fi
if [ -x /usr/bin/mapscrn ]; then
    if [ -n "$SCRNMAP" ]; then
        /usr/bin/mapscrn $SCRNMAP
    fi
fi
else
    echo "can't set font"
    exit 1
fi
```

Как видно, при вызова скрипта `/sbin/setsysfont` выполняются команды `"setfont Cyr_a8x16"` и `"mapscrn koi2alt"`. После этого, для включения в ядре кодовой таблицы пользователя, необходимо выдать на каждую виртуальную консоль последовательность `"\033(K"`. Это реализовано путем добавления этой последовательности к файлу `/etc/issue`, который генерируется при загрузке системы скриптом `/etc/rc.d/rc.local` и вызывается на исполнение при логировании каждого пользователя. Вот пример скрипта `/etc/rc.d/rc.local` из версии 5.2:

```
#!/bin/sh
```

```
# This script will be executed after all the other init scripts.
# You can put your own initialization stuff in here if you don't
# want to do the full Sys V style init stuff.
if [ -f /etc/blackcat-release ]; then
    R=$(cat /etc/blackcat-release)
elif [ -f /etc/redhat-release ]; then
    R=$(cat /etc/redhat-release)
else
    R="release 3.0.3"
fi
arch=$(uname -m)
a="a"
case "$arch" in
_a*) a="an";;
_i*) a="an";;
esac
# This will overwrite /etc/issue at every boot. So, make any changes you
# want to make to /etc/issue here or you will lose them when you reboot.
. /etc/sysconfig/i18n
echo "" > /etc/issue.net
echo "Black Cat Linux $R" >> /etc/issue.net
echo "Kernel $(uname -r) on $a $(uname -m)" >> /etc/issue.net
if [ -n "$SCRNMAP" ]; then
    echo -ne "\033(K" > /etc/issue
else
    echo "" > /etc/issue
fi
if [ -f /usr/bin/linux_logo ]; then
    /usr/bin/linux_logo -n -o 2 >> /etc/issue
    echo "" >> /etc/issue
fi
cat /etc/issue.net >> /etc/issue
echo "" >> /etc/issue
```

В версии 6.02 все происходит примерно так же. Просмотрите упомянутые выше файлы, и вы убедитесь в этом сами.

## kbd

Теперь же рассмотрим пакет kbd. Пакет consoletools вырос из kbd и сейчас мы это наглядно увидим.

В состав набора kbd входят следующие программы:

- loadkeys – загружает раскладку клавиатуры;
- setfont – устанавливает экранные шрифты;
- mapscrn – осуществляет перекодировку.

Пример русификации консоли:

```
loadkeys ru1
setfont Cyr_a8x16
mapscrn koi2alt
echo -ne "\033(K"
```

Для русификации всех терминалов, вместо последней строки можно использовать:

```
for I in 1 2 3 4 5 6
do
    echo -ne '\033(K' > /dev/tty$I
done
```

## Как это сделано в Slackware Linux

Редактируем (создаем) файл /etc/rc.d/rc.font. Вот что мы должны получить в результате:

```
# cat /etc/rc.d/rc.font
#!/bin/bash
setfont -v Cyr_a8x16
#
```

Редактируем (создаем) файл /etc/rc.d/rc.keymap. Вот так:

```
# cat /etc/rc.d/rc.keymap
#!/bin/bash
loadkeys ru1
mapscrn koi2alt
for I in 1 2 3 4 5 6
do
    echo -ne '\033(K' > /dev/tty$I
done
#
```

Делаем эти файлы исполняемыми для суперпользователя:

```
# chmod 700 /etc/rc.d/{rc.font,rc.keymap}
```

Теперь, после перезагрузки системы эти файлы будут применяться автоматически. Однако нам нужно увидеть результат уже сейчас! Запускаем эти скрипты:

```
# /etc/rc.d/rc.font
Loading 256-char 8x16 font from file
/usr/share/kbd/consolefonts/Cyr_a8x16.psfu.gz
Загружаю Unicode-таблицу...
#
# /etc/rc.d/rc.keymap
Loading /usr/share/kbd/keymaps/i386/qwerty/ru1.map.gz
#
```

Загрузка необходимых шрифтов и кодировок прошла успешно. Теперь можно в консоли переключить язык ввода сочетанием клавиш RSHIFT+RALT.

## Переключение кодировок

Теперь поговорим о том, как "на лету" изменить кодировку символов. Необходимость в этом возникает в тех случаях, когда просматриваешь какой-то файл и вместо читаемого текста видишь непонятную абракадабру. В таких случаях хочется превратить ее в нормальный текст нажатием пары горячих клавиш. Можно, конечно, для каждого из необходимых шрифтов создать специальную команду, записав в небольшой файл приведенные выше команды, естественно соответствующим образом модифицированные. Однако ясно, что это не совсем удобно. В этом отношении хочется работать так же комфортно, как в программе FAR E. Рошалья, где в том случае, когда просматриваешь файл по клавише <F3> или редактируешь его по <F4>, достаточно набрать <Shift>+<F8> и получаешь возможность перекодировать выводимый на экран текст в любую из нескольких кодировок. Можно считать, что это уже роскошь, но что поделаешь, все мы быстро привыкаем к хорошему, и отказываться уже трудно.

2 июля 2000 г на сайте linux.ru.net появилась сообщение о заплате (патче) к Midnight Commander версии 4.5.50, которая позволяет производить перекодировку как в FAR, по нажатию клавиш <Ctrl>+<T>. Тогда это можно было делать только во встроенной программе просмотра. На сегодня автор патча В. Студенников доработал его, так что перекодировка стала возможна и во встроенном редакторе (см. <http://www.linux.zp.ua:8100/mc/> или <http://www.sama.ru/~despaire/mc/my-patches.html>).

Это что касается перекодировки «на лету». Как видите, такая перекодировка является функцией отдельных программ просмотра файлов. Если перекодировка «на лету» недоступна, то можно перекодировать сам текстовый файл. Для преобразования потока символов из одной кодовой страницы (charset) в другую, в стандарте POSIX существуют утилита iconv и функция iconv().

## ***Отображение русских букв в файловых системах *vfat* и *ntfs****

В FAT русские буквы в именах файлов хранятся в кодировке cp866, а в NTFS в UTF-8. При монтировании этих файловых систем необходимо использовать специальные параметры монтирования, позволяющие «на лету» менять кодировки символов:

Для *vfat*, *iso9660* и *udf*:

- `iocharset=koi8-r,codepage=866`

Для *ntfs*:

- `nls=koi8-r` или `iocharset=koi8-r`

## Глава 15. Сети в Linux

Хотя мы рассматриваем в данном курсе только работу на локальном компьютере, компьютер этот не изолирован от других. Компьютер на вашем рабочем месте подключен к локальной сети.

UNIX известен своей сетевой производительностью. В самом деле, сетевой протокол TCP/IP был впервые реализован в BSD. Многие другие операционные системы стали использовать сетевой стек BSD из-за его высокой производительности и свободной лицензии.

Системный администратор должен понимать работу в сети. Большинство администраторов обычно знакомы с основами, но таких, которые понимают взаимодействие сетевых средств, не так уж и много. Знания об адресации IP, понимание работы сетевой маски и симбиотической связи между IP и TCP - вот что отличает новичка от профессионала. Здесь будут рассмотрены некоторые из этих тем.

Если вы знаете о том, что делает сеть /31 по большей части бесполезной, можете пропустить этот раздел. В противном случае прочтите его. Тестирование пройдет позднее - не на этом курсе, а в реальном мире.

### Немного истории

Идея сетей также стара, как и вообще идея телекоммуникаций. Рассмотрим людей, живших в каменном веке, когда для обмена сообщениями между людьми использовались барабаны. Предположим пещерный человек А хочет пригласить пещерного человека Б поиграть, но тот живет слишком далеко и не может услышать барабан, в который бьет А. Каковы же могут быть действия А? Он может взять барабан побольше или попросить В, живущего на полпути между А и Б, передать сообщение. Позже это стали называть сетями.

Конечно, мы далеко ушли от примитивных занятий и устройств наших предков. В наши дни мы пользуемся компьютерами которые общаются между собой посредством проводов, оптоволоконных кабелей, коротких волн, и т.д., позволяющих легко обмениваться данными. Далее мы с Вами будем обсуждать способы и пути, с помощью которых это можно сделать.

Мы с вами сосредоточимся на сетях TCP/IP, потому что это наиболее популярный набор протоколов в локальных (Local Area Networks, LAN) и глобальных сетях (Wide Area Networks, WAN), типа Internet.

Мы определяем сеть как совокупность компьютеров (хостов, от hosts), которые могут связаться друг с другом, часто полагаясь на услуги ряда выделенных компьютеров, передающих данные между участниками. Хостом может являться не обязательно полноценный компьютер, им вполне может быть X-терминал или интеллектуальный принтер. Маленькие скопления компьютеров также называют сайтами (sites).

Связь невозможна без некоторого языка или кода. В компьютерных сетях эти языки коллективно упоминаются как протоколы. Однако, здесь речь идет не о письменных протоколах, а скорее о высоко формализованном коде поведения, вроде встреч на высшем уровне между главами правительств или брачной песни лебедей. Протоколы (protocols), используемые в компьютерных сетях, это только очень строгие правила для обмена сообщениями между компьютерами.

### Сети TCP/IP

Первым средством связи между unix машинами была программа UUCP (Unix-to-Unix CoPy). Своё название uucp ведёт от имени команды копирования файлов в UNIX — cp, и фактически, является её расширением, позволяя копировать файлы с локальной машины на удалённую и с удалённой на локальную, в первую очередь, посредством модемного соединения (позже появились реализации "uucp over tcp"; они были как в виде встроенной в программу функции, так и в виде отдельного драйвера, перехватывающего обращение к модему и эмулирующего модемный звонок посредством установки tcp-сессии). Достаточно быстро uucp стал использоваться для передачи почты и новостей (телеконференций).

Хотя UUCP, может быть, и разумный выбор для дешевых сетей связи по телефону, существует большое количество ситуаций, в которых техника "сохранил-передал" оказывается слишком не гибкой, например, в локальных сетях (LAN). Они обычно состоят из небольшого числа машин, расположенных в одном здании или даже на одном этаже, которые связаны для создания однородной рабочей среды. Вы хотели бы разбросать файлы между этими хостами или запускать одно приложение на различных машинах. Эти задачи требуют совершенно другого подхода к организации сети. Вместо отправления полных файлов наряду с описанием работы, все данные разбиваются на маленькие пакеты, которые немедленно отправляются нужному хосту, где они повторно собираются. Этот тип сети называется пакетной (packet-switching) сетью. Среди прочего, это позволяет запускать по сети диалоговые приложения. Цена этого, конечно, резкое увеличение сложности программного обеспечения.



Решение, которое приняли Unix-системы и большинство не-Unix сайтов, известно как TCP/IP. В этом разделе мы будем рассматривать его концепции.

## Введение в сети TCP/IP

TCP/IP происходит от проекта, финансируемого американским DARPA (Defense Advanced Research Projects Agency) в 1969. Это была экспериментальная сеть ARPANET, которая была преобразована в эксплуатационную в 1975 после того, как была доказана ее полезность.

В 1983 новый протокол TCP/IP был принят как стандарт и от всех хостов в сети требовалось его использование. Когда ARPANET, наконец, вырос в Internet (ARPANET непосредственно окончил свое существование в 1990г.), использование TCP/IP распространилось и на сети вне Internet.

Для более конкретного рассмотрения TCP/IP повсюду далее мы будем пользоваться как примером некоторым Государственным Университетом (GU, Government University), большинство его отделов используют собственную локальную сеть, а другие используют несколько таких сетей. Они все связаны и подключены к Internet через единственную быстродействующую линию.

Предположим, что Ваша Linux-машина связана с сетью из unix-машин в отделе математики, имя Вашей машины *erdos*. Для доступа к хосту *quark* в отделе физики, введите следующую команду:

```
$ ssh quark.physics
Welcome to the Physics Department at GU
login as:
```

В ответ на приглашение Вы вводите Ваше имя (логин), скажем, *andrey* и пароль. Вам дают shell (оболочку) на *quark*, к которой Вы можете обращаться как будто Вы сидите за системной консолью *quark*. После того, как Вы покинете оболочку, Вы возвращаетесь к приглашению собственной машины. Сейчас Вы использовали только одно из диалоговых приложений, которые предлагает TCP/IP: remote login.

Другое очень важное приложение в unix-сетях TCP/IP: NFS (сетевая файловая система, Network File System). Это другая форма создания прозрачной сети, она позволяет монтировать каталоги с других хостов так, чтобы они рассматривались подобно локальным файловым системам. Например, домашние каталоги всех пользователей могут быть на центральной машине, с которой все другие хосты в локальной сети монтируют требуемые каталоги. В результате пользователи могут войти в сеть с любой машины и находиться в том же самом домашнем каталоге. NFS подробно рассмотрена в курсе «Сетевое администрирование Linux».

Конечно, это не единственные примеры того, что Вы можете делать в сетях TCP/IP. Ваши возможности почти безграничны.

Теперь мы поближе познакомимся с работой TCP/IP. Вы будете нуждаться в этом, чтобы понять, как и почему Вы должны конфигурировать свою машину. Мы начнем с исследования аппаратных средств.

## Ethernet

Среди сетевых аппаратных средств наиболее широко используется повсюду в локальных сетях Ethernet. Он состоит из единственного кабеля с хостами. Скорость передачи зависит от типа Ethernet и составляет 10, 100 или около 1000 мегабит в секунду.

Ethernets бывает трех типов: толстый (thick), тонкий (thin) и витая пара (twisted pair). Тонкий и толстый Ethernet использует коаксиальный кабель, отличающийся толщиной (отсюда и название) и способом подключения машины к кабелю. Тонкий Ethernet использует BNC-коннектор в форме буквы T, в который Вы вставляете кабель и вкручиваете сзади компьютера в гнездо сетевой платы. Толстый Ethernet требует, чтобы Вы проделали маленькую дырочку в кабеле и воткнули коннектор "методом вампира" (vampire tap). Один или больше хостов может быть присоединено к одному такому коннектору. Тонкий и толстый кабель Ethernet может иметь длину не больше 200 и 500 метров соответственно, поэтому они также названы 10base-2 и 10base-5. Витая пара использует кабель, сделанный из медных проводов, но обычно требует дополнительных аппаратных средств (активных концентраторов, active hubs). Он также известен как 10base-T. Здесь "base" является сокращением от "baseband modulation", это значит, что данные непосредственно поданы на кабель без каких-либо модемов и других подобных устройств. Число в начале показывает скорость в мегабитах в секунду, число в конце показывает максимальную длину кабеля. "T" обозначает витую пару. Длина кабеля «витой пары» не должна превышать 100 метров. В случае необходимости соединения хостов, находящихся на большем расстоянии — используйте промежуточное активное сетевое оборудование, например, hub или switch. Витая пара на 100 мегабит в секунду обозначается как "100base-T".

Добавление хоста к толстому Ethernet не слишком сложно, оно даже не разрушает временно сеть. Чтобы

добавить машину к сети с тонким Ethernet Вы должны прервать работу сети по крайней мере на несколько минут потому, что надо разрезать кабель, чтобы вставить T-connector. Витая пара в этом плане очень удобна: в ней используется концентратор (hub), который является точкой обмена данными. Машина может быть подключена к нему или отключена от него независимо от работы других машин в сети.

Использование толстого и тонкого Ethernet сегодня выглядит не привлекательно. Витая пара в плане подключения новых хостов очень удобна, а по цене сейчас тоже стала вполне доступной. По удобству же этому варианту просто нет равных, так что не стоит удивляться тому, что в серьезных сетях именно витая пара сейчас занимает ведущие позиции.

Один из недостатков Ethernet: ограниченная длина кабеля, который позволяет использовать его только для локальных сетей. Однако, несколько сегментов Ethernet могут быть связаны друг с другом с помощью повторителей (repeaters), мостов (bridges) или маршрутизаторов (routers). Повторители просто копируют сигналы между двумя или больше сегментами так, что все сегменты вместе действуют, как будто это один Ethernet. Но между двумя любыми машинами сети не может быть больше четырех повторителей, иначе они исказят сигнал. Мосты и маршрутизаторы более сложные. Они анализируют поступающие данные и отправляют их только тогда, когда хоста получателя нет в местном сегменте Ethernet.

Ethernet работает подобно системной шине, где хост может послать пакеты (или кадры, frames) до 1500 байтов длиной другому хосту на том же самом сегменте Ethernet. Хост идентифицируется адресом, состоящим из шести байт, зашитых в Ethernet-плату при ее создании. Эти адреса обычно записываются как последовательность шестнадцатеричных чисел с двумя цифрами отделяемыми двоеточиями, например, aa:bb:cc:dd:ee:ff.

Структура, посланная одной машиной, видна и всем остальным, но только хост места назначения подбирает и обрабатывает ее. Если две машины пробуют послать сообщение одновременно, происходит столкновение (collision), которое решается двумя машинами с помощью остановки передачи и повторной попытки передачи данных несколько позже. Число столкновений доходит до 30% от пропускной способности сети, но при сильной нагрузке, этот показатель будет расти. Это нормально. Беспокоиться стоит начинать, когда он достигнет 60%. Тут может сильно помочь витая пара: в ней пакеты идут по двум разным проводам и поэтому не сталкиваются. Именно поэтому, все крупные сети делают на витой паре.

### **Хабы, свитчи и коммутаторы**

Хаб Ethernet (hub) - это центральная часть аппаратного обеспечения, обеспечивающего физическое соединение устройств Ethernet. Он просто осуществляет переадресацию и пересылку информации уровня Ethernet между устройствами, которые к нему подключены. Весь поступающий трафик Ethernet хабы переадресуют каждому присоединенному хосту.

Свитч (switch) – оборудование, по своей сути аналогичное хабу. Разница заключается в том, что у свитча есть внутренние динамические таблицы, хранящие информацию лишь пока поступает питание. В этих таблицах хранятся Ethernet-адреса (MAC) всех устройств, которые к нему подключены. А пакеты, проходящие через свитч направляются исключительно адресату.

Коммутатор предоставляет усовершенствованный способ соединения устройств Ethernet, при котором есть возможность управлять внутренними таблицами. Используя коммутатор, вы можете настраивать различные списки доступа, например, создать приватную сеть. Информация во внутренних таблицах хранится вне зависимости от наличия электрического питания.

### **Другие типы оборудования**

В больших сооружениях, типа университета GU, Ethernet обычно не единственный тип используемого оборудования. В GU локальная сеть каждого отдела связана с университетской магистралью, которая является оптическим кабелем FDDI (Fiber Distributed Data Interface). FDDI использует совершенно другой подход к передаче данных, который основывается на рассылке определенных маркеров (tokens) и только, если машина получает этот маркер, она может послать некий блок (один или несколько пакетов) информации. Главное преимущество FDDI скорость, достигающая 100 Mbps, и максимальная длина кабеля до 200 км. Кроме того, там нет столкновений пакетов из-за того, что сеть основана на передаче маркера.

Если Вы работаете с аппаратурой IBM, часто придется иметь дело с сетью IBM Token Ring. Token Ring является альтернативой Ethernet в LAN и использует решения, аналогичные применяемым в FDDI, но на низкой скорости (4 Mbps или 16 Mbps). В Linux Token Ring настраивается аналогично Ethernet, так что я не буду рассматривать его отдельно. Раньше за право быть наиболее распространенным стандартом бились Ethernet и Token Ring. В конце концов победил Ethernet, но стандарты по своим возможностям находятся примерно на одном уровне.

Есть и совершенно другие сетевые технологии, например, ArcNet и DECNet. Linux поддерживает и их, но из-за

малой популярности, я не буду говорить о них здесь.

Многие сетевые операторы предлагают пакетные протоколы передачи. Вероятно, наиболее популярный из них стандарт X.25. Многие публичные сети подобно Tynnet в США, Austrac в Австралии и Datex-P в Германии предлагают это обслуживание. X.25 определяет собственный набор протоколов, но часто используется, чтобы соединить сети, работающие под TCP/IP и другими протоколами. Так как IP-пакеты не могут быть прямо отображены на X.25 (и наоборот), они просто вставляются в X.25-пакеты и посылаются по сети. X.25 требует синхронной связи и, следовательно, специальные синхронные аппаратные средства последовательного порта. Можно использовать X.25 с нормальными последовательными портами, если Вы используете специальное устройство PAD (Packet Assembler Disassembler). PAD автономное устройство. Оно обеспечивает асинхронные и синхронные последовательные порты. Это управляет протоколом X.25 так, чтобы простые устройства терминала могли создавать и принимать X.25-подключения. Имеется экспериментальная реализация протокола X.25 для Linux.

Более современный протокол называется Frame Relay. Frame Relay использует ряд технических свойств X.25, но гораздо больше похож в поведении на IP. Подобно X.25 Frame Relay требует специальных синхронных последовательных аппаратных средств. Из-за их сходства много плат поддерживают оба эти протокола. Доступен вариант, который не требует никаких специальных внутренних аппаратных средств. Это Frame Relay Access Device (FRAD), который формирует пакеты Frame Relay из пакетов Ethernet для передачи по сети. Frame Relay идеально подходит для переноса TCP/IP между сайтами. Linux обеспечивает драйверы многих внутренних устройств Frame Relay.

Если Вы нуждаетесь в работе с высокоскоростной сетью, Вас должен заинтересовать ATM (Asynchronous Transfer Mode). ATM является новой сетевой технологией, специально созданной для того, чтобы обеспечить управляемый, быстрый, с низким временем ожидания обмен данными. Она обеспечивает контроль над качеством обслуживания (QoS). Много компаний создают сети ATM, потому что это позволяет свести много услуг на одну платформу. ATM часто используется для передачи TCP/IP. Networking-HOWTO предлагает информацию относительно поддержки ATM в Linux.

Часто радиолюбители используют свое оборудование для создания сети из своих компьютеров. Это называется пакетное радио (packet radio) или ham-радио. Протокол, используемый ham-радио, назван AX.25 (он получен из X.25). Он также используется для передачи TCP/IP и других протоколов. AX.25 подобно X.25 требует последовательных аппаратных средств, способных к синхронному действию, или внешнее устройство Terminal Node Controller, чтобы преобразовать пакеты. Есть немало плат с поддержкой этого протокола, они имеют общее название "Z8530 SCC based" (по имени наиболее популярного контроллера связи, используемого в проектах). Еще два протокола, которые работают поверх AX.25: NetRom и Rose, являются протоколами сетевого уровня. Linux поддерживает полные реализации AX.25, NetRom и Rose. AX25-HOWTO является хорошим источником информации по использованию этих протоколов в Linux.

Есть методы, которые используют специально для медленных, но дешевых телефонных линий. Они требуют других протоколов для передачи пакетов типа SLIP или PPP.

## Internet Protocol

Конечно, Вы не хотели бы, чтобы Ваша сеть ограничивалась только Ethernet. В идеале хотелось бы использовать сеть независимо от того, какими аппаратными средствами это достигается. Например, в больших сооружениях (GU) Вы обычно имеете набор отдельных Ethernet, которые должны быть связаны некоторым образом. В GU в математическом отделе используются два Ethernet: одна сеть быстрых машин для профессоров и студентов последних курсов и другая с медленными машинами для студентов первых курсов (обе связаны с FDDI).

Эта связь управляется специальным хостом, так называемым gateway, который направляет поступающие и исходящие пакеты, копируя их между двумя Ethernet и FDDI. Например, если Вы в математическом отделе и хотите получить доступ к машине quark в локальной сети физического отдела, сетевое программное обеспечение не может послать пакеты непосредственно quark, потому что он находится на другом Ethernet. Поэтому этим занимается gateway. Gateway (назовем его sophus) посылает эти пакеты другому gateway (niels) в отделе физики, niels же отправляет их на требуемую машину.

Эта схема направления данных удаленному хосту называется маршрутизацией (routing), а пакеты часто называют дэитаграммами (datagrams). Для простоты обмен дэитаграммами управляется в соответствии с отдельным протоколом, который является независимым от используемых аппаратных средств: IP или Internet Protocol. Далее мы будем рассматривать IP и routing более подробно.

Основное преимущество IP в том, что он преобразует физически несовместимые сети в одну с виду однородную сеть. Это называется internetworking, в результате получаем "метасеть" называемую internet. Обратите внимание на различие между internet и Internet! Последнее является официальным названием одного

специфического глобального inetnet.

Конечно, IP также требует машинезависимой схемы адресации. Это достигается с помощью назначения каждому хосту уникального номера размером в 32 бита, названного IP-адресом (IP address). IP-адрес обычно пишется как четыре десятичных числа для каждой 8-битовой части, разделенных точками. Например, quark мог бы иметь IP-адрес 0x954C0C04, который будет записан как 149.76.12.4. Этот формат также назван dotted decimal notation или dotted quad notation, в общем, формат записи с точками. Это известно под именем IPv4 (Internet Protocol, Version 4), потому что новый стандарт IPv6 предлагает намного более гибкую адресацию и другие современные свойства. Но новый стандарт IPv6 войдет в повсеместное использование еще не скоро...

Теперь мы имеем три различных типа адресов: имя хоста, например quark, IP-адрес и, наконец, адреса аппаратных средств, например, Ethernet-адрес с 6 байтами. Все они так или иначе соответствуют друг другу, так что, когда Вы пишете ssh quark, программное обеспечение находит его IP-адрес. Когда IP пересылает данные в Ethernet отдела физики, так или иначе по IP-адресу выясняется Ethernet-адрес.

Мы не будем здесь вдаваться в подробности этого процесса, а сделаем это позднее. Пока достаточно помнить, что эти шаги называются hostname resolution, поиск IP-адреса по имени хоста, и address resolution, поиск физического адреса по IP.

### **IP на последовательных линиях**

Здесь стандартом является SLIP (Serial Line IP) или IP для последовательных линий. Есть модификация CSLIP (сжатый SLIP, Compressed SLIP), который использует сжатие IP-заголовков, чтобы оптимизировать IP для относительно низкой пропускной способности последовательной связи. PPP или протокол Point-to-Point Protocol еще один протокол для последовательных линий. PPP имеет большее число возможностей, чем SLIP, включая стадии переговоров о начале связи. Его главное преимущество по сравнению со SLIP в том, что он не ограничивается только транспортировкой IP-пакетов, а предназначен для передачи любого типа дэйтаграм.

## **Transmission Control Protocol**

Но, конечно, посылка пакетов от одного хоста к другому это не все, если Вы вошли на quark. Вы хотите иметь надежную связь между Вашим процессом ssh на erdos и процессом оболочки на quark. Таким образом, информация, посылаемая туда и обратно, должна быть разбита на пакеты отправителем и повторно собираться в поток приемником. Хотя это кажется тривиальным, здесь появляется несколько достаточно сложных задач.

Очень важно знать об IP, что он ненадежен. Предположим, что десять людей на Вашем Ethernet начали загружать самый последний выпуск некоторого программного обеспечения с GMU FTP-сервера. Такая активность может оказаться слишком большой для того, чтобы gateway переварил ее, потому что он слишком медленный и ограничен количеством памяти. Теперь если Вы пошлете пакет с quark, sophus может не хватить места в буфере, и поэтому он не сможет отправить этот пакет. IP решает эту проблему, просто забывая про данный пакет. Пакет безвозвратно потерян. Таким образом, ответственность за целостность данных перекладывается на поддерживающие связь хосты.

Это происходит в соответствии с другим протоколом, TCP (Transmission Control Protocol), который надстраивается над IP для создания связи с проверкой целостности данных. Существенный плюс TCP в том, что он использует IP. Это создает иллюзию простой связи между двумя процессами на Вашем хосте и удаленной машине так, что Вы не заботитесь о том, как и по какому маршруту Ваши данные фактически путешествуют. А TCP создает дуплексную связь, позволяющую одновременно как посылать, так и получать информацию.

В TCP точки связи определяются IP-адресами хостов и номерами так называемых портов (port) на каждом из хостов. Порты служат для определения процесса, с которым устанавливается связь. Если обратиться к примеру с номерами телефонов, то IP-адрес соответствует кодам городов, а номер порта местному номеру телефона.

В примере с ssh приложение-клиент (ssh) открывает порт на erdos и соединяется с портом 22 на quark, который прослушивает сервер sshd. Таким образом и устанавливается TCP-связь. Используя эту связь, sshd выполняет процедуру определения прав доступа и запускает оболочку. Стандартный ввод/вывод этой оболочки перенаправляются на TCP-связь, таким образом, все, набранное Вами в ssh на Вашей машине, будет передано через TCP-поток на стандартный ввод оболочки.

## **User Datagram Protocol**

Конечно, TCP не единственный протокол пользователя в сетях TCP/IP. Хотя он и подходит для приложений подобных ssh, но он излишне надежен и не нужен для приложений типа NFS. Вместо него в них использует UDP (User Datagram Protocol). Подобно TCP, UDP также позволяет приложению-клиенту войти в контакт с сервером, обслуживающим определенный порт на удаленной машине, но он не устанавливает связь

для этого. Вместо этого Вы можете использовать его, чтобы посылать отдельные пакеты к месту назначения.

Предположим, что Вы смонтировали директорию TeX с центрального NFS-сервера galois и хотите просмотреть документ, описывающий, как использовать LaTeX. Вы запускаете редактор, который сначала читает указанный файл. Однако, требуется слишком много времени, чтобы установить TCP-связь с galois, послать файл и повторять это снова. Вместо этого на запрос, посланный к galois, тот посылает файл в паре UDP-пакетов, что происходит гораздо быстрее. Однако, UDP не приспособлен для борьбы с потерей пакетов. Этим приходится заниматься NFS.

## Сетевые порты

Вы когда-нибудь замечали, что компьютеры имеют слишком много портов? Что ж, добавим в список порты TCP и UDP. Благодаря наличию сетевых портов один сервер может предоставлять различные сетевые сервисы. Порты являются средством для организации множественных соединений между машинами.

Пакет (TCP или UDP), поступающий в систему, необходимо доставить к определенному порту. Различные порты предоставляют различные сервисы. Например, почтовая служба Интернета называется SMTP. Как указано в файле `/etc/services`, SMTP работает через порт 25. Если запрос на соединение TCP запрашивает порт 25, можно сделать вывод, что происходит обращение к почтовому серверу. Порты позволяют организовывать множественные соединения между несколькими машинами.

Файл `/etc/services` содержит список номеров портов и сервисов, которые с ними ассоциированы. Почти любой сервис можно запускать на произвольном порту, но таким образом можно сбить с толку другие хосты Интернета, которые пытаются соединиться с вашей системой. Формат файла очень прост: официальное имя сервиса, номер порта, протокол, все псевдонимы (aliases) для этого сервиса и комментарии. Поля разделены табуляцией. Например, одним из первых сервисов на хостах UNIX был сервис qotd (Quote of the Day, цитата дня). В файле `/etc/services` можно найти соответствующую запись:

```
qotd          17/tcp      quote          #Quote of the Day
qotd          17/udp      quote          #Quote of the Day
```

Многие сервисы работают через порты TCP и UDP, которым назначены определенные номера, но есть и другие сервисы, работающие только по одному протоколу.

Многие программы читают `/etc/services`, чтобы выяснить, какой порт надо использовать (к какому порту осуществить привязку, bind to). Надо отредактировать этот файл, чтобы в зависимости от программы назначить тот или иной протокол тому или иному порту. Подобно всем другим стандартам, содержимое файла `/etc/services` не является догмой. Служба sshd обычно занимает порт 22, однако я запустил ее на порту 80, чтобы в чрезвычайных обстоятельствах можно было обойти ограничения брандмауэра. Все зависит от программы, задействованной для предоставления того или иного сервиса.

Порты с номерами 1024 и ниже называются портами с низкими номерами (low-numbered ports). Они зарезервированы для ключевых протоколов инфраструктуры Интернета, таких как DNS, telnet и HTTP. Стандарты их применения словно высечены в камне. Порты с номерами 1024 и выше не настолько стандартизованы, поэтому изредка возникают конфликты, когда различные протоколы пытаются использовать один и тот же порт. По существу, клиент инициирует соединение с портом, номер которого выше 1024, и запрашивает соединение с сервером через порт с низким номером.

Время от времени к `/etc/services` обращаются службы, работающие поверх протоколов, отличных от TCP и UDP. Некоторые службы используют протокол доставки дейтаграмм (Datagram Delivery Protocol, DDP). О них беспокоиться не надо - для этого нет оснований. В то же время почти любой программе может потребоваться внесение произвольных записей в `/etc/services`.

## Библиотека сокетов

В Unix-подобных операционных системах программное обеспечение, выполняющее все задачи и протоколы, описанные выше, обычно является частью ядра, аналогично сделано и в Linux. Интерфейс программирования, наиболее общий для мира Unix, библиотека гнезд Berkeley (Berkeley Socket Library). Свое название она получила из-за популярной аналогии, которая рассматривает порты как гнезда (розетки). Правда, в современной русской сетевой терминологии устоялся термин "сокет". Она обеспечивает запрос bind, который определяет удаленный хост, транспортный протокол и сервис, к которому программа может присоединиться или слушать (используя системные вызовы connect, listen и accept). Библиотека сокетов обеспечивает не только TCP/IP-сокеты (сокеты типа AF\_INET), но также класс, который обрабатывает подключения, локальные для машины (AF\_UNIX). Некоторые реализации могут также обрабатывать другие классы подобно XNS (Xerox Networking System) или X.25.

В Linux библиотека сокетов является частью стандартной C-библиотеки `libc`. Типы сокетов, которые она поддерживает в настоящее время, указаны в таблице:

Таблица. Типы сокетов, поддерживаемых `libc`.

<i>Протокол</i>	<i>Классы сокетов</i>
TCP/IP	AF_INET и AF_INET6
Локальные соединения	AF_UNIX
Сети Novell	AF_IPX
X.25	AF_X25
ATM	AF_ATMPVC и AF_ATMSVC
Amateur Radio	AF_AX25, AF_NETROM и AF_ROSE

Ведутся работы по добавлению сокетов и для других протоколов.

## Сети в Linux

Поскольку ОС Linux есть результат совместных усилий программистов во всем мире, она не была бы возможна без глобальной сети. Так что не удивительно, что на ранних стадиях разработки многие из них стали работать над сетевыми возможностями. Реализация UUCP была в Linux почти с самого начала, а TCP/IP появилась осенью 1992, когда Ross Biro и другие создали пакет Net-1.

После выхода Ross из активной разработки в мае 1993, Fred van Kempen начал работать над новой реализацией, переписывая главные части кода. Этот проект был известен как Net-2. Первый публичный выпуск Net-2d был сделан летом 1993 (как часть ядра 0.99.10) и с тех пор поддерживается и расширяется командой, особенное место в которой занимает Alan Cox. Первоначальная работа Алана была известна как Net-2Debugged. После тяжелой отладки, многочисленных усовершенствований и выпуска Linux 1.0 он изменил название на Net-3. Net-3 был далее разработан для Linux 1.2 и Linux 2.0. Ядра 2.2 и более поздние используют поддержку сети Net-4, которая остается стандартным пакетом на текущий момент.

Net-4 Linux Network предлагает широкое разнообразие драйверов устройств и продвинутых свойств. Стандартные протоколы Net-4 включают SLIP и PPP (для работы с последовательными линиями), PLIP (для параллельных линий), IPX (для Novell-совместимых сетей), Appletalk (для сетей Apple), AX.25, NetRom и Rose (для любительских радиосетей). Другие стандартные свойства Net-4 включают IP firewalling, IP accounting и IP Masquerade (рассматриваются в курсе «Сетевое администрирование Linux»). IP tunnelling поддерживан в паре различных разновидностей и продвинутой стратегии маршрутизации. Поддерживается очень большое число устройств Ethernet. К тому же, есть поддержка многих плат FDDI, Token Ring, Frame Relay, ISDN и ATM.

Дополнительно, имеется ряд других свойств, которые значительно расширяют гибкость Linux. Эти свойства включают реализацию файловой системы SMB, что позволяет работать с lanmanager и Microsoft Windows. Пакет называется Samba, написан Andrew Tridgell. Есть поддержка Novell NCP (NetWare Core Protocol).

## Различные направления разработки

Fred продолжил разработку после того, как Net-2Debugged был сделан официальной сетевой реализацией. Эта разработка привела к пакету Net-2e. Fred пытался стандартизовать интерфейс драйверов устройств (Device Driver Interface, DDI), но сейчас работа над Net-2e уже закончена.

Другая реализация работы с сетями TCP/IP исходила от Matthias Urlichs. Он написал драйвер ISDN для Linux и FreeBSD. Для этого драйвера он интегрировал часть BSD-кода работы с сетями в ядро Linux. Этот проект также уже свернут.

Было много изменений, они и сейчас вносятся, поскольку разработка Linux не закончена. Иногда это означает, что изменения также должны произойти в другом программном обеспечении типа сетевых инструментальных средств конфигурации.

Сетевая реализация Net-4 стала стандартом и используется на очень большом числе машин во всем мире. Много работы было выполнено по улучшению эффективности Net-4, и теперь она конкурирует с самыми лучшими реализациями, доступными для тех же самых аппаратных платформ. Linux распространяется в среде Internet Service Provider и часто используется, чтобы формировать дешевые и надежные серверы World Wide Web, e-mail и news для этих организаций.

## Сетевые интерфейсы

Чтобы скрыть разнообразное оборудование, которое может использоваться в сетевой среде, TCP/IP определяет абстрактный интерфейс (interface), через который можно обращаться к аппаратным средствам. Этот интерфейс предлагает набор действий, который является одинаковым для всех типов аппаратных средств и в основном имеет дело с посылкой и получением пакетов данных.

Для каждого периферийного устройства, которое вы хотите использовать, в ядре должен быть представлен соответствующий интерфейс. Например, Ethernet-интерфейсы в Linux названы eth0 и eth1, интерфейсы PPP (для модема) ppp0 и ppp1, а FDDI-интерфейсы fddi0 и fddi1. Эти названия интерфейса используются при конфигурировании, когда вы хотите определить ядру специфическое физическое устройство. Они не имеют никакого назначения кроме этого.

Чтобы работать в сети TCP/IP, интерфейсу должен быть назначен IP-адрес, который служит как идентификатор при общении с остальным миром. Этот адрес различен в зависимости от названия интерфейса: если вы сравниваете интерфейс с дверью, тогда адрес подобен пластине с именем, прикрепленной на ней.

Конечно, имеются и другие параметры устройства которые необходимо отрегулировать. Один из них, максимальный размер пакета, который может быть обработан данной частью аппаратуры, также называемый Maximum Transfer Unit (MTU). Другие параметры будут представлены позже.

## ip адреса

IP-адрес - это 32-разрядное число, обычно разделенное на четыре группы по 8 бит в каждой. В переводе на человеческий язык это означает четыре числа, принимающих значения от 0 до 255, разделенных точками. Например, 192.168.1.87 - это действительный IP-адрес, а 192.259.0.87 - нет, поскольку одно из чисел превышает 255. Ну, с «адресом» 607.322.843.999 все ясно.

Каждое устройство в Интернете имеет уникальный IP-адрес - если не применяется преобразование сетевых адресов (Network Address Translation, NAT) или другие трюки.

В локальной сети, которая не связана с другими сетями, вы можете назначить IP-адреса согласно вашим персональным предпочтениям. Однако, для участков Internet, они назначаются NIC (Network Information Center). Есть несколько диапазонов IP-адресов, зарезервированных для частных сетей. Они перечислены в таблице:

Таблица. Диапазоны IP-адресов, зарезервированных для частных сетей

<i>Класс</i>	<i>Сети</i>
A	от 10.0.0.0 до 10.255.255.255
B	от 172.16.0.0 до 172.31.0.0
C	от 192.168.0.0 до 192.168.255.0

Для более легкого чтения, IP-адреса разбивают на четыре 8-битовых числа, названные octets. Например, quark.physics.gu.edu имеет IP-адрес 0x954C0C04, который записывается как 149.76.12.4. Этот формат часто называют dotted quad notation (запись с точками-разделителями).

## Что такое бит?

Сетевому администратору частенько встречаются такие термины, как 32 бита и 48 бит. Их смысл необходимо понимать, чтобы можно было распознавать неверные значения.

Мы знаем, что компьютер интерпретирует данные как нули и единицы. Одна единица или один ноль составляют один бит. Когда протокол задает биты, он указывает это число согласно представлениям компьютера - в двоичной форме. (Двоичные операции, или операции с числами по основанию 2, наверняка изучались вами в средней школе и быстро забылись. Пришло время освежить эти знания. Двоичная система - это просто другой способ представления чисел, с которыми люди работают каждый день.)

В десятичной математике (числа с основанием 10), ежедневно используемой людьми, задействованы цифры от 0 до 9. Для того чтобы получить число, превышающее самую большую цифру, слева к этой цифре добавляется другая, а текущая становится равной 0. (Это правило «переноса 1», которое вы изучили много лет назад и сейчас используете неосознанно.) В двоичной математике есть только цифры 0 и 1. Чтобы получить число, превышающее самую большую цифру, слева к этой цифре добавляется другая, а текущая становится равной 0. Все то же самое, только цифр меньше.

Вот как десятичные числа преобразуются в двоичные.

Десятичные	Двоичные	Десятичные	Двоичные
0	0	8	1000
1	1	9	1001
2	10	10	1010
3	11	11	1011
4	100	12	1100
5	101	13	1101
6	110	14	1110
7	111	15	1111

Число из 32 бит, например IP-адрес, представляет собой строку из 32 нулей и единиц. Однако это число не представляет собой неделимое целое. IP-адрес разбит на четыре 8-битных числа (причина будет объяснена в следующем разделе.)

**Внимание!** Преобразование из двоичной формы в десятичную и обратно реализовано во многих калькуляторах. Такую операцию может выполнить даже калькулятор Windows, если перевести его в инженерный режим. Возьмите IP-адрес, поместите каждое из четырех составляющих его чисел в калькулятор и преобразуйте их в двоичную форму.

## Сетевые маски

При подключении компании к Интернету провайдер выделяет ей блок IP-адресов. Эти адреса используются для локальной сети Ethernet. Как правило, этот блок невелик, скажем, 16 или 32 IP-адреса. Если речь идет о небольшой группе идентичных серверов, компания получит лишь несколько IP-адресов. Все зависит от ее потребностей. Размер блока IP-адресов определяет *сетевую маску*.

Раньше довольно часто можно было встретить сетевую маску 255.255-255.0. Также общеизвестно, что неверная сетевая маска приведет к неработоспособности системы. Сейчас такая простая маска встречается все реже и реже. Чтобы понять, в чем тут дело, необходимо разобраться, что представляет собой сетевая маска и как выделяются блоки IP-адресов.

Много лет назад IP-адреса выделялись блоками трех размеров, соответствующих классу А, классу В и классу С. Со временем эта терминология устарела, но мы возьмем ее за основу.

Класс А был очень простым: первое из четырех чисел: IP-адресов было фиксированным. InterNIC мог выделить блок IP-адресов класса А, скажем, 10.0.0.0. Оставшиеся три числа владелец IP-адреса мог назначать по своему усмотрению. Например адреса с 10.1.0.0 по 10.1.1.255 можно было выделить для информационного центра, адреса с 10.1.2.0 по 10.1.7.255 - для бостонского офиса и т. д. Блоки адресов класса А получали только очень большие компании, такие как Ford и Херох, а также влиятельные академические организации, связанные с компьютерными технологиями.

В блоках класса В фиксированными были первые два из четырех чисел, составляющих IP-адрес. Блок адресов класса В мог выглядеть, например, как 64.29.0.0. Каждый IP-адрес, использовавшийся во внутренней сети, начинался с 64.29, а по своему усмотрению можно было назначать последние два числа. Многие компании среднего размера получили блок адресов класса В.

Подобным образом в блоке класса С фиксированы первые три числа адреса. Обычно такие блоки получали маленькие компании. Провайдер услуг Интернета выделял номер, подобный 209.69.9, и предоставлял владельцу блока возможность назначать последнее число каждого адреса.

Такая схема приводила к растрачиванию IP-адресов. Многие мелкие фирмы не нуждались в 256 IP-адресах, а многие компании среднего размера занимали больше 256, но меньше 65 000 адресов в блоке класса В. И почти никому не требовались все 16 миллионов адресов из блока класса А. Однако таков был выбор, и до бума Интернета такое положение всех устраивало.

Сегодня IP-адреса выделяются вместе с *префиксной длиной (prefix length)*, отделяемой *косой чертой (slash)*. Блок IP-адресов может выглядеть так: 192.168.1.128/25. Подобная запись может быть не очень понятной, но делает возможным более тонкое использование классов адресов. Известно, что каждое число в IP-адресе состоит из 8 бит. При использовании классов «фиксированным» считается определенное количество бит - их нельзя изменить в локальной сети. Адрес класса А имеет 8 фиксированных бит, класса В - 16 бит и класса С — 24 бита.



В приведенных примерах IP-адреса того или иного класса не были представлены в двоичной форме, и я не буду заставлять вас выполнять это преобразование. Однако под IP-адресом следует понимать строку из двоичных чисел. В адресах локальной сети можно изменить биты в правой части каждого адреса, но не биты в левой части.

Нет никаких причин, по которым длина каждой части адреса должна быть кратной 8. Префиксная длина - это просто некоторое количество фиксированных битов. Запись /25 означает, что фиксированными являются 25 бит, то есть на один бит больше, чем в адресе класса C. В этом случае изменять можно 7 бит. В следующей записи фиксированные биты представлены единицами, а изменяемые — нулями:

11111111. 11111111. 11111111.10000000

Это по-детски просто - конечно, если вы мыслите двоичными категориями. С двоичными числами вовсе не обязательно работать каждый день, но тем, кто не понимает базовые концепции двоичной математики, преобразование чисел из двоичной формы в десятичную кажется совершенно не понятным. В ходе практической деятельности можно научиться видеть десятичные числа в логичной двоичной записи.

Что это означает на практике?

Прежде всего, блоки IP-адресов выделяются в количестве, кратном 2. Если изменяемыми являются 4 бита, то возможны 16 адресов ( $2 \times 2 \times 2 \times 2 = 16$ ), а если 8 бит, то ( $2^8$ ) 256 IP-адресов. Если кто-либо говорит, что доступно 13 IP-адресов, то либо речь идет о разделяемой сети Ethernet, либо высказанное утверждение ошибочно.

Сетевая маска - это всего лишь другой способ указать количество фиксированных битов. Двоичное число, состоящее из 8 разрядов, может иметь значение от 0 до 255. В случае префиксной длины /24 сетевая маска будет такой: 255.255.255.0. Если префиксная длина представлена как /25, то фиксированными являются все биты в первых трех числах и один бит в последнем числе. В предыдущем примере последнее число было представлено в двоичной форме так: 10000000. Несложные вычисления по преобразованию двоичного числа в десятичную форму, выполненные на калькуляторе, дадут 255.255.255.128.

**Внимание!** Такое преобразование стоит выполнить и на бумаге, по крайней мере несколько раз. Таким образом можно научиться гораздо большему. Вперед, смелее!

Нередко можно увидеть IP-адрес хоста вместе с его сетевой маской, например 192.168.3.4/26. Запись /32 представляет собой не сеть, а отдельный хост и применяется, когда необходимо ясно показать, что речь не идет о сети.

## Трюки с сетевой маской

Преобразование из десятичной формы в двоичную не всегда доставляет удовольствие. Вот как можно вычислить сетевую маску.

Прежде всего выясним, сколько реальных IP-адресов есть в наличии. Это число будет кратно 2. Почти всегда количество адресов меньше значения, вычисляемого по записи /24 (обычная адресация класса C). Из числа 256 вычтем количество адресов, имеющихся в наличии, и получим последнее число сетевой маски.

Например, при наличии 64 IP-адресов последний компонент сетевой маски вычисляется так:  $256 - 64 = 192$ , а сетевая маска будет иметь вид: 255.255. 255.192.

Для того чтобы избежать двоичных преобразований, обратимся к логике. Составление списка IP-адресов, допустимых в той или иной сети, - дело нелегкое. При наличии IP-адреса 192.168.54.187/25 фиксированными являются 25 бит. Значит, используется блок из 128 IP-адресов. Рассмотрим последнее число этого адреса, 187. Конечно, оно находится не в интервале между 0 и 127, а в интервале между 128 и 255. Другие хосты из этого блока имеют IP-адреса от 192.168.54.128 до 192.168.54.255.

## Шестнадцатеричные сетевые маски

Все ясно? Хорошо. К сожалению, в UNIX принято представлять сетевую маску в шестнадцатеричной форме (основание 16), а не в десятичной или двоичной. Рано или поздно вы увидите сетевую маску

`0xff ff f00`

Шестнадцатеричная цифра занимает 4 бита, поэтому каждая 8-битная часть сетевой маски может быть представлена как две шестнадцатеричных цифры. (IP-адреса также можно представить в такой форме, но это не принято.) Шестнадцатеричные числа всегда начинаются с «0х», поэтому распознать их легко.

Для преобразования числа в шестнадцатеричную форму проще всего воспользоваться калькулятором или специальной таблицей. В таблице ниже для удобства представлено преобразование «запись с косой чертой/шестнадцатеричная форма/двоичная форма/десятичная форма» для сетевых масок от /24 до /32.

Таблица. Преобразование сетевых масок

Префикс	Двоичная маска	Десятичная маска	Шестнадцатеричная маска	Доступные IP- адреса
/24	00000000	0	0x00	256
/25	10000000	128	0x80	128
/26	11000000	192	0xc0	64
/27	11100000	224	0xe0	32
/28	11110000	240	0xf0	16
/29	11111000	248	0xf8	8
/30	11111100	252	0xfc	4
/31	11111110	254	0xfe	2
/32	11111111	255	0xff	1

## Неиспользуемые адреса

Теперь ясно, что запись /26 подразумевает 64 IP-адреса. К сожалению, не все из них можно использовать. Первый IP-адрес - это номер сети (network number). Он предназначен для внутреннего управления сетевыми ресурсами. В любой группе IP адресов последний номер является широковещательным адресом (broadcast address). Согласно спецификации IP, каждая машина в сети должна откликаться на запрос, посылаемый по этому адресу. Такая возможность позволяет проверять доступность адресатов и выяснять, какие IP-адреса используются в сети. Например, в типичной, сети /24 широковещательный адрес будет иметь вид: x.y.z.255. В конце 90-х годов отправка пакетов по широковещательному адресу применялась при организации атак на сети. Сейчас эта возможность отключена в большинстве операционных систем.

В любом случае интерфейсу нельзя назначить первый и последний IP-адреса сети. Но можно попробовать.

Если помните, немного выше я упоминал, что маска /31 практически бесполезна. Такая запись подразумевает два IP-адреса, а первый и последний адреса использовать нельзя. Остается не так уж много жеста для серверов и даже для клиентов.

## Преобразование адресов

Теперь, когда Вы знаете, как составлены IP-адреса, можно задаться вопросом, как они используются в сетях Ethernet или Token Ring. В конце концов эти протоколы имеют их собственные адреса для идентификации компьютеров, которые не имеют абсолютно ничего общего с адресом IP, не так ли?

Необходим механизм, чтобы отобразить адреса IP на адреса основной сети. Этим механизмом является Address Resolution Protocol (ARP, протокол преобразования адресов). Фактически, ARP не ограничен сетями Ethernet или Token Ring, он используется и на других типах сетей, типа протокола AX.25 для любительского радио. Идея, лежащая в основе ARP, точно такая, какую большинство людей использует, когда они должны найти человека X в толпе из 150 людей: человек, который хочет его видеть, зовет достаточно громко, чтобы каждый в толпе мог его услышать, а тот, кого зовут, отвечает. Когда он отвечает, мы узнаем, который человек нам нужен.

Когда ARP хочет найти адрес Ethernet, соответствующий данному IP-адресу, он использует свойство Ethernet под названием broadcasting, в котором пакеты адресуются всем машинам в сети одновременно. Пакет, который посылает ARP, содержит запрос IP-адреса. Каждый компьютер, получивший запрос, сравнивает его содержимое с собственным IP-адресом и, если он совпадает с указанным в запросе, возвращает ответ. Запрашивающий компьютер может теперь извлекать адрес Ethernet из ответа.

Возникает проблема: как обратиться к Internet-адресу, который может быть в другой сети в другом полушарии? Ответ на этот вопрос называется routing, а именно нахождение физического расположения компьютера в сети. Мы обсудим эту проблему в следующем разделе.

Давайте поговорим более подробно об ARP. Как только компьютер обнаружил адрес Ethernet, он сохраняет этот адрес в кэше ARP так, чтобы не делать запрос снова, когда в следующий раз потребуется послать пакет рассматриваемому компьютеру. Однако, неблагоприятно хранить эту информацию всегда; плата Ethernet удаленного компьютера может быть заменена из-за технических проблем, так что ARP-запись становится неактуальной. Следовательно, записи в кэше ARP будут сброшены через некоторое время, чтобы вызвать другой запрос для поиска IP-адреса.

Иногда необходимо найти адрес IP, связанный с данным адресом Ethernet. Это случается, когда машина без диска хочет загружаться с сервера по сети, что является частой ситуацией в локальных сетях. Клиент без диска, однако, не имеет фактически никакой информации относительно себя, кроме адреса Ethernet! Так что он передает сообщение, содержащее запрос к серверу начальной загрузки, чтобы обеспечить себя адресом IP. Для этой ситуации имеется другой протокол Reverse Address Resolution Protocol (RARP, протокол обратного преобразования адреса). Наряду с протоколом BOOTP он служит, чтобы определить процедуру начальной загрузки клиентов без диска по сети.

## IP маршрутизация

Как пакеты находят путь к машине назначения? Эта задача подробно рассматривается ниже.

### IP сети

Когда вы пишете письмо, вы обычно помещаете на конверте полный адрес. После того, как вы опускаете его в почтовый ящик, почта доставит его по месту назначения: оно будет послано в обозначенную страну, чья национальная почта пошлет его в требуемый регион и т.д. Преимущество этой иерархической схемы довольно очевидно: везде, где вы отправляете по почте письмо, местный начальник почтового отделения будет точно знать, куда передать это письмо и не должен заботиться, каким путем письмо будет путешествовать.

IP-сети построены подобным же образом. Весь Internet состоит из набора сетей, названных автономными системами (autonomous systems). Каждая такая система производит всю маршрутизацию между своими членами так, что задача посылки пакетов сведена к обнаружению пути к сети с требуемым хостом. Это означает, что как только пакет вручен любому хосту, который находится в нужной сети, обработка выполняется исключительно данной сетью.

### Подсети

Эта структура отражена в разбиении IP-адреса на хост и сетевую часть, как объяснено выше. По умолчанию, сеть места назначения получается из сетевой части IP-адреса. Таким образом, хосты с идентичными IP-адресами сети должны располагаться в пределах одной подсети и наоборот.

Имеет смысл предложить подобную схему также и внутри сети, так как она может состоять из набора сотен меньших сетей, где самыми маленькими единицами являются физические сети типа Ethernet. Поэтому IP позволяет поделить IP-сеть на несколько подсетей (subnets).

Подсеть принимает ответственность за доставку пакетов для определенного диапазона IP-адресов. Как с классами A, B или C она идентифицируется сетевой частью IP-адресов. Однако сетевая часть теперь расширена, чтобы включить некоторые биты части хоста. Число битов, которые интерпретируются как номер в подсети, задается так называемой маской подсети (subnet mask) или netmask. Это 32-разрядное число, которое определяет разрядную маску для сетевой части IP-адреса.

Сеть университета GU является примером такой сети. Она имеет класс B с сетевым адресом 149.76.0.0 и netmask, поэтому равна 255.255.0.0.

Внутри сеть GU состоит из нескольких меньших сетей типа локальных сетей различных отделов. Так что диапазон IP-адресов разбит на 254 подсети: от 149.76.1.0 до 149.76.254.0. Например, отдел теоретической физики имеет адрес 149.76.12.0. Университетский оптоволоконный кабель тоже является сетью с собственным адресом 149.76.1.0. Эти подсети имеют одинаковый сетевой IP-адрес, в то время как третья часть адреса (octet) используется, чтобы различать их между собой. Таким образом, они будут использовать сетевую маску 255.255.255.0.

Стоит заметить, что subnetting (техника создания подсетей) чисто внутреннее дело сети. Подсети создаются сетевым владельцем (или администратором). Часто подсети создаются, чтобы отразить существующие границы, будь они физические (две сети Ethernet), административные (между двумя отделами) или географические. Однако, эта структура воздействует только на внутреннее поведение сети и полностью невидима для внешнего мира.

### Gateways

Subnetting не только организационное деление, но часто и естественное следствие границ аппаратных средств. Знания хостов о строении данной физической сети, типа Ethernet, являются очень ограниченными: единственные хосты, с которыми они способны говорить непосредственно, те, что находятся в той же сети. Ко всем другим хостам они могут обращаться только через так называемый gateway (шлюз). Gateway, это хост,

который связан с двумя или больше физическими сетями одновременно и сконфигурирован так, чтобы передавать пакеты между ними.

По IP достаточно легко распознать, находится ли хост в местной физической сети: различные физические сети должны принадлежать различным IP-сетям. Например, сетевой адрес 149.76.4.0 зарезервирован для хостов в локальной сети математиков. При посылке пакетов машине quark сетевое программное обеспечение на машине erdos немедленно увидит по IP-адресу 149.76.12.4, что хост места назначения находится в другой физической сети, и поэтому может быть достигнут только через gateway (допустим, машину sophus ).

Машина sophus непосредственно связана с двумя подсетями: отделом математики и университетской магистралью. Они доступны через различные интерфейсы (eth0 и fddi0 соответственно). Но какой IP-адрес мы ей назначаем? 149.76.1.0 или 149.76.4.0?

Ответ: оба. При работе с сервером в локальной сети математиков машина sophus использует IP-адрес 149.76.4.1, а при работе с хостом на магистрали она должна использовать 149.76.1.1. Таким образом, gateway получает по одному IP-адресу на каждую сеть, к которой подключен. Эти адреса (вместе с netmask) привязаны к интерфейсу, через который обращаются подсети. Значит, интерфейсы и адреса sophus связаны так:

<i>Интерфейс</i>	<i>Адрес</i>	<i>Маска подсети</i>
eth0	149.76.4.1	255.255.255.0
fddi0	149.76.1.1	255.255.255.0
lo	127.0.0.1	255.0.0.0

Последняя запись определяет loopback интерфейс lo.

Вообще, вы можете не обращать внимание на различия между адресами хоста и интерфейса. Относитесь к адресу хоста, который находится только в одной сети, как к адресу того и другого, хотя строго говоря, это Ethernet-интерфейс имеет IP-адрес. Однако, это различие ощутимо только, когда вы работаете с gateway.

## Таблица маршрутов (Routing Table)

Теперь сосредоточим наше внимание на том, как IP выбирает gateway при доставке пакетов определенной сети.

Как мы видели раньше, erdos, когда передавал пакеты для quark, проверил место назначения и нашел, что его нет в местной сети. Поэтому он посылает пакет gateway sophus, который теперь сталкивается с той же самой задачей. Машина sophus определяет, что quark не находится в сетях, с которыми он непосредственно связан, так что он передает этот пакет другому gateway, чтобы он перенаправил его дальше. Правильный выбор был бы niels (gateway отдела физики). Но шлюзовая машина sophus нуждается в некоторой информации, чтобы определить подходящий gateway.

Для этого используется таблица маршрутизации IP, которая определяет какие сети присоединены с помощью каких gateways. Обязательно должен быть указан маршрут по умолчанию (default route), по которому будут направляться все пакеты с адресами в неизвестных сетях. Этот gateway связан с сетью 0.0.0.0. На sophus эта таблица могла бы напоминать эту:

<i>Сеть</i>	<i>Netmask</i>	<i>Gateway</i>	<i>Интерфейс</i>
149.76.1.0	255.255.255.0	-	fddi0
149.76.2.0	255.255.255.0	149.76.1.2	fddi0
149.76.3.0	255.255.255.0	149.76.1.3	fddi0
149.76.4.0	255.255.255.0	-	eth0
149.76.5.0	255.255.255.0	149.76.1.5	fddi0
...	...	...	...
0.0.0.0	0.0.0.0	149.76.1.2	fddi0

Маршруты к сетям, с которыми sophus связан непосредственно, обозначаются знаком "-" в столбце gateway.

Таблицы маршрутизации могут быть построены различными средствами. Для маленькой сети наиболее эффективно строить их вручную и передавать их IP, используя команду route во время загрузки системы. Для больших сетей они строятся и регулируются во время работы маршрутизирующих демонов; они запускаются на центральном хосте и обмениваются информацией с другими компьютерами для вычисления оптимального маршрута между членами сетей.

Процесс определения правильного маршрута прост, но требует знания логики и двоичной арифметики:

маршрут совпадает с пунктом назначения, если  $(\text{адрес сети AND netmask}) = (\text{адрес назначения AND netmask})$ .

Это означает, что маршрут правильный, если числу бит адреса сети, указанное netmask (начиная с левого бита) соответствует то же самое число битов в адресе назначения.

Когда реализация IP ищет самый лучший маршрут к адресату, может найтись ряд записей маршрутизации, которые соответствуют целевому адресу. Например, мы знаем, что заданный по умолчанию маршрут соответствует каждому адресату, но пакеты, предназначенные для местных сетей, будут соответствовать и их локальному маршруту. Как узнать, какой маршрут использовать? Здесь netmask играет важную роль. В то время как оба маршрута соответствуют адресату, один из маршрутов имеет большую netmask, чем другой. Я уже упомянул, что netmask используется, чтобы разбить наше адресное пространство на меньшие сети. Большой netmask более определенно задает целевой адрес, значит, при маршрутизации мы должны всегда выбирать маршрут, который имеет самую большую netmask. Заданный по умолчанию маршрут имеет netmask из нулевых битов, и в конфигурации, приведенной выше, местные сети имеют 24-разрядный netmask. Если пакет соответствует местной сети, он будет направлено на соответствующее устройство, а не на заданный по умолчанию маршрут, потому что локальный сетевой маршрут соответствует netmask с большим числом битов. Единственный пакет, который будет направлен через заданный по умолчанию маршрут это тот, который не соответствует вообще никакому другому маршруту.

В зависимости от размера сети используются различные протоколы маршрутизации. Для маршрутизации в автономной системе (типа университетского городка) лучше подходит внутренний протокол маршрутизации (internal routing protocols), например, RIP (Routing Information Protocol, протокол маршрутной информации), который предложен в демоне BSD routed. Для маршрутизации между автономными системами используются внешние протоколы маршрутизации (external routing protocols) типа EGP (External Gateway Protocol) или BGP (Border Gateway Protocol). Они были предложены в демоне gated (разработка University of Cornell's).

## Метрические значения

Динамическая маршрутизация, основанная на RIP, выбирает самый лучший маршрут к некоторому хосту или сети, основываясь на наборе hops (переходов), то есть с помощью пакетов, рассылаемых перед передачей основной информации. Чем более короткий маршрут будет определен, тем лучше RIP его оценивает. Очень длинные маршруты с 16 или больше hops рассматриваются как неподходящие и отвергаются.

Чтобы использовать RIP для управления информацией, маршрутизируемой внутри вашей сети, вы должны запустить gated на всех хостах. Во время загрузки gated проверяет все активные сетевые интерфейсы. Если имеется больше одного активного интерфейса (не считая loopback), он предполагает, что хост передает пакеты между несколькими сетями и будет активно обмениваться маршрутной информацией. Иначе, он будет только пассивно получать RIP-пакеты и модернизировать локальную таблицу маршрутизации.

Получив информацию из локальной таблицы маршрутизации, gated вычисляет длину маршрута по так называемому метрическому значению (metric value), связанному с записью в таблице. Это метрическое значение задается администратором системы при конфигурировании маршрута и должно отражать фактическую трудоемкость использования этого маршрута. Поэтому размер маршрута к подсети, с которой хост непосредственно связан, должен всегда быть установлен в ноль, в то время как маршрут, проходящий через два шлюза, должен иметь размер два. Обратите внимание на то, что Вы не должны беспокоиться относительно метрического значения, когда не используете RIP или gated.

## Internet Control Message Protocol

IP имеет вспомогательный протокол, о котором мы еще не говорили. Это Internet Control Message Protocol (ICMP), используемый ядром Linux для передачи сообщений об ошибках на другие компьютеры. Допустим, что вы с системы erdos пытаетесь связаться по telnet с портом 12345 на quark, но на той машине нет никаких процессов на этом порте. Когда первый пакет TCP попадет на этот порт quark, сетевой демон увидит этот пакет и немедленно возвратит ICMP-сообщение на erdos с сообщением "Port Unreachable".

ICMP-протокол обеспечивает несколько различных сообщений, определяемых условиями ошибки. Однако, имеется очень интересное сообщение, названное Redirect. Оно генерируется модулем маршрутизации, когда он обнаруживает, что другой компьютер использует его как шлюз, несмотря на наличие намного более короткого маршрута. Например, после загрузки таблица маршрутизации sophus не была завершена. Она могла бы содержать маршруты к сети Mathematics network, к каналу FDDI и заданному по умолчанию маршруту, указывающему на GU Computing Center (gcc1). Таким образом, пакеты для quark были бы посланы gcc1, а не niels, шлюзу кафедры физики. При получении такого пакета gcc1 обратит внимание, что это неправильная маршрутизация и передаст пакет на niels, одновременно возвращая сообщение ICMP Redirect на sophus.

Это, очевидно, очень хороший способ избежать настройки маршрутов, кроме базисных, вручную. Однако

помните, что доверять сообщениям Redirect протоколов RIP и ICMP можно не всегда! ICMP Redirect и RIP не дают возможности проверить подлинность информации. Эта ситуация позволяет злонамеренному человеку сделать очень многое. Сетевой демон Linux обрабатывает сообщения Network Redirect, как будто они были сообщениями Host Redirects. Это минимизирует повреждение при нападении, ограничивая их одной машиной, а не целой сетью.

## Преобразование имен машин

Как описано выше, адресация в сети TCP/IP крутится вокруг 32-разрядных адресов. Однако, вам будет трудно запомнить даже некоторые из них. Поэтому хосты чаще известны под "обычными" именами типа gauss или strange. Очевидно, требуются программы для получения IP-адреса по имени машины. Этот процесс назван получением или разрешением имени (hostname resolution).

Приложение, которое хочет найти IP-адрес по заданному имени хоста, не должно пытаться сделать это собственными силами. Вместо этого оно обращается к библиотечным функциям, которые для этого и написаны, они называются gethostbyname и gethostbyaddr. Традиционно эти и ряд других процедур были сгруппированы в отдельной библиотеке, названной resolverlibrary. В Linux это часть стандартной libc. Настройка преобразователя имен детально рассматривается в курсе «Сетевое администрирование Linux».

В маленькой сети не очень трудно поддерживать таблицу, сопоставляющую имена хоста с IP-адресами. Эта информация обычно хранится в файле /etc/hosts. При добавлении или перемещении хоста, или при переназначении адресов, все что вы должны сделать, это изменить файл hosts на всех хостах. Очевидно, что это будет достаточно трудно в сетях с большим количеством машин.

Одно из решений этой проблемы: сетевая информационная система (Network Information System, NIS), разработанная Sun Microsystems, также известная как YP или желтые страницы. NIS хранит файл hosts и другую информацию в базе данных на главном хосте, с которого клиенты могут восстановить свои файлы, если это необходимо. Этот способ подходит только для сетей среднего размера, потому что он требует поддерживать полную базу данных как на центральной машине, так и на всех остальных. Установка и настройка NIS здесь не рассматривается, так как в данный момент не все компоненты системы NIS являются свободным программным обеспечением.

В Internet информация об адресах первоначально хранилась в единственном файле HOSTS.TXT. Этот файл поддерживался в NIC (Network Information Center) и должен был загружаться всеми участвующими сайтами. Когда сеть выросла, возникло несколько проблем. Постоянное обновление и постоянная перекачка файла HOSTS.TXT требовали все больше ресурсов, нагрузка на сервер, который этим занимался, стала слишком высока. Но еще большей проблемой стало придумывание новых (не совпадающих с прежними) имен.

Вот почему в 1984 г. была введена новая схема Domain Name System (DNS), разработанная Paul Mockapetris и решившая обе проблемы одновременно. Domain Name System подробно рассматривается в курсе «Сетевое администрирование Linux».

## Обзор сетевых устройств в Linux

Ядро Linux поддерживает несколько драйверов для различных типов оборудования. Этот раздел дает краткий обзор семейств доступных драйверов и имен интерфейсов, используемых для них.

В Linux имеется ряд стандартных имен интерфейсов, которые описаны ниже. Большинство драйверов поддерживают больше, чем один интерфейс, тогда интерфейсы перечисляются как eth0 и eth1:

### lo

Локальный интерфейс для loopback. Он используется для отладки, а также рядом сетевых приложений. Он работает подобно замкнутому циклу, возвращая все пакеты, переданные ему, сетевому уровню того же хоста. В ядре имеется всего одно loopback-устройство, и нет большого смысла в наличии меньшего или большего количества.

### eth0, eth1, ethn

n-ая Ethernet-карта. Это имя интерфейса генерируется для большинства Ethernet-плат.

### tr0, tr1, trn

Карты сети Token Ring. Имена используются большинством карт Token Ring, включая не-IBM карты.

### sl0, sl1, sln

n-ый SLIP-интерфейс. Первая последовательная линия, отконфигурированная под SLIP становится sl0, и т.д.

**ppp0, ppp1, pppn**

n-ый PPP-интерфейс. Подобно SLIP-интерфейсам, PPP-интерфейс связан с последовательной линией, если только она отконфигурирована для PPP.

**plip0, plip1, plipn**

n-ый PLIP-интерфейс. PLIP-транспортирует IP-пакеты по параллельным линиям. Они устанавливаются PLIP-драйвером при загрузке системы и отображают параллельные порты. В ядрах 2.0.x имеется прямая связь между именем устройства и адресом ввода-вывода параллельного порта, но в более поздних ядрах имена устройства распределены последовательно точно, как для SLIP и PPP.

**ax0, ax1, axn**

n-ый интерфейс AX.25. AX.25 является главным протоколом операторов amateur radio. Интерфейсы AX.25 распределяются аналогично интерфейсам SLIP.

Есть еще немало разных интерфейсов для сетевых драйверов. Я перечислил лишь самые распространенные.

В следующих разделах мы будем обсуждать детали использования драйверов, описанных выше. Много полезного по настройке сетевых устройств можно узнать в Networking HOWTO, а AX25 HOWTO подробно описывает настройку сетевых устройств Amateur Radio.

## Установка Ethernet

Сейчас Linux поддерживает различные модели Ethernet карт. Больше всего драйверов было написано Donald Becker (becker@super.org). Он автор семейства драйверов для карт основанных на чипе National Semiconductor 8390. Они стали известными как серия Becker-драйверов. Есть также драйверы для пары изделий D-Link, среди них адаптер D-Link, который предлагает доступ к Ethernet через параллельный порт. Этот драйвер был написан Bjrn Ekwall (bj0rn@blox.se). DEPCA драйвер был написан David C. Davies (davies@wanton.lkg.dec.com). Список поддерживаемых карт постоянно растет, так что Ваша карта почти наверняка поддерживается.

Когда-то раньше в руководствах по Linux пытались перечислить все допустимые карты. Сейчас такой список займет слишком много места. К счастью, Paul Gortmaker поддерживает Ethernet HOWTO, в котором есть список всех карт и информация о том, как заставить их работать в Linux. Документ рассылается в группу новостей comp.os.linux.answers и доступен на зеркалах Linux Documentation Project.

Даже если Вы уверены, что знаете, как установить специфический тип платы Ethernet на своей машине, все равно стоит ознакомиться с Ethernet HOWTO на предмет возможных хитростей. Это поможет сэкономить немало сил для DMA-карт Ethernet, которые используют тот же канал DMA, что и контроллер Adaptec 1542 SCSI. Если их не разделить, Вы получите карту Ethernet, записывающую данные из ip-пакетов в произвольное место на Вашем жестком диске.

Чтобы использовать любую из поддерживаемых плат Ethernet с Linux, Вы можете использовать готовое ядро одного из дистрибутивов Linux. Они вообще имеют модули для всех поддерживаемых драйверов, и процесс установки современных дистрибутивов вообще лишен раздела выбора драйвера для вашей сетевой карты так как это происходит автоматически. Однако, лучше формировать собственное ядро и компилировать только те драйверы, в которых Вы фактически нуждаетесь: это сохраняет дисковое пространство и память.

## Автоопределение Ethernet

Многие из драйверов Linux Ethernet достаточно интеллектуальны, чтобы самостоятельно определить параметры платы Ethernet. Это избавляет от необходимости сообщать такие данные ядру вручную. Ethernet HOWTO перечисляет, использует ли специфический драйвер автопоиск, и в каком порядке он ищет адрес ввода-вывода платы.

Имеются три ограничения на автопоиск. Во-первых, ядро не может распознавать все платы правильно. Это особенно верно для некоторых дешевых аналогов популярных плат. Во-вторых, ядро не будет искать больше, чем одну плату, если нет дополнительных указаний об этом. Это было сознательным решением проекта, поскольку считается что Вы захотите иметь контроль над тем, какие настройки какому интерфейсу предназначены. Самый лучший способ получения надежной системы: вручную конфигурировать платы Ethernet. В-третьих, драйвер избегает некоторых адресов, чтобы не конфликтовать с другим оборудованием. Так что еще не факт, что адрес платы вообще будет проверен.

PCI-платы должны быть опознаны правильно, но все равно Вы имеете способ инструктировать ядро относительно марки и адреса карты, если используете больше одной платы, или автопоиск потерпел неудачу.

Во время загрузки можно передавать параметры и информацию ядру, которую любой из ядерных компонентов может читать. Этот механизм позволяет Вам передавать ядру информацию, которую драйверы Ethernet могут использовать без проведения автопоиска.

Если Вы используете lilo для загрузки, параметры сетевых интерфейсов можно передать, указав их через аргумент `append` в файле `lilo.conf`. Например, можно передать ядру следующие параметры, чтобы проинструктировать его о плате Ethernet:

```
ether=irq, base_addr, [param1,][param2,] name
```

Первые четыре параметра задают числа, последний задает имя устройства. Параметры `irq`, `base_addr` и `name` обязательны, но два `param` опциональны. Числовые значения могут быть установлены в ноль, в этом случае ядро будет их искать, исходя из имеющейся информации.

Первый параметр задает IRQ. По умолчанию, ядро попытается его найти само. Например, драйвер 3c503 имеет специальное свойство, из-за которого ядро выбирает свободный IRQ из 5, 9, 3, 4 и настраивает плату, чтобы использовать эту линию. Параметр `base_addr` задает адрес ввода-вывода карты, значение 0 сообщает, чтобы ядро исследовало возможные адреса карты.

Различные драйверы используют следующие два параметра по-разному. Для карт с разделением памяти (`shared-memory cards`), подобных WD80x3, они определяют первый и последний адреса разделяемой памяти. Другие платы обычно используют `param1`, чтобы установить уровень выдачи отладочной информации. Значения от 1 до 7 задают повышенные уровни отладки, 8 выключает ее, 0 выдает нормальное число сообщений. Драйвер 3c503 использует `param2`, чтобы выбрать между своим трансивером (по умолчанию) и внешним (значение 1). Аргументы `param` не должны быть включены вообще, если нет ничего особо нуждающегося в настройке.

Первый нечисловой параметр интерпретируется ядром как имя устройства. Вы должны определить имя устройства для каждой платы Ethernet.

Если у Вас две платы Ethernet cards, можно поручить определение первой ядру, а самому настроить вторую через lilo. Можно и вручную конфигурировать обе платы. Если Вы решите поручить ядру поиск первой платы, надо выяснить, какую плату считаете первой Вы, а какую ядро. Ваши мнения совпадут не всегда. Решить проблему позволяет опция `lilo reserve`, которая однозначно запретит ядру поиск в адресном пространстве первой карты. Например, чтобы Linux нашла вторую карту Ethernet на адресе 0x300 с именем `eth1` задайте параметры ядра:

```
reserve=0x300,32 ether=0,0x300,eth1
```

Опция `reserve` запрещает какому-либо драйверу обращаться в указанное адресное пространство. Вы можете также использовать параметры ядра, чтобы отменить автопоиск для `eth0`:

```
reserve=0x340,32 ether=0,0x340,eth0
```

Автопоиск можно и вовсе запретить (допустим, Вы временно сняли одну из плат Ethernet). Чтобы запретить автопоиск задайте аргумент `base_addr` как -1:

```
ether=0,-1,eth0
```

Для передачи этих параметров ядру при загрузке Вы их вводите в ответ на приглашение lilo "boot:". Чтобы lilo выдал запрос "boot:", надо при загрузке держать нажатой одну из клавиш Control, Alt или Shift. Если в ответ на приглашение нажать Tab, будет выведен список ядер, которые можно загрузить. Для загрузки ядра с параметрами введите его имя, пробел и все нужные параметры. Затем нажмите клавишу Enter, и lilo загрузит заданное ядро с заданными параметрами.

Чтобы параметры воспринимались при каждой перезагрузке автоматически, внесите их в `/etc/lilo.conf`, используя ключевое слово `append=`. Например:

```
boot=/dev/hda
root=/dev/hda2
install=/boot/boot.b
map=/boot/map
vga=normal
delay=20
append="ether=10,300,eth0"
image=/boot/vmlinuz-2.2.14
label=2.2.14
read-only
```

После правки `lilo.conf`, выполните команду `lilo` для активизации изменений.



## Другие типы сетей

Большинство других типов сетей настраиваются аналогично Ethernet. Параметры, переданные загружаемым модулям, будут другими, и некоторые драйверы не могут поддерживать больше одной платы, но все остальное примерно так же. Документация для этих плат доступна в подкаталоге `/usr/src/linux/Documentation/networking` исходных текстов ядра Linux.

## Драйверы сетевых устройств в ядре

Естественно, что для того, чтобы работать с локальной сетью, необходимо иметь сетевую карту (плату) и подключение к сети. Я не буду объяснять, как физически подключиться к сети, предполагая, что вы уже установили сетевую карту в ваш компьютер и соединили его (кабелем или витой парой) с существующей сетью.

Заметим, что в каталоге `/dev` нет специального файла для сетевой карты. В Linux сетевые устройства создаются динамически, и поэтому не требуют наличия соответствующих файлов в каталоге `/dev`.

Прежде, чем пытаться подключаться к сети, вы должны убедиться, что установленное в вашей системе ядро скомпилировано с поддержкой сетевых возможностей. Некоторые системные администраторы утверждают, что признаком этого является наличие каталога `/proc/net`. Но можно посмотреть и протокол загрузки системы (файл `/var/log/dmesg`), в котором должны найтись примерно такие строки:

```
NET: Linux TCP/IP 1.0 for NET4.0
IP Protocols: ICMP, UDP, TCP, IGMP
IP: routing cache hash table of 512 buckets, 4Kbytes
TCP: Hash tables configured (established 8192 bind 8192)
NET: Unix domain sockets 1.0/SMP for Linux NET4.0.
NET: Ethernet Bridge 008 for NET4.0
```

Далее нужно убедиться, что в состав ядра включен драйвер для вашей сетевой карты. Вообще-то, ядра, включаемые в стандартные дистрибутивы, обеспечивают поддержку большинства распространенных сетевых плат (что, конечно, увеличивает объем ядра). Так что с очень высокой вероятностью нужный драйвер входит в ядро. В процессе загрузки ядра выполняется процедура автоматического обнаружения сетевой карты. Если такое обнаружение было успешным, то в файле `/var/log/dmesg` вы найдете соответствующие сообщения. У меня они имеют вид:

```
eth0: registered as PCnet/FAST III 79C973.
eth0: link up, 100Mbps, full-duplex
eth0: no IPv6 routers present
```

Если карта не обнаружена, то вам придется перекомпилировать ядро (или поменять карту). Перекомпиляция ядра может иметь смысл и в том случае, если вы хотите удалить из ядра ненужные драйверы устройств, которые вы не используете. Но все же в большинстве случаев стандартное ядро успешно решает задачи поддержки сетевых возможностей, так что на вопросе перекомпиляции ядра сейчас мы останавливаться не будем. Рассмотрим только вопрос о динамическом подключении драйвера сетевой карты.

## Динамическое подключение драйверов

Для динамического подключения драйвера надо подгрузить модуль ядра, отвечающий за взаимодействие с данным сетевым устройством (например, сетевой картой) и передать ему параметры устройства. Сделать это можно с помощью команды `insmod`, вызов которой осуществляется следующим образом:

```
insmod [ filename ] [ module options ... ]
```

**Внимание!** В последних версиях Linux вместо `insmod` используется `modprobe`.

Ядро именует драйверы Ethernet как `eth0`, `eth1` и т. д., так что для подключения, например, второй сетевой карты надо `eth0` заменить на `eth1`.

В некоторых случаях в команде требуется дополнительно задать номер порта и номер используемого прерывания, а также некоторые другие опции, но подробнее об этом см. на странице `man insmod`.

Для просмотра уже загруженных модулей служит программа `lsmod`. Вот что я вижу в своей системе:

```
# lsmod
Module                Size  Used by
ipv6                  220580  10
fuse                  40468   2
```

```
agpgart          26416  0
lp               9156  0
parport_pc      23588  0
parport         29896  2 lp,parport_pc
psmouse         36112  0
serio_raw       4996   0
evdev           8576   0
ac              4100   0
thermal         12572  0
button          6032   0
processor       21680  1 thermal
i2c_piix4       7052   0
pcnet32         29700  0
mii             4480   1 pcnet32
#
```

Для того, что бы получить подробную информацию о загруженном модуле, необходимо воспользоваться программой modinfo:

**modinfo [-O][-F field][-k kernelversion] module...**

Вот что я увидел, попросив информацию о модуле pcnet32.ko:

```
# modinfo pcnet32
filename:        /lib/modules/2.6.24.5/kernel/drivers/net/pcnet32.ko
license:         GPL
description:     Driver for PCnet32 and PCnetPCI based ethercards
author:          Thomas Bogendoerfer
alias:           pci:v00001023d00002000sv*sd*bc02sc00i*
alias:           pci:v00001022d00002000sv*sd*bc*sc*i*
alias:           pci:v00001022d00002001sv*sd*bc*sc*i*
depends:          mii
vermagic:        2.6.24.5 mod_unload 486
parm:            debug:pcnet32 debug level (int)
parm:            max_interrupt_work:pcnet32 maximum events handled per interrupt
(int)
parm:            rx_copybreak:pcnet32 copy breakpoint for copy-only-tiny-frames
(int)
parm:            tx_start_pt:pcnet32 transmit start point (0-3) (int)
parm:            pcnet32vlb:pcnet32 Vesa local bus (VLB) support (0/1) (int)
parm:            options:pcnet32 initial option setting(s) (0-15) (array of int)
parm:            full_duplex:pcnet32 full duplex setting(s) (1) (array of int)
parm:            homepna:pcnet32 mode for 79C978 cards (1 for HomePNA, 0 for
Ethernet, default Ethernet (array of int)
#
```

Обратите внимание на строку

```
depends:          mii
```

Она сообщает нам, что модуль pcnet32.ko зависит от модуля mii.ko. Ту же информацию мы видели в выводе программы lsmod в строке

```
mii             4480   1 pcnet32
```

Только здесь указаны модули, которые зависят от этого модуля.

Зависимость модулей друг от друга означает, что модуль pcnet32.ko не сможет быть загружен командой insmod до тех пор, пока не будет загружен модуль mii.ko. И наоборот, модуль mii.ko не сможет быть выгружен, пока не будут выгружены из памяти модули, зависящие от него. В нашем случае, пока не будет выгружен модуль pcnet32.ko

Для выгрузки модулей ядра используется программа rmmod:

**rmmod [-fhswwV] modulename ...**

Опции:

- -f (or --force) принудительная выгрузка модуля ядра, может привести к краху системы. Требуется, что бы

ядро было скомпилировано с опцией - Forced Module Removal

- -h (or --help) краткая информация об использовании программы
- -s (or --syslog) использование системы syslog, ошибки не будут выводиться на stderr
- -v (or --verbose) включает опцию подробных сообщений
- -V (or --version) сообщает версию программного кода
- -w (or --wait) включается функция ожидания, если модуль ядра занят в данный момент

Давайте рассмотрим зависимости на примере указанных модулей:

Выгрузим модуль mii.ko

```
# rmmod mii
ERROR: Module mii is in use by pcnet32
#
```

Как видим – это не возможно без выгрузки модуля pcnet32.ko. Выгрузим последовательно модули pcnet32.ko и mii.ko

```
# rmmod pcnet32
# rmmod mii
# lsmod
Module                Size  Used by
ipv6                   220580  10
fuse                   40468   2
agpgart                26416   0
lp                     9156   0
parport_pc             23588   0
parport               29896   2 lp,parport_pc
psmouse               36112   0
serio_raw              4996   0
evdev                  8576   0
ac                     4100   0
thermal               12572   0
button                 6032   0
processor              21680   1 thermal
i2c_piix4              7052   0
#
```

Как видим модули ядра выгрузились. Теперь загрузим модули pcnet32.ko и mii.ko используя программу modprobe. Преимущество modprobe перед insmod заключается в том, что в общем случае ей не надо указывать ни каких дополнительных параметров, в том числе и полного пути к модулю ядра. Так же modprobe автоматически подгружает все модули, которые определяют зависимости. Подробную информацию по этой программе смотрите в документации – man modprobe.

```
# modprobe pcnet32
# lsmod
Module                Size  Used by
pcnet32               29700   0
mii                   4480   1 pcnet32
ipv6                   220580  10
fuse                   40468   2
agpgart                26416   0
lp                     9156   0
parport_pc             23588   0
parport               29896   2 lp,parport_pc
psmouse               36112   0
serio_raw              4996   0
evdev                  8576   0
ac                     4100   0
thermal               12572   0
button                 6032   0
processor              21680   1 thermal
i2c_piix4              7052   0
```

Как видите, были подгружены оба модуля.

### Загрузка модулей сетевых карт в Slackware Linux

Для того, что бы модули сетевых карт автоматически подгружались в процессе загрузки системы необходимо явно прописать вызов программы `modprobe` с указанием имени подгружаемого модуля в файле `/etc/rc.d/rc.netdevices`. Если вы еще не выполняли такую процедуру, то этого файла у вас нет. Его нужно создать и сделать исполняемым. Рекомендуемые права для таких файлов – 700.

Файл должен выглядеть примерно так:

```
#!/bin/bash
/sbin/modprobe pcnet32
```

## Определение HOSTNAME

Большинство сетевых приложений предполагает, что Вы установите имя компьютера в какое-то приемлемое значение. Эта установка обычно бывает сделана в течение процедуры начальной загрузки, выполняя команду `hostname`. Для установки имени компьютера в `name` введите:

```
# hostname name
```

Обычной практикой является использование локального имени (без имени домена). Например, машины Virtual Brewery могут иметь имена `vale.vbrew.com` и `vlager.vbrew.com`. Эти имена полные, то есть с именем домена (fully qualified domain names, FQDNs). Их локальные имена будут первыми компонентами полных, например, `vale`. Однако, поскольку локальные имена часто используется, чтобы искать IP-адрес компьютера, Вы должны удостовериться, что сервер имен способен найти адрес по имени. Это обычно означает, что Вы должны внести имя в файл `/etc/hosts`.

Некоторые люди предлагают использовать команду `domainname`, чтобы сообщить ядру оставшуюся часть FQDN. Этим путем Вы могли бы объединять вывод из `hostname` и `domainname` для получения FQDN. Однако, это в самом лучшем случае только наполовину верно. `domainname` вообще используется, чтобы установить домен NIS, который может полностью отличаться от домена DNS, к которому компьютер принадлежит. Вместо этого, чтобы гарантировать, что короткая форма пригодна к использованию в поиске, все новые версии `hostname` добавляют соответствующую запись в сервер имен или пишут полное доменное имя в файл `/etc/hosts`. Затем можно использовать параметр `--fqdn` в команде `hostname` для получения полного имени.

## Получение сетевого адреса

Если Вы конфигурируете программное обеспечение на Вашей системе для автономной работы (например, чтобы быть способным выполнить INN Netnews), Вы можете безопасно пропустить этот раздел, потому что единственный адрес IP, в котором Вы будете нуждаться, это адрес для кольцевого (loopback) интерфейса, который всегда 127.0.0.1.

Дела немного более сложны с реальными сетями, подобными Ethernet. Если Вы хотите подключить Ваш компьютер к существующей сети, Вы должны просить, чтобы администраторы дали Вам IP-адрес в этой сети. Если сеть строите сами, можете назначать IP-адреса лично.

Компьютеры в локальной сети обычно должны совместно использовать адреса из той же самой логической IP-сети. Следовательно, Вы должны назначить сетевой адрес IP. Если Вы имеете несколько физических сетей, Вы должны назначить им различные сетевые адреса или использовать подсети, чтобы разделить адресный интервал IP на несколько подсетей. Последний способ подробно рассмотрен ниже в разделе "Создание подсетей".

При выборе сетевого IP-адреса многое зависит от того, предполагаете ли Вы выходить в Internet в ближайшем будущем. Если да, получите официальный IP-адрес сейчас, чтобы потом не перенастраивать систему. Попросите, чтобы Ваш провайдер помог Вам.

Если Ваша сеть не связана с Internet и не будет связана в ближайшем будущем, Вы свободны выбрать любой допустимый сетевой адрес. Только убедитесь, что никакие пакеты не могут из Вашей сети попасть в Internet! Чтобы быть уверенным в том, что даже в случае их попадания в Internet, они не создадут проблем, используйте один из сетевых адресов, зарезервированных для частного использования. Internet Assigned Numbers Authority (IANA) отложил несколько сетевых адресов из классов A, B и C, которые Вы можете использовать без регистрации. Эти адреса имеют силу только внутри частной сети и непригодны для сетевого взаимодействия в Internet. Они определены в RFC 1597.

Выбор адресов из одной из этих подсетей не только полезен для сетей, полностью не связанных с Internet; Вы можете организовать ограниченный доступ, используя одну машину как шлюз. Для вашей локальной сети шлюз

доступен по внутреннему IP-адресу в то время, как внешний мир знает лишь официально зарегистрированный адрес (назначенный Вам провайдером). Мы вернемся к этому понятию на описании подключения с IP masquerade в курсе «Сетевое администрирование Linux».

В дальнейшем мы предполагаем, что используется сеть класса C, скажем 192.168.0.0.

Поскольку вы собираетесь устанавливать машину с Linux в уже существующую сеть, то следующим вашим шагом при подключении к сети должно стать обращение к администратору сети за получением сетевого адреса. Точнее, вы должны получить следующую информацию:

- IP адрес вашего компьютера;
- IP адрес сети;
- широковещательный IP-адрес;
- имя домена, в который будет включен ваш компьютер;
- маску подсети;
- IP адрес маршрутизатора (router);
- IP адрес сервера имен (DNS-сервера).

Заодно согласуйте с администратором сетевое имя, которое вы выбрали для своего компьютера (чтобы избежать его совпадения с именем какого-то из уже включенных в сеть компьютеров).

## Создание подсетей

Чтобы оперировать несколькими Ethernet (или другими сетями), Вы должны разбить сеть на несколько подсетей. Обратите внимание, что подсети требуются только в случае, если Вы имеете больше, чем одну широковещательную сеть (broadcast network): Point-to-point связь не учитывается. Например, если Вы имеете один Ethernet и одно или более SLIP-соединений с внешним миром, Вы не нуждаетесь в подсети.

Например, администратор сети Konstantin обращается к NIC за сетевым адресом класса B. Чтобы разместить две сети Ethernet, он решает использовать восемь битов части хоста как дополнительные подсетевые биты. Это оставляет другие восемь битов для хоста, то есть по 254 хоста в каждой подсети. Он также назначает подсеть 1 brew-net и 2 wine-net. Их сетевые адреса таким образом 172.16.1.0 и 172.16.2.0. Сетевая маска 255.255.255.0.

vlager, который является gateway между двумя сетями, получил адрес хоста, равный 1 в обоих из них, что дает ему IP-адреса 172.16.1.1 и 172.16.2.1 соответственно.

Обратите внимание, что в этом примере я использую сеть класса B для простоты: сеть класса C была бы более реалистичной. С новым сетевым кодом деление на подсети не ограничено границами байта, так что даже сеть класса C может быть разделена на несколько подсетей. Например, Вы могли бы использовать 2 бита части хоста для сетевой маски, что дает четыре возможные подсети с 64 хостами в каждой.

## Создание файлов hosts и networks

После того, как Вы разделили на подсети свою сеть, Вы должны подготовиться к простому поиску адреса по имени, использующего файл /etc/hosts. Если Вы не собираетесь использовать DNS или NIS для этого, Вы должны помещать все хосты в файл hosts.

Даже если Вы хотите использовать DNS или NIS, можно иметь некоторое подмножество имен и в /etc/hosts. Например, если Вы хотите иметь некоторый вид поиска по имени даже, когда сетевые интерфейсы не запущены, например, во время загрузки. Это не только вопрос удобства, но это также позволяет Вам использовать символические имена хостов в стартовых скриптах rc. Таким образом, при изменении IP-адресов, Вы должны будете только копировать обновленный файл hosts на все машины вместо того, чтобы редактировать большое количество файлов rc. Обычно Вы будете помещать все локальные имена и адреса в hosts добавлением их на любой gateway и NIS-сервер, если они используются.

Также при проверке Вы должны удостовериться, что сервер имен использует информацию только из файла hosts. Программное обеспечение DNS или NIS может иметь файлы примеров, которые могут дать странные результаты при их использовании. Чтобы заставить все приложения использовать исключительно /etc/hosts при поиске IP-адреса хоста, Вы должны отредактировать файл /etc/host.conf. Закомментируйте все строки, начинающиеся с ключевого слова order и вставьте строку: order hosts

Конфигурация библиотеки сервера имен будет подробно описана в курсе «Сетевое администрирование Linux».

Файл hosts содержит по одной записи на строку, состоящую из IP-адреса, имени хоста и необязательного списка

псевдонимов. Поля отделяются пробелами или табуляцией, поле адреса должно начинаться в первой колонке. Все, что следует после символа #, расценивается как комментарий и игнорируется.

Имя хоста может быть полностью квалифицированным или заданным относительно локального домена. Для server Вы ввели бы в hosts полностью квалифицированное имя, server.unix.class, а также коротко server, чтобы было известно и официальное имя и более короткое локальное.

Пример файла hosts для Virtual Brewery дан ниже. Два специальных имени, vlager-if1 и vlager-if2, задают адреса для обоих интерфейсов, используемых на vlager:

```
#
# Hosts file for Virtual Brewery/Virtual Winery
#
# IP                FQDN                aliases
#
127.0.0.1           localhost
#
172.16.1.1          vlager.vbrew.com      vlager vlager-if1
172.16.1.2          vstout.vbrew.com      vstout
172.16.1.3          vale.vbrew.com        vale
#
172.16.2.1          vlager-if2
172.16.2.2          vbeaujolaais.vbrew.com vbeaujolaais
172.16.2.3          vbardolino.vbrew.com  vbardolino
172.16.2.4          vchianti.vbrew.com    vchianti
```

Точно так же, как с IP-адресами хостов, можно дать символическое имя подсетям. Поэтому файл hosts имеет компаньона /etc/networks, который отображает имя сети на сетевой адрес и наоборот. В Virtual Brewery мы могли бы устанавливать файл networks подобно этому:

```
# /etc/networks for the Virtual Brewery
brew-net      172.16.1.0
wine-net      172.16.2.0
```

## Настройка сетевых интерфейсов

Интерфейсом с точки зрения ОС является устройство, через которое система получает и передает IP-пакеты. Роль интерфейса локальной сети может выполнять одно (или несколько) из следующих устройств: Ethernet-карта, ISDN-адаптер или модем, подключенный к последовательному порту. Каждое устройство (не весь компьютер!) имеет свой IP-адрес. Для выхода в локальные сети используется, как правило, Ethernet-карта, что и будет рассматриваться в настоящем разделе. Заодно мы рассмотрим и настройку модемного интерфейса, поскольку настраивается он вполне аналогично, и знания о методах его настройки будут необходимы нам при рассмотрении вопроса о выходе в Интернет.

## Расположение конфигурационных файлов

Отметим сразу, что все приводимые ниже команды можно выполнять из командной строки, но тогда придется повторять эти операции при каждом перезапуске компьютера. Поэтому может быть удобнее записать их в один из инициализационных файлов, автоматически запускаемых при старте системы. Как уже рассматривалось ранее, в разных дистрибутивах процесс загрузки организован по-разному. В "Linux NET-3-HOWTO" приводится следующая таблица:

Дистрибутив	Настройка интерфейса и маршрутизации	Запуск демонов
Debian	/etc/init.d/network	/etc/init.d/netbase /etc/init.d/netstd_init /etc/init.d/netstd_nfs /etc/init.d/netstd_misc
Slackware	/etc/rc.d/rc.inet1	/etc/rc.d/rc.inet2
RedHat	/etc/sysconfig/network-scripts/ifup-<ifname>	/etc/rc.d/init.d/network

## Программа *ifconfig*

После подключения драйверов вы должны настроить те интерфейсы, которые вы предполагаете использовать. Настройка интерфейса заключается в присвоении IP-адресов сетевому устройству и установке нужных значений для других параметров сетевого подключения. Наиболее часто для этого используется программа *ifconfig* (ее название происходит от "interface configuration").

Запустите ее без аргументов (или с единственным аргументом *-a*) и вы узнаете, какие параметры установлены в данный момент для активных сетевых интерфейсов (в частности, для сетевой карты). Кстати, имеет смысл выполнить эту команду еще до подключения модулей: а вдруг у вас поддержка интерфейсов встроена в ядро и необходимые настройки сделаны в процессе инсталляции системы. Тогда вы в ответ можете получить информацию о параметрах вашей Ethernet-карты и так называемого "кольцевого интерфейса" или "обратной петли" - Local Loopback (интерфейс Ethernet при единственной сетевой карте обозначается как *eth0*, а кольцевой интерфейс - как *lo*). Если же по этой команде вы ничего не получите, то надо переходить к подключению модулей и настройке, и начинать надо с интерфейса обратной петли.

### Все про *ifconfig*

Имеются еще несколько параметров для *ifconfig*. Вот полное описание:

*ifconfig interface [address [parameters]]*

*interface* имя интерфейса и *address* IP-адрес, который требуется назначить интерфейсу. Это может быть IP-адрес или имя, которое *ifconfig* будет искать в файле */etc/hosts*.

Если *ifconfig* используется только с именем интерфейса, он показывает конфигурацию этого интерфейса. Когда он вызывается без параметров, он показывает все интерфейсы, которые Вы отконфигурировали; опция *-a* вынуждает его показать и бездействующие. Вывод для Ethernet-интерфейса *eth0* может напоминать это:

```
# ifconfig eth0
eth0      Link encap 10Mbps Ethernet  HWaddr 00:00:C0:90:B3:42
          inet addr 172.16.1.2 Bcast 172.16.1.255 Mask 255.255.255.0
          UP BROADCAST RUNNING MTU 1500 Metric 0
          RX packets 3136 errors 217 dropped 7 overrun 26
          TX packets 1752 errors 25 dropped 0 overrun 0
#
```

Поля *MTU* и *Metric* показывают текущее *MTU* и метрическое значение для этого интерфейса. Метрическое значение традиционно используется некоторыми операционными системами, чтобы вычислить сложность маршрута. Linux не использует это значение, но определяет его для совместимости.

Строки *RX* и *TX* показывают, сколько пакетов были получены или переданы без ошибок, сколько произошло ошибок, сколько пакетов были потеряны, вероятно, из-за нехватки памяти и сколько были потеряны из-за переполнения. Переполнение приемника обычно случается, когда пакеты ходят быстрее, чем ядро может их обслужить. Значения флагов, выводимые *ifconfig*, передают дополнительную информацию об именах и опциях командной строки, они будут объяснены ниже.

Следующий список параметров используется *ifconfig* с соответствующими названиями флагов, заданными в скобках. Опция, которая просто включает некоторую возможность, также позволяет ее выключать, если названию опции предшествует тире (-).

#### **up**

Эта опция делает интерфейс доступным для IP-уровня. Она подразумевается, когда задается IP-адрес. Также используется для перезапуска интерфейса, который временно выключен опцией *down*.

Соответствует флагам *UP* и *RUNNING*.

#### **down**

Она делает интерфейс недоступным IP-уровню. Эффективно отключает любой IP-трафик через интерфейс. Обратите внимание, что она не удаляет все маршрутизационные записи, которые используют этот интерфейс. Если Вы постоянно выключаете некий интерфейс, Вы должны удалить эти записи сами и предоставить, если возможно, альтернативные маршруты.

#### **netmask mask**

Назначает маску подсети для использования интерфейсом. Здесь можно задавать как любое шестнадцатеричное число с 32 битами, которому предшествует *0x*, так и десятичные числа, разделенные точками.

#### **pointopoint address**

Эта опция используется для point-to-point IP-соединений. Необходима, чтобы отконфигурировать, например, SLIP или PLIP интерфейсы. Если адрес для point-to-point был установлен, `ifconfig` показывает флаг `POINTOPOINT`.

### **broadcast address**

Широковещательный адрес обычно создается из сетевого адреса установкой всех бит части машины. Некоторые реализации IP используют другую схему, эта опция помогает приспособиться к этим странным средам. Если широковещательный (broadcast) адрес был установлен, `ifconfig` показывает флаг `BROADCAST`.

### **irq**

Задаёт номер IRQ для устройства. Обычно используется для PLIP, но может пригодиться и для некоторых карт Ethernet.

### **metric number**

Эта опция может использоваться для назначения метрического значения записи таблицы маршрутизации, созданной для интерфейса. Эта метрика используется в RIP для построения таблиц маршрутизации. Установленное по умолчанию значение равно нулю. Если Вы не используете RIP-демон, Вы не нуждаетесь в этой опции вообще; если используете, Вы редко должны будете изменять это значение.

### **mtu bytes**

Эта опция устанавливает Maximum Transmission Unit (максимальную длину передаваемого пакета). Для Ethernet MTU по умолчанию 1500, для SLIP интерфейсов 296.

### **arp**

Эта опция определена для широковещательных сетей типа пакетного радио или Ethernet. Она позволяет использовать ARP, протокол поиска адреса, используемый для определения физического адреса машины, включенной в сеть. Для широковещательных сетей, включено по умолчанию. Если ARP выключен, `ifconfig` отобразит флаг `NOARP`.

### **-arp**

Запрещает использование ARP на этом интерфейсе.

### **promisc**

Помещает интерфейс в состояние promiscuous. В широковещательной сети это заставляет интерфейс получать все пакеты независимо от того, были ли они предназначены для этой машины или нет. Это позволяет, используя фильтры пакетов, анализировать сетевой трафик. Обычно, это хорошая техника охоты на сетевые проблемы, которые иначе трудно отловить. Здесь весьма полезна утилита `tcpdump`. С другой стороны, это позволяет хакерам исследовать движение паролей по сети и делать другие черные дела. Одна защита против этого типа нападения: не позволять присоединяться к сети чужим компьютерам. Другой способ: использовать безопасные опознавательные протоколы, типа Kerberos, или SRA login. Эта опция соответствует флагу `PROMISC`.

### **-promisc**

Запрещает режим promiscuous.

### **allmulti**

Multicast-адреса представляют собой особый подвид широковещательных адресов позволяющих обращаться к группе машин, которые не обязательно должны быть в той же самой подсети. Они весьма полезны при сетевых голосовых переговорах и видеоконференциях. Поддерживаются многими, но не всеми картами Ethernet. Эта опция соответствует флагу `ALLMULTI`.

### **-allmulti**

Запрещает multicast-адреса.

## **Настройка интерфейса lo**

Этот интерфейс используется для связи программ IP-клиентов с IP-серверами, запущенными на той же машине, так что его необходимо настроить даже в том случае, если вы вообще не подключаете никаких сетевых устройств.

Локальный интерфейс настраивается очень просто:

```
# ifconfig lo 127.0.0.1
```



Теперь, чтобы проверить работоспособность протоколов TCP/IP на вашей машине, дайте команду:

```
# ping 127.0.0.1
PING 127.0.0.1 (127.0.0.1) 56(84) bytes of data.
64 bytes from 127.0.0.1: icmp_seq=1 ttl=64 time=1.03 ms
64 bytes from 127.0.0.1: icmp_seq=2 ttl=64 time=0.126 ms
64 bytes from 127.0.0.1: icmp_seq=3 ttl=64 time=0.168 ms

--- 127.0.0.1 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 1998ms
rtt min/avg/max/mdev = 0.126/0.443/1.036/0.419 ms
#
```

## Настройка Ethernet интерфейса eth0

Для того чтобы ваш компьютер вошел в сеть с IP-адресом, полученным вами у администратора (пусть для примера это будет адрес 192.168.0.15), вы должны запустить команду `ifconfig` примерно следующим образом:

```
# ifconfig eth0 192.168.0.15 netmask 255.255.255.0 up
```

Если не указывать явно маску, то она будет выставлена по-умолчанию. Для `ifconfig` эта маска устанавливается стандартом по IP (в зависимости от адреса может быть 8, 16 или 24)

В некоторых случаях необходимо бывает изменить адрес прерывания, используемого сетевой картой, порта ввода-вывода или типа соединения, используемого в сети. Это можно сделать, выполнив следующую команду:

```
# ifconfig eth0 irq 5 io_addr 220 media 10baseT
```

Не все устройства (платы) поддерживают динамическое изменение этих параметров (т. е. может потребоваться переустановить переключатели на плате).

## Использование нескольких IP адресов на одном интерфейсе

В некоторых случаях, возникает необходимость использования нескольких IP адресов на одном интерфейсе. ОС MS Windows позволяет на одном сетевом интерфейсе использовать до 4 сетевых адресов. В то время как в unix-like системах есть возможность использования до 2000 сетевых IP адресов на одном интерфейсе.

Если необходимо добавить несколько IP адресов, то к имени сетевого интерфейса нужно добавить “:0”, “:1” и т.д. Например:

```
# ifconfig eth0:0 172.16.0.1 netmask 255.255.255.0
```

Посмотрим, что у нас получилось:

```
# ifconfig eth0:0 172.16.0.1 netmask 255.255.255.0
# ifconfig
eth0      Link encap:Ethernet  HWaddr 08:00:27:95:18:35
          inet addr:192.168.0.1  Bcast:192.168.0.255  Mask:255.255.255.0
          inet6 addr: fe80::a00:27ff:fe95:1835/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:6 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:0 (0.0 B)  TX bytes:468 (468.0 B)
          Interrupt:11 Base address:0xc020

eth0:0    Link encap:Ethernet  HWaddr 08:00:27:95:18:35
          inet addr:172.16.0.1  Bcast:172.16.0.255  Mask:255.255.255.0
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          Interrupt:11 Base address:0xc020

eth1      Link encap:Ethernet  HWaddr 08:00:27:aa:7e:b5
          inet addr:192.168.2.1  Bcast:192.168.2.255  Mask:255.255.255.0
          inet6 addr: fe80::a00:27ff:feaa:7eb5/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:1608 errors:0 dropped:0 overruns:0 frame:0
          TX packets:791 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
```

```

RX bytes:191253 (186.7 KiB)  TX bytes:174474 (170.3 KiB)
Interrupt:10 Base address:0xc060

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          inet6 addr: ::1/128 Scope:Host
          UP LOOPBACK RUNNING  MTU:16436  Metric:1
          RX packets:19 errors:0 dropped:0 overruns:0 frame:0
          TX packets:19 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:1736 (1.6 KiB)  TX bytes:1736 (1.6 KiB)

# ping 172.16.0.1
PING 172.16.0.1 (172.16.0.1) 56(84) bytes of data.
64 bytes from 172.16.0.1: icmp_seq=1 ttl=64 time=1.25 ms
64 bytes from 172.16.0.1: icmp_seq=2 ttl=64 time=0.110 ms
64 bytes from 172.16.0.1: icmp_seq=3 ttl=64 time=0.141 ms

--- 172.16.0.1 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 1999ms
rtt min/avg/max/mdev = 0.110/0.503/1.258/0.534 ms

Удалить такой адрес можно выключением соответствующего псевдоинтерфейса:

# ifconfig eth0:0 down

# ifconfig
eth0      Link encap:Ethernet  HWaddr 08:00:27:95:18:35
          inet addr:192.168.0.1  Bcast:192.168.0.255  Mask:255.255.255.0
          inet6 addr: fe80::a00:27ff:fe95:1835/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:12 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:0 (0.0 B)  TX bytes:720 (720.0 B)
          Interrupt:11 Base address:0xc020

eth1      Link encap:Ethernet  HWaddr 08:00:27:aa:7e:b5
          inet addr:192.168.2.1  Bcast:192.168.2.255  Mask:255.255.255.0
          inet6 addr: fe80::a00:27ff:feaa:7eb5/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:1608 errors:0 dropped:0 overruns:0 frame:0
          TX packets:791 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:191253 (186.7 KiB)  TX bytes:174474 (170.3 KiB)
          Interrupt:10 Base address:0xc060

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          inet6 addr: ::1/128 Scope:Host
          UP LOOPBACK RUNNING  MTU:16436  Metric:1
          RX packets:19 errors:0 dropped:0 overruns:0 frame:0
          TX packets:19 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:1736 (1.6 KiB)  TX bytes:1736 (1.6 KiB)

```

## Интерфейс для последовательного порта

Последовательный порт используется для подключения модема, через который осуществляется соединение с сетью по телефонной линии. Для настройки интерфейса этого типа тоже можно использовать программу `ifconfig`. Однако, такие программы как `rppd` и `dip`, используемые для соединения с сетью по модему, способны автоматически конфигурировать сетевой интерфейс, поэтому обычно для этого случая применять `ifconfig` не требуется.

## Настройка маршрутизации

Правила маршрутизации определяют, куда отправлять IP-пакеты. Данные маршрутизации хранятся в одной из таблиц ядра. Вести таблицы маршрутизации можно статически или динамически. Статический маршрут — это маршрут, который задается явно с помощью команды `route`. Динамическая маршрутизация выполняется процессом-демоном (`routed` или `gated`), который ведет и модифицирует таблицу маршрутизации на основе сообщений от других компьютеров сети. Для выполнения динамической маршрутизации разработаны специальные протоколы: RIP, OSPF, IGRP, EGP, BGP и т. д.

Динамическая маршрутизация необходима в том случае, если у вас сложная, постоянно меняющаяся структура сети и одна и та же машина может быть доступна по различным интерфейсам (например, через разные Ethernet или SLIP интерфейсы). Маршруты, заданные статически, обычно не меняются, даже если используется динамическая маршрутизация.

Для персонального компьютера, подключаемого к локальной сети, в большинстве ситуаций бывает достаточно статической маршрутизации командой `route`. Прежде чем пытаться настраивать маршруты, просмотрите таблицу маршрутизации ядра. `Route` печатает всю таблицу маршрутизации, если его вызвать без аргументов (опция `-n` указывает печатать вместо адресов имена машин):

```
# route -n
Kernel IP routing table
Destination      Gateway          Genmask         Flags Metric Ref    Use Iface
192.168.2.0       0.0.0.0          255.255.255.0   U        0      0        0 eth1
192.168.0.0       0.0.0.0          255.255.255.0   U        0      0        0 eth0
127.0.0.0         0.0.0.0          255.0.0.0       U        0      0        0 lo
0.0.0.0           192.168.0.100    0.0.0.0         UG       0      0        0 eth0
#
```

Если таблица пуста, то вы увидите только заголовки столбцов. Тогда надо использовать `route`. С помощью команды `route` можно добавить или удалить один (за один раз) статический маршрут. Вот ее формат:

`route операция [-тип] адресат шлюз [dev] интерфейс`

Здесь аргумент *операция* может принимать одно из двух значений: `add` (маршрут добавляется) или `delete` (маршрут удаляется). Аргумент *адресат* может быть IP-адресом машины, IP-адресом сети или ключевым словом `default`. Аргумент *шлюз* — это IP-адрес компьютера, на который следует пересылать пакет (этот компьютер должен иметь прямую связь с вашим компьютером).

Как правило, бывает необходимо настроить маршрутизацию по упоминавшимся выше трем интерфейсам:

- локальный интерфейс (`lo`),
- интерфейс для платы Ethernet (`eth0`),
- интерфейс для последовательного порта (PPP или SLIP).

Локальный интерфейс поддерживает сеть с IP-номером 127.0.0.0. Поэтому для маршрутизации пакетов с адресом 127.... используется команда:

```
# route add -net 127.0.0.0 netmask 255.0.0.0 lo
```

Если у вас для связи с локальной сетью используется одна плата Ethernet, и все машины находятся в этой сети (сетевая маска 255.255.255.0), то для настройки маршрутизации достаточно вызвать:

```
# route add -net 192.168.36.0 netmask 255.255.255.0 eth0
```

Если же вы имеете несколько интерфейсов, то вам надо определиться с сетевой маской и вызвать команду `route` для каждого интерфейса.

Поскольку очень часто IP-пакеты с вашего компьютера могут отправляться не в одну единственную сеть, а в разные сети (например, при просмотре разных сайтов в Интернете), то в принципе надо было бы задать очень много маршрутов. Очевидно, что сделать это было бы очень сложно, точнее просто невозможно. Поэтому решение проблемы маршрутизации пакетов перекладывают на плечи специальных компьютеров — маршрутизаторов, а на обычных компьютерах задают маршрут по умолчанию, который используется для отправки всех пакетов, не указанных явно в таблице маршрутизации. С помощью маршрута по умолчанию вы говорите ядру "а все остальное отправляй туда". Маршрут по умолчанию настраивается следующей командой:

```
# route add default gw 192.168.0.100 eth0
```

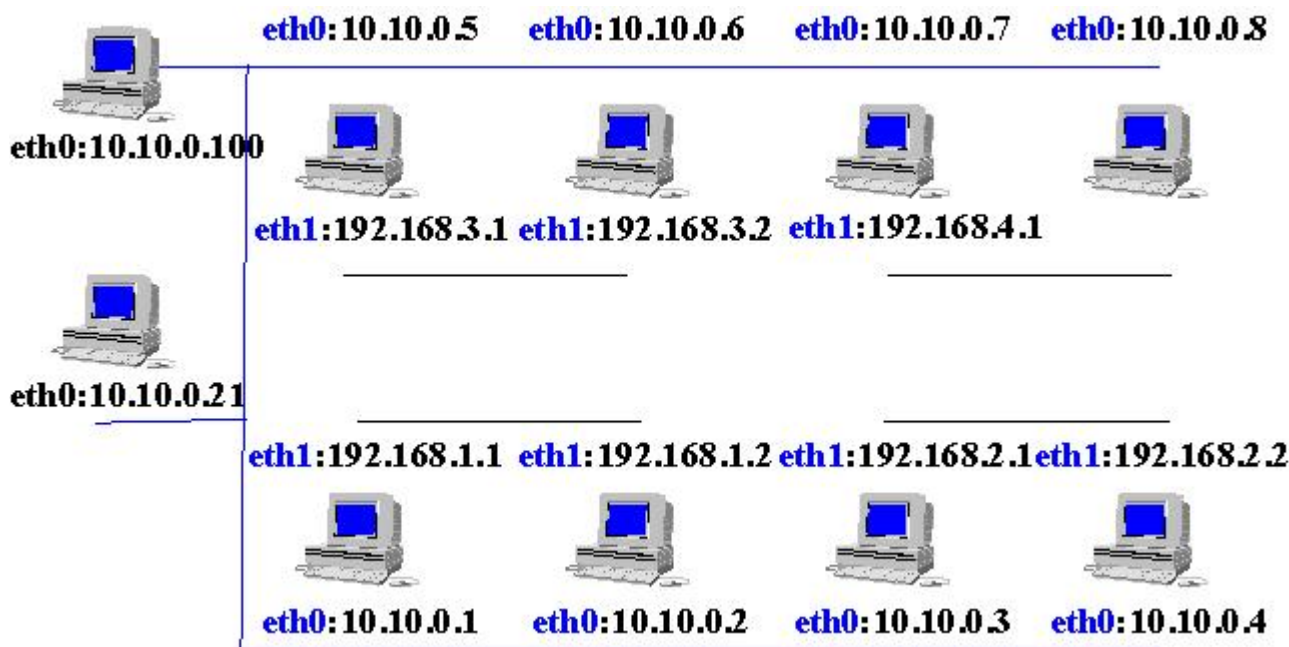
Опция `gw` указывает программе `route`, что следующий аргумент — это IP-адрес или имя маршрутизатора, на который надо отправлять все пакеты, соответствующие этой строке таблицы маршрутизации.

После настройки маршрутизации можно проверить, что у вас получилось. Для этого снова дайте команду

```
# route -n
```

## Лабораторная работа

Посмотрите схему сети класса.



Четные компьютеры будут выступать в роли рабочих станций. Нечетные – в роли роутеров.

На четных компьютерах выключаем интерфейс eth0 и добавляем маршрут по умолчанию на интерфейс eth1 роутера.

Посмотрим, какие действия нужно предпринять на примере первой пары.

Действия, которые необходимо выполнить на четном компьютере:

```
root@c2:~# ifconfig eth0 down
root@c2:~# ifconfig eth1 192.168.1.2 netmask 255.255.255.0
root@c2:~# route add default gw 192.168.1.1
root@c2:~# route -n
Kernel IP routing table
Destination      Gateway          Genmask         Flags Metric Ref    Use Iface
192.168.1.0      0.0.0.0          255.255.255.0   U        0      0        0 eth1
127.0.0.0        0.0.0.0          255.0.0.0       U        0      0        0 lo
0.0.0.0          192.168.1.1      0.0.0.0         UG       0      0        0 eth1
root@c2:~#
```

На нечетных компьютерах (роутерах) необходимо включить маршрутизацию пакетов. Выше, мы уже говорили, что главной особенностью роутера (router, маршрутизатор) является способность пересылать IP-пакеты между сетями с различной адресацией. Для того, чтобы включить возможность маршрутизации необходимо выполнить следующие действия.

Проверяем, включена ли маршрутизация с помощью виртуальной файловой системы proc. Как мы ранее уже говорили, виртуальная файловая система proc отображает текущее состояние параметров ядра.

```
root@c1:~# cat /proc/sys/net/ipv4/ip_forward
0
root@c1:~#
```

Значение «0» определяет отключенное состояние маршрутизации.

Проверяем наличие стартового скрипта rc.ip\_forward и права доступа:

```
root@c1:~# ls -l /etc/rc.d/rc.ip_forward
-rw-r--r-- 1 root root 1924 2003-09-14 04:10 /etc/rc.d/rc.ip_forward
root@c1:~#
```

Делаем скрипт исполняемым и запускаем его:

```
root@c1:~# chmod a+x /etc/rc.d/rc.ip_forward
root@c1:~# ls -l /etc/rc.d/rc.ip_forward
-rwxr-xr-x 1 root root 1924 2003-09-14 04:10 /etc/rc.d/rc.ip_forward*
root@c1:~# /etc/rc.d/rc.ip_forward
usage /etc/rc.d/rc.ip_forward start|stop|restart
root@c1:~# /etc/rc.d/rc.ip_forward start
Activating IPv4 packet forwarding.
root@c1:~#
```

Проверяем состояние параметра ip\_forward ядра:

```
root@c1:~# cat /proc/sys/net/ipv4/ip_forward
1
root@c1:~#
```

Теперь, маршрутизация будет запускаться всякий раз при загрузке системы.

Настроим таблицу маршрутизации. Пример настройки для первого компьютера (свою сеть описывать не надо):

```
root@c1:~# ifconfig eth1 192.168.1.1 netmask 255.255.255.0
root@c1:~# route -n
Kernel IP routing table
Destination      Gateway          Genmask          Flags Metric Ref    Use Iface
192.168.1.0      0.0.0.0          255.255.255.0    U        0      0        0 eth1
10.10.0.0         0.0.0.0          255.255.255.0    U        0      0        0 eth0
127.0.0.0         0.0.0.0          255.0.0.0        U        0      0        0 lo
0.0.0.0          10.10.0.100      0.0.0.0          UG       0      0        0 eth0
root@c1:~# route add -net 192.168.2.0/24 gw 10.10.0.3
root@c1:~# route add -net 192.168.3.0/24 gw 10.10.0.5
root@c1:~# route add -net 192.168.4.0/24 gw 10.10.0.7
root@c1:~# route add -net 192.168.5.0/24 gw 10.10.0.9
root@c1:~# route -n
Kernel IP routing table
Destination      Gateway          Genmask          Flags Metric Ref    Use Iface
192.168.5.0      10.10.0.9        255.255.255.0    UG       0      0        0 eth0
192.168.4.0      10.10.0.7        255.255.255.0    UG       0      0        0 eth0
192.168.3.0      10.10.0.5        255.255.255.0    UG       0      0        0 eth0
192.168.2.0      10.10.0.3        255.255.255.0    UG       0      0        0 eth0
192.168.1.0      0.0.0.0          255.255.255.0    U        0      0        0 eth1
10.10.0.0         0.0.0.0          255.255.255.0    U        0      0        0 eth0
127.0.0.0         0.0.0.0          255.0.0.0        U        0      0        0 lo
0.0.0.0          10.10.0.100      0.0.0.0          UG       0      0        0 eth0
root@c1:~#
```

Теперь, если все компьютеры настроены правильно, то каждый компьютер должен видеть любой другой компьютер в учебном классе. Проверим это:

```
# ping 192.168.3.2
PING 192.168.3.2 (192.168.3.2) 56(84) bytes of data.
64 bytes from 192.168.3.2: icmp_seq=1 ttl=126 time=41.6 ms
64 bytes from 192.168.3.2: icmp_seq=2 ttl=126 time=31.6 ms
64 bytes from 192.168.3.2: icmp_seq=3 ttl=126 time=3.21 ms
64 bytes from 192.168.3.2: icmp_seq=4 ttl=126 time=3.93 ms
64 bytes from 192.168.3.2: icmp_seq=5 ttl=126 time=7.71 ms

--- 192.168.3.2 ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 4006ms
rtt min/avg/max/mdev = 3.215/17.638/41.639/15.922 ms
root@c2:~#
```

И посмотрим, каким маршрутом пойдут пакеты между этими компьютерами:

```
root@c2:~# traceroute -n 192.168.3.2
traceroute to 192.168.3.2 (192.168.3.2), 30 hops max, 38 byte packets
 1  192.168.1.1  7.554 ms  31.171 ms  90.319 ms
 2  10.10.0.5  150.954 ms  15.406 ms  3.280 ms
 3  192.168.3.2  2.559 ms  8.184 ms  34.058 ms
root@c2:~#
```

## Конфигурационный файл rc.inet1

Для того, что бы после перезагрузки компьютера были сконфигурированы все сетевые интерфейсы и таблица маршрутизации, эти настройки должны быть определены в файле /etc/rc.d/rc.inet1

Изначально, файл rc.inet1 является сценарием оболочки, на основании которого производятся сетевые настройки в процессе инсталляции системы и последующей настройки. Однако имеет смысл самостоятельно определить все сетевые настройки.

Для дальнейшей работы в этом файле необходимо описать конфигурацию сетевых интерфейсов lo, eth0 и eth1, а также добавить в таблицу маршрутизации маршрут к сети 127.0.0.0 и маршрут по умолчанию.

**Внимание!** В конце файла должно остаться 5 пустых строк.

Вот как должен выглядеть файл rc.inet1 для первого компьютера:

```
root@c1:~# cat /etc/rc.d/rc.inet1
#!/bin/sh
/sbin/ifconfig lo 127.0.0.1
/sbin/ifconfig eth0 10.10.0.1 netmask 255.255.255.0
/sbin/ifconfig eth1 192.168.1.1 netmask 255.255.255.0
/sbin/route add -net 127.0.0.0 netmask 255.0.0.0 lo
/sbin/route add default gw 10.10.0.100
```

```
root@c1:~#
```

Теперь проверяем правильность написания файла rc.inet1:

```
root@c1:~# ifconfig eth0 down
root@c1:~# ifconfig eth1 down
root@c1:~# ifconfig lo down
root@c1:~# /etc/rc.d/rc.inet1
root@c1:~# ifconfig
eth0      Link encap:Ethernet  HWaddr 08:00:27:95:18:35
          inet addr:10.10.0.1  Bcast:10.10.0.255  Mask:255.255.255.0
          inet6 addr: fe80::a00:27ff:fe95:1835/64  Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:82 errors:0 dropped:0 overruns:0 frame:0
          TX packets:70 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:7396 (7.2 KiB)  TX bytes:5588 (5.4 KiB)
          Interrupt:11 Base address:0xc020

eth1      Link encap:Ethernet  HWaddr 08:00:27:aa:7e:b5
          inet addr:192.168.1.1  Bcast:192.168.1.255  Mask:255.255.255.0
          inet6 addr: fe80::a00:27ff:feaa:7eb5/64  Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:53 errors:0 dropped:0 overruns:0 frame:0
          TX packets:87 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:4358 (4.2 KiB)  TX bytes:7424 (7.2 KiB)
          Interrupt:10 Base address:0xc060

lo        Link encap:Local Loopback
```

```

inet addr:127.0.0.1  Mask:255.0.0.0
inet6 addr: ::1/128 Scope:Host
UP LOOPBACK RUNNING  MTU:16436  Metric:1
RX packets:4 errors:0 dropped:0 overruns:0 frame:0
TX packets:4 errors:0 dropped:0 overruns:0 carrier:0
collisions:0 txqueuelen:0
RX bytes:344 (344.0 B)  TX bytes:344 (344.0 B)

```

```

root@c1:~# route -n
Kernel IP routing table
Destination      Gateway          Genmask         Flags Metric Ref    Use Iface
192.168.1.0      0.0.0.0          255.255.255.0   U        0      0        0 eth1
10.10.0.0        0.0.0.0          255.255.255.0   U        0      0        0 eth0
127.0.0.0        0.0.0.0          255.0.0.0       U        0      0        0 lo
0.0.0.0          10.10.0.100     0.0.0.0         UG       0      0        0 eth0
root@c1:~#

```

## Программа netstat

Netstat полезный инструмент для проверки вашей сетевой конфигурации и активности. Он фактически является набором из нескольких инструментов, собранных вместе. Мы рассмотрим каждую из ее функций.

При вызове netstat с параметром -r, он показывает таблицу маршрутизации ядра, подготовленную с помощью route. На c1.unix.class он выдаст:

```

root@c1:~# netstat -r
Kernel IP routing table
Destination      Gateway          Genmask         Flags   MSS Window  irtt Iface
192.168.5.0      10.10.0.9        255.255.255.0   UG            0  0          0 eth0
192.168.4.0      10.10.0.7        255.255.255.0   UG            0  0          0 eth0
192.168.3.0      10.10.0.5        255.255.255.0   UG            0  0          0 eth0
192.168.2.0      10.10.0.3        255.255.255.0   UG            0  0          0 eth0
192.168.1.0      *                255.255.255.0   U            0  0          0 eth1
10.10.0.0        *                255.255.255.0   U            0  0          0 eth0
loopback        *                255.0.0.0       U            0  0          0 lo
default         10.10.0.100     0.0.0.0         UG            0  0          0 eth0
root@c1:~#

```

Опция -n заставляет netstat печатать IP-адреса вместо имен хостов и сетей. Это особенно полезно, когда вы хотите избежать поисков адреса по сети (например, через DNS или NIS-сервер).

Вторая колонка вывода netstat показывает маршрутизационную запись gateway. Если шлюз не используется, печатается звездочка. Третья колонка показывает сетевую маску (Genmask) маршрута. Когда дается IP-адрес, чтобы найти подходящий маршрут для него, ядро просматривает все записи таблицы маршрутизации, берет побитовое AND адреса и genmask и лишь затем сравнивает результат с целью маршрута.

Значения флагов программы netstat:

- **G** маршрут использует шлюз (gateway).
- **U** интерфейс, который нужно использовать, работает.
- **H** только отдельный хост может быть достигнут через данный маршрут (не сеть!). Например, для записи loopback 127.0.0.1.
- **D** устанавливается, если запись таблицы была произведена по приходу перенаправляемого сообщения ICMP, или если запись таблицы была создана демоном динамической маршрутизации, наподобие gated.
- **M** устанавливается, если запись таблицы была изменена перенаправляемым сообщением ICMP.
- **!** путь неверен (закрыт), все пакеты будут отброшены.

Еще три колонки показывают MSS, Window и irtt, применяемые для соединений TCP через этот маршрут. MSS (Maximum Segment Size) определяет максимальный размер пакета для этого маршрута. Window задает максимальное количество данных, которое система примет в одном пакете с удаленного компьютера. irtt означает "initial round trip time". TCP протокол гарантирует, что данные будут надежно доставлены между компьютерами, повторно передавая пакеты, если они были потеряны. При этом ведется счетчик времени:

сколько можно ждать, пока пакет дойдет до машины назначения, и оттуда придет подтверждение. Если время вышло, пакет будет послан еще раз. Этот процесс называется *round-trip time*. *initial round-trip time* задает значение, которое используется при установке подключения. Для большинства сетей подходит значение по умолчанию, но для некоторых медленных сетей (особенно ряд типов пакетного радио) время слишком короткое, что вызывает ненужные повторы. Параметр *irtt* может быть установлен, используя команду *route*. По умолчанию в этом поле ноль.

Последнее поле показывает, к какому сетевому интерфейсу относится маршрут.

## Отображение статистики интерфейса

Когда *netstat* вызывается с параметром *-i*, он показывает статистику для сетевых интерфейсов. Если, кроме того, дается опция *-a*, он будет печатать все интерфейсы, представленные в ядре, а не только те, которые были отконфигурированы в настоящее время. На *cl.unix.class* вывод *netstat* будет напоминать это:

```
root@cl:~# netstat -i
Kernel Interface table
Iface      MTU Met    RX-OK RX-ERR RX-DRP RX-OVR      TX-OK TX-ERR TX-DRP TX-OVR Flg
eth0       1500 0        15     0     0 0        21     0     0 0 BMRU
eth1       1500 0        17     0     0 0        23     0     0 0 BMRU
eth2       1500 0       900     2     0 0       372     0     0 0 BMRU
lo         16436 0         24     0     0 0         24     0     0 0 LRU
root@cl:~#
```

Поля *MTU* и *Met* показывают текущий *MTU* и метрическое значение для этого интерфейса. Колонки *RX* и *TX* показывают сколько пакетов были получены или переданы без ошибок (*RX-OK* / *TX-OK*), повредились (*RX-ERR* / *TX-ERR*), сколько было потеряно (*RX-DRP* / *TX-DRP*) и сколько было потеряно из-за переполнения (*RX-OVR* / *TX-OVR*).

Последняя колонка показывает флаги, установленные для этого интерфейса. Здесь используется односимвольная версия флагов, которые печатает *ifconfig*:

- **B** Установлен широковещательный адрес.
- **L** Этот интерфейс задает устройство *loopback*.
- **M** Интерфейс получает все пакеты (режим *promiscuous*).
- **O** *ARP* выключен для этого интерфейса.
- **P** Это соединение *point-to-point*.
- **R** Интерфейс работает.
- **U** Интерфейс активен.

## Отображение соединений

*Netstat* поддерживает множество опций для отображения активных и пассивных соединений. Опции *-t*, *-u*, *-w* и *-x* показывают активные *TCP*, *UDP*, *RAW* или *UNIX* соединения. Если вы зададите параметр *-a*, сокет, который ждет соединения (то есть, слушает сеть), также показывается. Это даст вам список всех серверов, которые в настоящее время работают в вашей системе.

```
root@cl:~# netstat -ta
Active Internet connections (servers and established)
Proto Recv-Q Send-Q Local Address           Foreign Address         State
tcp      0      0 *:ssh                   *:                        LISTEN
tcp6     0      0 [::]:ssh                [::]:                    LISTEN
root@cl:~#
```

Видно, что все сервисы просто ждут соединения.

При использовании параметра *-a* будут отображаться все сокеты всех семейств сразу.

## Проверка ARP-таблиц

В некоторых случаях бывает полезно просмотреть или изменить содержание *ARP*-таблицы, например, когда Вы подозреваете, что двойной адрес является причиной сетевой неустойчивости. Утилита *arp* была



сделана для исправления подобных вещей. Синтаксис командной строки:

```
# arp --help
Usage:
  arp [-vn]    [<HW>] [-i <if>] [-a] [<hostname>]          <-Display ARP cache
  arp [-v]      [-i <if>] -d <host> [pub]                 <-Delete ARP entry
  arp [-vnD]    [<HW>] [-i <if>] -f [<filename>]           <-Add entry from file
  arp [-v]      [<HW>] [-i <if>] -s <host> <hwaddr> [temp] <-Add entry
  arp [-v]      [<HW>] [-i <if>] -Ds <host> <if> [netmask <nm>] pub <-'''
```

Аргумент hostname может быть как именем, так и IP адресом в стандарте dotted quad.

Первая строка отображает ARP-запись для IP-адреса, указанного хоста или всех известных хостов, если hostname не задается. Например, вызов `arp` на `cl.unix.class` может выдать:

```
root@cl:~# arp -a
? (192.168.1.2) at 08:00:27:65:8B:D2 [ether] on eth1
root@cl:~#
```

Опция `-s` используется, чтобы добавить Ethernet-адрес хоста к ARP-таблицам. Аргумент `hwaddr` определяет адрес аппаратных средств, который по умолчанию предполагается Ethernet-адресом, указанным как шесть шестнадцатеричных байт, разделяемых двоеточиями.

Одна из проблем, которая может потребовать, чтобы Вы вручную добавили IP-адрес к ARP-таблице, это когда по некоторым причинам ARP-запросы для удаленного хоста не доходят, например, когда есть сбой ARP-драйвера, или имеется другой хост в сети, который ошибочно опознает себя с IP-адресом другого хоста. Твердая установка IP-адреса в ARP-таблице также является мерой защиты себя от хостов в Вашем Ethernet, которые прикидываются кем-то другим.

Вызов `arp` с использованием ключа `-d` удаляет все ARP-записи, касающиеся данного хоста. Это может быть необходимо, чтобы вынудить интерфейс повторно получить Ethernet-адрес для данного IP. Это полезно, когда переконфигурированная система имеет неправильную ARP-информацию.

Опция `-s` может также использоваться, чтобы создать проху ARP. Это специальная техника когда хост, скажем, `gate` действует как `gateway` для другого хоста, назовем его `fnord`, делая вид, что оба адреса относятся к тому же самому хосту, а именно, `gate`. Это делается так: на `gate` создается ARP-запись о `fnord`, которая указывает на его собственный Ethernet-интерфейс. Теперь, когда хост посылает ARP-запрос о `fnord`, `gate` будет возвращать ответ, содержащий собственный Ethernet-адрес. Спрашивающий хост будет посылать все пакеты `gate`, который перенаправит их к `fnord`.

Эта схема может быть необходима, например, когда Вы хотите работать с `fnord` с DOS-машины с нестандартным TCP, которое плохо работает с маршрутизацией. Когда Вы используете проху ARP, DOS-машине будет казаться, что `fnord` находится в локальной подсети, так что ей не требуется что-либо знать относительно маршрутов и `gateway`.

Другое очень полезное применение проху ARP, когда один из Ваших хостов действует как `gateway` к некоторому другому хосту только временно, например, по телефону. Конечно, это будет работать только, если адрес хоста, для которого Вы хотите обеспечить проху ARP, находится в той же самой IP-подсети, что и Ваш `gateway`.

## Программа tcpdump

Программа `tcpdump` работает на всех платформах UNIX, а сейчас существует и аналог этой программы для платформ Windows - `Windump`.

Программа `tcpdump`, включаемая в большинство дистрибутивов UNIX, выводит заголовки пакетов для сетевого интерфейса в соответствии с заданным логическим выражением. Программа также допускает использование с флагом `-w` для записи пакетов в файл, которым может впоследствии использоваться для анализа. Возможен и просмотр заголовков из таких файлов с помощью флага `-r`. Во всех случаях `tcpdump` имеет дело только с пакетами, соответствующими заданному логическому выражению (фильтру).

`tcpdump` (если в команде не был указан флаг `-c`) продолжает собирать пакеты до тех пор, пока процесс не будет прерван сигналом SIGINT (например, при нажатии клавиш `control-C`) или SIGTERM (например, в результате команды `kill(1)`). Если команда используется с флагом `-c`, сбор пакетов кроме описанных выше способов может быть прекращен также после обработки определенного числа пакетов.

При завершении работы `tcpdump` выводит значения счетчиков для собранных (captured) пакетов (число пакетов, полученных и обработанных `tcpdump`); полученных фильтром (received by filter) пакетов; толкование этого значения зависит от ОС, под управлением которой работала программа `tcpdump` (в некоторых ОС указывается

число пакетов независимо от числа совпадений с условиями фильтрации, а в других – число пакетов, соответствующих фильтрам); отброшенных ядром (dropped by kernel) пакетов (число пакетов, отброшенных ядром по причине нехватки ресурсов или фильтрации внутри ядра).

Опция	Описание
-------	----------

- |                                   |   |
|-----------------------------------|---|
| ● <b>-A</b>                       | задает вывод каждого пакета (без заголовков канального уровня) в формате ASCII. Этот режим удобен для сбора трафика HTTP.   |
| ● <b>-c &lt;число пакетов&gt;</b> | задает завершение работы программы после захвата заданного числа пакетов.   |
| ● <b>-C &lt;размер файла&gt;</b>  | задает необходимость проверки размера файла захвата перед записью в него каждого нового пакета. Если размер файла превышает значение параметра file_size, этот файл закрывается и создается новый файл для записи в него пакетов. Для файлов захвата используется имя, заданное параметром -w и, начиная со второго файла к имени добавляется в качестве суффикса номер файла. Переменная file_size задает размер файла в миллионах байтов (не в мегабайтах = 1 048 576 байт).  |
| ● <b>-d</b>                       | задает вывод дампа скомпилированного кода соответствия пакетов (packet-matching code) в понятном человеку формате и завершение работы программы.  |
| ● <b>-dd</b>                      | выводит дамп кода соответствия в виде фрагмента C-программы.  |
| ● <b>-ddd</b>                     | выводит дамп кода соответствия в виде строки десятичных значений, перед которой следует строка со значением счетчика.   |
| ● <b>-D</b>                       | выводит список сетевых интерфейсов системы, с которых tcpdump может собирать пакеты. Для каждого сетевого интерфейса указывается имя и номер, за которыми может следовать текстовое описание интерфейса. Имя и номер интерфейса могут использоваться с флагом -i для задания сбора пакетов с одного интерфейса. Эта опция может быть весьма полезна для систем, не дающих информации об имеющихся сетевых интерфейсах. Флаг -D не поддерживается, если программа tcpdump была скомпилирована со старой версией libpcap, которая не поддерживает функцию pcap_findalldevs().   |
| ● <b>-e</b>                       | выводит заголовок канального уровня в каждой строке дампа.  |
| ● <b>-E</b>                       | задает использование алгоритма и секрета spi@ipaddr для расшифровки пакетов IPsec ESP, направленных по адресу ipaddr и содержащих and в поле Security Parameter Index значение spi. Комбинация spi и адреса может быть повторена с использованием в качестве разделителя запятой или новой строки. Отметим, что установка секрета для пакетов IPv4 ESP в настоящее время поддерживается. В качестве алгоритмов могут использоваться des-cbc, 3des-cbc, blowfish-cbc, rc3-cbc, cast128-cbc или none. По умолчанию применяется алгоритм des-cbc. Возможность дешифровки пакетов обеспечивается только в тех случаях, когда при компиляции tcpdump были включены опции поддержки криптографии. Параметр secret содержит ASCII-текст секретного ключа ESP. Если секрет начинается с символов 0x, будет считываться шестнадцатеричное значение. Опция предполагает использование ESP в соответствии с RFC 2406, а не RFC 1827. Эта опция поддерживается только для отладки и использовать ее с реальными секретными ключами не следует, поскольку введенный в командной строке ключ IPsec доступен другим пользователям системы. Кроме явного указания параметров в командной строке их можно задать в файле опций, который tcpdump будет читать при получении первого пакета ESP. |
| ● <b>-f</b>                       | задает вывод чужих адресов IPv4 в числовом формате. Использование этой опции позволяет избавиться от проблем, возникающих на серверах Sun NIS при попытках трансляции нелокальных адресов. Проверка чужеродности адреса IPv4 осуществляется с использованием адреса и маски принявшего пакет интерфейса. Если адрес и маска интерфейса недоступны (например, при использовании unnumbered-интерфейсов или при захвате пакетов со всех адресов в Linux с использованием фиктивного интерфейса any), эта опция будет работать некорректно.  |
| ● <b>-F &lt;файл&gt;</b>          | задает использование фильтров, содержащихся в указанном файле. В этом случае заданные в командной строке фильтры игнорируются.  |
| ● <b>-i &lt;интерфейс&gt;</b>     | задает сбор пакетов с указанного интерфейса. Если интерфейс не задан, tcpdump ищет в системе список доступных интерфейсов и выбирает в нем активное устройство с минимальным номером (исключая loopback). В системах Linux, начиная с ядра 2.2 поддерживается фиктивный интерфейс с именем any, обеспечивающий сбор пакетов со всех активных интерфейсов системы. Отметим, что сбор пакетов с устройства any осуществляется в обычном (не promiscuous) режиме. Если в системе поддерживается флаг -D, можно в качестве аргумента задавать номер интерфейса, выводимый при использовании этого флага.  |

- **-l** задает буферизацию строк stdout. Эта опция полезна в тех случаях, когда вы хотите просматривать данные во время сбора пакетов. Например, команды `tcpdump -l | tee dat` или `tcpdump -l > dat & tail -f dat` обеспечивают запись пакетов в файл `dat` и одновременный вывод на консоль.
- **-L** задает вывод списка известных типов канального уровня и завершение работы программы.
- **-m <файл>** загружает модуль определений SMI MIB из указанного файла. Эта опция может использоваться неоднократно для загрузки нескольких модулей MIB.
- **-n** отключает преобразование адресов и номеров портов в символьные имена.
- **-N** задает использование только имен хостов, а не полных доменных имен. Например, вместо `lhotze.bilim-systems.net` при использовании этой опции моя рабочая станция будет обозначаться как `lhotze`.
- **-O** отключает оптимизатор кода проверки соответствия пакетов условиям фильтрации. Используйте эту опцию, если вам покажется, что оптимизатор работает с ошибками.
- **-p** указывает программе, что интерфейс не нужно переводить в режим захвата<sup>5</sup>. Опцию `-p` нельзя использовать вместе с фильтром `ether host {local-hw-addr}` or `ether broadcast`.
- **-q** задает вывод минимального объема информации.
- **-R** при установке этого флага предполагается, что пакеты ESP/AH используют старый вариант спецификации<sup>6</sup> и `tcpdump` не будет выводить поля `replay prevention` (защита от воспроизведения). Поскольку спецификация ESP/AH не включает поля с номером версии, `tcpdump` не может определить версию протокола ESP/AH по заголовкам пакетов.
- **-r <файл>** задает чтение данных из файла, созданного ранее с использованием команды `tcpdump -w` или с помощью другой программы, поддерживающей формат `tcpdump` (например, `Ethereal`). Если в качестве имени файла задан символ `-`, используется поток данных от стандартного устройства ввода (`stdin`).
- **-S** задает вывод абсолютных порядковых номеров TCP взамен относительных.
- **-s** задает захват из каждого пакета `snaplen` байтов вместо отбираемых по умолчанию 68 байтов<sup>7</sup>. Значение 68 подходит для протоколов IP, ICMP, TCP и UDP но может приводить к потере протокольной информации для некоторых пакетов DNS и NFS. Потеря части пакетов по причине малого размера кадра захвата (`snapshot`) указывается в выходных данных полями вида `[proto]`, где `proto` – имя протокольного уровня, на котором произошло отсечение части пакета<sup>8</sup>. Отметим, что увеличение кадра захвата приведет к дополнительным временным затратам на обработку пакетов и уменьшению числа буферизуемых пакетов, что может привести к потере части пакетов. Используйте минимальное значение `snaplen`, которое позволит обойтись без потери информации об интересующем вас протоколе. Установка `snaplen = 0` приведет к захвату полных пакетов.
- **-T <mun>** задает интерпретацию пакетов, выбранных с помощью фильтра, как пакетов указанного параметром типа. В настоящее время поддерживаются типы `aodv9`, `cnfp10`, `rpc11`, `rtpl2`, `rtcp13`, `snmp14`, `tftp15`, `vat16` и `wb17`.
- **-t** отключает вывод временных меток в каждой строке дампа.
- **-tt** задает вывод в каждой строке дампа неформатированных временных меток.
- **-ttt** задает вывод временных интервалов (в микросекундах) между захватом предыдущего и данного пакетов в каждой строке дампа.
- **-tttt** задает вывод временных меток в принятом по умолчанию формате для каждой строки дампа.
- **-u** задает вывод манипуляторов (`handle`) NFS без декодирования.
- **-U** задает режим “буферизации на уровне пакетов” для файлов, сохраняемых с помощью опции `-w`. В этом режиме каждый пакет записывается в выходной файл как только он будет захвачен (не дожидаясь заполнения выходного буфера). Флаг `-U` не будет поддерживаться, если программа `tcpdump` была скомпилирована со старой опцией `libpcap`, не поддерживающей функцию `pcap_dump_flush()`.
- **-v** задает вывод дополнительной информации при захвате файлов. К такой информации может относиться значение TTL (время жизни), идентификация, общий размер, опции IP и т. п. При использовании этого флага также выполняется дополнительная проверка целостности пакетов с помощью контрольных сумм (например, для протоколов IP и ICMP).
- **-vv** задает дополнительное увеличение объема выводимой информации (например, полное декодирование пакетов SMB, вывод дополнительных полей откликов NFS и т. п.).

- **-vvv** задает максимальный объем выводимой информации (например, полностью выводятся опции telnet SB ... SE). При использовании вместе с ключом **-X** опции Telnet выводятся также в шестнадцатеричном представлении.
- **-w <файл>** задает запись необработанных (raw) пакетов. Собранные в файл пакеты можно впоследствии просматривать с использованием флага **-g** или передавать для анализа другим программам (например, Ethereal). Если в качестве имени файла указан символ **-**, запись осуществляется на стандартное устройство вывода (stdout).
- **-x** задает вывод шестнадцатеричного дампа (без заголовка канального уровня) для каждого захваченного пакета. Объем выводимой информации определяется меньшим из двух значений - размер пакета и значение параметра `snaplen`. Отметим, что при захвате полных кадров канального уровня дампы могут включать также байты заполнения, если пакет сетевого уровня имеет малый размер.
- **-xx** задает вывод шестнадцатеричного дампа для каждого пакета с включением заголовков канального уровня.
- **-X** задает вывод дампа в шестнадцатеричном и ASCII-формате без заголовков канального уровня. Эта опция может быть очень удобна при анализе новых протоколов.
- **-XX** задает вывод дампа в шестнадцатеричном и ASCII-формате с включением заголовков канального уровня.
- **-y<тип>** задает тип канального уровня, используемого при захвате пакетов. Поддерживаемые значения можно посмотреть с помощью флага **-L**.

## Фильтрация при сборе пакетов

В командной строке `tcpdump` наряду с опциями могут задаваться выражения, определяющие фильтрацию пакетов на этапе их сбора. Если никакого фильтра не задано, программа будет собирать все пакеты.

Каждое выражение, задающее фильтр, включает один или несколько примитивов, состоящих обычно из одного или нескольких идентификаторов объекта и предшествующих ему классификаторов. Идентификатором объекта может служить его имя или номер. Классификаторы объектов могут относиться к одному из трех видов:

### type

указывает тип объекта, заданного идентификатором. В качестве типа объектов могут указываться значения `host` (хост), `net` (сеть) и `port` (порт). Если тип объекта не указан, предполагается значение `host`.

### dir

задает направление по отношению к объекту. Для этого классификатора поддерживаются значения `src` (объект является отправителем), `dst` (объект является получателем), `src or dst` (отправитель или получатель) и `src and dst` (отправитель и получатель). Например, `src foo` указывает на пакеты, отправленные с хоста `foo`, `dst net 128.3` - пакеты, адресованные в сеть `128.3.0.0/16`, а `src or dst port ftp-data` - пакеты данных протокола FTP (порт `ftp-data`), передаваемые в обоих направлениях. Если классификатор `dir` не задан, предполагается значение `src or dst`. Для некоторых типов соединений (например, SLIP) и режимов захвата (например, захват с фиктивного интерфейса `any` в Linux-системах) могут использоваться классификаторы `inbound` и `outbound`.

### proto

задает протокол, к которому должны относиться пакеты. Этот классификатор может принимать значения `ether`, `fdi1`, `tr2`, `wlan3`, `ip`, `ip6`, `arp`, `rarp`, `decnet`, `tcp` и `udp`. Если примитив не содержит классификатора протокола, предполагается, что данному фильтру удовлетворяют все протоколы, совместимые с типом объекта.

Кроме объектов и квалификаторов примитивы могут содержать ключевые слова `gateway` (шлюз), `broadcast` (широковещательный), `less` (меньше), `greater` (больше) и арифметические выражения.

Сложные фильтры могут содержать множество примитивов, связанных между собой с использованием логических операторов `and`, `or` и `not` (например, `host foo and not port ftp and not port ftp-data`). Для сокращения задающих фильтры выражений можно опускать идентичные списки квалификаторов. Например, выражение `tcp dst port ftp or ftp-data or domain` будет краткой формой выражения

```
tcp dst port ftp or tcp dst port ftp-data or tcp dst port domain
```

## Логические выражения

Выражения типа

`expr <операция> expr`

возвращают логическое значение, соответствующее отношениям между левой и правой частью. В качестве операции могут использоваться `>`, `<`, `>=`, `<=`, `=`, `!=`, а операнды `expr` могут быть арифметическими выражениями, включающими целые константы (запись в стандарте C), бинарные операторы `+`, `-`, `*`, `/`, `&`, `|`, `<<`, `>>`, оператор длины (`offset`) и данные из пакетов. Для получения значений полей из пакетов применяется синтаксис:

`proto [offset : size]`

Параметр `proto` может содержать идентификатор одного из протоколов (`ether`, `fddi`, `tr`, `wlan`, `ppp`, `slip`, `link`, `ip`, `arp`, `garp`, `tcp`, `udp`, `icmp`, `ip6`) и задает уровень протокола<sup>15</sup>, для которого извлекаются данные. Отметим, что `tcp`, `udp` и другие протоколы верхних уровней относятся только к пакетам IPv4, а не IPv6. Параметр `offset` задает смещение в байтах относительно начала заголовка указанного уровня. Необязательный параметр `size` определяет размер интересующего поля в байтах. Допустимы значения размера 1, 2 и 4, по умолчанию просматривается 1 байт.

Оператор длины, указываемый ключевым словом `len`, определяет размер пакета в байтах.

Например, выражению

`ether[0] & 1 != 0`

будет соответствовать весь multicast-трафик, выражение

`ip[0] & 0xf != 5`

позволяет собрать все пакеты IP, в которых присутствует поле опций, а фильтр

`ip[6:2] & 0x1fff = 0`

соберет только нефрагментированные дейтаграммы и первые фрагменты.

При выборе полей из заголовков учитывается структура пакетов соответствующего уровня. Например, `tcp[0]` всегда будет возвращать первый байт заголовка TCP, игнорируя фрагменты.

Некоторые поля и значения смещений могут задаваться не только числами, но и именами. В частности, для протокола поддерживается параметр `icmp-type` (поле типа ICMP), который может принимать значения:

`icmp-echoreply`, `icmp-unreach`, `icmp-sourcequench`, `icmp-redirect`, `icmp-echo`, `icmp-routeradvert`, `icmp-routersolicit`, `icmp-timxceed`, `icmp-paramprob`, `icmp-tstamp`, `icmp-tstamp-reply`, `icmp-ireq`, `icmp-ireqreply`, `icmp-maskreq`, `icmp-maskreply`.

Для флагов TCP можно использовать следующие идентификаторы:

`tcp-fin`, `tcp-syn`, `tcp-rst`, `tcp-push`, `tcp-ack` и `tcp-urg`.

Примитивы в выражениях можно группировать с использованием

- скобок;
- отрицания (! или not);
- логического пересечения (&& или and);
- логического объединения (|| или or).

Оператор отрицания имеет высший уровень приоритета, операции объединения и пересечения имеют одинаковый приоритет и выполняются слева направо в порядке следования. Отметим, что для операции логического пересечения недостаточно просто указать операнды рядом, а требуется явно задать операцию (&& или and).

Если идентификатор указан без ключевого слова, предполагается ключевое слово, которое до этого использовалось последним. Например, выражение

`not host vs and ace`

является простым сокращением от

`not host vs and host ace`

Отметим, что эти выражения не эквивалентны фильтру `not ( host vs or ace )`.

Аргументы выражений могут передаваться программе `tcpdump` как один или множество аргументов

(используйте более удобную для вас форму). В общем случае выражения, содержащие мета-символы командного интерпретатора, должны передаваться как один аргумент, заключенный в кавычки.

## Примеры фильтров

Фильтр	Выполняемые действия
● <b>tcpdump host sundown</b>	Выводит все пакеты, принимаемые и передаваемые хостом sundown
● <b>tcpdump host helios and ( hot or ace )</b>	Выводит пакеты, передаваемые между хостом helios и любым из хостов hot или ace.
● <b>tcpdump ip host ace and not helios</b>	Выводит пакеты передаваемые между хостом ace и любым хостом, за исключением helios.
● <b>tcpdump net ucb-ether</b>	Выводит все пакеты, передаваемые или принимаемые хостами сети ucb-ether.
● <b>tcpdump 'gateway snup and (port ftp or ftp-data)'</b>	Выводит весь трафик ftp, проходящий через шлюз snup
● <b>tcpdump ip and not net localnet</b>	Выводит весь трафик, за исключением исходящего от локальных хостов и адресованного им.
● <b>tcpdump 'tcp[tcpflags] &amp; (tcp-syn tcp-fin) != 0 and not src and dst net localnet'</b>	Выводит стартовые (SYN) и конечные (FIN) пакеты TCP, исключая соединения локальных хостов.
● <b>tcpdump 'gateway snup and ip[2:2] &gt; 576'</b>	Выводит переданные через шлюз snup пакеты IP, размер которых превышает 576 байтов.
● <b>tcpdump 'ether[0] &amp; 1 = 0 and ip[16] &gt;= 224'</b>	Выводит широковещательные и групповые пакеты, которые не были переданы с использованием широковещательных и групповых адресов Ethernet.
● <b>tcpdump 'icmp[icmptype] != icmp-echo and icmp[icmptype] != icmp-echoreply'</b>	Выводит все пакеты ICMP, кроме запросов и откликов echo (т. е., кроме пакетов ping).

Программа tcpdump является очень гибкой и имеет огромный функционал. Вы можете найти больше информации по адресу

<http://www.protocols.ru/modules.php?name=News&file=article&sid=32>

## Настройка клиента DNS

Как уже обсуждалось выше, сеть TCP/IP может полагаться на различные схемы преобразования имен в адреса. Самый простой способ - таблица хостов, сохраненная в /etc/hosts. Это полезно только для маленьких LAN, которые управляются одним администратором, и не имеют никакого IP общения с внешним миром.

С другой стороны Вы можете использовать BIND ( Berkeley Internet Name Domain) для перевода имен хостов в IP-адреса. Конфигурация BIND может показаться сложной задачей, но если только Вы сделаете это, то изменения в сетевой топологии могут быть легко отслежены. На Linux, как и на многих других Unix-подобных системах, сервис обеспечивается через программу, называемую named. При запуске эта программа загружает множество основных файлов в собственный кэш и ждет запрос от удаленных или локальных пользовательских процессов. Имеющиеся способы требуют, чтобы Вы обязательно ввели имя сервера для каждого хоста.

## Библиотека Resolver

Когда мы говорим о "resolver", то не подразумеваем никакого специального приложения, речь идет только об интерфейсе: системе функций, которая может быть найдена в стандартной библиотеке C. Основными вызовами являются gethostbyname и gethostbyaddr, которые ищут все IP-адреса, принадлежащие хосту, и наоборот. Они могут быть сконфигурированы при простом просмотре информации о хосте, при запросе ряда серверов, или при использовании баз данных hosts NIS (Network Information Service).

Функции читают файл настройки. Он сообщает, какой сервис использовать и в каком порядке. Старая стандартная библиотека Linux, libc, использовала файл /etc/host.conf, но библиотека GNU Version 2, glibc, использует файл /etc/nsswitch.conf.

## Файл *host.conf*

Опции в */etc/host.conf* должны быть на отдельных строках. Области могут быть отделены пустым пространством (пробелами или табуляцией). Знак *#* вводит комментарий до конца строки. Доступны следующие опции:

### **order**

Эта опция определяет порядок, в котором перебираются все доступные сервисы. Значение *hosts* связывает запросы сервера с поиском хостов в */etc/hosts*, *bind* с пакетом *named*, а *nis* с NIS-поиском. Любое или все значения сразу могут быть определены. Порядок, в котором они появляются в строках, определяет последовательность, в которой будут перебираться сервисы.

### **multi**

Эта опция определяет, разрешено ли хосту в */etc/hosts* иметь несколько IP-адресов, которые обычно называются "multi-homed". Этот флаг не действует на DNS или NIS запросы. Допустимые значения: *on* или *off*. По умолчанию *off*.

### **nospoof**

Как уже объяснено в предыдущем разделе, DNS позволяет найти имя хоста, принадлежащего IP-адресу, используя домен *in-addr.агра*. Попытки серверов поддерживать ложное имя хоста называются "спуфинг" ("spoofing"). Чтобы обезопасить себя от этого, библиотека может быть сконфигурирована на проверку, является ли настоящий IP-адрес фактически связанным с полученным именем хоста. Если нет, то этому имени будет отказано в преобразовании и возвращен код ошибки. Это поведение зависит от того, включен ли *nospoof*.

### **alert**

Эта опция может принимать значения *on* или *off*. Если она включена, то любые попытки спуфинга будут внесены в протокол *syslog*.

### **trim**

Эта опция берет как аргумент имя домена, которое будет удалено из имени хоста перед поиском. Это полезно для информационных элементов, где Вы хотите точно определить имя хоста с локальным доменом. При поиске хоста с именем локального домена будет удалено имя этого домена. Таким образом, легко осуществить поиск в */etc/hosts*. Добавляемое имя домена должно оканчиваться точкой (*.*), например *linux.org.au.*

Опция *trim* позволяет рассматривать Ваш хост как локальный для нескольких доменов.

Файл, созданный по умолчанию на *cl.unix.class*:

```
root@cl:~# cat /etc/host.conf
order hosts, bind
multi on
root@cl:~#
```

## **Resolver и переменные окружения**

Установки из файла *host.conf* могут быть изменены, используя ряд переменных окружения:

### **RESOLV\_HOST\_CONF**

Определяет файл, который будет считан вместо */etc/host.conf*.

### **RESOLV\_SERV\_ORDER**

Отменяет опцию *order* в *host.conf*. Сервисы, заданные как *hosts*, *bind*, и *nis*, разделяются пробелом, запятой, двоеточием или точкой с запятой.

### **RESOLV\_SPOOF\_CHECK**

Определяет действия, принимаемые против спуфинга. Эта установка полностью отключается опцией *off*. Значения *warn* и *warn off* включают проверку спуфинга, но соответственно включают и выключают протоколирование (*logging*). Значение *\** включает проверку, но оставляет *logging*, как определено в файле *host.conf*.

### **RESOLV\_MULTI**

Эта переменная может быть использована для перекрытия опции *multi* из *host.conf*. Допустимы значения *on* или *off*.

### **RESOLV\_OVERRIDE\_TRIM\_DOMAINS**

Определяет список trim-доменов, который отключает те, что заданы в host.conf.

### RESOLV\_ADD\_TRIM\_DOMAINS

Определяет список trim-доменов, которые добавляются к заданным в host.conf.

## Файл nsswitch.conf

Version 2 GNU standard library включает более мощный и гибкий механизм, чем старый host.conf. Понятие сервиса имен было расширено с включением ряда различных типов информации. Опции конфигурации для всех функций, которые делают запрос к базам данных, были вынесены в один файл конфигурации nsswitch.conf.

Файл nsswitch.conf позволяет администратору конфигурировать большое число разных баз данных. Мы ограничим наше обсуждение параметрами, которые касаются хостов и поиска адресов IP.

Опции в nsswitch.conf должны располагаться на отдельных строках. Поля разделяются пробелами или табуляцией. Знак # как обычно определяет комментарий. Каждая строка описывает свой сервис. Первое поле в каждой строке задает имя базы данных, заканчивающееся двоеточием. База данных для преобразования адресов hosts. Связанная база данных, которая используется для преобразования имен сетей в адреса, networks.

Доступны следующие параметры:

### dns

Использовать Domain Name System (DNS), чтобы найти адрес. Это имеет смысл только для адресов машин, но не сетей. Этот механизм использует файл /etc/resolv.conf, который мы обсудим позже.

### files

Искать локальный файл для хоста или имени сети и связанный адрес. Опция использует традиционные /etc/hosts и /etc/network.

### nis или nisplus

Использовать Network Information System (NIS), чтобы найти адрес.

Порядок, в котором перечислены сервисы, определяет последовательность их опроса. Список порядка опроса находится в описании сервиса в файле /etc/nsswitch.conf. Сервисы опрашиваются слева направо, и по умолчанию опрос прекращается когда преобразование выполнено.

Простой пример спецификации баз данных host и network показан ниже:

```
#
# /etc/nsswitch.conf
#
# An example Name Service Switch config file. This file should be
# sorted with the most-used services at the beginning.
#
# The entry '[NOTFOUND=return]' means that the search for an
# entry should stop if the search in the previous entry turned
# up nothing. Note that if the search failed due to some other reason
# (like no NIS server responding) then the search continues with the
# next entry.
#
# Legal entries are:
#
#     nisplus or nis+      Use NIS+ (NIS version 3)
#     nis or yp           Use NIS (NIS version 2), also called YP
#     dns                 Use DNS (Domain Name Service)
#     files               Use the local files
#     [NOTFOUND=return]   Stop searching if not found so far
#
hosts:      dns files
networks:   files
```

Этот пример заставляет систему искать компьютеры сначала в Domain Name System, а в случае неудачи в файле /etc/hosts. Поиск имени сети использует только файл /etc/networks.

Вы способны управлять поисковой таблицей более точно, используя "action items", которые описывают, какое



действие использует результат предыдущего поиска. Action items появляются между сервисными спецификациями и включены в квадратные скобки ([ ]). Общий синтаксис здесь такой: [ [!] status = action ... ]

Имеются два возможных действия:

#### **return**

Управление возвращается программе, которая запросила преобразование имени. Если попытка поиска была успешна, resolver вернет подробные данные, иначе нулевой результат.

#### **continue**

Resolver перейдет к следующему сервису в списке и будет пытаться использовать его.

Факультативный символ ! определяет, что значение состояния должно быть инвертировано перед тестированием.

Доступные значения состояния, которые мы можем использовать:

#### **success**

Запрошенная запись была найдена без ошибки. Заданное по умолчанию действие для этого состояния return.

#### **notfound**

Не было ошибки в поисковой таблице, но компьютер адресата или сеть не найдены. Заданное по умолчанию действие для этого состояния continue.

#### **unavail**

Сервис недоступен. Файл hosts или networks нечитаем для сервиса files, сервер имен или NIS-сервер не отвечают на запросы сервисов dns или nis. Заданное по умолчанию действие для этого состояния continue.

#### **tryagain**

Это состояние означает, что сервис временно недоступен. Для сервиса files это обычно указывает, что файл был заблокирован неким процессом. Для других сервисов это может означать, что сервер временно не может принимать подключения. Заданное по умолчанию действие для этого состояния continue.

Простой пример того, как Вы могли бы использовать этот механизм, показан ниже

```
hosts:      dns [!UNAVAIL=return] files
networks:   files
```

Этот пример пытается искать компьютеры через Domain Name Service. Если состояние возврата не unavailable, resolver возвращает любой найденный адрес. Если, и только если, DNS возвращает unavailable, resolver делает попытку использовать локальный /etc/hosts. Это означает, что он должен использовать файл hosts, только если наш сервер имен является недоступным.

### **Файл resolv.conf**

При конфигурировании библиотеки resolver для того, чтобы использовать сервис BIND для поиска хостов, Вы обязательно должны сообщить, какое имя сервера используете. Существует отдельный файл, предназначенный специально для этого, называемый resolv.conf. Если этот файл не существует или пуст, то resolver примет имя сервера, определенного для Вашего локального хоста. Если Вы запускаете сервер на Вашем локальном хосте, то должны установить это имя отдельно, как это сделать будет объяснено позже в следующем разделе. Если в локальной сети есть возможность использовать существующее имя сервера, то ему должно отдаваться предпочтение.

Самая важная опция в resolv.conf это name server, которая задает IP-адрес используемого сервера. Если Вы определите несколько имен серверов, используя опцию name server несколько раз, то они будут проверяться в том порядке в котором вы их поместили в файл. Поэтому Вы должны поместить наиболее надежный сервер первым. Могут поддерживаться не более трех серверов. Если опция name server не задана, то resolver попытается соединиться с сервером на локальном хосте.

Две других опции, domain и search, имеют дело с заданными по умолчанию доменами, которые берутся из имени хоста, если BIND не может найти адрес с первого запроса. Опция search определяет список доменов, которые необходимо проверить. Пункты списка разделены пробелом или табуляцией. Обычно при подключении компьютера, в локальном домене не указывают полностью квалифицированное имя, но использование имени вроде gauss в командной строке приведет на локальный домен, например, mathematics.gu.edu.

Инструкция domain. Она позволяет Вам определять заданное по умолчанию имя домена, которое будет

добавлено, когда DNS терпит неудачу при поиске имени хоста. Например, когда дано имя gauss, resolver не может найти машину gauss. в DNS, поскольку такого домена верхнего уровня нет. Когда для domain определено mathematics.gu.edu как значение по умолчанию, resolver повторяет запрос для gauss с заданным по умолчанию доменом.

Но как только Вы выйдете за пределы домена отдела математики, хорошая жизнь кончится. Конечно, Вы также хотели бы иметь записи вроде quark.physics для компьютеров в отделе Физики.

Это делается с помощью search list. Список может быть определен, используя опцию search, которая является обобщением инструкции domain. Он задает перечень доменов, которые будут использовать для поиска машины с коротким именем.

Инструкции search и domain исключают друг друга и не могут появляться больше одного раза. Если никакая опция не задана, resolver пробует взять заданный по умолчанию домен из локального имени хоста (hostname) системным вызовом getdomainname. Если локальный hostname не имеет доменной части, заданный по умолчанию домен будет принят как корневой домен.

Если Вы помещаете инструкцию search в resolv.conf, Вы должны внимательно следить, какие домены туда попадают. Библиотека Resolver до BIND 4.9 создавала заданный по умолчанию список поиска из имени домена, когда список поиска не был задан. Этот заданный по умолчанию список был составлен из заданного по умолчанию домена и всех родительских доменов до корневого. Это вызывало проблемы потому, что запросы DNS попадали на сервера, которые никогда не имели дела с нужным доменом.

Предположим, что Вы из сети Virtual Brewery хотите обратиться к foot.gu.edu. Но ошибочно вместо foot набираете foo, который не существует. Сервер имен GU сообщит Вам, что не знает такой компьютер. Со списком поиска старого стиля resolver теперь продолжил бы пробовать найти имя с добавлением vbrew.com и com. Последнее проблематично, потому что gu.edu.com вполне может оказаться реальным доменом. Их сервер имен мог бы даже найти в своем домене foo. В результате Вы попадете на машину, к которой совсем не стремились! Вряд ли Вы на ней зарегистрируетесь, но с толку будете сбиты основательно.

Для некоторых прикладных программ поддельные поисковые таблицы могут быть проблемой защиты. Следовательно, Вы должны обычно ограничивать домены в Вашем списке поиска. В отделе математики GU список поиска обычно установлен в maths.gu.edu и gu.edu.

Если заданный по умолчанию домен создает проблемы, посмотрите часть файла resolv.conf для Virtual Brewery:

```
# /etc/resolv.conf
# Our domain
domain          vbrew.com
#
# We use vlager as central name server:
name server     172.16.1.1
```

Когда запрашивается имя vale, resolver ищет vale, а в случае неудачи еще и vale.vbrew.com.

## Ошибкоустойчивость Resolver

Если Вы создаете LAN внутри большей сети, непременно должны использовать центральные сервера имен, если они доступны. Преимущество этого состоит в том, что они имеют богатые кэши, так как все запросы направлены к ним. Эта схема имеет недостаток: когда сгорел базовый кабель в нашем университете при пожаре, невозможно было дальше работать в LAN нашего отдела, потому что resolver не мог достичь какого-либо из серверов. Не было login на X-терминалах, печати на принтерах и т.д.

Хотя опускаться до пожаров для университетского городка никуда не годиться, каждый обязан соблюдать технику безопасности, чтобы избежать случаев, подобных этим.

Один из способов это обойти: устанавливать локальный сервер, который определяет имена из Вашего локального домена и пересылает все запросы для других имен к главным серверам. Конечно, это применимо только тогда, когда Вы используете Ваш собственный домен.

Альтернатива: Вы можете поддерживать резервную ведущую таблицу для домена или LAN в /etc/hosts. Это очень просто, Вы просто гарантируете, что библиотека resolver делает сначала запрос DNS, а во вторую очередь файла hosts. В файле /etc/host.conf Вы используете для этого order bind,hosts, а в файле /etc/nsswitch.conf следует задать hosts: dns files, чтобы resolver вернулся обратно к файлу hosts, если центральный сервер имен недоступен.

Принцип работы системы DNS и настройка сервера BIND подробно изучаются в курсе «Сетевое администрирование Linux».

## Настройка удаленного управления

Часто очень полезно выполнить команду на удаленном компьютере. Причем ввод и вывод команды осуществляются по сети.

Традиционные команды, используемые для выполнения задач на удаленных компьютерах `telnet`, `rlogin`, `rsh` и `rcp`. Однако эти программы имеют серьезные проблемы защиты. Сегодня в качестве замены этим программам существует пакет `ssh`. Он обеспечивает команды `slogin`, `ssh` и `scp`.

Каждая из этих команд порождает оболочку на удаленном компьютере и позволяет пользователю выполнять команды. Конечно, пользователь должен быть зарегистрирован на том удаленном компьютере, где должна быть выполнена команда. Таким образом, все эти команды используют процесс авторизации. Старые `r`-команды просто обменивались с удаленным компьютером именем пользователя и его паролем в открытом виде. Поэтому любая программа сетевого наблюдения легко перехватывала пароли. Пакет `ssh` обеспечивает более высокий уровень защиты: он использует методику Public Key Cryptography для авторизации и шифрования обменов между компьютерами, чтобы гарантировать, что ни пароли, ни другие данные сеанса не перехвачены. В принципе, их как и раньше можно перехватить, но они будут зашифрованы стойким криптоалгоритмом, так что никакой пользы перехватившему от них не будет.

Иногда желательно ослабить проверку доступа для некоторых пользователей. Например, если Вы часто должны регистрироваться на других машинах Вашей локальной сети, неплохо бы быть признанным без ввода пароля каждый раз. Это всегда было возможно с `r`-командами, но пакет `ssh` позволяет Вам сделать это немного легче. Но такой подход опасен тем, что если на одной машине логин пользователя вскрыт, это дает доступ без пароля на другие машины.

## Шифрование

Шифрование — способ преобразования информации, применяемый для хранения важной информации в ненадежных источниках или передачи её по незащищенным каналам связи. Согласно ГОСТ 28147-89, шифрование — процесс зашифрования или расшифрования.

В зависимости от структуры используемых ключей методы шифрования подразделяются на:

- *симметричное шифрование*: посторонним лицам может быть известен алгоритм шифрования, но неизвестна небольшая порция секретной информации — ключа, одинакового для отправителя и получателя сообщения;
- *асимметричное шифрование*: посторонним лицам может быть известен алгоритм шифрования, и, возможно открытый ключ, но неизвестен закрытый ключ, известный только получателю.

Различные виды шифрования обладают различной криптостойкостью.

### Ключи

Ключ — секретная информация, используемая криптографическим алгоритмом при шифровании/расшифровке сообщений, постановке и проверке цифровой подписи, вычислении кодов аутентичности (MAC). При использовании одного и того же алгоритма результат шифрования зависит от ключа. Для современных алгоритмов сильной криптографии утрата ключа приводит к практической невозможности расшифровать информацию.

Согласно принципу Керхгоффса, надёжность криптографической системы должна определяться сокрытием секретных ключей, но не сокрытием используемых алгоритмов или их особенностей.

#### Длина ключа

Количество информации в ключе, как правило, измеряется в битах.

Для современных симметричных алгоритмов (AES, CAST5, IDEA, Blowfish, Twofish) основной характеристикой криптостойкости является длина ключа. Шифрование с ключами длиной 128 бит и выше считается сильным, так как для расшифровки информации без ключа требуются годы работы мощных суперкомпьютеров. Для асимметричных алгоритмов, основанных на проблемах теории чисел (проблема факторизации — RSA, проблема дискретного логарифма — Elgamal) в силу их особенностей минимальная надёжная длина ключа в настоящее время — 1024 бит. Для асимметричных алгоритмов, основанных на использовании теории эллиптических кривых (ECDSA, ГОСТ Р 34.10-2001, ДСТУ 4145-2002), минимальной надёжной длиной ключа считается 163 бит, но рекомендуются длины от 191 бит и выше.

#### Классификация ключей

Криптографические ключи различаются согласно алгоритмам, в которых они используются.

*Секретные (Симметричные) ключи* — ключи, используемые в симметричных алгоритмах (шифрование, выработка кодов аутентичности). Главное свойство симметричных ключей: для выполнения как прямого, так и обратного криптографического преобразования (шифрование/расшифровывание, вычисление MAC/проверка MAC) необходимо использовать один и тот же ключ (либо же ключ для обратного преобразования легко вычисляется из ключа для прямого преобразования, и наоборот). С одной стороны, это обеспечивает более высокую конфиденциальность сообщений, с другой стороны, создаёт проблемы распространения ключей в системах с большим количеством пользователей.

*Асимметричные ключи* — ключи, используемые в асимметричных алгоритмах (шифрование, ЭЦП); вообще говоря, являются ключевой парой, поскольку состоят из двух ключей:

Закрытый ключ (Private key) — ключ, известный только своему владельцу

Открытый ключ (Public key) — ключ, доступный всем пользователям криптографической системы.

Главное свойство ключевой пары: по секретному ключу легко вычисляется открытый ключ, но по известному открытому ключу практически невозможно вычислить секретный. В алгоритмах ЭЦП подпись обычно ставится на секретном ключе пользователя, а проверяется на открытом. Таким образом, любой может проверить, действительно ли данный пользователь поставил данную подпись. Тем самым асимметричные алгоритмы обеспечивают не только целостность информации, но и её аутентичность. При шифровании же наоборот, сообщения шифруются на открытом ключе, а расшифровываются на секретном. Таким образом, расшифровать сообщение может только адресат и больше никто (включая отправителя). Использование асимметричных алгоритмов снимает проблему распространения ключей пользователей в системе, но ставит новые проблемы: достоверность полученных ключей. Эти проблемы более-менее успешно решаются в рамках инфраструктуры открытых ключей (PKI).

*Сеансовые (сессионные) ключи* — ключи, вырабатываемые между двумя пользователями, обычно для защиты канала связи. Обычно сеансовым ключом является общий секрет — информация, которая вырабатывается на основе секретного ключа одной стороны и открытого ключа другой стороны. Существует несколько протоколов выработки сеансовых ключей и общих секретов, среди них, в частности, алгоритм Диффи — Хеллмана.

*Подключи* — ключевая информация, вырабатываемая в процессе работы криптографического алгоритма на основе ключа. Зачастую подключа вырабатываются на основе специальной процедуры развёртывания ключа.

## Симметричное шифрование

Симметричные криптосистемы (также симметричное шифрование, симметричные шифры) — способ шифрования, в котором для (за)шифрования и расшифрования применяется один и тот же криптографический ключ. До изобретения схемы асимметричного шифрования единственным существовавшим способом являлось симметричное шифрование. Ключ алгоритма должен сохраняться в секрете обеими сторонами. Ключ алгоритма выбирается сторонами до начала обмена сообщениями.

### Общая схема

В настоящее время симметричные шифры - это:

- *блочные шифры*. Обработывают информацию блоками определённой длины (обычно 64, 128 бит), применяя к блоку ключ в установленном порядке, как правило, несколькими циклами перемешивания и подстановки, называемыми раундами. Результатом повторения раундов является лавинный эффект - нарастающая потеря соответствия битов между блоками открытых и зашифрованных данных.
- *поточные шифры*, в которых шифрование проводится над каждым битом либо байтом исходного (открытого) текста с использованием гаммирования. Поточный шифр может быть легко создан на основе блочного (например, ГОСТ 28147-89 в режиме гаммирования), запущенного в специальном режиме.

### Параметры алгоритмов

Существует множество (не менее двух десятков) алгоритмов симметричных шифров, существенными параметрами которых являются:

- стойкость
- длина ключа
- число раундов
- длина обрабатываемого блока

- сложность аппаратной/программной реализации

### **Распространенные алгоритмы**

DES и TripleDES (3DES)

AES (Rijndael)

ГОСТ 28147-89

Blowfish

Twofish

CAST

Serpent

NUSH

### **Сравнение с асимметричными криптосистемами**

#### *Достоинства*

- скорость (по данным Applied Cryptography — на 3 порядка выше)
- простота реализации (за счёт более простых операций)
- меньшая требуемая длина ключа для сопоставимой стойкости
- изученность (за счёт большего возраста)

#### *Недостатки*

- сложность управления ключами в большой сети. Означает квадратичное возрастание числа пар ключей, которые надо генерировать, передавать, хранить и уничтожать в сети. Для сети в 10 абонентов требуется 45 ключей, для 100 уже 4950, для 1000 — 499500 и т. д.
- сложность обмена ключами. Для применения необходимо решить проблему надёжной передачи ключей каждому абоненту, так как нужен секретный канал для передачи каждого ключа обоим сторонам.

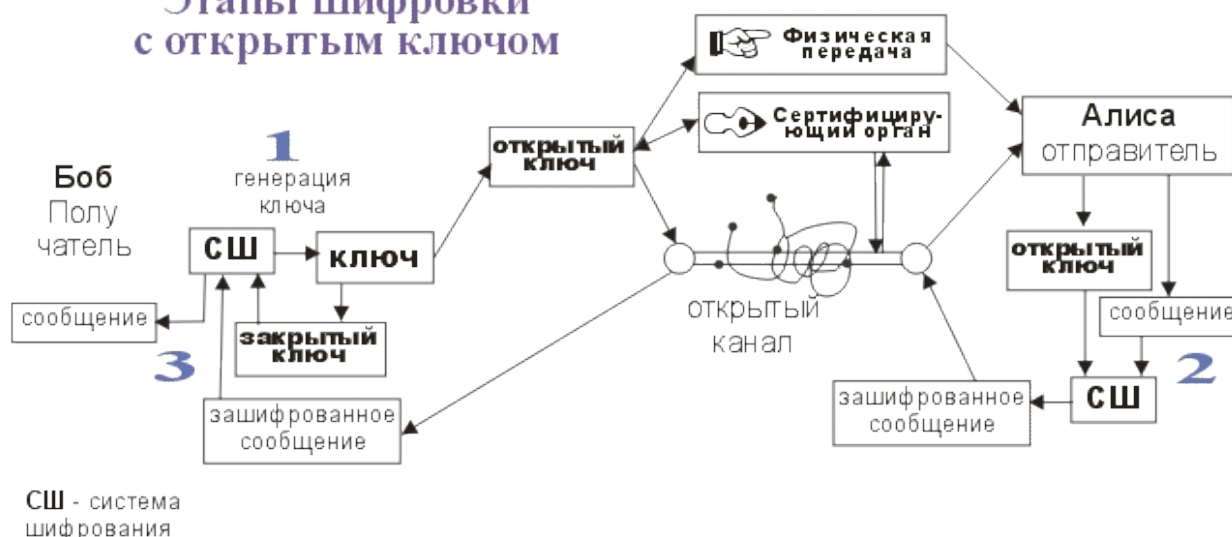
Для компенсации недостатков симметричного шифрования в настоящее время широко применяется комбинированная (гибридная) криптографическая схема, где с помощью асимметричного шифрования передаётся сеансовый ключ, используемый сторонами для обмена данными с помощью симметричного шифрования.

Важным свойством симметричных шифров является невозможность их использования для подтверждения авторства, так как ключ известен каждой стороне.

## **Асимметричное шифрование**

Криптографическая система с открытым ключом (или Асимметричное шифрование, Асимметричный шифр) — система шифрования и/или электронной цифровой подписи, при которой открытый ключ передаётся по открытому (то есть незащищённому, доступному для наблюдения) каналу, и используется для проверки ЭЦП и для шифрования сообщения. Для генерации ЭЦП и для расшифрования сообщения используется секретный ключ. Криптографические системы с открытым ключом в настоящее время широко применяются в различных сетевых протоколах, в частности, в протоколах TLS и его предшественнике SSL (лежащих в основе HTTPS), а так же SSH, PGP, S/MIME и т. д.

## Этапы шифровки с открытым ключом



1. Получатель генерирует 2 ключа. Один из них открытый, другой закрытый (секретный). При этом закрытый ключ не должен передаваться по открытому каналу, либо его подлинность должна быть гарантирована некоторым сертифицирующим органом.
2. Отправитель с помощью открытого ключа шифрует сообщение.
3. Получатель с помощью закрытого ключа дешифрует сообщение.

### Научная основа

Начало асимметричным шифрам было положено в 1976 году в работе Уитфилда Диффи и Мартина Хеллмана «Новые направления в современной криптографии». Они предложили систему обмена общим секретным ключом на основе проблемы дискретного логарифма. Вообще, в основу известных асимметричных криптосистем кладётся одна из сложных математических проблем, которая позволяет строить односторонние функции и функции-ловушки. Например, криптосистема Ривеста-Шамира-Адельмана использует проблему факторизации больших чисел, а криптосистемы Меркля-Хеллмана и Хора-Ривеста опираются на так называемую задачу об укладке рюкзака.

### Особенности системы

#### Преимущества

Преимущество асимметричных шифров перед симметричными шифрами состоит в отсутствии необходимости предварительной передачи секретного ключа по надёжному каналу. Сторона, желающая принимать зашифрованные тексты, в соответствии с используемым алгоритмом вырабатывает пару «открытый ключ — закрытый ключ». Значения ключей связаны между собой, однако вычисление закрытого ключа по открытому должно быть невозможным с практической точки зрения. Открытый ключ публикуется в открытых справочниках и используется для шифрования информации контрагентом, и проверки ЭЦП. Закрытый ключ держится в секрете и используется для расшифровывания сообщения, переданного владельцу пары ключей, и для создания ЭЦП. Для удостоверения аутентичности самих публичных ключей, передаваемых по открытому каналу или получаемых из справочника, обычно используют сертификаты.

#### Недостатки

В чистом виде асимметричные криптосистемы требуют существенно больших вычислительных ресурсов, потому на практике используются в сочетании с другими алгоритмами.

Для ЭЦП сообщение предварительно подвергается хешированию, а с помощью асимметричного ключа подписывается лишь относительно небольшой результат хеш-функции.

Для шифрования они используются в форме гибридных криптосистем, где большие объёмы данных шифруются симметричным шифром на сеансовом ключе, а с помощью асимметричного шифра передаётся только сам сеансовый ключ.

Хотя сообщения надёжно шифруются, но «засвечиваются» получатель и отправитель самим фактом пересылки зашифрованного сообщения.

### Виды асимметричных шифров

RSA

Elgamal

Rabin

ГОСТ Р 34.10-2001

## Установка и конфигурирование ssh

OpenSSH свободная версия набора программ ssh. Linux-версия есть на <http://violet.ibs.com.au/openssh> и во многих дистрибутивах Linux, включая Slackware. Здесь я не буду рассматривать компиляцию: подробные указания есть в пакете с исходными текстами. Если Вы можете установить программу из откомпилированных модулей, лучше всего это сделать.

Для работы ssh нужно два компонента: клиент ssh, который Вы должны сконфигурировать и запустить на локальном компьютере, и ssh daemon, который должен работать на удаленном компьютере, к которому нужно подключиться.

### Ssh daemon

Sshd daemon это программа, которая слушает сетевые подключения клиентов ssh, управляет авторизацией и выполняет запрошенную команду. Он имеет один основной файл конфигурации /etc/ssh/sshd\_config и специальный файл, содержащий ключ, используемый при авторизации и шифровании. Каждый компьютер и каждый пользователь имеет собственный ключ.

Утилита ssh-keygen генерирует случайные ключи. Это обычно используется один раз при установке для генерации главного ключа, который администратор системы обычно хранит в файле /etc/ssh/ssh\_host\_key. Ключи могут иметь длину 512 бит или больше. По умолчанию ssh-keygen генерирует ключи длиной 1024 бита, и большинство людей использует значение по умолчанию. Чтобы генерировать ключ, вызовите команду ssh-keygen:

```
ssh-keygen -f /etc/ssh/ssh_host_key
```

Вас спросят о фразе-пароле. Но ключи хостов такую фразу использовать не должны, так что оставьте ее пустой и нажмите Enter. Вывод программы будет выглядеть примерно так:

```
# ssh-keygen -f /etc/ssh/ssh_host_key
Generating public/private rsa key pair.
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /etc/ssh/ssh_host_key.
Your public key has been saved in /etc/ssh/ssh_host_key.pub.
The key fingerprint is:
62:67:e6:bd:94:cb:41:8a:c5:e8:38:06:a3:d6:af:e7 root@master
#
```

В конце сказано, что два файла были созданы. Первый называется секретным ключом, и должен быть сохранен в тайне. Он попал в файл /etc/ssh/ssh\_host\_key. Второй называется публичным ключом и должен распространяться. Он хранится в файле /etc/ssh/ssh\_host\_key.pub.

Теперь нужно создать файл конфигурации. Пакет ssh очень мощный, и файл конфигурации может содержать много параметров. Следующий код показывает безопасный и минимальный файл конфигурации для sshd. Остальные опции детально описаны на man-странице sshd(8):

```
# cat /etc/ssh/sshd_config

# The IP addresses to listen for connections on. 0.0.0.0 means all
# local addresses.
ListenAddress 0.0.0.0

# The TCP port to listen for connections on. The default is 22.
Port 22

# The name of the host key file.
HostKey /etc/ssh/ssh_host_key
```

```
# The length of the key in bits.
ServerKeyBits 1024

# Should we allow root logins via ssh?
PermitRootLogin no

# Should the ssh daemon check users' home directory and files permissions?
# are safe before allowing login?
StrictModes yes

# Should we allow old ~/.rhosts and /etc/hosts.equiv authentication method?
RhostsAuthentication no
# Should we allow pure RSA authentication?
RSAAuthentication yes
# Should we allow password authentication?
PasswordAuthentication yes

# Should we allow /etc/hosts.equiv combined with RSA host authentication?
RhostsRSAAuthentication no
# Should we ignore ~/.rhosts files?
IgnoreRhosts yes
# Should we allow logins to accounts with empty passwords?
PermitEmptyPasswords no
```

Очень важно правильно задать права доступа к файлу настроек. Используйте следующие команды для их задания:

```
# chown -R root:root /etc/ssh
# chmod 755 /etc/ssh
# chmod 600 /etc/ssh/ssh_host_key
# chmod 644 /etc/ssh/ssh_host_key.pub
# chmod 644 /etc/ssh/sshd_config
```

Заключительная стадия настройки sshd daemon это его запуск. Следует создать для него rc-файл или вписать вызов в уже существующий, чтобы сервис автоматически вызывался при начальной загрузке. Daemon выполняется автономно. Он должен быть выполнен от имени пользователя root. Синтаксис очень прост:

```
# /usr/sbin/sshd
```

sshd автоматически переходит в фоновый режим. Теперь он готов к приему соединений по протоколу ssh.

## Клиент ssh

Имеется ряд клиентов ssh: slogin, scp и ssh. Они используют один файл конфигурации, обычно `/etc/ssh/ssh_config`, и файлы настроек из каталога `.ssh` в домашнем каталоге пользователя, который их вызвал. Наиболее важные из этих файлов: `.ssh/config`, который может содержать параметры, отменяющие заданные в `/etc/ssh/ssh_config`, `.ssh/identity`, который содержит собственный секретный ключ пользователя и `.ssh/identity.pub`, содержащий публичный ключ пользователя. Другие важные файлы: `.ssh/known_hosts` и `.ssh/authorized_keys`; они будут рассмотрены ниже. Сначала надо создать глобальный файл настроек и файл ключа пользователя.

Файл `/etc/ssh/ssh_config` очень похож на файл настройки сервера. Есть большое количество свойств, которые Вы можете конфигурировать, но минимальная конфигурация приведена в примере 12-5. Остальная часть параметров рассмотрена на map-странице sshd(8). Вы можете добавлять секции для конкретных компьютеров или их групп. Параметром команды Host может быть любое полное имя компьютера или набор символов подстановки. В данном примере задано соответствие всем хостам. Можно, например, создать запись Host `*.vbrew.com`, соответствующую всем хостам в домене `vbrew.com`.

Пример 12-5. Клиентский файл настроек ssh# `/etc/ssh/ssh_config`

```
# Default options to use when connecting to a remote host
Host *
# Compress the session data?
Compression yes
# .. using which compression level? (1 - fast/poor, 9 - slow/good)
CompressionLevel 6
```



```
# Fall back to rsh if the secure connection fails?
FallBackToRsh no

# Should we send keep-alive messages? Useful if you use IP masquerade
KeepAlive yes

# Try RSA authentication?
RSAAuthentication yes
# Try RSA authentication in combination with .rhosts authentication?
RhostsRSAAuthentication yes
```

Я упомянул в разделе конфигурации сервера, что каждый компьютер и пользователь имеет ключ. Ключ пользователя сохранен в его файле `~/.ssh/identity`. Чтобы генерировать ключ, используйте команду `ssh-keygen`, но Вы не должны определять имя файла, в котором сохраняете ключ. По умолчанию `ssh-keygen` использует правильное расположение, но попросит Вас ввести имя файла в случае, если Вы хотите сохранить ключ в другом месте. Иногда полезно иметь много авторизационных файлов. Как и прежде `ssh-keygen` запросит у Вас пароль, что добавляет другой уровень защиты, и его стоит использовать. Пароль не будет отображен на экране, когда Вы его напечатаете.

**Внимание!** Пароль нельзя восстановить, если он забыт. Требования к паролю такие же, как и ко всем прочим паролям. Он должен быть длиной от 10 до 30 символов. Если пароль забыт, придется сгенерировать новый ключ.

Вы должны попросить каждого из Ваших пользователей, чтобы они выполнили `ssh-keygen` для правильного создания ключа. Команда `ssh-keygen` создаст их каталоги `~/.ssh/` с соответствующими правами доступа, а также их секретный и публичный ключи в `.ssh/identity` и `.ssh/identity.pub`, соответственно. Типовой сеанс:

```
stavr@master:~$ ssh-keygen
Generating public/private rsa key pair.
Enter file in which to save the key (/home/stavr/.ssh/id_rsa):
Created directory '/home/stavr/.ssh'.
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /home/stavr/.ssh/id_rsa.
Your public key has been saved in /home/stavr/.ssh/id_rsa.pub.
The key fingerprint is:
42:38:e7:17:a7:2f:87:d9:42:1c:71:4b:c6:29:d4:be stavr@master
stavr@master:~$
```

Теперь `ssh` готов к работе.

## Использование ssh

Теперь пакет `ssh` и связанные с ним команды установлены и готовы к выполнению.

Сначала опробуем удаленный вход в систему. Можно использовать программу `slogin`. Первый раз, когда Вы делаете попытку подключения к компьютеру, `ssh` получит публичный ключ хоста и запросит подтверждение с помощью электронной подписи `fingerprint`.

Администратор удаленного компьютера должен предоставить электронную подпись, которую Вы должны добавить к Вашему файлу `.ssh/known_hosts`. Если он этого не сделал, Вы можете соединиться с удаленным компьютером, но `ssh` предупредит Вас о возникшей ситуации. Допустим, что Вы уверены в том, что работаете с тем компьютером, с которым хотели связаться (никто не перехватил DNS). Тогда ответьте на запрос положительно. Соответствующий ключ будет автоматически сохранен в Вашем файле `.ssh/known_hosts` без повторных запросов. Если при следующей попытке подключения, публичный ключ с этого компьютера не соответствует тому, который сохранен, Вы будете предупреждены, потому что это представляет потенциальную опасность.

Первый вход в систему на удаленном компьютере будет выглядеть так:

```
root@cl:~# slogin stavr@master.unix.class
The authenticity of host 'master.unix.class (10.10.0.21)' can't be established.
RSA key fingerprint is 23:ef:8a:39:4f:4e:2e:81:e4:f1:29:09:72:ee:26:86.
Are you sure you want to continue connecting (yes/no)? yes
```

```
Warning: Permanently added 'master.unix.class,10.10.0.21' (RSA) to the list of
known hosts.
stavr@master.unix.class's password:
Last login: Fri Sep 12 14:28:10 2008 from 10.10.0.1
Linux 2.6.24.5-smp.
stavr@master:~$
```

На запрос о пароле нужно указать свой пароль на удаленной системе. Этот пароль не будет отображен на экране, когда Вы напечатаете его.

Без специальных параметров `slogin` будет пытаться войти с тем же идентификатором пользователя, который был применен на локальной машине. Вы можете отменить это, указанием параметра `-l` с именем для регистрации на удаленном компьютере.

Можно копировать файлы по сети командой `scp`. Синтаксис подобен стандартной `cp` за исключением того, что Вы можете определять имя машины перед именем файла, указывая, что файл находится на определенном компьютере. Следующий пример иллюстрирует синтаксис `scp`, копируя локальный файл `/tmp/media` в каталог `/home/stavr/` на удаленный компьютер `master.unix.class`:

```
root@c1:~# ls -l /media > /tmp/media
root@c1:~# scp /tmp/media stavr@master.unix.class:/home/stavr
stavr@master.unix.class's password:
media                                100% 1214      1.2KB/s   00:00
root@c1:~#
```

Проверяем:

```
root@master:~# ls -l /home/stavr/media
-rw-r--r-- 1 stavr users 1214 2008-09-12 14:44 /home/stavr/media
root@master:~#
```

Конечно, будет запрошен пароль. Команда `scp` по умолчанию отображает полезные сообщения о ходе процесса копирования. Вы можете копировать файл и с удаленного компьютера: просто определите имя хоста и путь к файлу как источник и локальный путь как адресат. Можно даже копировать файлы с удаленного компьютера на другой удаленный компьютер, но это не очень удобно, поскольку все данные пойдут транзитом через Ваш компьютер.

Вы можете выполнять команды на удаленных компьютерах, использующих `ssh`. Синтаксис очень прост. Допустим, пользователь `stavr` просматривает оглавление корневого каталога на машине `c1.unix.class`. Команда:

```
stavr@master:~$ ssh c1.unix.class ls -CF /
stavr@c1.unix.class's password:
bin/  dev/  home/  media/  opt/  root/  srv/  tmp/  var/
boot/ etc/  lib/   mnt/    proc/  sbin/  sys/  usr/
stavr@master:~$
```

Можно помещать `ssh` в поток команд с перенаправлением ввода-вывода ("трубопровод"). Все будет работать как и в любой другой команде Linux с той лишь разницей, что ввод-вывод `ssh` пойдет по сети через защищенное соединение.

Набор инструментальных средств `ssh` очень мощен и имеет многие другие полезные свойства и параметры. Для дальнейшего его изучения обратитесь к `man`-страницам и другой документации, которая поставляется вместе с пакетом.

## Глава 16. Поиск файлов в файловой системе

Бывает, что вы знаете, что файл или каталог существует, но не знаете, как его найти. Для поиска необходимого файла в Linux нет необходимости вручную рыться во множестве каталогов, т.к. для этого существуют быстрые и эффективные методы поиска.

### Программа *which*

Программа производит поиск файлов директориях, указанных в переменной PATH. В результате выводится полный путь к файлу. По умолчанию, программа останавливает поиск после нахождения первого соответствующего файла.

Опции:

- **-a**       показать все найденные файлы

```
root@c1:~# which ls
/usr/bin/ls
root@c1:~# which -a ls
/usr/bin/ls
/bin/ls
root@c1:~#
```

### Программа *whereis*

Программа производит поиск файлов директориях, указанных в переменной PATH, а также страницах документации manpages. В результате выводится полный путь к файлу.

Опции:

- **-b**       искать только в директориях, описанных в переменной PATH;
- **-m**       искать только в страницах документации manpages.

```
root@c1:~# whereis ls
ls: /bin/ls /usr/bin/ls /usr/X11R6/bin/ls /usr/bin/X11/ls /usr/X11/bin/ls
/usr/man/man1/ls.1.gz /usr/share/man/man1/ls.1.gz /usr/X11/man/man1/ls.1.gz
root@c1:~# whereis -b ls
ls: /bin/ls /usr/bin/ls /usr/X11R6/bin/ls /usr/bin/X11/ls /usr/X11/bin/ls
root@c1:~# whereis -m ls
ls: /usr/man/man1/ls.1.gz /usr/share/man/man1/ls.1.gz /usr/X11/man/man1/ls.1.gz
root@c1:~#
```

Программа *whereis* сообщила нам не только, где находится программа, а также и местонахождение документации к ней.

### Программа *locate*

Программа *locate* выполняет поиск по специальной базе данных, а не по самой файловой системе.

База данных содержит индексированную информацию о файлах в файловой системе. Создание и обновление этой базы данных производится с помощью программ *slocate* и *updatedb*.

Программы *locate* и *updatedb* обычно являются символическими ссылками на программу *slocate*.

Опции программы *slocate*:

- **-u**       создание базы данных начиная с корневой директории;
- **-e dir,dir...**   не включать в поиск указанные директории;
- **-f fs\_type,fs\_type...**   не включать в поиск указанные типы файловых систем;
- **-c**       читать конфигурационные параметры из файла */etc/updatedb.conf*.

```
root@c1:~# slocate -u -e /proc,/dev,/tmp,/var/tmp
root@c1:~# locate ls
```

```

.....
.....
/usr/share/xscreensaver/config/glcalls.xml
/usr/share/xscreensaver/config/glsnake.xml
/usr/share/xscreensaver/config/sballs.xml
/usr/sbin/alsaconf
/usr/sbin/alsactl
/usr/sbin/mailstats
/usr/libexec/gcc/i486-slackware-linux/4.2.3/install-tools
/usr/libexec/gcc/i486-slackware-linux/4.2.3/install-tools/fixinc.sh
/usr/libexec/gcc/i486-slackware-linux/4.2.3/install-tools/fixincl
/usr/libexec/gcc/i486-slackware-linux/4.2.3/install-tools/mkheaders
/usr/libexec/hal-ipw-killswitch-linux
/usr/libexec/xscreensaver/glcalls
/usr/libexec/xscreensaver/glslideshow
/usr/libexec/xscreensaver/glschool
/usr/libexec/xscreensaver/metaballs
/usr/libexec/xscreensaver/sballs
/usr/libexec/xscreensaver/pulsar
/usr/libexec/xscreensaver/glsnake
/usr/libexec/xscreensaver/fluidballs
/sbin/lsmmod
/sbin/lspci
/sbin/lspcmcia
/sbin/lssusb
root@c1:~# locate ls | grep /ls$
/bin/ls
/usr/bin/ls
/usr/share/locale/l10n/ls
/usr/share/epic/help/8_Scripts/ls
/usr/share/epic/script/ls
root@c1:~#

```

## Программа find

Команда `find` позволяет пользователю выполнять поиск по файловой системе с помощью большого набора поисковых аргументов. Пользователи могут искать файлы по именам с использованием шаблонов подстановки, диапазонов времени их изменения или создания и других расширенных свойств. Программа не использует базы данных и специальные переменные.

Общий синтаксис команды `find` имеет следующий вид:

`find [список_каталогов] критерий_поиска`

Параметр "список\_каталогов" определяет, где искать нужный файл. Проще всего задать в качестве начального каталога поиска корневой каталог `/`, однако, в таком случае поиск может затянуться очень надолго, так как будет просматриваться вся структура каталогов, включая смонтированные файловые системы (в том числе сетевые, если таковые есть). Можно сократить объем поиска, если задать вместо одного корневого каталога список из нескольких каталогов (естественно, тех, в которых может находиться искомый файл).

Началом "критерия\_поиска", определяющего, что именно должна искать программа `find`, считается первый аргумент, начинающийся на `-` ( ) , !. Все аргументы, предшествующие "критерию\_поиска", трактуются как имена каталогов, в которых надо производить поиск. Если не указано ни одного пути, поиск производится только в текущем каталоге и его подкаталогах.

### Критерии поиска:

- **-mount** или **-xdev** осуществлять поиск только в пределах одной физической файловой системы.
- **-name шаблон** поиск файла по его имени.
- **-iname шаблон** то же, что и `name`, но без учета регистра.
- **-regex шаблон** то же, что и `name`, но в качестве шаблона используется регулярное выражение.
- **-type тип\_файла** поиск файлов указанного типа.
- **-user login** искать файлы принадлежащие указанному пользователю.

- **-uid *UID*** искать файлы принадлежащие пользователю с указанным UID.
- **-nouser** искать файлы не принадлежащие ни одному пользователю.
- **-group *группа*** искать файлы принадлежащие указанной группе.
- **-gid *GID*** искать файлы принадлежащие группе с указанным GID.
- **-nogroup** искать файлы не принадлежащие ни к одной группе.
- **-atime *N*** искать файлы, доступ к которым был *N* суток назад.
- **-mtime *N*** искать файлы, которые изменялись *N* суток назад.
- **-size *N[bckw]*** искать файлы, размер которых *Nb* – *N* блоков, *Nc* – *N* байт, *Nk* – *N* килобайт.
- **-exec *программа*** выполнить указанную программу и передать ей имя файла.
- **-ok *программа*** то же, что и *exec*, но перед выполнением программы у пользователя спрашивается разрешение на ее выполнение.

Например:

```
root@c1:~# find /usr/share/doc /usr/doc -name readme
/usr/doc/samba-3.0.28a/examples/validchars/readme
root@c1:~#
```

Чтобы найти в системе файл `xinitrc`, можно воспользоваться следующей командой (`/dev/null` позволит не выводить на экран многочисленные сообщения “Отказано в доступе” при использовании команды от обычного пользователя):

```
stavr@c1:~$ find / -name xinitrc 2>/dev/null
/etc/X11/xinit/xinitrc
/etc/xfce/xdg/xfce4/xinitrc
stavr@c1:~$
```

Другие простые критерии вы можете узнать, если просмотрите man-страницу о команде `find`. Здесь же надо только сказать, что из простых критериев можно строить более сложные с помощью логических операций `and`, `or` или операции отрицания, знаком которой служит восклицательный знак. Например, если вы хотите найти все файлы, имена которых оканчиваются на `.txt` и `.doc`, то критерий можно записать как `(-name *.txt -or -name *.doc)`. Можно комбинировать таким образом любое число критериев (и не только простых!). Если операция не указана явно, то подразумевается `-and`, т. е. вместо `(-name *.txt -and -name *.doc)` можно записать просто `(-name *.txt -name *.doc)`. Если применяется только одна операция `-and` или `!`, то скобки обычно можно опустить, а с операцией `-or` и в сложных выражениях скобки необходимы. Перед скобкой нужно поставить обратную косую черту, а после скобки — пробел. Например, если вы хотите найти каталог по его имени, то можно сделать это следующей командой:

```
$ find /usr -name doc -type d
```

или (с соблюдением правил построения сложных критериев)

```
$ find /usr \( -name doc -and -type d \)
```

Чтобы закончить рассмотрение команды `find`, надо сказать еще о том, что после критерия поиска в этой команде можно сразу же задать операцию, которая будет применяться ко всем файлам, найденным по указанному критерию. Простейшим примером использования такой возможности является указание команды `-print`

```
root@c1:~# find /usr/bin -name *.sh -print
/usr/bin/gettext.sh
/usr/bin/lesspipe.sh
/usr/bin/lprsetup.sh
/usr/bin/pv.sh
/usr/bin/unix-lpr.sh
/usr/bin/winpopup-install.sh
/usr/bin/kopete_latexconvert.sh
/usr/bin/winpopup-send.sh
/usr/bin/kmail_clamav.sh
/usr/bin/kmail_sav.sh
/usr/bin/kmail_antivir.sh
/usr/bin/kmail_fprot.sh
/usr/bin/build-progress.sh
```

```
/usr/bin/isc-config.sh
/usr/bin/gpgsm-gencert.sh
/usr/bin/lp2pap.sh
/usr/bin/fontprop.sh
/usr/bin/fontname.sh
root@c1:~#
```

по которой выдается на экран список имен всех найденных файлов с указанием полного пути к файлу. Эта операция применяется по умолчанию, т. е. когда никаких операций вообще не указано (как это было во всех приведенных выше примерах).

Другим примером операции, применяемой ко всем найденным файлам, может служить операция `-exec cmd {} \;`, где `cmd` — произвольная команда оболочки `shell`. То есть в этом случае ко всем найденным файлам (их именами заменяются поочередно фигурные скобки) применяется команда `cmd`. За `cmd {}` в этом случае должна следовать точка с запятой, экранированная обратной косой чертой.

Например, если вы хотите удалить в текущем каталоге все файлы, к которым пользователи не обращались в течение 30 дней, дайте команду:

```
# find . -type f -atime +30 -exec rm {} \;
```

Вместо `-exec` можно поставить `-ok`, тогда перед выполнением указанной команды `cmd` применительно к каждому файлу будет запрашиваться подтверждение.

В общем, команда `find` является очень мощным, полезным и чрезвычайно адаптируемым инструментом поиска в файловой системе. Все ее возможности здесь не перечислены, изучайте соответствующую `man`-страницу. И будьте очень осторожны с применением таких возможностей команды, как вызов других команд, применяемых ко всем найденным файлам. Помните, что изменения часто необратимы!

## Глава 17. Архивация и резервное копирование

С программами архивирования приходится встречаться довольно часто, хотя бы потому, что вы будете встречать архивированные файлы в Интернете.

Основным средством архивирования в UNIX (а, следовательно, и в Linux) является комплекс из двух программ — `tar` и `gzip` (`bzip2`). Хотя никто не запрещает пользоваться `arj`, `pkzip`, `lha`, `rar` и т. д. — версии этих программ для Linux общедоступны. Просто уж исторически сложилось, что пользователи Unix чаще применяют именно `tar` и `gzip` (`bzip2`), и именно в таком формате распространяется большая часть программного обеспечения для Unix. Поэтому овладеть работой с `tar` и `gzip` (`bzip2`) — дело чести любого пользователя Linux.

### Программы для архивации данных

#### Программа `tar`

Вы, вероятно, привыкли к архиваторам типа `winrar` и `winzip`, которые собирают файлы в единый архив и сразу "сжимают" их. Естественно у вас может возникнуть вопрос "А зачем использовать две программы?" Все дело в том, что `tar` (Tape ARchiver), не сжимает данные, а лишь объединяет их в один файл (архив) для последующей записи на ленту. По умолчанию этот архивный файл создается на ленточном накопителе, точнее на устройстве `/dev/rmt0`. Если вы хотите создать архивный файл на диске, то необходимо использовать команду `tar` с опцией `f`, после которой указывается имя архивного файла.

У программы `tar` имеется 8 опций, отличающихся от остальных тем, что при вызове программы должна обязательно задаваться одна из этих опций. Эти опции определяют основные функции программы:

- `-A, --catenate, --concatenate`      Добавляет файлы в существующий архив
- `-c, --create`      Создает новый архив
- `-d, --diff, --compare`      Найти различия между архивом и файловой системой
- `--delete`      Удалить из архива (не может использоваться с магнитной лентой!)
- `-r, --append`      Дописывает файлы в конец архива
- `-t, --list`      Выводит список файлов архива
- `-u, --update`      Добавляет только файлы, которые новее, чем имеющаяся в архиве копия
- `-x, --extract, --get`      Извлечь файлы из архива

Если вы работаете с файлами архивов на дисках, а не с ленточным устройством, то, очевидно, обязательной будет и опция `f`. Другие опции не являются обязательными, они служат только для конкретизации задания программе. Например, опция `v` заставляет программу выводить список обрабатываемых файлов.

Однобуквенные опции программы `tar` могут перечисляться друг за другом (вы увидите это в приводимых ниже примерах).

Я не буду давать здесь описание всех опций команды `tar`, просто приведу несколько командных строк для выполнения самых необходимых действий с архивами.

Чтобы создать один `tar`-архив из нескольких файлов, используется команда:

```
tar -cf имя_архива файл1 файл2 ...
```

где опция `-c` сообщает программе, что необходимо создать (`create`) архив, а опция `f` говорит о том, что архив должен создаваться в виде файла (имя которого должно следовать сразу за этой опцией).

В именах файлов, которые сохраняются в архиве, можно использовать шаблоны имен файлов, в том числе просто символы-заместители `*` и `?`. Благодаря этому можно очень короткой командой отправить в архив сразу много файлов. Например, для того, чтобы создать архив, содержащий все файлы одного из подкаталогов (пусть это будет `sub_dir`) текущего каталога, достаточно дать команду

```
$ tar -cvf имя_архива ./sub_dir/*
```

или даже просто

```
$ tar -cvf имя_архива sub_dir
```

По этой команде в архиве будут сохранены не только файлы, расположенные непосредственно в подкаталоге sub\_dir, но и рекурсивно все файлы из подкаталогов каталога sub\_dir. При этом в архиве сохраняется вся структура подкаталогов каталога sub\_dir.

Заметим, что если в только что приведенном примере не указать имя подкаталога, то будут архивироваться все файлы (и подкаталоги) текущего каталога.

Например, вот некоторые действия по созданию архива:

```
root@c1:~# mkdir test
root@c1:~# cp /usr/share/doc/Linux-HOWTOs/X* test
root@c1:~# tar -cvf test.tar test
test/
test/X-Big-Cursor
test/XDMCP-HOWTO
test/XDM-Xterm
test/XFree86-HOWTO
test/XFree86-R200
test/XFree86-Second-Mouse
test/XFree86-Touch-Screen-HOWTO
test/XFree86-Video-Timings-HOWTO
test/XFree86-XInside
test/XFree-Local-multi-user-HOWTO
test/Xinerama-HOWTO
test/XML-RPC-HOWTO
test/Xterminals
test/Xterm-Title
test/XWindow-Overview-HOWTO
test/XWindow-User-HOWTO
root@c1:~# ls
bad_list  DEADJOE  loadlin16c.txt  loadlin16c.zip  test/  test.tar
root@c1:~#
```

Получить список файлов архива можно командой:

**tar -tf имя\_архива**

Вот так:

```
root@c1:~# tar -tf test.tar
test/
test/X-Big-Cursor
test/XDMCP-HOWTO
test/XDM-Xterm
test/XFree86-HOWTO
test/XFree86-R200
test/XFree86-Second-Mouse
test/XFree86-Touch-Screen-HOWTO
test/XFree86-Video-Timings-HOWTO
test/XFree86-XInside
test/XFree-Local-multi-user-HOWTO
test/Xinerama-HOWTO
test/XML-RPC-HOWTO
test/Xterminals
test/Xterm-Title
test/XWindow-Overview-HOWTO
test/XWindow-User-HOWTO
root@c1:~#
```

Или более детально:

```
root@c1:~# tar -tvf test.tar
drwxr-xr-x root/root          0 2008-09-16 18:32 test/
-rw-r--r-- root/root      9509 2008-09-16 18:32 test/X-Big-Cursor
-rw-r--r-- root/root    48517 2008-09-16 18:32 test/XDMCP-HOWTO
```



```

-rw-r--r-- root/root      54271 2008-09-16 18:32 test/XDM-Xterm
-rw-r--r-- root/root      15763 2008-09-16 18:32 test/XFree86-HOWTO
-rw-r--r-- root/root      38994 2008-09-16 18:32 test/XFree86-R200
-rw-r--r-- root/root      10172 2008-09-16 18:32 test/XFree86-Second-Mouse
-rw-r--r-- root/root       9367 2008-09-16 18:32 test/XFree86-Touch-Screen-HOWTO
-rw-r--r-- root/root      81428 2008-09-16 18:32 test/XFree86-Video-Timings-HOWTO
-rw-r--r-- root/root      15408 2008-09-16 18:32 test/XFree86-XInside
-rw-r--r-- root/root     400281 2008-09-16 18:32 test/XFree-Local-multi-user-
HOWTO
-rw-r--r-- root/root      36250 2008-09-16 18:32 test/Xinerama-HOWTO
-rw-r--r-- root/root      58754 2008-09-16 18:32 test/XML-RPC-HOWTO
-rw-r--r-- root/root      22609 2008-09-16 18:32 test/Xterminals
-rw-r--r-- root/root      15872 2008-09-16 18:32 test/Xterm-Title
-rw-r--r-- root/root      30026 2008-09-16 18:32 test/XWindow-Overview-HOWTO
-rw-r--r-- root/root     157838 2008-09-16 18:32 test/XWindow-User-HOWTO
root@c1:~#

```

Теперь вы знаете как создать архив, а для того, чтобы распаковать (извлечь) файлы из архива, нужно дать команду:

**tar -xvf имя\_архива файлы**

```

root@c1:~# rm -R test
root@c1:~# ls
bad_list DEADJOE loadlin16c.txt loadlin16c.zip test.tar
root@c1:~# tar -xvf test.tar
test/
test/X-Big-Cursor
test/XDMCP-HOWTO
test/XDM-Xterm
test/XFree86-HOWTO
test/XFree86-R200
test/XFree86-Second-Mouse
test/XFree86-Touch-Screen-HOWTO
test/XFree86-Video-Timings-HOWTO
test/XFree86-XInside
test/XFree-Local-multi-user-HOWTO
test/Xinerama-HOWTO
test/XML-RPC-HOWTO
test/Xterminals
test/Xterm-Title
test/XWindow-Overview-HOWTO
test/XWindow-User-HOWTO
root@c1:~# ls
bad_list DEADJOE loadlin16c.txt loadlin16c.zip test/ test.tar
root@c1:~# ls test
X-Big-Cursor          XFree86-Touch-Screen-HOWTO  Xterminals
XDMCP-HOWTO           XFree86-Video-Timings-HOWTO  Xterm-Title
XDM-Xterm             XFree86-XInside              XWindow-Overview-HOWTO
XFree86-HOWTO         XFree-Local-multi-user-HOWTO  XWindow-User-HOWTO
XFree86-R200          Xinerama-HOWTO
XFree86-Second-Mouse  XML-RPC-HOWTO
root@c1:~#

```

Программа tar является удобным средством для создания резервных копий файлов. Конечно, существуют специальные утилиты резервного архивирования, но даже если вы о них еще не знаете, то по меньшей мере, вы можете сделать следующее:

```
tar -Mcvf /dev/fd0H1440 /каталог
```

Такая команда создаст на дискетах архив с содержимым каталога, разбивая его на тома. Монтировать дискеты перед запуском команды не нужно, программа просто пишет на устройство потоком (в данном случае на дискету по секторам). При этом никакой файловой системы на дискете не создается. После заполнения дискеты вам будет выдан запрос на смену дискеты. Только, прежде чем запускать такую команду на выполнение, приготовьте достаточное число свободных дискет (помните, что tar не сжимает файлы), которые лучше всего

соответствующим образом пометить и обязательно пронумеровать. Кроме того, имейте в виду, что вся информация на дискетах будет молча уничтожена.

Чтобы восстановить сохраненные данные, воспользуйтесь командой:

```
tar -Mxpvf /dev/fd0H1440
```

Если вы ошибетесь в порядке вставляемых дискет, программы сообщит вам об этом и попросит заменить том.

В заключение раздела заметим, что всегда можно получить подсказку по использованию программы tar, дав команду

```
tar --help
```

При этом, если вы используете русифицированный или локализованный дистрибутив Linux то подсказка будет выдаваться по-русски.

## Программа cpio

Утилита cpio - это мощная производная от tar. cpio работает как с архивами tar так и с другими форматами и, кроме того, имеет множество встроенных опций. Cpio можно использовать с большим количеством опций, позволяющих отфильтровать скопированные файлы. Cpio также имеет дополнительные опции, поддерживающие удаленное резервное копирование (вместо того, чтобы использовать канал с применением csh и др.) Главное преимущество, которое имеет cpio по сравнению с tar в том, что вы можете как добавить файлы к уже имеющемуся архиву, так и удалить файлы из архива. Архив cpio может быть многотомным.

Режимы работы программы cpio:

- **-i** (*copy in*) **cpio -i** выбирает файлы из стандартного входа.
- **-o** (*copy out*) **cpio -o** читает список имен файлов из стандартного входа и копирует их в стандартный выход.
- **-p** (*pass*) **cpio -p** читает список имен файлов из стандартного входа.

Опции:

- **-a** Сбрасывает времена обращения к входным файлам после их копирования. Времена обращения не сбрасываются для связанных файлов при указании опций cpio -pla (взаимоисключающих с опцией -m).
- **-A** Добавляет файлы в архив. Опция -A требует обязательного указания опции -O. Допустима только для архивов в файлах, на дискетах или на разделах жесткого диска.
- **-b** Изменяет порядок байтов в каждом слове (Используется только с опцией -i.)
- **-B** Выполняет ввод/вывод в виде блоков-записей по 5120 байтов. Если не указана эта опция и опция -C, используется стандартный размер блока - 8192 байтов. Опция -B не применима в режиме передачи; -B имеет смысл только при обмене данными со специальным символьным устройством, например, /dev/rmt/0m.
- **-c** Читает и записывает информацию заголовка в форме ASCII-текста для переносимости. Этот формат заголовка не накладывает никаких ограничений на значения UID или GID. Эта опция используется при обмене архивами между SVR4-системами; для обмена между заранее неизвестными машинами имеет смысл использовать опцию -H odc. Опция -c предполагает использование расширенных номеров устройств, которые поддерживаются только в SVR4-системах. При переносе файлов между SunOS 4 или Interactive UNIX и ОС Solaris 2.6 или более поздними версиями этой ОС используйте опцию -H odc.
- **-C размер\_буфера** Объединяет ввод/вывод в записи указанного размера (размер задается положительным целым числом). Стандартный размер буфера, если не указана эта опция и опция -B, - 8192 байта. Опция -C имеет смысл только если происходит обмен данными со специальным символьным устройством, например, /dev/rmt/0m.
- **-d** Создает каталоги при необходимости.
- **-E файл** Задает входной файл, содержащий список имен файлов, которые необходимо извлечь из архива (по одному имени в строке).
- **-f** Копирует все файлы, за исключением соответствующих шаблону. (Описание шаблонов см. в разделе ОПЕРАНДЫ.)

- **-H формат** Читает или записывает заголовок в указанном формате. Всегда используйте эту опцию или опцию **-с**, если исходная и целевая машины - разных типов (взаимоисключающих, как **-с** и **-6**). Допустимы следующие значения формата:
  - **bar** Заголовок и формат bar. Используется вместе с опцией **-i** (только для чтения).
  - **crc|CRC** Заголовок ASCII с расширенными номерами и дополнительной контрольной суммой для каждого файла. Этот формат заголовка не предполагает никаких ограничений на значения UID или GID.
  - **odc** Заголовок ASCII с небольшими номерами устройств. Этот заголовок и формат архива **сrio** соответствует стандарту IEEE/P1003 Data Interchange Standard. Это - наиболее переносимый из всех форматов заголовков. Этот формат официально принят для обмена файлами между системами, соответствующими стандарту POSIX (см. standards(5)). Используйте этот формат для обмена информацией с SunOS 4 и Interactive UNIX. Этот формат заголовка поддерживает хранение в заголовке идентификаторов UID и GID со значениями вплоть до 262143.
  - **tar|TAR** Заголовок и формат tar. Этот формат заголовка позволяет хранить в заголовке значения UID и GID со значениями вплоть до 2097151.
  - **ustar|USTAR** Заголовок и формат tar, соответствующий стандарту IEEE/P1003 Data Interchange Standard. Этот формат заголовка позволяет хранить в заголовке значения UID и GID со значениями вплоть до 2097151.

Файлы с идентификаторами UID и GID, большими указанного выше предела, попадают в архив как имеющие UID и GID 60001. Для переноса больших файлов (8 Гбайт - 1 байт) можно использовать только заголовки форматов tar/TAR, ustar/USTAR или odc.
- **-I файл** Читает содержимое файла как входной архив. Если указан файл специального символического устройства и текущий носитель полностью прочитан, замените носитель и нажмите клавишу Enter для чтения следующего носителя. Эта опция используется только совместно с опцией **-i**.
- **-k** Пытается пропустить поврежденные заголовки файлов и игнорировать получаемые ошибки ввода/вывода. Если необходимо избирательно скопировать файлы с поврежденного носителя, эта опция позволяет прочитать файлы с неповрежденными заголовками. (Если архив **сrio** содержит другие архивы **сrio**, в случае ошибки **сrio** может закончить работу досрочно. **сrio** найдет следующий неповрежденный заголовок, и если это окажется заголовок вложенного архива, завершит работу, обнаружив признак завершения вложенного архива.) Используется только совместно с опцией **-i**.
- **-l** По возможности связывает файлы, а не копирует их. (Используется только с опцией **-р**.)
- **-L** Следует по символическим связям. По умолчанию утилита не следует по символическим связям.
- **-m** Оставляет прежнее время изменения файла. Эта опция не действует для копируемых каталогов (эта опция и опция **-а** - взаимоисключающие).
- **-М сообщение** Задаёт сообщение, выдаваемое при смене носителей. При задании специального символического устройства в опциях **-O** или **-I**, эта опция позволяет задать сообщение, выдаваемое когда достигнут конец носителя. В сообщении можно один раз включить шаблон **%d**, заменяемый последовательным номером следующего носителя.
- **-O файл** Направляет результат работы **сrio** в файл. Если файл является специальным символическим устройством и текущий носитель заполнен, замените носитель и нажмите клавишу Enter для продолжения работы со следующим носителем. Используется только с опцией **-o**.
- **-P** Сохраняет списки контроля доступа (ACL). Если эта опция используется при выдаче информации, существующие списки ACL записываются вместе с другими атрибутами в стандартный выходной поток. Списки ACL создаются как специальные файлы соответствующего типа. Если опция задана при считывании информации, существующие в архиве списки ACL извлекаются вместе с другими атрибутами из стандартного входного потока. Опция распознает файлы соответствующего специального типа. Учтите, что при попытке извлечения файлов из архива со списками ACL предыдущими версиями **сrio** произойдет ошибка. Эту опцию не надо использовать вместе с опцией **-с**, поскольку списки ACL поддерживаются не во всех системах, и поэтому не переносимы. Для обеспечения переносимости используйте заголовки ASCII.
- **-r** Интерактивно переименовывает файлы. Если пользователь просто нажимает Enter, файл пропускается. Если пользователь вводит точку (**.**), оставляется исходное имя файла. (Опция не поддерживается для **сrio -р**.)

- **-R id** Устанавливает пользователя id (и его первичную группу) в качестве владельца каждого файла (id должно быть одним из известных регистрационных имен в файле /etc/passwd). Эту опцию может задавать только пользователь root.
- **-s** Переставляет байты в каждом полуслове.
- **-S** Переставляет полуслова в каждом слове.
- **-t** Выдает содержимое входного архива. Никакие файлы при этом не создаются (эта опция взаимоисключающая с опцией -V).
- **-u** Копирует безусловно (обычно старый файл не заменяет более новый с тем же именем).
- **-v** Выдача дополнительной информации. Выдается список имен обрабатываемых файлов. При использовании с опцией -t, список имен файлов выглядит аналогично результатам команды ls -l.
- **-V** Выдача дополнительной информации специального вида. Выдается точка для каждого прочитанного или записанного файла. Позволяет пользователю убедиться, что процесс cpio работает, не выдавая при этом имена всех файлов.
- **-b** Обрабатывает формат файла архива UNIX System Sixth Edition. Используется только совместно с опцией -i (взаимоисключающая опция по отношению к опциям -c и -H).

### Создание архива

`cpio -o [опции] < file_list > archive`

На стандартный ввод подается список файлов, которые необходимо добавить в архив. Архив выдается на стандартный вывод. Пример создания архива:

```
root@c1:~# find test -print | cpio -o > test.cpio
cpio: test
cpio: test/X-Big-Cursor
cpio: test/XDMCP-HOWTO
cpio: test/XDM-Xterm
cpio: test/XFree86-HOWTO
cpio: test/XFree86-R200
cpio: test/XFree86-Second-Mouse
cpio: test/XFree86-Touch-Screen-HOWTO
cpio: test/XFree86-Video-Timings-HOWTO
cpio: test/XFree86-XInside
cpio: test/XFree-Local-multi-user-HOWTO
cpio: test/Xinerama-HOWTO
cpio: test/XML-RPC-HOWTO
cpio: test/Xterminals
cpio: test/Xterm-Title
cpio: test/XWindow-Overview-HOWTO
cpio: test/XWindow-User-HOWTO
1965 blocks
root@c1:~# ls
bad_list  DEADJOE  loadlin16c.txt  loadlin16c.zip  test/  test.cpio  test.tar
root@c1:~#
```

### Извлечение данных из архива

`cpio -i [опции] [шаблон] < archive`

Файл с архивом передают на стандартный ввод.

Если необходимо извлечь только определенные файлы, указывайте шаблон или список файлов разделенных пробелами.

```
root@c1:~# cpio -it < test.cpio
test
test/X-Big-Cursor
test/XDMCP-HOWTO
test/XDM-Xterm
test/XFree86-HOWTO
test/XFree86-R200
test/XFree86-Second-Mouse
```

```

test/XFree86-Touch-Screen-HOWTO
test/XFree86-Video-Timings-HOWTO
test/XFree86-XInside
test/XFree-Local-multi-user-HOWTO
test/Xinerama-HOWTO
test/XML-RPC-HOWTO
test/Xterminals
test/Xterm-Title
test/XWindow-Overview-HOWTO
test/XWindow-User-HOWTO
1965 blocks
root@cl:~# cpio -ivt < test.cpio
drwxr-xr-x  2 root    root          0 Sep 16 18:32 test
-rw-r--r--  1 root    root       9509 Sep 16 18:32 test/X-Big-Cursor
-rw-r--r--  1 root    root      48517 Sep 16 18:32 test/XDMCP-HOWTO
-rw-r--r--  1 root    root      54271 Sep 16 18:32 test/XDM-Xterm
-rw-r--r--  1 root    root      15763 Sep 16 18:32 test/XFree86-HOWTO
-rw-r--r--  1 root    root     38994 Sep 16 18:32 test/XFree86-R200
-rw-r--r--  1 root    root     10172 Sep 16 18:32 test/XFree86-Second-Mouse
-rw-r--r--  1 root    root      9367 Sep 16 18:32 test/XFree86-Touch-Screen-HOWTO
-rw-r--r--  1 root    root     81428 Sep 16 18:32 test/XFree86-Video-Timings-HOWTO
-rw-r--r--  1 root    root     15408 Sep 16 18:32 test/XFree86-XInside
-rw-r--r--  1 root    root    400281 Sep 16 18:32 test/XFree-Local-multi-user-HOWTO
-rw-r--r--  1 root    root     36250 Sep 16 18:32 test/Xinerama-HOWTO
-rw-r--r--  1 root    root     58754 Sep 16 18:32 test/XML-RPC-HOWTO
-rw-r--r--  1 root    root     22609 Sep 16 18:32 test/Xterminals
-rw-r--r--  1 root    root     15872 Sep 16 18:32 test/Xterm-Title
-rw-r--r--  1 root    root     30026 Sep 16 18:32 test/XWindow-Overview-HOWTO
-rw-r--r--  1 root    root    157838 Sep 16 18:32 test/XWindow-User-HOWTO
1965 blocks
root@cl:~# rm -R test
root@cl:~# ls
bad_list  DEADJOE  loadlin16c.txt  loadlin16c.zip  test.cpio  test.tar
root@cl:~# cpio -id < test.cpio
1965 blocks
root@cl:~# ls test
X-Big-Cursor          XFree86-Touch-Screen-HOWTO      Xterminals
XDMCP-HOWTO           XFree86-Video-Timings-HOWTO     Xterm-Title
XDM-Xterm             XFree86-XInside                 XWindow-Overview-HOWTO
XFree86-HOWTO         XFree-Local-multi-user-HOWTO    XWindow-User-HOWTO
XFree86-R200          Xinerama-HOWTO
XFree86-Second-Mouse XML-RPC-HOWTO
root@cl:~#

```

### Копирование файлов

`cpio -p [опции] dir < file_list`

Копирование файлов, определенных в списке `file_list`, в директорию `dir` с сохранением путей к ним.

Программа `cpio` по умолчанию не создает необходимых директорий. Используйте опцию `-d` для автоматического создания директорий.

```

root@cl:~# find /usr -name *.gif | cpio -pd gifs
9117 blocks
root@cl:~# ls
bad_list  gifs/          loadlin16c.zip  test.cpio
DEADJOE  loadlin16c.txt test/           test.tar
root@cl:~# ls -R gifs
.....
.....
gifs/usr/share/swat/using_samba/figs:
sam2_0101.gif  sam2_0311.gif  sam2_0339.gif  sam2_0406.gif  sam2_0803.gif
sam2_0102.gif  sam2_0312.gif  sam2_0340.gif  sam2_0407.gif  sam2_0804.gif
sam2_0103.gif  sam2_0313.gif  sam2_0341.gif  sam2_0408.gif  sam2_0805.gif
sam2_0104.gif  sam2_0314.gif  sam2_0342.gif  sam2_0409.gif  sam2_0806.gif
sam2_0105.gif  sam2_0315.gif  sam2_0343.gif  sam2_0410.gif  sam2_0807.gif

```

```
sam2_0106.gif sam2_0316.gif sam2_0344.gif sam2_0411.gif sam2_0808.gif
sam2_0107.gif sam2_0317.gif sam2_0345.gif sam2_0412.gif sam2_0809.gif
sam2_0108.gif sam2_0318.gif sam2_0346.gif sam2_0413.gif sam2_0810.gif
sam2_0109.gif sam2_0319.gif sam2_0347.gif sam2_0414.gif sam2_0811.gif
sam2_0110.gif sam2_0320.gif sam2_0348.gif sam2_0415.gif sam2_0901.gif
sam2_0111.gif sam2_0321.gif sam2_0349.gif sam2_0416.gif sam2_0902.gif
sam2_0112.gif sam2_0322.gif sam2_0350.gif sam2_0417.gif sam2_0903.gif
sam2_0113.gif sam2_0323.gif sam2_0351.gif sam2_0501.gif sam2_1001.gif
sam2_0114.gif sam2_0324.gif sam2_0352.gif sam2_0502.gif sam2_1002.gif
sam2_0201.gif sam2_0325.gif sam2_0353.gif sam2_0503.gif sam2_1003.gif
sam2_0202.gif sam2_0326.gif sam2_0354.gif sam2_0504.gif sam2_1004.gif
sam2_0203.gif sam2_0327.gif sam2_0355.gif sam2_0505.gif sam2_1005.gif
sam2_0204.gif sam2_0328.gif sam2_0356.gif sam2_0506.gif sam2_1006.gif
sam2_0301.gif sam2_0329.gif sam2_0357.gif sam2_0507.gif sam2_1101.gif
sam2_0302.gif sam2_0330.gif sam2_0358.gif sam2_0508.gif sam2_1102.gif
sam2_0303.gif sam2_0331.gif sam2_0359.gif sam2_0601.gif sam2_af01.gif
sam2_0304.gif sam2_0332.gif sam2_0360.gif sam2_0602.gif sam2_af02.gif
sam2_0305.gif sam2_0333.gif sam2_0361.gif sam2_0603.gif sam2_af03.gif
sam2_0306.gif sam2_0334.gif sam2_0401.gif sam2_0604.gif sam2_af04.gif
sam2_0307.gif sam2_0335.gif sam2_0402.gif sam2_0605.gif sam2_af05.gif
sam2_0308.gif sam2_0336.gif sam2_0403.gif sam2_0701.gif sam2_af06.gif
sam2_0309.gif sam2_0337.gif sam2_0404.gif sam2_0801.gif sam2_af07.gif
sam2_0310.gif sam2_0338.gif sam2_0405.gif sam2_0802.gif
```

```
gifs/usr/share/xml:
docbook/
```

```
gifs/usr/share/xml/docbook:
xsl-stylesheets-1.73.2/
```

```
gifs/usr/share/xml/docbook/xsl-stylesheets-1.73.2:
images/
```

```
gifs/usr/share/xml/docbook/xsl-stylesheets-1.73.2/images:
caution.gif important.gif note.gif tip.gif warning.gif
home.gif next.gif prev.gif up.gif
```

```
gifs/usr/src:
linux-2.6.24.5/
```

```
gifs/usr/src/linux-2.6.24.5:
Documentation/
```

```
gifs/usr/src/linux-2.6.24.5/Documentation:
logo.gif
root@c1:~#
```

## Сжатие (компрессия) данных

Хотя программа tar создает архивы, она, как было сказано, не сжимает архивы, а просто соединяет отдельные файлы в единый архивный файл. Для сжатия этого файла применяют программы компрессии. Мир Unix очень консервативен, и многим его «жителям» показалось привычным использовать программу tar. Тем более что ленточные накопители, возможно, еще долго будут использоваться для систем резервного копирования. И по этой причине появившиеся программы сжатия данных чаще всего работают с одним файлом, а не с целыми каталогами.

Файлам, сжатым с помощью программы bzip2, принято давать расширение .bz2, файлам, сжатым с помощью программы gzip, принято давать расширение .gz и файлам, сжатым с помощью программы compress, принято давать расширение .Z.

Для декомпрессии файлов, сжатых с помощью программы gzip, используется программа gunzip, файлов, сжатых с помощью программы bzip2, используется программа bunzip2 (bzip2) и файлов, сжатых с помощью программы compress, используется программа uncompress.

## Программа gzip

gzip это GNU программа сжатия. Она берёт один файл и сжимает его. Пример обычного использования выглядит вот так:

### gzip infile

Выходной файл будет назван *infile.gz* и почти всегда будет меньше входного. Обратите внимание, что *infile.gz* заменит *infile*. Это значит, что *infile* прекратит своё существование. Останется только его сжатая копия. Обычные текстовые файлы существенно сожмутся, в то время как jpeg картинки, mp3, и другие подобные файлы почти не сожмутся, так как они уже сжаты. Приведённый выше пример, это нечто среднее между качеством сжатия и затраченным временем. Максимальное сжатие может быть получено при помощи такой команды:

```
$ gzip -9 infile
```

Это займёт больше времени, но выходной файл будет настолько сжатым, насколько gzip вообще может его сжать. Использование меньших значений займёт меньше времени, но соответственно и качество компрессии будет хуже.

Еще один вариант команды для сжатия с ключом -c, перенаправляющим стандартный вывод:

```
$ gzip -c infile > infile.gz
```

В этом случае исходный файл остается неизменным.

Распаковывание gzipped (запакованных GNU zip) файлов может быть выполнено при помощи двух команд, которые на самом деле являются одной и той же программой. gzip распакует любой файл с узнаваемым им расширением. Вот список расширений, которые узнаёт команда: .gz, -gz, .z, -z, .Z, или -Z. Первый метод - применить команду gunzip к файлу:

```
$ gunzip infile.gz
```

Выполнение этой команды приведёт к тому, что вместо указанного файла в этом же каталоге появится его распакованная версия и .gz часть его имени исчезнет.

Второй метод распаковывания gzipped файла, это вызвать gzip в применении к файлу:

```
$ gzip -d infile.gz
```

Это приведёт к точно такому же результату, как и вызов gunzip.

Поэтому вы можете на ваше усмотрение пользоваться любой из команд, для распаковки gzipped файлов.

```
root@c1:~# ls
bad_list  gifs/                loadlin16c.zip  test.cpio
DEADJOE   loadlin16c.txt      test/           test.tar
root@c1:~# gzip test.tar
root@c1:~# ls
bad_list  gifs/                loadlin16c.zip  test.cpio
DEADJOE   loadlin16c.txt      test/           test.tar.gz
root@c1:~# gunzip test.tar.gz
root@c1:~# ls
bad_list  gifs/                loadlin16c.zip  test.cpio
DEADJOE   loadlin16c.txt      test/           test.tar
root@c1:~# gzip -c test.tar > test.tar.gz
root@c1:~# ls
bad_list  gifs/                loadlin16c.zip  test.cpio  test.tar.gz
DEADJOE   loadlin16c.txt      test/           test.tar
```

## Программа bzip2

bzip2 это альтернативная программа сжатия. Она использует алгоритм отличный от gzip, который имеет как преимущества, так и недостатки. Главное преимущество bzip2 это размер сжатых файлов. bzip2 почти всегда сожмёт лучше, чем gzip. Иногда файлы получаются гораздо меньше, чем файлы сжатые gzip-ом. Это может быть значительным преимуществом для людей, с медленным internet-соединением.

Недостаток bzip2 в том, что она более интенсивно использует CPU, чем gzip. А это значит, что использование bzip займёт больше времени и будет более требовательно к процессору, чем gzip. Когда вы решаете, каким архиватором пользоваться, надо взвесить это соотношение скорость - сжатие, и выбрать, что важнее.

Использование bzip2 очень похоже на использование gzip, так что мы не станем много времени тратить на её обсуждение. Просто вызовите bzip2, указав имя файла:

### bzip2 infile

Вывод обычно будет меньше, чем входной файл, и получит название infile.bz2. Как и с gzip, входной файл будет заменён сжатым.

Вы можете так же указывать числовой аргумент, чтобы балансировать скоростью и качеством сжатия, как и с gzip. Следующий пример показывает, как достигнуть максимального сжатия при помощи bzip2:

```
$ bzip2 -9 infile
```

Ключ -с, перенаправляющий стандартный вывод, работает и здесь:

```
$ bzip2 -c infile > infile.bz2
```

В этом случае исходный файл также остается неизменным.

Есть два способа распаковывания файлов, заканчивающихся .bz2 расширением, как и с gzip. Вы можете использовать bzip2 или bunzip2 для распаковки bziped файлов. Использование bzip2 потребует указания аргумента:

```
$ bzip2 -d infile.bz2
```

Эта команда распакует bziped файл и заменит его распакованной копией. Этот результирующий файл потеряет .bz2 расширение. Аналогично, вы можете использовать bunzip2 для распаковки файла:

```
$ bunzip2 infile.bz2
```

все произведённые программой действия будут абсолютно идентичными.

```
root@c1:~# bzip2 -c test.tar > test.tar.bz2
root@c1:~# ls
bad_list  gifs/          loadlin16c.zip  test.cpio  test.tar.bz2
DEADJOE  loadlin16c.txt test/           test.tar   test.tar.gz
root@c1:~#
```

## Программа compress

Программа compress является старейшей программой для сжатия файлов. В существующих unix-like системах она оставлена так как входит в стандарт POSIX и является обязательным инструментом.

Все действия с программой аналогичны описанным для программ gzip и bzip2.

Сжатие файла:

### compress infile

Использование перенаправления стандартного вывода:

```
$ compress -c infile > infile.Z
```

Распаковывание сжатого файла:

```
$ uncompress infile.Z
```

или

```
$ compress -d infile.Z
```

```
root@c1:~# compress -c test.tar > test.tar.Z
root@c1:~# ls
bad_list  gifs/          loadlin16c.zip  test.cpio  test.tar.bz2  test.tar.Z
DEADJOE  loadlin16c.txt test/           test.tar   test.tar.gz
root@c1:~#
```

## Сравнение программ сжатия

```
root@c1:~# ls -l test.tar*
-rw-r--r-- 1 root root 1024000 2008-09-16 18:33 test.tar
-rw-r--r-- 1 root root 176893 2008-09-17 19:42 test.tar.bz2
-rw-r--r-- 1 root root 226929 2008-09-17 19:34 test.tar.gz
```



```
-rw-r--r-- 1 root root 296410 2008-09-17 19:51 test.tar.Z
root@c1:~#
```

## Вызов программ сжатия из программы tar

Вы наверняка согласитесь со мной, что последовательное использование двух программ для процедур архивации и сжатия очень неудобно. Очень хотелось бы выполнять вместо двух процедур — лишь одну. И такое решение есть!

Современная программа tar довольно гибка. В ее арсенале появились средства с помощью которых возможно вызвать практически любую программу компрессии в процессе ее выполнения.

Более того, программа tar во многих своих реализациях, включая дистрибутив Slackware Linux может сама, используя специфическую часть файла оставленную программой сжатия, вызвать эту программу во время своей работы.

Опции программы tar для вызова программы компрессии:

- **-Z**      вызов программы compress;
- **-z**      вызов программы gzip;
- **-j**      вызов программы bzip2.

Пример:

### извлечение данных

```
root@c1:~# rm -r test
root@c1:~# ls
bad_list  gifs/          loadlin16c.zip  test.tar        test.tar.gz
DEADJOE  loadlin16c.txt test.cpio        test.tar.bz2    test.tar.Z
root@c1:~# tar -xzvf test.tar.gz
test/
test/X-Big-Cursor
test/XDMCP-HOWTO
test/XDM-Xterm
test/XFree86-HOWTO
test/XFree86-R200
test/XFree86-Second-Mouse
test/XFree86-Touch-Screen-HOWTO
test/XFree86-Video-Timings-HOWTO
test/XFree86-XInside
test/XFree-Local-multi-user-HOWTO
test/Xinerama-HOWTO
test/XML-RPC-HOWTO
test/Xterminals
test/Xterm-Title
test/XWindow-Overview-HOWTO
test/XWindow-User-HOWTO
root@c1:~# ls
bad_list  gifs/          loadlin16c.zip  test.cpio  test.tar.bz2  test.tar.Z
DEADJOE  loadlin16c.txt test/           test.tar   test.tar.gz
```

### создание сжатого архива

```
root@c1:~# tar -cjvf gifs.tar.bz2 gifs
.....
.....
gifs/usr/share/swat/images/samba.gif
gifs/usr/share/swat/images/shares.gif
gifs/usr/share/swat/images/wizard.gif
gifs/usr/share/swat/images/home.gif
gifs/usr/share/swat/images/viewconfig.gif
gifs/usr/share/swat/images/globals.gif
gifs/usr/share/swat/images/passwd.gif
gifs/usr/share/swat/images/status.gif
```

```
gifs/usr/share/swat/images/printers.gif
gifs/usr/share/swat/lang/
gifs/usr/share/swat/lang/tr/
gifs/usr/share/swat/lang/tr/images/
gifs/usr/share/swat/lang/tr/images/samba.gif
gifs/usr/share/swat/lang/tr/images/shares.gif
gifs/usr/share/swat/lang/tr/images/home.gif
gifs/usr/share/swat/lang/tr/images/viewconfig.gif
gifs/usr/share/swat/lang/tr/images/globals.gif
gifs/usr/share/swat/lang/tr/images/passwd.gif
gifs/usr/share/swat/lang/tr/images/status.gif
gifs/usr/share/swat/lang/tr/images/printers.gif
root@c1:~# ls
bad_list  gifs/          loadlin16c.txt  test/          test.tar        test.tar.gz
DEADJOE   gifs.tar.bz2   loadlin16c.zip  test.cpio      test.tar.bz2    test.tar.Z
root@c1:~#
```

## Резервное копирование и восстановление данных

Linux - это стабильная и надежная система. Тем не менее, в любой компьютерной системе могут происходить непредвиденные события, такие как, например, сбой оборудования. Надежное резервное копирование важной конфигурационной информации и данных -- неотъемлемая часть плана администрирования системы. Существует множество подходов к созданию резервных копий в Linux. Они варьируются от простых методов, основанных на выполнении сценариев, до использования сложных коммерческих продуктов. Резервное копирование может выполняться на удаленные сетевые устройства или на магнитную ленту и другие извлекаемые устройства. Вы можете копировать файлы или создавать образ диска. Вариантов множество, и вы можете корректировать и смешивать различные подходы, чтобы ваш план копирования наилучшим образом подходил для вашей ситуации.

## Какова Ваша стратегия?

Существует много различных подходов к созданию резервных копий системы. Что именно вы будете копировать, зависит исключительно от причин, побуждающих вас выполнять резервное копирование. Вы хотите подстраховаться от серьезных сбоев, таких, как отказ оборудования? Или же вы архивируете старые файлы, которые могут понадобиться впоследствии? Будет ли отправной точкой при восстановлении система в исходном состоянии или же предварительно загруженная система, находящаяся в ждущем режиме?

В рамках этой главы мы не можем обсудить все стратегии и решения для резервного копирования вашей системы и ваших данных. Здесь мы лишь познакомимся с системой резервного копирования в целом. Рекомендую уделить этой системе как можно больше времени и ваши усилия не пропадут даром. Я это знаю по собственному опыту. Всегда легче восстановить данные из резервной копии чем пытаться выдрать их из «сдохшего винчестера».

## Что копировать?

Ориентированная на файлы природа Linux дает огромное преимущество при выполнении резервного копирования и при восстановлении системы. В Windows реестр крайне системнозависим. Настройка системы и установка оборудования -- это гораздо больше, нежели простое добавление файлов в систему. Таким образом, восстановление системы требует программного обеспечения, которое учитывает такие индивидуальные особенности. В Linux ситуация иная. Конфигурационные файлы здесь имеют текстовый вид и практически не зависят от системы, за исключением моментов, когда они напрямую связаны с оборудованием. В современном подходе аппаратные драйвера представляют собой динамически загружаемые модули, что позволяет сделать ядро менее зависимым от системы. Поэтому резервное копирование не превращается в решение головоломки о том, каким образом на имеющееся оборудование была инсталлирована система, а фактически представляет собой архивирование и разархивирование файлов.

Вот некоторые каталоги, которые имеет смысл копировать:

/etc

Содержит все ваши ключевые конфигурационные файлы. В их число входят сетевые настройки, имя системы, правила брандмауэра, пользователи, группы и другие системные элементы.

/var

Содержит информацию, используемую вашими системными демонами (службами), в том числе настройки DNS, DHCP leases, файлы почтового буфера, файлы HTTP сервера, конфигурации db2 и другие.

#### **/home**

Содержит домашние каталоги по умолчанию для всех ваших пользователей. Хранит данные о персональных настройках, загруженных файлах и другую ценную для ваших пользователей информацию.

#### **/root**

Домашний каталог привилегированного пользователя root.

#### **/opt**

Каталог, для не системного программного обеспечения. Сюда устанавливается программное обеспечение IBM, OpenOffice, JDKs и другое программное обеспечение по умолчанию так же устанавливается в этот каталог.

Ниже приведены каталоги, для которых не надо выполнять резервное копирование.

#### **/proc**

Никогда не выполняйте резервное копирование для этого каталога. В нем лежат не реальные файлы, а лишь виртуальный образ работающего ядра и среды. Он содержит такие файлы, как /proc/kcore -- виртуальный образ всей используемой памяти. Копирование этого каталога -- лишь пустая трата ресурсов.

#### **/dev**

Содержит файловое представление ваших аппаратных устройств. Вы можете выполнить резервное копирование каталога /dev, если планируете начинать восстановление с пустой системы. Однако, если вы планируете восстановление с уже инсталлированным Linux, то нет необходимости копирования /dev.

Другие директории содержат системные файлы и установленные программные пакеты. В случае сервера большая часть этой информации остается неизменной. Большинство изменений происходит в каталогах /etc и /home. Но для полноты вы можете скопировать и их.

В случае production-систем, где я хочу быть уверен, что мои данные не потеряются, я, скорее всего, выполню резервное копирование всей системы, за исключением директории /proc. Если бы я главным образом волновался о данных пользователей и настройках, я выполнил бы только резервное копирование систем /etc, /var, /home и /root.

## **Инструменты резервного копирования**

Как уже упоминалось выше, резервное копирование в Linux заключается, главным образом, в архивировании и разархивировании файлов, а также копировании архивов на надежный носитель. Это позволяет вам использовать существующие системные утилиты и писать сценарии для выполнения резервного копирования и не покупать коммерческое программное обеспечение. В большинстве случаев такой резервной копии достаточно, и это предоставляет администратору широкие возможности для контроля ситуации. Запуск сценария резервного копирования можно автоматизировать, используя команду cron, которая в Linux управляет выполнением запланированных событий.

Вы уже знакомы с утилитами архивирования и сжатия данных. Полученные архивные файлы можно сохранить на любых доступных сменных носителях типа CDR/RW, DVDR/RW или ленточных накопителях. Так же, архивные файлы можно хранить на различных сетевых ресурсах, используя сервисы NFS или SMB. Важной задачей, сопряженной с резервным копированием, является обеспечение безопасности и ограничения доступа к резервным копиям данных.

Еще одним инструментом, для решения задач резервного копирования, является связка программ dump/restore. Программа dump сохраняет в указанный файл переданную ей файловую систему. При этом, dump копирует файловую систему поблочно. Программа restore служит для восстановления данных из файла, созданного программой dump.

К сожалению программа dump имеет одно неудобство — она работает только с файловыми системами ext2/ext3. И по этому в чистом виде практически не используется.

Однако для тех, кто пользуется файловой системой XFS существует реализация программы xfsdump/xfsrestote.

Рекомендую познакомиться с этими программами более серьезно.

Конечно, существуют и специальные программные комплексы для организации системы резервного копирования. Одним из таких программных комплексов, относящихся к категории открытого и свободного программного обеспечения, является программа Amanda (сайт проекта <http://www.amanda.org/>).

Пакет AMANDA (Advanced Maryland Automatic Network Disk Archiver) контролирует процесс проведения серий полных и инкрементальных резервирований данных на промежуточное дисковое хранилище хоста ленточных или дисковых устройств для некоторого набора сетевых клиентов. Затем полученные наборы данных переносятся на ленточные или дисковые носители. При условии корректной установки пакет Amanda работает полностью автоматически. Процесс резервирования обычно запускается ночью и осуществляет все операции копирования, последовательно устанавливая соединения хоста копирования с каждым из клиентов. Существует возможность устанавливать эти соединения параллельно, а также контролировать и регулировать создаваемую нагрузку на сеть. Модуль планировщика определяет, какой уровень инкрементального копирования должен быть выполнен для каждой файловой системы каждого из хостов.

## Глава 18. Управление программным обеспечением

Необходимость в установке новых программных пакетов под LINUX возникает в двух основных случаях:

- когда появляется новая версия одного из уже установленных у вас пакетов;
- когда возникает желание или необходимость использовать какой-то пакет, еще не установленный в системе.

Во втором случае это может быть один из пакетов, имеющихся на вашем установочном диске, но не установленный в процессе инсталляции. Однако чаще всего новое ПО вы будете находить в Интернете, тем более, что значительная часть этого ПО бесплатна. Как бы то ни было, но рано или поздно вы все равно окажетесь перед необходимостью установить новый пакет.

Для дистрибутивов, основанных на Slackware Linux, существует две основных формы распространения ПО: в исходных текстах и в виде бинарных пакетов. В первом случае пакет ПО обычно поставляется в виде tar-gz архива, во втором случае — в виде tgz-пакета (фактически это бинарные файлы в виде tar-gz-архива).

Проще всего установить ПО, представленное в виде tgz-пакета, содержащего исполняемые файлы, этот способ и рассмотрим первым. Отметим только, что для инсталляции новых пакетов вы должны войти в систему как пользователь root.

### Менеджер пакетов Slackware

Пакет с программным обеспечением - это набор связанных программ, уже готовых к установке. Когда вы загружаете программу в виде архива с исходными текстами, вам необходимо вручную сконфигурировать, откомпилировать и установить её. В программных пакетах это уже сделано за вас. Всё, что вам нужно сделать - это установить пакет. Другой удобной функцией пакетов с ПО является то, что их очень легко удалить и обновить, если вы пожелаете сделать это. В состав Slackware входят все программы, необходимые для управления пакетами. Вы легко можете устанавливать, удалять, обновлять, создавать и изучать пакеты.

После того, как RedHat выпустили свой менеджер пакетов RedHat (RedHat Package Manager), появился миф о том, что в Slackware нет утилиты для управления пакетами. Это просто заблуждение. В Slackware всегда был менеджер пакетов, даже ещё до того, как появился RedHat. И хотя он не настолько наворочен или распространён как rpm (или подобный ему deb-формат), pkgtool и связанные с этим менеджером программы настолько же удобны для установки пакетов, как и rpm. Проблема с pkgtool заключается не в том, что он не существует, а в том, что он не проверяет зависимости.

Очевидно многие люди в сообществе Linux думают, что менеджер пакетов по определению должен включать в себя проверку зависимостей. Ну что ж, это просто не тот случай, поскольку в Slackware как раз так и не сделано. Это не означает, что у пакетов в Slackware отсутствуют зависимости, просто менеджер пакетов не проверяет их. Отслеживание зависимостей остаётся на совести системного администратора, и нам нравится такой подход.

### Формат пакетов

Перед тем, как приступить к изучению утилит, вам следует разобраться с форматом пакетов Slackware. В Slackware пакет - это просто tar-архив, сжатый gzip'ом. Собранные пакеты предназначены для распаковки в корневой каталог.

Вот пример некоторой программы и её пакета:

```
./
usr/
usr/bin/
usr/bin/makehejaz
usr/doc/
usr/doc/makehejaz-1.0/
usr/doc/makehejaz-1.0/COPYING
usr/doc/makehejaz-1.0/README
usr/man/
usr/man/man1
usr/man/man1/makehejaz.1.gz
install/
```

```
install/doinst.sh
```

Система работы с пакетами распакует этот файл в корневой каталог и установит его. В базе данных пакетов будет создана запись с содержимым этого пакета, чтобы позже его можно было удалить или обновить.

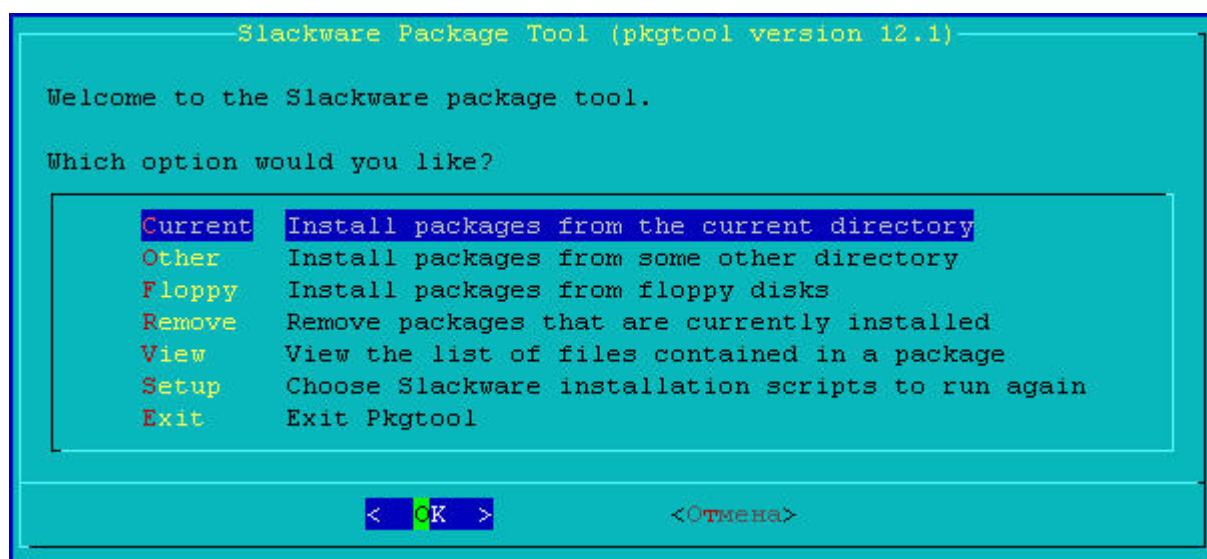
Обратите внимание на подкаталог install/. Это специальный каталог, в котором находится скрипт, выполняемый после установки, под названием doinst.sh. Если менеджер пакетов найдёт этот файл, он запустит его после установки пакета.

## Программы для работы с пакетами

Для управления пакетами существуют четыре основные утилиты. Они выполняют установку, удаление и обновление пакетов.

### **pkgtool**

pkgtool - это программа с поддержкой меню, которая позволяет устанавливать и удалять пакеты. Главное меню показано ниже:



Установка может быть выполнена из текущего каталога, другого каталога или с дискет. Просто выберите нужный вам метод установки и pkgtool начнёт искать в указанном вами месте пригодные к установке пакеты.

Также вы можете просмотреть список установленных пакетов.

Если вам нужно удалить пакеты, выберите опцию Remove и вам будет представлен список всех установленных пакетов. Отметьте те, которые вы хотите удалить, и нажмите OK. pkgtool выполнит их удаление.

Некоторые пользователи отдадут предпочтение этой утилите, а не утилитам командной строки. Однако, следует отметить, что эти консольные утилиты предлагают гораздо больше функций. Кроме того, обновление пакетов возможно только с помощью утилит командной строки.

### **Проверка установленного программного обеспечения**

Список установленных пакетов можно узнать посмотрев список файлов в директории /var/log/packages.

Имя файла соответствует названию пакета.

Внутри файла находятся описание пакета и список установленных файлов.

```

root@c1:~# ls /var/log/packages
.....
.....
xscreensaver-5.05-i486-1
xset-1.0.4-i486-1
xsetmode-1.0.0-i486-1
xsetpointer-1.0.1-i486-1
xsetroot-1.0.2-i486-1
    
```

```

xsm-1.0.1-i486-1
xstdcmap-1.0.1-i486-1
xterm-232-i486-1
xtrans-1.1-noarch-1
xtrap-1.0.2-i486-1
xv-3.10a-i486-5
xvidtune-1.0.1-i486-1
xvinfo-1.0.2-i486-1
xwd-1.0.1-i486-1
xwininfo-1.0.3-i486-1
xwud-1.0.1-i486-1
xxgdb-1.12-i486-2
yptools-2.9-i486-1
ytalk-3.3.0-i486-1
zdl211-firmware-1.4-fw-1
zlib-1.2.3-i486-2
zoo-2.10-i486-1
zsh-4.3.6-i486-1
root@c1:~# cat /var/log/packages/zoo-2.10-i486-1
PACKAGE NAME:      zoo-2.10-i486-1
COMPRESSED PACKAGE SIZE:      56 K
UNCOMPRESSED PACKAGE SIZE:    110 K
PACKAGE LOCATION: /var/log/mount/slackware/a/zoo-2.10-i486-1.tgz
PACKAGE DESCRIPTION:
zoo: zoo (archiving and compressing utility)
zoo:
zoo: Zoo is used to create and maintain collections of files in compressed
zoo: form. It uses a Lempel-Ziv compression algorithm that gives space
zoo: savings in the range of 20% to 80% depending on the type of file data.
zoo: Zoo can store and selectively extract multiple generations of the same
zoo: file. Data can be recovered from damaged archives by skipping the
zoo: damaged portion and locating undamaged data with the help of fiz(1).
zoo:
zoo: Zoo was written by Rahul Dhesi.
zoo:
FILE LIST:
./
usr/
usr/bin/
usr/bin/fiz
usr/bin/zoo
usr/man/
usr/man/man1/
usr/man/man1/fiz.1.gz
usr/man/man1/zoo.1.gz
usr/doc/
usr/doc/zoo-2.10/
usr/doc/zoo-2.10/Copyright
usr/doc/zoo-2.10/Install
install/
install/slack-desc
root@c1:~#

```

## **installpkg**

Утилита `installpkg` выполняет установку в систему новых пакетов. Её синтаксис следующий:

`installpkg` опция имя\_пакета

У `installpkg` есть только три опции. Одновременно может быть использована только одна опция.

Опции:

- **-m** Выполняет действие `makepkg` в текущем каталоге.
- **-warn** Показывает, что произойдёт, если вы установите указанный пакет. Это полезно для

ответственных систем, когда, перед тем как устанавливать что-либо, вы можете узнать что при этом произойдёт.

- **-r** Рекурсивно устанавливает все пакеты из текущего каталога и ниже. Имя пакета может содержать маску, которая будет использована при рекурсивном поиске пакетов для установки.

```
root@c1:~# mount /dev/cdrom /mnt/dvd/
mount: блочное устройство /dev/hdc защищен от записи, монтируется только для
чтения
root@c1:~# ls /mnt/dvd/
ANNOUNCE.12_1      FILELIST.TXT      RELEASE_NOTES
BOOTING.TXT        GPG-KEY           slackbook/
ChangeLog.txt      isolinux/         slackware/
CHANGES_AND_HINTS.TXT kernels/          Slackware-HOWTO
CHECKSUMS.md5      PACKAGES.TXT      source/
CHECKSUMS.md5.asc  pasture/          SPEAK_INSTALL.TXT
COPYING            patches/          SPEAKUP_DOCS.TXT
COPYING3           README_CRYPT.TXT  testing/
COPYRIGHT.TXT      README.initrd     UPGRADE.TXT
CRYPTO_NOTICE.TXT   README_LVM.TXT    usb-and-pxe-installers/
extra/             README_RAID.TXT
FAQ.TXT            README.TXT
root@c1:~# installpkg /mnt/dvd/slackware/y/bsd-games-2.13-i486-8.tgz
Installing package bsd-games-2.13-i486-8 ([OPT])...
PACKAGE DESCRIPTION:
bsd-games: bsd-games (Classic BSD text games collection)
bsd-games:
bsd-games: Games that go in /usr/games:  adventure arithmetic atc backgammon
bsd-games:  banner battlestar bcd caesar canfield cfscores countmail cribbage
bsd-games:  factor fish fortune gomoku hangman hunt mille monop morse number
bsd-games:  phantasia pig pom ppt primes quiz rain random robots rot13 sail
bsd-games:  snake snscore teachgammon trek wargames worm worms wump
bsd-games:
bsd-games: Adds a call to 'fortune' to /etc/profile.d/ so that users will get a
bsd-games: fortune message when they log in.
bsd-games:
Executing install script for bsd-games-2.13-i486-8...

root@c1:~#
```

Если вы определите переменную окружения ROOT перед запуском installpkg, этот путь будет использован в качестве корневого каталога. Это полезно для настройки новых дисков для размещения на них корневых каталогов. Обычно они монтируются в /mnt или какой-то другой каталог, отличный от /.

Запись в базе данных об установленном пакете хранится в /var/log/packages. Запись представляет собой обычный текстовый файл, по одному на пакет. Имя файла соответствует названию пакета. Внутри файла находятся описание пакета и список установленных файлов.

```
root@c1:~# ls /var/log/packages | grep game
bsd-games-2.13-i486-8
kdegames-3.5.9-i486-2
xgames-0.3-i486-1
root@c1:~# cat /var/log/packages/bsd-games-2.13-i486-8
PACKAGE NAME:      bsd-games-2.13-i486-8
COMPRESSED PACKAGE SIZE: 2273 K
UNCOMPRESSED PACKAGE SIZE: 5460 K
PACKAGE LOCATION: /mnt/dvd/slackware/y/bsd-games-2.13-i486-8.tgz
PACKAGE DESCRIPTION:
bsd-games: bsd-games (Classic BSD text games collection)
bsd-games:
bsd-games: Games that go in /usr/games:  adventure arithmetic atc backgammon
bsd-games:  banner battlestar bcd caesar canfield cfscores countmail cribbage
bsd-games:  factor fish fortune gomoku hangman hunt mille monop morse number
bsd-games:  phantasia pig pom ppt primes quiz rain random robots rot13 sail
bsd-games:  snake snscore teachgammon trek wargames worm worms wump
```



```

bsd-games:
bsd-games: Adds a call to 'fortune' to /etc/profile.d/ so that users will get a
bsd-games: fortune message when they log in.
bsd-games:
FILE LIST:
./
etc/
etc/profile.d/
etc/profile.d/bsd-games-login-fortune.sh
etc/profile.d/bsd-games-login-fortune.csh
var/
.....
.....
install/
install/doinst.sh
install/slack-desc
root@cl:~#

```

Если в пакете есть постановочный скрипт, он записывается в каталог /var/log/scripts/.

Вы можете указать несколько пакетов или использовать маски в именах файлов. Учтите, что `installpkg` не сообщит вам, если будет перезаписываться уже установленный пакет. Он просто установит его поверх старого. Если вы хотите, чтобы старые файлы из предыдущего пакета были безопасно удалены, используйте утилиту `upgradepkg`.

## **upgradepkg**

Утилита `upgradepkg` обновляет установленные пакеты Slackware. Её синтаксис следующий:

`upgradepkg имя_пакета`

или

`upgradepkg старое_имя_пакета%новое_имя_пакета`

`upgradepkg` сначала устанавливает новый пакет, а затем удаляет старый пакет, чтобы в системе больше не осталось старых файлов. Если у обновляемого пакета изменилось имя, используйте в команде знак процента для указания старого имени пакета (того, что установлен) и нового имени пакета (до которого вы выполняете обновление).

Если вы определите переменную окружения `ROOT` перед запуском `upgradepkg`, этот путь будет использован в качестве корневого каталога. Это полезно для настройки новых дисков для размещения на них корневых каталогов. Обычно они монтируются в `/mnt` или какой-то другой каталог, отличный от `/`.

`upgradepkg` не лишён недостатков. Вам всегда следует создавать резервные копии своих конфигурационных файлов. Если они будут удалены или перезаписаны, для нормальной работы вам потребуется восстановить их оригиналы.

Как и в случае с `installpkg` вы можете указать несколько пакетов или использовать маски в именах пакетов.

## **removepkg**

Утилита `removepkg` удаляет из системы установленные пакеты. Её синтаксис следующий:

`removepkg опция имя_пакета`

У `removepkg` есть только четыре опции. Одновременно может быть использована только одна опция.

Опции:

- **-copy**           Пакет копируется в специальный каталог для пакетов. Так можно создать дерево из оригинальных пакетов, не удаляя их.
- **-keep**           Сохраняет временные файлы, созданные во время удаления. В действительности полезно только в целях отладки.
- **-preserve**       Пакет удаляется, однако в тоже время он копируется в каталог для резервных копий.
- **-warn**           Показывает, что бы произошло, если бы вы удалили пакет.

Если вы определите переменную окружения `ROOT` перед запуском `removepkg`, этот путь будет использован в качестве корневого каталога. Это полезно для настройки новых дисков для размещения на них корневых каталогов. Обычно они монтируются в `/mnt` или какой-то другой каталог, отличный от `/`.

`removepkg` выполняет поиск и в остальных установленных пакетах, но удаляет только те файлы, которые являются уникальными для указанного вами пакета. Он также найдёт постановочный скрипт для указанного пакета и удалит все созданные им символические ссылки.

Во время процесса удаления на экран выводится отчёт о ходе его выполнения. После удаления запись из базы данных пакетов перемещается в каталог `/var/log/removed_packages`, а скрипт, выполняемый после установки, перемещается в `/var/log/removed_scripts`.

```
root@c1:~# removepkg bsd-games-2.13-i486-8

Removing package /var/log/packages/bsd-games-2.13-i486-8...
Removing files:
--> /usr/man/man6/worm.6.gz was found in another package. Skipping.
--> Deleting /etc/profile.d/bsd-games-login-fortune.csh
--> Deleting /etc/profile.d/bsd-games-login-fortune.sh
--> Deleting /usr/bin/strfile
--> Deleting /usr/doc/bsd-games-2.13/AUTHORS
--> Deleting /usr/doc/bsd-games-2.13/BUGS
--> Deleting /usr/doc/bsd-games-2.13/INSTALL
--> Deleting /usr/doc/bsd-games-2.13/NEWS
--> Deleting /usr/doc/bsd-games-2.13/PACKAGING
--> Deleting /usr/doc/bsd-games-2.13/README
--> Deleting /usr/doc/bsd-games-2.13/README.fortune
--> Deleting /usr/doc/bsd-games-2.13/README.hunt
--> Deleting /usr/doc/bsd-games-2.13/README.linux.hunt
--> Deleting /usr/doc/bsd-games-2.13/README.linux.trek
--> Deleting /usr/doc/bsd-games-2.13/README.phantasia
--> Deleting /usr/doc/bsd-games-2.13/SECURITY
--> Deleting /usr/doc/bsd-games-2.13/THANKS
--> Deleting /usr/doc/bsd-games-2.13/TODO
--> Deleting /usr/doc/bsd-games-2.13/YEAR2000
.....
.....
--> Deleting empty directory /usr/share/misc/
--> Deleting empty directory /usr/share/games/quiz/
--> Deleting empty directory /usr/share/games/fortunes/
--> Deleting empty directory /usr/share/games/atc/
--> Deleting empty directory /usr/share/games/
--> Deleting empty directory /usr/share/dict/
--> Deleting empty directory /usr/games/
--> Deleting empty directory /usr/doc/bsd-games-2.13/
root@c1:~# ls /var/log/packages | grep game
kdegames-3.5.9-i486-2
xgames-0.3-i486-1
root@c1:~#
```

Как и в случае с `installpkg` вы можете указать несколько пакетов или использовать маски в именах пакетов.

### **swaret**

Однако для Slackware Linux был написан пакет `swaret` (slackware tool), который позволяет выполнять все вышеописанные действия более комфортно. А также этот пакет позволяет обновить всю систему в целом до состояния `current`.

В своей работе `swaret` использует репозитории доступные через сеть `internet`.

Этот пакет не входит в официальный дистрибутив и поддерживается группой энтузиастов. Найти его можно в сети `internet`.

Вот некоторые возможности `swaret`:

Обновить список пакетов:

```
# swaret --update
```

Если все прошло нормально и вы увидели "=> Создаем список файлов... ЗАВЕРШЕНО!", система автоматизированного обновления настроена и готова к дальнейшему использованию.

Обновить систему:

```
# swaret --upgrade
```

При этом система может задавать вам различные вопросы.

Обновление системы в автоматическом режиме

```
# swaret --upgrade -a
```

Установка нового пакета

```
# swaret --install имя_пакета
```

Здесь достаточно указать лишь имя пакета без указания его версии. Swaret найдет и установит пакет в последней доступной версии. Так же будут проверены и удовлетворены все зависимости!

Удаление установленного пакета

```
# swaret --remove имя_пакета
```

Проверка зависимостей для пакета

```
# swaret --dep имя_пакета
```

При возникновении проблем или вопросов обратитесь к документации: `man swaret`, `swaret --htus`, `swaret --faq`, `swaret --manual`, `swaret --manual -c`.

## ***rpm2tgz/rpm2targz***

Менеджер пакетов Red Hat (Red Hat Package Manager, RPM) - это популярная на сегодняшний день система работы с пакетами. Многие распространители программного обеспечения предлагают свои продукты в формате RPM. Поскольку он не является нашим “родным” форматом, мы не рекомендуем вам использовать его. Однако некоторые вещи бывают доступны только в виде RPM (даже исходные тексты).

Мы предлагаем программу для преобразования RPM-пакетов в наш родной формат .tgz. Это позволит вам развернуть пакет (возможно, с помощью `explodepkg`) во временный каталог и изучить его содержимое.

Программа `rpm2tgz` создаст пакет Slackware с расширением .tgz, а `rpm2targz` создаёт архив с расширением .tar.gz.

## **Создание пакетов**

Создание пакетов Slackware может быть как простым, так и сложным. Не существует чёткого метода для сборки пакетов. Единственное требование - пакет должен быть tar-файлом, сжатым gzip'ом, и если в нём есть постановочный скрипт, это должен быть `/install/doinst.sh`.

Если вы заинтересованы в создании пакетов для своей системы или для обслуживаемой вами сети компьютеров, вам следует ознакомиться с различными сборочными скриптами, доступными в дереве исходных текстов Slackware. При создании пакетов мы используем несколько различных методов.

### ***explodepkg***

Утилита `explodepkg` делает то же самое, что и `installpkg` делает при установке пакета, однако на самом деле она не устанавливает его и не делает запись в базе данных пакетов. Она просто извлекает содержимое пакета в текущий каталог.

Если вы посмотрите на дерево исходных текстов Slackware, вы увидите, как мы используем эту команду для пакетов “framework”. Эти пакеты содержат каркас будущего пакета. В них находятся все необходимые имена файлов (нулевой длины), права доступа и владельцы. Скрипт сборки извлекает содержимое пакета из каталога с исходными текстами и помещает его в каталог сборки.

## **makepkg**

Утилита `makepkg` упаковывает содержимое текущего каталога в пакет Slackware. Он найдёт в дереве символические ссылки и добавит в скрипт, выполняемый после установки, блок команд, создающих эти ссылки во время установки пакета. Он также предупреждает вас о наличии в дереве пакета файлов нулевого размера.

Эта команда обычно запускается после того, как вы создали дерево своего пакета.

## **Скрипты SlackBuild**

При необходимости пакеты Slackware могут быть собраны и другими способами. Не всё программное обеспечение в виде пакетов написано программистами для компиляции одним и тем же способом. Во многих есть опции компиляции, которые не используются в пакетах Slackware. Возможно, когда-то вам понадобятся эти функциональные возможности. Тогда вам придётся собрать свой собственный пакет. К счастью у многих пакетов Slackware в дереве исходных текстов пакета вы можете найти скрипты SlackBuild.

Что же такое скрипт SlackBuild? Скрипты SlackBuild представляют собой исполняемые скрипты командного процессора, запускаемые вами под `root`'ом для настройки, компиляции и создания пакетов Slackware. Вы можете свободно изменять эти скрипты в дереве исходных текстов и запускать их для создания своих собственных версий стандартных пакетов Slackware.

## **Менеджер пакетов RPM**

Программа `rpm`. Название этой программы (или команды) является аббревиатурой от Redhat Package Manager. Такая расшифровка дается в большинстве книг и руководств по Linux и кажется мне более правильной и логичной, хотя в главе 6 "The Official Red Hat Linux Reference Guide" говорится: "The RPM Package Manager (RPM), is an open packaging system available for any-one to use, and works on Red Hat Linux as well as other Linux and UNIX systems", т. е. предлагается рекурсивная расшифровка названия RPM, подобная расшифровке GNU — GNU is Not Unix).

Программа `rpm` в некотором смысле аналогична программам типа `setup wizard` для MS Windows. Преимуществом использования этой программы по сравнению с установкой `tar gz` архивов является то, что она автоматически проделает все необходимые действия по установке ПО: создаст необходимые каталоги, распределит по ним файлы, создаст ссылки. Кроме того, она может быть использована не только для установки нового пакета, но и для обновления версий ПО, получения перечней установленного ПО и проверки установки, а также для деинсталляции отдельных пакетов (например, если после периода пробной работы с программой вы решили отказаться от ее дальнейшего использования). С помощью той же программы `rpm` можно самому создать пакет формата `rpm`, однако для начинающих лучше, наверное, этим не заниматься, а воспользоваться готовыми `rpm`-пакетами.

`Rpm`-пакеты — это специальным образом подготовленные архивы, предназначенные для обработки программой `rpm`. Название `rpm`-пакетов оканчивается на суффикс `.rpm`, например, `xzip-180-1.i386.rpm` или `xzip-180-1.src.rpm`. Как видите, перед суффиксом `.rpm` стоит еще один суффикс. Если это `.i386`, `.i686` или `.i586`, то в пакете находятся исполняемые файлы (оптимизированные для соответствующего типа процессора), а если этот суффикс `.src`, — то в пакете исходные тексты, которые после установки еще надо скомпилировать. Обычно как на установочных компакт-дисках, так и в интернет-каталогах `rpm`-пакеты с исполняемыми файлами располагаются в каталогах с названием `RPMS`, а `rpm`-пакеты с исходными текстами — в подкаталогах `SRPMS`. Часто встречаются также `rpm`-пакеты с суффиксом `.noarch.rpm`, содержащие файлы, которые просто без всякой дополнительной обработки устанавливаются в соответствующие каталоги (например, файлы страниц интерактивного руководства `man`). И, наконец, если `rpm`-пакет рассчитан на версию Linux, предназначенную для другой аппаратной платформы (AMD, DEC Alpha, SUN Sparc, MIPS, PowerPC), это тоже будет отображено в имени пакета: вместо `i386` в суффиксе будет стоять, соответственно, `athlon`, `alpha`, `sparc`, `mips` или `ppc`.

В Интернете `rpm`-пакеты можно найти на различных серверах. По моему опыту наиболее удобным сервером в Интернете для поиска `rpm`-архивов является сервер <http://rufus.w3.org> (недаром он имеет другое имя <http://rpmfind.net>). На нем установлена поисковая система, которая позволяет упорядочивать список пакетов наиболее удобным для вас способом:

- по именам пакетов;
- по дистрибутивам;
- по группам приложений;
- по датам;
- по поставщикам (производителям) ПО.

Общий объем архива rpm-пакетов на этом сервере составляет более 66 Гигабайт. Очень богатые архивы хранят также два ftp-сервера в России: <ftp://ftp.chg.ru/pub/Linux> и <ftp://ftp.nc.orc.ru/>.

Необходимо только заметить, что если для перекачки пакетов из Интернета вы используете компьютер, работающий под Windows 95, то все имена пакетов у вас будут, скорее всего, искажены. Дело в том, что Windows "не любит" имена, в которых несколько точек (например, glib-1.0.6-3.i386.rpm) и заменит "лишние", по его мнению, точки на знаки подчеркивания — glib-1\_0\_6-3\_i386.rpm. Так что после получения пакета (при переносе его на ПК с ОС Linux) желательно эти "исправления" устранить, вернувшись к исходным именам UNIX. Правда, делать это не обязательно, поскольку внутри rpm-пакет все равно правильно идентифицирован, но для единообразия и облегчения поиска файлов все же целесообразно.

Итак, вы нашли и скачали rpm-архив с исполняемой версией нужного вам пакета. Если вы ставите совершенно новый пакет (у вас не было на компьютере предыдущих версий этого ПО), то для установки пакета из этого архива достаточно перейти в тот каталог, где находится архив, и дать команду (для самых нетерпеливых: не спешите выполнять эту рекомендацию, прочитайте еще хотя бы пару абзацев)

**rpm -i имя\_rpm-архива**

Если у вас была установлена предыдущая версия пакета, то в простейшем случае надо дать команду следующего формата:

**rpm -U --force имя\_rpm-архива**

Здесь параметр -U говорит программе, что надо произвести обновление (upgrade) пакета, а опция --force требует безусловно (и без лишних вопросов) обновить все входящие в пакет файлы. Заметьте, что это очень сильное требование, и в некоторых случаях может быть лучше сохранить какие-то (например, конфигурационные) файлы от предыдущей версии. Если установка проходит нормально, и никаких дополнительных сообщений не появляется, то после завершения работы программы (после появления приглашения оболочки) вы можете пользоваться вновь установленным пакетом.

К сожалению, не всегда все так просто. Приведу конкретный пример. У меня был установлен RedHat Linux версии 5.2, причем программа Midnight Commander (mc) была версии 4.1.36. На ftp-сервере я увидел версию 4.5.30 этой программы (пакет mc-4.5.30-12.i386.rpm) и, естественно, решил ее поставить. Однако оказалось, что для этого необходимо установить еще 4 других пакета, о чем rpm мне и сообщила:

```
ошибка: неудовлетворенные зависимости:
redhat-logos нужен для mc-4.5.30-12
libglib-1.2.so.0 нужен для mc-4.5.30-12
libc.so.6(GLIBC_2.1) нужен для mc-4.5.30-12
libc.so.6(GLIBC_2.0) нужен для mc-4.5.30-12
```

Это не удивительно, программа rpm автоматически проверяет все зависимости и старается удовлетворить их сама, но если таковое не возможно выдает соответствующее сообщение. Однако в случае инсталляции с CD-ROM все необходимые пакеты находятся на том же диске, а здесь мне пришлось вначале поискать нужные пакеты. Два пакета (redhat-logos-1.0.5-1.noarch.rpm и glibc-2.1.1-6.i386.rpm) я нашел без труда, после чего rpm перестала просить и GLIBC\_2.0. А вот с libglib.so.1 вышло сложнее. Во-первых, я никак не мог найти пакета с таким названием. Как оказалось, такого пакета и не существует, файл libglib.so.1 входит в состав пакета glib-1.0.6-3.i386.rpm.

Программа rpm позволяет выяснить, какие файлы установит тот или иной пакет. Для этого надо дать следующую команду (только учтите, что текущим должен быть каталог, содержащий интересующий вас пакет):

**rpm -qpl имя\_rpm-архива**

А для получения информации о том, для чего служит ПО, содержащееся в rpm-пакете, используйте команду

**rpm -qpi имя\_rpm-архива**

Дело в том, что файлы RPM кроме собственно архива файлов содержат информацию о пакете, включая имя, версию и краткое описание. С помощью той же программы rpm вы можете просмотреть эту дополнительную информацию. Например, для пакета glib-1.0.6-3.i386.rpm вывод команды

```
# rpm -qpi glib-1.0.6-3.i386.rpm
```

будет примерно таким:

```
Name : glib Relocations: (not relocateable)
Version : 1.0.6 Vendor: Red Hat Software
Release : 3 Build Date: Суб 10 Окт 1998 04:49:03
Install date: (not installed)
```

```
Build Host : porky.redhat.com
Group : Libraries Source RPM: glib-1.0.6-3.i386.rpm
Size : 55305
Packager : Red Hat Software <bug@redhat.com>
Summary : Handy library of utility functions
Description : Handy library of utility functions. Development libs and headers
are in gtk+-devel.
```

Если дать команду:

```
# rpm -qpl glib-1.0.6-3.i386.rpm
```

будет выдан список входящих в пакет файлов с указанием того, куда они будут установлены:

```
/usr/lib/libglib.so.1
/usr/lib/libglib.so.1.0.6
```

RPM также предоставляет мощную систему запросов по установленным в системе пакетам. По команде

```
# rpm -qa
```

вы получите перечень всех установленных в системе пакетов (перечень будет очень большим, так что лучше сразу направить вывод в фильтр more или в файл, который потом просматривать с помощью less или встроенной программы просмотра из оболочки Midnight Commander). Вы можете искать информацию об отдельном пакете или об отдельных файлах. Например, вы можете легко найти, какому пакету принадлежит файл и откуда появился. Команда

```
# rpm -qf /etc/bashrc
```

сообщит:

```
bash-1.14.7-16.
```

Если вы беспокоитесь о том, что случайно удалили важный файл из установленного пакета, просто проверьте это:

```
# rpm -Va
```

Вы будете оповещены об любых аномалиях. Потом можно переустановить пакет, если это необходимо. Любые конфигурационные файлы будут сохранены.

Как видите, rpm это очень полезная утилита, и у нее имеется много разных опций. Выше приведено только несколько примеров. Всего rpm имеет 16 основных режимов работы, которые можно объединить в 6 групп (после двоеточия приводится формат команды для соответствующего режима).

### **Запросы.**

*Запрос:* rpm [--query] [queryoptions]

*Показать метки запросов (Querytags):* rpm [--querytags]

### **Установка и поддержка установленных пакетов.**

*Установка:* rpm [--install] [installoptions] [package\_file]+

*Обновление:* rpm [--freshen|-F] [installoptions] [package\_file]+

*Деинсталляция:* rpm [--uninstall|-e] [uninstalloptions] [package]+

*Проверка:* rpm [--verify|-V] [verifyoptions] [package]+

**Подписи** (пакеты подписываются электронной цифровой подписью в формате PGP, с целью обеспечения неизменяемости и сохранения авторства пакетов).

*Проверка подписи:* rpm [--verify|-V] [verifyoptions] [package]+

*Переподписывание:* rpm [--resign] [package\_file]+

*Добавление подписи:* rpm [--addsign] [package\_file]+

### **Работа с базой.**

*Инициализация базы:* rpm -i [--initdb]

*Обновление базы (Rebuild Database):* rpm -i [--rebuilddb]

**Создание rpm-пакетов.**

Создать пакет: `rpm [-b|t] [package_spec]+`

Перекомпилировать пакет: `rpm [--rebuild] [sourcerpm]+`

Скомпилировать пакет из tar-архива: `rpm [--tarbuild] [tarredsource]+`

**Разное.**

Показать конфигурацию программы rpm: `rpm [--showrc]`

Задать пользователей: `rpm [--setperms] [package]+`

Задать группы: `rpm [--setgids] [package]+`

Подробное описание всех возможностей команды rpm выходит за рамки этого курса. Его вы можете найти в RPM-HOWTO, на страницах `man` и `info`.

**Примечание**

Как и другие программы для Linux, программа rpm постоянно развивается и совершенствуется. При этом при замене версии этой программы могут возникнуть трудности с установкой пакетов, созданных в предыдущих версиях. Так было, например, при переходе с третьей на четвертую версию rpm. Так что надо использовать пакеты, соответствующие установленной у вас версии программы.

Приведенное выше описание программы rpm предполагает, что она запускается с консоли или в эмуляторе терминала. Между тем в разных дистрибутивах имеются графические оболочки для управления rpm-пакетами. В составе графической среды KDE такая оболочка называется `kpackage`. Вы можете запустить ее либо из командной строки, либо из основного меню KDE. Однако, на мой взгляд, она не дает никаких преимуществ по сравнению с работой из командной строки.

**Компиляция программного обеспечения из исходных текстов**

Если бинарные пакеты с необходимым вам программным обеспечением нужно еще поискать (и не всегда можно найти), то `tar-gz`-архив любого ПО для Linux найдется в Интернете непременно. В некоторых случаях такие архивы содержат исполняемые модули приложений. Тогда установка приложения лишь немного сложнее, чем в случае установки из бинарного пакета: необходимо просто развернуть архив с помощью программ `gunzip` и `tar`, перейти в созданный каталог и можно уже запускать полученное приложение. Но чаще всего приложения поставляются в исходных текстах, т. е. в виде программы на языке Си. Установить их в этом случае немного сложнее, хотя и тут нет ничего невозможного даже для начинающего пользователя. Давайте рассмотрим, как это делается.

**Необходимые сведения о программировании на языке Си**

Начать стоит с того, что операционная система UNIX родилась на свет одновременно с языком программирования C (Си). Более того, язык C был создан специально для разработки этой ОС, значительная часть UNIX была написана на языке C. ОС Linux тоже написана на Си. Поэтому, а также в соответствии с принципом свободного распространения исходных кодов, многие приложения для Linux распространяются в виде текстов на C (а в последнее время — и на C++). Естественно, что для установки и запуска такого приложения на исполнение, его необходимо предварительно скомпилировать. Для выполнения процедур компиляции обычно используется программа `gcc` (хотя существуют и некоторые альтернативные разработки).

**Внимание!** Изначально аббревиатура GCC имела смысл GNU C Compiler, но в апреле 1999 года сообщество GNU решило взять на себя более сложную миссию и начать создание компиляторов для новых языков с новыми методами оптимизации, поддержкой новых платформ, улучшенных runtime-библиотек и других изменений (<http://gcc.gnu.org/gccmission.html>). Поэтому сегодня GCC расшифровывается как GNU Compiler Collection (коллекция компиляторов GNU) и содержит в себе компиляторы для языков C, C++, Objective C, Chill, Fortran, Ada и Java, а также библиотеки для этих языков (`libstdc++`, `libgcj`, ...).

GNU-компилятор с языка C `gcc`, содержит в себе 4 основных компонента, соответствующие четырем этапам преобразования исходного кода в исполняемую программу.

Первый компонент — это препроцессор, который модифицирует исходный код программы перед компиляцией в соответствии с командами препроцессора, содержащимися в C-программе. В соответствии с этими командами выполняются простые подстановки текста. Второй — собственно компилятор, который обрабатывает исходный код и преобразует его в код на языке ассемблера. Третий компонент — ассемблер, который генерирует объектный код. И, наконец, четвертый компонент — компоновщик, который собирает исполняемый файл из

файлов объектного кода. Дело в том, что большие программы обычно пишутся по частям, в виде множества отдельных файлов, содержащих исходный код соответствующей части. Компилятор обрабатывает каждый такой файл отдельно и создает отдельные объектные модули (файлы таких модулей обычно имеют расширение `o`). Создание единой исполняемой программы из таких модулей и является задачей компоновщика. При таком подходе, если в какой-то модуль программист вносит исправление, нет необходимости заново компилировать всю программу: достаточно откомпилировать исправленный модуль и заново запустить компоновщик.

Для выполнения стандартных операций программист может использовать функции из стандартных библиотек. Самый характерный пример — это библиотека `libc`, которая содержит функции, выполняющие такие задачи, как управление памятью и операции ввода-вывода. Программисты могут создать свои собственные библиотеки и использовать их при написании новых программ.

Библиотеки бывают статическими, разделяемыми и динамическими. Статическая библиотека — это библиотека, код которой встраивается в программу при компиляции. Код разделяемой библиотеки не встраивается в программу, а загружается в память одновременно с программой и программа получает доступ к функциям этой библиотеки. Динамические библиотеки — разновидность разделяемых, но библиотечные функции загружаются в память только тогда, когда из программы поступит вызов соответствующей функции. В процессе выполнения программы они могут выгружаться и заменяться другими функциями из той же или другой библиотеки. Имена статических библиотек обычно имеют суффикс `.a`, а имена разделяемых библиотек — суффикс `.so`, за которым следует старший и младший номера версии. Имя может быть любой строкой, которая однозначно характеризует библиотеку. Обычно имена библиотек начинаются с `lib`. Примеры: `libm.so.5` — общая математическая библиотека, `libX11.so.6` — библиотека для работы с системой X Window. Библиотека `libc.so.5` компонуется автоматически, в то время как большинство других библиотек необходимо явно указывать в командной строке при вызове программы `gcc`. Это делается через опцию `-l`, за которой следует уникальная часть имени библиотеки, например, для вызова математической библиотеки достаточно указать `-lm`.

Многие системные библиотеки располагаются в системных каталогах, например, в `/usr/lib` и `/lib`, но некоторые могут располагаться и в других местах. Список этих каталогов помещается в файл `/etc/ld.so.conf`. Каждый раз, когда разделяемая библиотека изменяется или устанавливается вновь, нужно выполнять команду `ldconfig`, чтобы обновить файл `/etc/ld.so.conf`, а также ссылки на него. Если библиотека устанавливается из RPM-пакета, это обычно делается автоматически, хотя и не всегда.

При компиляции больших программ, использующих фрагменты исходного кода, расположенные в разных файлах, бывает очень трудно отследить, какие файлы нужно перекомпилировать, а какие только компоновать. В таких случаях очень помогает утилита `make`, которая автоматически определяет, следует ли компилировать файл исходного кода, по дате его последней модификации. Утилита `make` оперирует файлами, исходя из их зависимости друг от друга. Эти зависимости определяются файлом с именем `makefile`. Строка файла `makefile` состоит из трех частей: имени целевого файла, списка файлов, от которых он зависит, и команды. Если какой-либо файл из списка изменился после целевого файла, то выполняется указанная в строке команда. В строке может быть указано несколько команд. Обычно команда — это вызов компилятора для компиляции файла исходного кода или компоновки файлов объектного кода. Строки, определяющие зависимости, отделяются друг от друга пустой строкой.

## Инсталляция программного обеспечения из исходных кодов

Теперь, когда мы получили общее представление о компиляции программ на языке C, можно рассмотреть обращение с пакетами программ, распространяемыми в виде исходных кодов. Первое, что надо сказать в этой связи, это то, что для установки таких пакетов вы, естественно, должны иметь в своей системе утилиты `gcc` и `make`.

Непосредственно процесс инсталляции пакета состоит из следующих шагов:

- Перейти (с помощью команды `cd`) в каталог, содержащий исходные коды устанавливаемого пакета.
- Выполнить команду `./configure`, которая осуществляет конфигурирование пакета в соответствии с вашей системой. Процесс выполнения этой команды занимает довольно длительное время, причем команда выдает на экран сообщения о том, какие именно особенности системы испытываются. Результат работы программы отображается в файле `makefile` или `Makefile`.
- Выполнить команду `make`, для того, чтобы скомпилировать пакет.
- После этого можно выполнить (это шаг не является обязательным) команду `make check`, которая вызывает запуск процедур самотестирования, которые поставляются с пакетом.
- Выполнить команду `make install` для установки программ, а также файлов данных и документации.
- Заключительный этап состоит в выполнении команды `make clean`, которая удаляет промежуточные



объектные и двоичные файлы из каталога с исходными кодами. Для удаления временных файлов, которые создала команда `configure` (после чего пакет можно компилировать для другого типа компьютеров), надо выполнить команду `make distclean`.

В большинстве случаев выполнение этой последовательности команд достаточно для установки нового пакета.

Основная проблема, с которой приходится сталкиваться при инсталляции программ из исходных кодов, связана с конфликтами версий: для вновь устанавливаемого пакета требуются новые версии каких-то системных утилит, которые пока еще не установлены в вашей системе. Более того, часто возникает целая цепочка (или даже дерево): для программы нужна какая-то новая версия утилиты, для последней нужно обновить еще какие-то утилиты, и т. д.. Но, если вы не очень давно устанавливали (или обновляли) дистрибутив, то таких проблем не возникает, и обновление пакета пройдет без затруднений.

## Программа *make*

Программа `make` предназначена для сборки программ из исходных кодов.

После запуска, `make` в текущей директории ищет файл `makefile` или `Makefile`. В этих файлах описаны правила сборки проектов.

Файл `Makefile` (`makefile`) предназначен для описания правил сборки.

Если в этом файле в начале строки встречаются слова, заканчивающиеся на «:» — эти слова обозначают цель выполнения (`target`). После них написаны правила, которые необходимо выполнить.

Например:

```
all:
install:
clean:
```

Если программу `make` запустить без параметров, то будут выполнены все правила, описанные в цели `all`. Для выполнения правил, описанных другой целью, следует запускать `make` с указанием имени цели.

Например:

```
make install
make install clean
```

## Пример сборки программ из исходных кодов. Программа *screen*

А теперь давайте рассмотрим процесс сборки и установки программного обеспечения из исходных кодов на примере программы `screen`.

Давайте проверим, установлена ли у нас эта программа, и если «да», то удалим ее:

```
root@cl:~# ls /var/log/packages | grep screen
font-screen-cyrillic-1.0.1-noarch-2
screen-4.0.3-i486-1
xscreensaver-5.05-i486-1
root@cl:~# removepkg screen-4.0.3-i486-1

Removing package /var/log/packages/screen-4.0.3-i486-1...
Removing files:
--> Deleting symlink /usr/bin/screen
--> /etc/screenrc.new no longer exists. Skipping.
--> /etc/skel/.screenrc.new no longer exists. Skipping.
--> Deleting /usr/bin/screen-4.0.3
--> Deleting /usr/doc/screen-4.0.3/COPYING
--> Deleting /usr/doc/screen-4.0.3/ChangeLog
--> Deleting /usr/doc/screen-4.0.3/FAQ
.....
--> Deleting empty directory /usr/share/screen/utf8encodings/
--> Deleting empty directory /usr/share/screen/
--> Deleting empty directory /usr/doc/screen-4.0.3/
root@cl:~#
```

Инсталляционный DVD диск у меня уже смонтирован в /mnt/dvd. Исходные коды должны находится в корневой директории диска в каталоге slacksource.

Но давайте проверим, с помощью программы find:

```
root@c1:~# find /mnt/dvd -name screen*
/mnt/dvd/slackware/ap/screen-4.0.3-i486-1.tgz.asc
/mnt/dvd/slackware/ap/screen-4.0.3-i486-1.tgz
/mnt/dvd/slackware/ap/screen-4.0.3-i486-1.txt
/mnt/dvd/source/ap/screen
/mnt/dvd/source/ap/screen/screen.SlackBuild
/mnt/dvd/source/ap/screen/screen-4.0.3.tar.bz2
/mnt/dvd/source/l/ncurses/screeninfo.src
root@c1:~#
```

Скопируем директорию с исходниками screen на жесткий диск и распакуем архив:

```
root@c1:~# cp -r /mnt/dvd/source/ap/screen /usr/src/
linux/                linux-2.6.24.5/ rpm/
root@c1:~# cp -r /mnt/dvd/source/ap/screen /usr/src
root@c1:~# cd /usr/src/screen/
root@c1:/usr/src/screen# tar -xjf screen-4.0.3.tar.bz2
root@c1:/usr/src/screen# ls
screen-4.0.3/  screen-4.0.3.tar.bz2  screen.SlackBuild*  slack-desc
root@c1:/usr/src/screen# cd screen-4.0.3
root@c1:/usr/src/screen/screen-4.0.3#
```

Программа ./configure предназначена для подготовки исходных кодов к сборке. Функция --help позволяет просмотреть возможные параметры.

```
# ./configure --help
`configure' configures this package to adapt to many kinds of systems.
```

```
Usage: ./configure [OPTION]... [VAR=VALUE]...
```

To assign environment variables (e.g., CC, CFLAGS...), specify them as VAR=VALUE. See below for descriptions of some of the useful variables.

Defaults for the options are specified in brackets.

Configuration:

-h, --help	display this help and exit
--help=short	display options specific to this package
--help=recursive	display the short help of all the included packages
-V, --version	display version information and exit
-q, --quiet, --silent	do not print `checking...' messages
--cache-file=FILE	cache test results in FILE [disabled]
-C, --config-cache	alias for `--cache-file=config.cache'
-n, --no-create	do not create output files
--srcdir=DIR	find the sources in DIR [configure dir or `..']

Installation directories:

--prefix=PREFIX	install architecture-independent files in PREFIX [/usr/local]
--exec-prefix=EPREFIX	install architecture-dependent files in EPREFIX [PREFIX]

By default, `make install' will install all the files in `/usr/local/bin', `/usr/local/lib' etc. You can specify an installation prefix other than `/usr/local' using `--prefix', for instance `--prefix=\$HOME'.

For better control, use the options below.

```
.....
.....
```

Use these variables to override the choices made by `configure' or to help it to find libraries and programs with nonstandard names/locations.

#

Выполним программу ./configure указав путь для установки программы /opt/screen

```
# ./configure --prefix=/opt/screen
this is screen version 4.0.3
checking for gcc... gcc
checking for C compiler default output... a.out
checking whether the C compiler works... yes
checking whether we are cross compiling... no
checking for suffix of executables...
checking for suffix of object files... o
checking whether we are using the GNU C compiler... yes
checking whether gcc accepts -g... yes
.....
.....
Now please check the pathnames in the Makefile and in the user
configuration section in config.h.
Then type 'make' to make screen. Good luck.
#
```

Выполним сборку программы screen:

```
# make
CPP="gcc -E " srcdir=. sh ./osdef.sh
AWK=gawk CC="gcc -g -O2" srcdir=. sh ./comm.sh
AWK=gawk srcdir=. sh ./term.sh
gcc -c -I. -I. -g -O2 screen.c
gcc -c -I. -I. -g -O2 ansi.c
.....
.....
gcc -o screen screen.o ansi.o fileio.o mark.o misc.o resize.o socket.o search.o
tty.o term.o window.o utmp.o loadav.o putenv.o help.o termcap.o input.o
attacher.o pty.o process.o display.o comm.o kmapdef.o acls.o braille.o
braille_tsi.o logfile.o layer.o sched.o telnet.o nethack.o encoding.o -lcurses
-lutempter -lutil -lcrypt
#
```

Установим программу screen:

```
# make install
./etc/mkinstalldirs /opt/screen/bin `sed < config.h -n -e '/define
SCREENENCODINGS/s/^\.*"([^\"]*)"/\1/p`
mkdir /opt/screen
mkdir /opt/screen/bin
mkdir /opt/screen/share
mkdir /opt/screen/share/screen
mkdir /opt/screen/share/screen/utf8encodings
.....
.....
#
```

Удалим временные файлы и директории:

```
# make clean
rm -f screen.o ansi.o fileio.o mark.o misc.o resize.o socket.o search.o tty.o
term.o window.o utmp.o loadav.o putenv.o help.o termcap.o input.o attacher.o
pty.o process.o display.o comm.o kmapdef.o acls.o braille.o braille_tsi.o
logfile.o layer.o sched.o telnet.o nethack.o encoding.o screen config.cache
osdef0.c osdef1.sed osdef2.sed
rm -f tty.c term.h comm.h osdef.h kmapdef.c core
#
```

Теперь необходимо добавить пути к исполняемым файлам screen к переменной PATH. Вот один из вариантов:

```
root@c1:~# mcedit /etc/profile.d/screen.sh
#!/bin/bash
export PATH=$PATH:/opt/screen/bin
export MANPATH=$MANPATH:/opt/screen/man

root@c1:~# chmod a+x /etc/profile.d/screen.sh
root@c1:~#
```

Для того, что бы применить значения переменной PATH необходимо выйти из системы и войти вновь.

```
root@c1:~# echo $MANPATH
/usr/local/man:/usr/man:/usr/lib/java/man:/opt/screen/man
root@c1:~#
```

Все, программа screen установлена и функционирует.

### Описание программы screen

Программа screen очень удобный инструмент системного администратора. Она позволяет отвязать сессию работы с командным интерпретатором от консоли.

Представьте себе классическую ситуацию, когда ваш сервер находится достаточно далеко от вас. Например в серверной. У сервера нет консоли. К нему подключены лишь два провода — электрическое питание и сетевой информационный кабель. Вы подключаетесь к нему удаленно, например через ssh. Все запущенные приложения выполняются пока открыт ssh-клиент. Но это не безопасно, особенно для различных длительных процессов.

Здесь вам и поможет screen. Программа запускается следующей командой:

```
root@c1:~# screen

Screen version 4.00.03 (FAU) 23-Oct-06

Copyright (c) 1993-2002 Juergen Weigert, Michael Schroeder
Copyright (c) 1987 Oliver Laumann

This program is free software; you can redistribute it and/or modify it under
the terms of the GNU General Public License as published by the Free Software
Foundation; either version 2, or (at your option) any later version.

This program is distributed in the hope that it will be useful, but WITHOUT
ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS
FOR A PARTICULAR PURPOSE. See the GNU General Public License for more
details.

You should have received a copy of the GNU General Public License along with
this program (see the file COPYING); if not, write to the Free Software
Foundation, Inc., 59 Temple Place - Suite 330, Boston, MA 02111-1307, USA.

Send bugreports, fixes, enhancements, t-shirts, money, beer & pizza to
screen@uni-erlangen.de

[Press Space or Return to end.]
```

После этого, нажав клавиши Space или Enter, мы получаем доступ к виртуальной консоли:

```
root@c1:~# pwd
/root
root@c1:~#
```

Теперь отключимся от виртуальной консоли. При этом наша сессия останется неизменной.

Отключение выполняется с помощью комбинации клавиш:

<Ctrl>+A и <Ctrl>+D

```
root@c1:~# screen
[detached]
root@c1:~#
```

Возвращение к виртуальной консоли:

```
root@c1:~# screen -r

root@c1:~# pwd
/root
root@c1:~#
```

Мы увидели прежнюю картину.

Количество виртуальных консолей, полученных с помощью программы screen, весьма велико.

Новую виртуальную консоль получаем прежним способом.

```
[detached]
root@c1:~# screen
```

```
Screen version 4.00.03 (FAU) 23-Oct-06
```

```
Copyright (c) 1993-2002 Juergen Weigert, Michael Schroeder
Copyright (c) 1987 Oliver Laumann
```

```
This program is free software; you can redistribute it and/or modify it under
the terms of the GNU General Public License as published by the Free Software
Foundation; either version 2, or (at your option) any later version.
```

```
This program is distributed in the hope that it will be useful, but WITHOUT
ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS
FOR A PARTICULAR PURPOSE. See the GNU General Public License for more
details.
```

```
You should have received a copy of the GNU General Public License along with
this program (see the file COPYING); if not, write to the Free Software
Foundation, Inc., 59 Temple Place - Suite 330, Boston, MA 02111-1307, USA.
```

```
Send bugreports, fixes, enhancements, t-shirts, money, beer & pizza to
screen@uni-erlangen.de
```

```
[Press Space or Return to end.]
```

```
root@c1:~# tail -f /var/log/messages
Sep 18 12:02:21 c1 -- MARK --
Sep 18 12:22:21 c1 -- MARK --
Sep 18 12:42:22 c1 -- MARK --
Sep 18 13:02:22 c1 -- MARK --
Sep 18 13:22:22 c1 -- MARK --
Sep 18 13:42:22 c1 -- MARK --
Sep 18 14:02:22 c1 -- MARK --
Sep 18 14:22:22 c1 -- MARK --
Sep 18 14:42:23 c1 -- MARK --
Sep 18 15:02:23 c1 -- MARK --
```

Теперь отключимся от этой консоли и попытаемся подключиться к первой.

```
[detached]
root@c1:~# screen -r
There are several suitable screens on:
    3170.pts-0.c1    (Detached)
    3174.pts-0.c1    (Detached)
Type "screen [-d] -r [pid.]tty.host" to resume one of them.
root@c1:~#
```

Как видим, программа `screen` указала нам PID имеющихся консолей. Рискнем предположить, что процесс с меньшим PID первичен.

```
root@c1:~# screen -r 3170
```

```
root@c1:~# pwd
```

```
/root
```

```
root@c1:~#
```

Для завершения работы программы `screen` выполните команду `exit`.

## Глава 19. Система печати в Linux

Исторически сложилось так, что в Linux существовало две системы печати — BSD и SystemV. Более популярна была система печати LPRng построенная на BSD системе. Относительно недавно появилась новая система печати CUPS, более мощная и универсальная, хотя и не лишенная своих недостатков.

Вначале мы рассмотрим систему печати LPRng. А потом настроим локальный и сетевой доступ к принтеру используя систему печати CUPS.

### Классические средства печати

Исторически для печати в UNIX-системах существовали две системы печати: LPD (Line Printer Daemon) [RFC1179], разработанная для Berkeley UNIX (или BSD-система), и AT&T Line Printer system. Эти системы печати были созданы в 70-х годах для печати текстов на построочно-печатающих (линейных) принтерах. Принимая во внимание, что аппаратные средства печати (проще говоря, принтеры) с тех пор существенно изменились, можно было бы предположить, что существенно переработаны и программные средства для управления печатью. Однако, этого не произошло. Хотя и были созданы различные улучшенные системы печати [LPRng, Palladin, PLP], однако ни одна из этих новых разработок не изменяла фундаментальные возможности этих систем. Впрочем, как показало время и практика, возможности этих систем вполне достаточны и при небольших доработках удовлетворяют и современные потребности.

Как и во всех UNIX-системах, в Linux файл, предназначенный для печати, вначале пересылается во временную область (проще говоря, временный каталог), которая называется областью спулинга. Дело в том, что принтеры являются относительно медленными устройствами, и система заботится о том, чтобы не задерживать работу на время распечатки файла. Фоновый процесс — демон печати — постоянно сканирует область спулинга в ожидании файлов, предназначенных для печати. Для каждого принтера, подключенного к системе, заводится своя область спулинга. Таким образом, область спулинга представляет собой очередь заданий на печать, ожидающих того момента, когда освободится соответствующий принтер и демон печати отправит данное задание на печать (в фоновом режиме).

В основу подсистемы печати в Linux положена BSD-система — LPD, а точнее, доработанный вариант этой системы LPRng. LPRng состоит из отдельных программ, которые обеспечивают выполнение отдельных функций подсистемы печати:

- **lpd** — демон системы печати. Обычно запускается на этапе загрузки системы из файла `rc`, но может быть запущен и пользователем.
- **lpr** — пользовательская команда печати. Программа `lpr` принимает подлежащие печати данные и помещает их в буферный каталог, где их находит `lpd` и выводит на печать. Программа `lpr` — единственная программа, которая может ставить новые задания в очередь печати. Другие программы, которым необходимо использовать печать, обращаются для этого к `lpr`.
- **lpq** — программа, позволяющая просматривать очередь заданий, ожидающих печати на указанном принтере.
- **lpc** — команда контроля системы `lpd`. С помощью `lpc` можно отключать принтеры, останавливать или переупорядочивать очереди печати и т.п. Некоторые из функций этой команды доступны пользователям, но в основном это средство для администратора.
- **lprm** — эта команда позволяет удалить одно или несколько заданий из очереди печати. При этом стираются соответствующие файлы данных и из системы печати удаляются все ссылки на них.

Взаимодействие `lpr` и других программ этого набора с демоном `lpd` осуществляется с использованием сетевых средств, так что они могут запускаться и на других компьютерах. Рассмотрим вкратце, как осуществляется печать файла в системе LPD.

Когда вызывается программа `lpr`, она первым делом выбирает принтер, на который будет производиться печать. Этот выбор определяется либо параметром командной строки `Pprinter`, либо значением переменной окружения `PRINTER`, либо же используется общесистемный принтер, заданный по умолчанию (это принтер с именем `lp`). Как только `lpr` узнает, на какой принтер отправлять текущее задание, она ищет описание этого принтера в базе данных об имеющихся принтерах, которая хранится в файле `/etc/printcap`. Из этой базы `lpr` получает имя каталога (области спулинга), в который следует помещать задания для найденного принтера. Обычно этот буферный каталог имеет имя `/var/spool/lpd/printer`. Такой каталог (область спулинга) должен существовать, если вы хотите печатать на принтере с именем `printer`.

Для каждого задания программа `lpr` помещает в буферный каталог два файла: `sfxxx` и `dfxxx`, где `xxx` — номер текущего задания. Файл `sfxxx` содержит справочную информацию и информацию об обработке задания. Источником этих сведений являются командная строка запуска программы, переменные среды процесса, который запустил эту программу, и глобальная конфигурация системы. Файл `dfxxx` содержит данные, подлежащие печати.

После постановки задания в очередь `lpr` уведомляет демона `lpd` о появлении задания на печать. Взаимодействие `lpr` с `lpd` происходит через именованный сокет `/dev/printer`. Демон `lpd` тоже обращается к файлу `/etc/printcap`, чтобы узнать, какой принтер должен использоваться для печати, и является он локальным или удаленным. Если в `/etc/printcap` указано, что принтер подключен локально, `lpd` проверяет наличие демона печати, обрабатывающего соответствующую очередь. Дело в том, что для обработки каждой очереди `lpd` создает отдельную копию самого себя. Если такой копии еще не имеется, она создается и ей передается обработка очереди. Если соответствующий принтер подключен к другой машине, `lpd` устанавливает соединение с демоном `lpd` удаленной машины и пересылает туда файл данных и управляющий файл.

Обслуживание заданий печати осуществляется по правилу "первым пришел — первым обслужен" (FIFO). Системный администратор может при желании изменить порядок печати с помощью программы `lprc`.

## Файл `/etc/printcap`

Файл `/etc/printcap` — это главная база данных системы печати LPD. Принтер будет получать задания на печать только в том случае, если он (принтер) описан в этом файле. Поэтому для получения возможности использовать принтер вы должны добавить очередь печати к `lpd`, т. е. создать новый буферный подкаталог в каталоге `/var/spool/lpd` и добавить соответствующую запись в файл `/etc/printcap`.

Запись в файле `/etc/printcap` выглядит примерно так:

```
# local djet500
lp|dj|deskjet:\
:sd=/var/spool/lpd/dj:\
:mx#0:\
:lp=/dev/lp0:\
:sh:
```

Каждый элемент файла `/etc/printcap` начинается со строки, задающей имена принтера (их может быть несколько), разделяемые вертикальной чертой. Затем следует ряд параметров конфигурации, разделенных двоеточиями (обычно каждый параметр заключается в двоеточия с обеих сторон). Каждый параметр имеет вид `xx=строка` или `xx#число`, где `xx` — двухсимвольное имя параметра, а `строка` и `число` — присваиваемые ему значения. Если никакого значения не присваивается, переменная является булевой, и ее присутствие означает "истина". Допускаются пустые операторы: два рядом стоящих двоеточия. Строки, начинающиеся символом `#`, содержат комментарий. Обратная косая черта в конце строки означает, что в следующей строке идет продолжение текущей строки.

Приведенный выше пример записи определяет принтер называемый `lp`, `dj`, или `deskjet`, его спул размещается в директории `/var/spool/lpd/dj`, максимальный размер задания не имеет ограничения, печать производится на устройство `/dev/lp0`, и страница с заголовком (с именем пользователя, который отправил задание на печать, и другой информацией) не добавляется в начало задания печати.

Подробнее о том, как создавать такие записи, можно прочитать на справочной странице для `printcap`. Но о некоторых самых важных (и обязательных) параметрах стоит рассказать здесь, хотя бы просто для примера:

- **sd=буферный\_каталог** У каждого принтера должен быть свой буферный каталог. Все буферные каталоги должны находиться в одном каталоге (обычно это `/var/spool/lpd`) и иметь имена, совпадающие с полными именами обслуживаемых ими принтеров. Буферный каталог нужен даже в том случае, если описываемый принтер подключен к другой машине: задания находятся на локальной машине до тех пор, пока они не будут переданы на печать.
- **lp=имя\_устройства** В таком виде этот параметр должен задаваться только для локального принтера. Если принтер находится на другой машине, вместо имени устройства указывается имя уникального файла, который существует и расположен на локальном диске. Чтобы принтер мог возвращать через указанный файл информацию о своем состоянии, необходимо задать в элементе булеву переменную `gw`, чтобы устройство было открыто и для чтения, и для записи.
- **rm и rp** Во многих случаях в качестве печатающего устройства используется сетевой принтер, подключенный к какому-то другому компьютеру в локальной (или даже глобальной) сети. В таком случае на вашей машине в файле `/etc/printcap` должны присутствовать две переменные `rm` и `rp`. В переменной `rm` определяется машина, на которую должны посылаться задания, а переменная `rp` задает



имя принтера на этой машине.

- **mx** Переменная `mx` используется для задания максимального размера (в байтах) печатаемого файла. Этот параметр может использоваться системными администраторами для того, чтобы предотвратить отрицательные последствия распространенной ошибки начинающих пользователей, состоящей в попытке на печать двоичных файлов (которые обычно имеют большую длину). Поскольку такие файлы могут содержать произвольные символы, в том числе символы, которые служат в качестве управляющих команд для принтера, печать двоичных файлов может приводить, например, к неоправданному расходованию бумаги и другим неприятным последствиям. На локальном принтере персонального компьютера этот параметр можно и не задавать. Я привел его для того, чтобы отметить, что для этого параметра, как и для всех других числовых параметров, значение должно отделяться от имени параметра знаком `#`, например, `mx#0`. Если написать `mx=0`, то такая запись не вызовет сообщений об ошибке, но она не изменяет значения переменной `mx`.

Еще один очень существенная группа параметров — это параметры `of`, `if` и `nf`. Но о них надо поговорить особо, что мы и сделаем чуть ниже в подразделе "Фильтры".

## Фильтры

Когда задание на печать дождется своей очереди, `lpd` создает ряд программных каналов между буферным файлом и печатающим устройством для передачи данных, подлежащих печати. Посередине этой цепочки `lpd` устанавливает процесс-фильтр, в задачи которого входит просмотр и дополнительная обработка потока данных, направляемых на принтер. Процессы-фильтры могут выполнять над данными различные преобразования, в частности, форматирование и поддержку различных протоколов, которые могут понадобиться для работы с данным принтером.

Фильтр — это, как правило, просто сценарий `shell`, который вызывает ряд конвертирующих программ. Фильтр можно указать в командной строке вызова программы `lpr`. Если в командной строке фильтр не указан, то используются фильтры, заданные параметрами `if`, `of` и `nf` соответствующей записи в файле `/etc/printcap`. Если в этой записи присутствует переменная `if`, а параметра `of` нет, то устройство (принтер) будет открываться один раз для каждого задания, а фильтр будет посылать одно задание на принтер и завершать работу. Если есть `of`, а `if` нет, то `lpd` однократно откроет устройство и вызовет программу-фильтр для отправки сразу всех заданий, стоящих в очереди. Это полезно для печати на тех устройствах, соединение с которыми требует большого времени. Одновременного использования параметров `of` и `if` следует избегать, а из двух предыдущих вариантов рекомендуется выбирать использование параметра `if`. Соответствующий элемент в записи файла `/etc/printcap` может иметь примерно такой вид:

```
:if=/var/spool/lpd/dj/filter:\
```

Если никакого фильтра вообще не задано, то вывод на печать может выглядеть очень некрасиво. Например, при печати обычного текстового файла вывод может выглядеть примерно так:

This is line one.

This is line two.

This is line three.

Печать файла в формате PostScript выдаст листинг команд PostScript, напечатанных с этим "лестничным эффектом", а не полезный вывод. В руководстве "Printing HOWTO" приводится следующий пример простого фильтра, предназначенного только для того, чтобы устранить "лестничный эффект":

```
#!/perl
# Предыдущая строка должна содержать полный путь к perl
# Скрипт должен быть исполнимым: chmod 755 filter
while(<STDIN>){chop $_; print "$_\r\n";};
# вы можете также добавить в конец прогон страницы: print "\f";
```

Этот текст надо сохранить в виде файла `/var/spool/lpd/dj/filter`, после чего будут нормально печататься обычные текстовые файлы.

Но печать простых ASCII-файлов — это только частный случай печати. В большинстве случаев в настоящее время печатаются файлы в других форматах, например, PostScript. Проблема вывода таких файлов на печать тоже решается путем использования фильтра, только гораздо более сложного. Таких фильтров разработано уже достаточно много, но самый важный из них — программа `ghostscript`.

## PostScript и Ghostscript

К сожалению пользователей, фирмы-производители принтеров долгое время не могли достигнуть согласия в вопросе о выборе управляющих сигналов для производимых ими устройств. В результате для каждого принтера до сих пор необходим особый драйвер. Однако со времен так называемой "революции настольных издательских систем" 80-х годов в качестве стандартного языка управления принтером постепенно утвердился язык PostScript, разработанный фирмой Adobe Systems, Inc. И не только в UNIX-среде, а в издательском деле вообще.

Этот язык представляет собой специальный язык программирования для описания выводимой на печать страницы с текстом или графикой. Adobe Systems, Inc., изначально разработавшая стандарт на PostScript, открыла его для свободного распространения. Отметим еще, что формат PDF (Формат Переносимого Документа Adobe) — это в действительности чуть больше чем несколько преобразованный PostScript в сжатом файле.

Идея, заложенная в основу разработки PostScript, проста: все, что можно напечатать, описывается с помощью специального языка программирования, принтер же должен этот язык понимать. И принтеры, "понимающие" язык PostScript, т. е. имеющие встроенный PostScript-интерпретатор (так называемые PostScript-принтеры), быстро появились. К сожалению, они оказались стабильно дороже обычных принтеров. Тогда были разработаны программные PostScript-интерпретаторы, которые берут данные в формате PostScript и преобразуют в специфический для данного принтера управляющий код. Это дает вам виртуальный PostScript-принтер и позволяет использовать принтеры, не имеющие аппаратного интерпретатора.

Вероятно, одним из лучших программных интерпретаторов языка PostScript является Ghostscript (<http://www.cs.wisc.edu/~ghost/>), или просто gs. Он существует в двух вариантах. Коммерческая версия Ghostscript, называемая Aladdin Ghostscript или AFPL Ghostscript, свободна для персонального использования, но не может распространяться с коммерческими дистрибутивами Linux. В составе последних доступен GNU Ghostscript, представляющий собой тот же gs, только версией ниже и с другим лицензионным соглашением. В составе Ghostscript имеется внушительный набор фильтров — аппаратно ориентированных модулей, позволяющих получать изображение на различных устройствах. Устройствах, а не принтерах, поскольку Ghostscript может обеспечить вывод на любое графическое устройство. Именно gs присутствует в качестве фильтра в `/etc/printcap` — конфигурационном файле `lpd`. Опции запуска gs в качестве фильтра определяются типом принтера.

## Шрифты для Ghostscript

Для пакета Ghostscript разработаны PostScript-шрифты, которые обеспечивают высокое качество печати на не-PostScript принтерах. Такие шрифты наверняка найдутся на вашем дистрибутивном диске в виде пакета `ghostscript-fonts`. Однако именно со шрифтами и связано большинство проблем, которые возникают при настройке принтера.

Дело в том, что программе Ghostscript надо точно знать, где расположены шрифты для нее. Но поскольку стандарт FHS (Filesystem Hierarchy Standard), о поддержке которого заявили все составители дистрибутивов, пока еще не утвердился окончательно, структура каталогов в Linux меняется от версии к версии даже в пределах одного дистрибутива. Поэтому файлы шрифтов могут оказаться где угодно. Очень часто — не там, где их будет искать Ghostscript. В результате при попытке распечатать какой-либо документ вы можете получить далеко не то, что ожидали: от несоответствия внешнего вида распечатанного документа вашему замыслу до искажения или отсутствия фрагментов текста, требующего, в соответствии с PostScript-файлом, того самого шрифта, который не смог загрузить Ghostscript. Положение усугубляется тем, что, по крайней мере, часть документов включает кириллицу, а некоторые дистрибутивы не имеют в каталогах, сканируемых по умолчанию Ghostscript, шрифтов с кириллицей.

Преодолеть эти трудности в принципе не сложно. Но, прежде чем рассказать, как это сделать, надо сказать, что в Linux имеется программа `ghostview` (`gv`), назначение которой — принять вывод `ghostscript` и вывести изображение на экран. Это дает инструмент, обеспечивающий возможность предварительного просмотра ("print preview") для любого приложения, генерирующего PostScript-файлы. С помощью `gv` вы сможете определить, связаны ли ваши проблемы с выбором типа принтера или с работой `gs` в целом. Видите на экране, но не получаете на печати — попробуйте другой фильтр (выберите другой принтер), не видите ничего "путного" — продолжаем разбираться с настройкой `ghostscript`.

Теперь надо отметить, что программу Ghostscript можно запускать не только в качестве фильтра для `LPD`, но и из командной строки (для этого надо дать команду `gs`). Этой возможностью и воспользуемся для целей отладки.

Вначале запустите команду `gs` с опцией `-help`. В результате вы получите, во-первых, краткий информативный список опций и доступных драйверов (заметим, что этот список является списком только вкомпилированных, а не всех доступных драйверов), и, во-вторых, перечень путей поиска. Этот список можно, конечно, изменить, но

для этого надо перекомпилировать программу. Если же вы не хотите заниматься компиляцией, надо поместить файлы шрифтов именно в эти каталоги.

Но этого еще недостаточно для того, чтобы Ghostscript могла использовать шрифты. Дело в том, что эта программа обращается к шрифтам по именам, записываемым в той нотации, в которой допускается их использование в PostScript-файлах. Соответствие между такими названиями шрифтов и именами реальных файлов шрифтов задается файлом Fontmap (или Fontmap.GS), который располагается в каталоге /usr/share/ghostscript/N.NN, где N.NN — номер версии программы ghostscript. Каждая строка (кроме строк комментариев) этого файла состоит из трех элементов.

Первым стоит имя, под которым шрифт будет известен программе Ghostscript, причем перед этим именем должен стоять слэш (/), либо имя должно быть заключено в круглые скобки.

Далее следует имя файла шрифта либо синоним (alias) имени шрифта. Если указывается имя файла шрифта, то оно должно быть заключено в круглые скобки и записано с указанием расширения (обычно это gsf, но допускаются также pfa и pfb), а также должно соответствовать правилам формирования имен файлов в MS-DOS, т. е. состоять из букв (в нижнем регистре), цифр и знаков подчеркивания. Если же это синоним, то указывается имя одного из уже известных программе Ghostscript шрифтов, причем перед этим именем должен стоять слэш (/).

Завершает строку точка с запятой, перед которой должен стоять, по крайней мере, один пробел или знак табуляции.

Пути к файлам шрифтов в файле Fontmap не указаны. Если вы не использовали предлагаемые руководством средства принудительной "ориентации" ghostscript (параметры командной строки и переменные окружения), то gs будет использовать "пути по умолчанию", заданные при компиляции. В этих каталогах должны иметься файлы fonts.dir, которые содержат описание фонтон в данном каталоге.

Таким образом, в зависимости от потребностей вы можете либо внести в Fontmap необходимый шрифт (предварительно поместив соответствующий файл в один из доступных программе каталогов и указав имя файла в добавляемой строке), либо назначить в качестве синонима нужного шрифта имя одного из уже известных программе шрифтов. Например, сделать шрифт /Courier синонимом изначально известного программе шрифта /NimbusMonL-Regu (которому, в свою очередь соответствует файл (n022024l.pfb)). Если задача — в основном печатать файлы, PostScript-содержимое которых вне вашего контроля, — подберите синонимы для нужных шрифтов из числа известных программе. Если PostScript-файл генерируется под вашим контролем — просто выбирайте один из имеющихся в системе шрифтов. Разумеется, не забыв при этом описать его в Fontmap.

После этого выполните команду

```
$ gv filename.ps
```

Если вы при этом увидите на экране весь текст из файла filename.ps, вы можете попытаться отпечатать файл и на принтере. Если же вместо текста увидите пустой лист или шрифт вам не нравится, продолжайте экспериментировать с настройкой шрифтов.

## Печать на удаленный принтер

Если ваш компьютер подключен к локальной сети, то не обязательно иметь принтер, непосредственно к нему подключенный, можно пользоваться принтером, подключенным к какому-то другому компьютеру. Настройка такого принтера на вашем компьютере требует только указания того, к какому компьютеру в сети подключен принтер (это делается с помощью задания переменных gm и gr в файле /etc/printcap, о чем было сказано выше). Если использовать утилиту printconf-gui, то достаточно при выборе типа очереди выбрать вариант "UNIX printer (lpd Queue)", если это другой Linux-компьютер. Если принтер подключен к Windows-компьютеру или отдан в сеть через Samba-сервер, то, естественно, надо выбирать тип очереди "Принтер Windows (ресурс Samba)"

На удаленном компьютере должен быть разрешен доступ к этому принтеру. В Linux это делается с помощью файла /etc/lpd.perms (см. соответствующую страницу руководства man).

## Система печати CUPS

Служба печати CUPS (Common Unix Printing System) предназначена для унификации доступа ко всем принтерам, находящимся в локальной сети. Возможно, будь Linux единственной ОС в мире, всем хватало бы возможностей стандартной Berkeley LPD и такая система не понадобилась, но в реальных условиях именно CUPS может обеспечить доступ к Windows и SMB-принтерам, обладая при этом возможностью периодически

обновлять информацию о принтерах и объединять их в группы (в зависимости от типа или других параметров).

Компания Easy Software Products, разработавшая CUPS, распространяет ее под лицензией GPL, предоставляя на платной основе поддержку и дополнительные функции .

Программу можно скачать с сайта <http://gazette.linux.ru.net/www.cups.org>, либо получить в виде бинарных пакетов в большинстве дистрибутивов Linux (любой современный дистрибутив содержит в своём составе пакеты CUPS).

## Как это работает?

По идее, CUPS становится полной заменой системы печати LPD, подставляя на место команды lpr и драйверов LPD свои аналоги. Программы Linux (и Unix) не замечают этой подмены, так как обе системы основаны на базе языка описания страниц Postscript.

В CUPS включена поддержка большинства принтеров, подключаемых через LPT, USB и даже COM-порты. Конечно, подключение через COM-порт - это не лучший вариант, но если у вас еще остались старые матричные принтеры, подключаемые через этот интерфейс, то CUPS позволит Вам собрать из кучи такого железа неплохой принт-сервер. Может быть, это как раз то, что нужно Вам?

## Обновление информации о принтерах

Служба CUPS обладает возможностью, привычной скорее для мира Windows, чем для Linux: она извещает все компьютеры в локальной сети о принтерах, доступных для печати и их состоянии.

Естественно (в отличие от Windows :), эта ее способность поддается настройке. Внося изменения в файл конфигурации CUPS (cupsd.conf), можно определить какие компьютеры (точнее, в каких подсетях) будут получать такие извещения и как часто, что позволяет сократить неизбежный в таком случае широковещательный трафик.

## Классификация принтеров

Группа принтеров или класс (в терминологии CUPS) - это несколько принтеров, которые пользовательские приложения воспринимают как один. К примеру, можно создать класс ЦВЕТНЫЕ ПРИНТЕРЫ, объединяющие все цветные принтеры. Пользователь может настроить свою программу так чтобы печатать на принтер ЦВЕТНЫЕ ПРИНТЕРЫ, и получить распечатку на любом из этих принтеров. На каком именно - будет зависеть от прав этого пользователя, либо от того, какой из принтеров будет доступен в это время.

В то же время, даже если какой-либо принтер является членом группы, это не мешает пользователю печатать именно на этот принтер. А вот уже с помощью списков доступа CUPS можно добиться того, что конкретный принтер будет отвергать такие попытки, а группа принтеров, в которые он входит - напечатает задание. В результате пользователи смогут печатать на группы принтеров, а не на один принтер - все зависит от Вас!

## Интеграция с MS Windows

CUPS может использовать принтеры, к которым на Windows-компьютере открыт общий доступ, а также сетевые принтеры, использующие протокол SMB - в ее терминах они называются "Windows printer using Samba". Все, что нужно - просто указать адрес принтера в формате: smb://server/printer.

Благодаря серверу Samba Windows-компьютеры также могут использовать принтеры CUPS. Настройка сервера SAMBA подробно рассматривается в курсе «Сетевое администрирование Linux». Так как Samba воспринимает принтер CUPS так же, как LPD-принтер, то при этом можно использовать любые стандартные опции.

## Настройка CUPS

Демон cupsd запускается при старте системы и открывает на прослушивание 631 порт. Для эмуляции BSD системы печати требуется запуск отдельного демона: /usr/lib/cups/daemon/cups-lpd

Для полноценной поддержки принтеров необходимо наличие PPD файлов, описывающих эти принтеры. Такие файлы поставляются как с CUPS, так и в виде отдельных пакетов. Если в системе нет PPD файла принтера, его можно найти либо на CD принтера, либо в Интернет ([www.linuxprinting.org](http://www.linuxprinting.org)). PPD файлы должны находиться в директории /usr/share/cups/model.

Для добавления принтера в командной строке следует пользоваться программой `lpadmin`. При добавлении принтера потребуется указать имя PPD файла, включая путь к нему относительно директории `/usr/share/cups/model`. Так же потребуется указать устройство принтера.

Чтобы узнать какие устройства поддерживаются системой печати следует использовать программу `lpinfo`.

```
# lpinfo -v
```

Вот что получилось у меня:

```
root@c1:~# lpinfo -v
lpinfo: Не удается подключиться к серверу: ????? ????
root@c1:~#
```

Очевидно система печати cups не запущена. Выполним действия по ее запуску:

```
root@c1:~# ls -l /etc/rc.d/rc.cups
-rw-r--r-- 1 root root 4318 2008-04-29 07:38 /etc/rc.d/rc.cups
root@c1:~# chmod a+x /etc/rc.d/rc.cups
root@c1:~# /etc/rc.d/rc.cups
Usage: cups {reload|restart|start|status|stop}
root@c1:~# /etc/rc.d/rc.cups start
cups: started scheduler. [ OK ]
root@c1:~#
```

Возвращаемся к программе `lpinfo`:

```
root@c1:~# lpinfo -v
network socket
network beh
direct hpfax
direct hp
network http
network ipp
network lpd
direct scsi
network smb
root@c1:~#
```

Ниже показаны примеры устройств принтеров:

- `parallel:/dev/lp0` — принтер, подключенный к параллельному порту.
- `usb:/dev/usb/lp0` — принтер, подключенный к USB разъему.
- `socket://192.168.0.1` — сетевой принтер (подключение на 9100 порт).
- `http://192.168.0.1:631/printers/printer` — IPP принтер.

Конфигурационные файлы CUPS находятся в директории `/etc/cups`:

- **cupsd.conf** — основной конфигурационный файл. Используется для ограничения доступа к принтерам и тонких настроек.
- **printers.conf** — содержит описание установленных принтеров.
- **classes.conf** — содержит описание классов принтеров.
- **client.conf** — настройки клиента печати.

## Модули системы печати CUPS

`lpr [опции] [файл ...]`

Программа предназначена для постановки задания в очередь на печать.

Опции:

-P printer — определяет принтер, на котором будет происходить печать.

### lpr [опции]

Программа предназначена для отображения состояния очереди на печать.

Опции:

-P printer — определяет принтер, на котором будет происходить печать.

### lprm [опции]

Программа предназначена для удаления задания из очереди на печать.

Опции:

-P printer — определяет принтер, на котором будет происходить печать.

## Пример подключения USB принтера HP LaserJet 1300

Для программы lpadmin необходимо указать точное местоположение файла драйвера принтера относительно директории /usr/share/cups/model

```
root@c1:~# lpadmin -p Laser -E -v usb:/dev/usb/lp0 -m HP/hp-laserjet_1300-  
hpijs.ppd.gz  
root@c1:~#
```

Отредактируйте файл /etc/cups/cupsd.conf. Имя принтера должно быть одинаковым, и для программы lpadmin и в файле cupsd.conf.

```
<Location /printers/Laser>  
Order Allow,Deny  
Allow From All  
</Location>
```

Напечатаем тестовую страницу:

```
root@c1:~# cd /usr/share/ghostscript/8.62/examples/  
root@c1:/usr/share/ghostscript/8.62/examples# lpr alphabet.ps  
root@c1:/usr/share/ghostscript/8.62/examples#
```

## Сетевая печать

CUPS – современная система печати, поддерживающая все протоколы сетевой печати:

- bsd (515 порт)
- ipr (631 порт)
- smb (требуется Samba).

Сетевой принтер настраивается также как и локальный с той лишь разницей, что указывается иной способ подключения принтера.

Для вышеприведенного примера с принтером HP LaserJet 1300:

```
root@c2:~# lpadmin -p MyPR -E -v http://10.10.0.1:631/printers/Laser -m HP/hp-  
laserjet_1300-hpijs.ppd.gz  
root@c2:~#
```

Аналогичным образом проверяем печать.

## WEB-интерфейс системы CUPS

Настраивать CUPS можно двумя способами. Можно непосредственно редактировать файл конфигурации /etc/cups/cupsd.conf, а можно набрать в браузере <http://localhost:631> и воспользоваться интуитивно понятным веб-интерфейсом, очень похожим на веб-интерфейсы многих сетевых служб.

Наберите в браузере адрес <http://localhost:631>

Или в командной строке:

```
root@c1:~# links http://localhost:631
```

## Глава 20. Ядро Linux

Каждый, кто хоть немного интересовался тем, что такое Linux, обязательно встречал в различных руководствах термин "ядро", по-английски — kernel. Ядро — это важная часть Linux, как и любой другой операционной системы, поскольку именно ядро обеспечивает взаимодействие с аппаратной частью компьютера, распределение ресурсов, управление процессами и многое другое. Когда вы загружаете какое-то приложение с жесткого диска в оперативную память, или переключаетесь между уже работающими приложениями, или когда какое-то приложение записывает информацию в файл на диске, операционная система или активное приложение должно запросить доступ к той части аппаратуры, которая ему необходима. Ядро обеспечивает исполнение таких запросов других частей операционной системы и приложений, а также распределяет память между запускаемыми приложениями. Ядро, таким образом, является посредником между аппаратным и программным обеспечением компьютера, обеспечивающим их взаимодействие.

Работа по совершенствованию ядра Linux ведется международным сообществом разработчиков постоянно, и регулярно появляются новые версии ядра. Естественно, что пользователи хотят иметь последнюю (или, по крайней мере, одну из последних) версий ядра ОС и рано или поздно вы приходите к выводу о том, что пора обновить ядро.

Можно задать вопрос: "В каких случаях это необходимо?". Действительно, если система неплохо работает со старым ядром, то стоит ли заниматься его обновлением? Основными причинами, приводящими к выводу о необходимости обновления ядра, являются:

- обновление аппаратуры компьютера, подключение новых устройств, которые не поддерживаются старым ядром;
- необходимость работы с новыми программами, которые рассчитаны на новую версию ядра и отказываются работать с версией, установленной у вас;
- обнаружение каких-то ошибок в старой версии ядра, в частности таких, которые представляют угрозы с точки зрения безопасности;
- желание повысить производительность системы, используя более совершенную версию ядра, либо оптимизировать ядро для работы с конкретным набором аппаратных средств, имеющихся на вашем компьютере;
- и, наконец, простое любопытство и желание работать с последней версией системы.

Обновить ядро можно двумя способами: установкой готового бинарного образа нового ядра из rpm-пакета и компиляцией ядра из исходных текстов. Первый способ проще, но надо иметь в виду, что скомпилированное где-то и кем-то ядро скорее всего не является оптимальным вариантом для вашей системы. Поэтому приходится применять второй способ — компиляцию ядра из исходных кодов. Для начинающих пользователей Linux компиляция ядра из исходных кодов кажется чем-то супер-сложным и недоступным. Однако я думаю, что, прочитав настоящую главу, вы убедитесь, что это не намного сложнее, чем установка ПО из rpm-пакета.

Так же хочу заметить, что поставщики современных дистрибутивов Linux часто настоятельно не рекомендуют самостоятельно перекомпилировать ядро. Они просят использовать готовые бинарные пакеты для этих целей, так как заплатки и добавления выполненные ими обычно не находятся в открытом доступе. А используя оригинальные исходные тексты ядра вы вероятно потеряете некоторый функционал дистрибутива Linux, используемого вами. Подумайте, прежде чем приступить к сборке своего ядра. И никогда не экспериментируйте на работающих системах!!!

### Нумерация версий ядра

Ядро Linux создается командой программистов под руководством Линуса Торвальдса — создателя первого стабильного ядра Linux. Ядро Linux распространяется на основе лицензии GNU GPL. Подробно узнать об этой лицензии и прочитать ее текст можно на огромном количестве ресурсов в сети internet. Рекомендую для начала обратиться к Википедии (<http://ru.wikipedia.org/wiki/GPL>).

Сайт проекта, занимающегося разработкой и тестированием ядра находится здесь — <http://www.kernel.org>. Здесь же вы найдете ссылки на зеркала для загрузки исходных кодов ядра Linux.

Прежде, чем браться за обновление ядра вы должны четко представлять себе, что за версию вы собираетесь установить. В первую очередь необходимо иметь в виду, что разработчики ядра поддерживают две ветки ядра: стабильную и экспериментальную. Все новшества, вносимые в ядро, вначале появляются в экспериментальных версиях. И только после того, как сообщество разработчиков и добровольных тестировщиков опробует эти

новшества, они переносятся в так называемую стабильную версию.

## Старая система нумерации

Версии ядра было принято нумеровать тремя цифрами, разделенными точками, например, 2.4.8, при этом четная вторая цифра в номере ядра обозначает стабильные версии ядра, а нечетная — экспериментальные версии. Так что принимая решение об установке новой версии ядра вы должны продумать ответ на вопрос, хотите ли вы участвовать в выявлении возможных ошибок в нестабильной версии или предпочитаете работать с уже оттестированным ядром.

Как заявил Линус Торвалдс в одном из своих интервью, он предпочитает как раз заниматься экспериментальной веткой, разрабатывать код, работающий с новыми устройствами. Основным координатором разработки стабильной ветки является в настоящее время Алан Кокс, регулярно выпускающий обновленные версии или "заплатки" к стабильным версиям ядра.

Иногда в номере версии ядра встречаются дополнительные символы после «тире» - это обозначение собственных сборок.

Примеры нумерации:

0.2.65 1.3.12 2.4.22-5 2.5.34 2.6.13

## Современная система нумерации

В новой версии нумерации ядер отказались от «ветки» для разработчиков.

Экспериментальные версии обозначаются после тире, например: -pre1.

В нумерацию добавлена четвертая цифра, определяющая версию исправления ошибок.

Примеры нумерации:

2.6.13 2.6.13.4 2.6.14-pre1

## О компиляции нового ядра

Как было сказано в начале данного раздела, основная функция ядра состоит в том, чтобы обеспечить взаимодействие с аппаратурой компьютера. Обслуживание некоторых составляющих аппаратного обеспечения (таких, как память, например) напрямую встроено в ядро. Для тех частей аппаратуры, которые могут быть нестандартными, имеются драйверы устройств, которые обеспечивают взаимодействие ОС с аппаратурой. Большинство пользователей компьютеров с ОС Windows знакомы с понятием драйвера хотя бы потому, что после установки нового оборудования они вынуждены устанавливать и программные драйверы для этого оборудования. Только после этого становится возможным использовать вновь установленную аппаратную составляющую. В терминологии, принятой в Linux, драйвера называются "модулями". Таким образом, поддержка аппаратных устройств может быть обеспечена двумя способами: либо путем встраивания такой поддержки в ядро, либо путем использования соответствующего модуля (драйвера).

Компании, которые выпускают дистрибутивы Linux (такие как RedHat, Caldera, Debian и т. д.) вынуждены встраивать в ядро поддержку как можно более широкого спектра устройств, потому что они не могут заранее знать, какие устройства (модели устройств) будут установлены на компьютере конкретного пользователя.

Поддержка в ядре широкого спектра устройств облегчает установку и поддержку системы для покупателей, избавляя их от ненужных сложностей.

Как результат, ядро, поставляемое в составе дистрибутива, скорее всего содержит код для поддержки устройств, которых никогда не будет на Вашем конкретном компьютере. С другой стороны, если у вас есть устройство, которое не поддерживается стандартным ядром, у вас может появиться обоснованное желание встроить поддержку этого устройства в ядро. Оптимизация ядра под конкретный набор аппаратных устройств ускоряет загрузку системы и экономит память.

Пользователи Linux имеют возможность или скомпилировать ядро с поддержкой всех устройств, имеющихся на конкретном компьютере, или скомпилировать ядро, поддерживающее минимальный набор оборудования, и загружать модули для поддержки остальных устройств. Если поддержка всего оборудования осуществляется в ядре, то такое ядро называется "монолитным". Ядро, скомпилированное таким образом, что поддержка части оборудования осуществляется с использованием модулей (драйверов), называется "модульным".

Какой тип ядра вам выбрать при компиляции? Однозначного ответа на это вопрос дать нельзя. Если вы не имеете привычки менять аппаратную конфигурацию компьютера, тогда вам лучше встроить поддержку всех



имеющихся компонентов в ядро. Необходимо только иметь в виду, что чем больше устройств поддерживаются непосредственно ядром, тем больше его объем. А поскольку ядро полностью загружается в оперативную память, повышаются требования к объему памяти. На медленных компьютерах из-за большого размера ядра может снизиться общая производительность. Если же вы часто меняете конфигурацию компьютера (например, у вас имеются съемные жесткие диски или другие временно подключаемые устройства), то, вероятно, имеет смысл использовать для управления ими подключаемые модули, которые загружаются в память только при необходимости (экономя тем самым системные ресурсы). Таким образом, в самом общем случае поддержка некоторой части устройств должна быть встроена в ядро, а остальные устройства должны поддерживаться за счет использования загружаемых модулей.

Кроме желания иметь ядро, оптимизированное для вашей системы, необходимость перекомпилировать ядро может быть вызвана обнаружением каких-то ошибок в старой версии ядра, в частности таких, которые представляют угрозы с точки зрения безопасности (когда еще появится грт-пакет с исправленной версией ядра?).

Я был вынужден заниматься установкой ядра из исходных кодов потому, что система виртуальных машин VMware отказалась работать с установленным у меня ядром 2.2.16, сообщив, что эта версия ядра не поддерживает работу с CDROM из VMware, и предложив мне либо установить более позднюю версию ядра, либо вернуться к версии 2.2.15. Попытки установить новую версию ядра из грт-пакетов тоже не решили проблему, потому что конфигурационный скрипт VMware сообщал, что ему не хватает header-файлов. Установка пакетов kernel-headers (полностью соответствующих ядру) тоже не привела к успеху, вот и пришлось сделать попытку установить ядро из исходных текстов.

Надо сказать, что к тому времени мой опыт установки программного обеспечения для Linux из исходников был очень ограничен. Поэтому приступал я к этой процедуре только под давлением обстоятельств (очень хотелось запускать MS Office под Linux, не прибегая к перезагрузке компьютера). Приводимый ниже текст является как раз описанием того, что я тогда делал. Поскольку мой эксперимент оказался удачным, я могу со спокойной совестью утверждать: ничего такого, что оказалось бы не под силу начинающему пользователю, в компиляции ядра из исходных кодов нет.

## Перед тем как начать...

Пожалуй, самое первое, к чему нужно быть готовым, приступая к компиляции ядра, — это то, что процедура эта длительная. Так что не рассчитывайте скомпилировать ядро "между делом", в свободную минутку. Заранее планируйте, что потратите на это несколько часов, иначе вы будете вынуждены прервать процедуру посередине.

Во-вторых, до начала компиляции ядра вам необходимо хотя бы в общих чертах представлять, какую именно конфигурацию аппаратного обеспечения будет обслуживать новое ядро, и решить, какие устройства будут обслуживаться самим ядром, а какие — модулями.

В-третьих, ядро версии 2.6.26.5 даже в архивированном (программой bzip2) виде занимает около 32 Мбайт, а после разархивации объем исходных текстов превышает 180 Мбайт. Еще столько же может потребоваться для промежуточного tar-архива, так что надо иметь не менее 400 Мбайт свободного места. Поэтому прежде чем приступить к компиляции, позаботьтесь о том, чтобы места на диске было достаточно.

В-четвертых, приготовьтесь отвечать на те вопросы, которые программа задаст вам в процессе конфигурирования ядра (а их около 500). Еще, после разворачивания исходников ядра вам необходимо найти файл Configure.help в подкаталоге linux/Documentation. Имеет смысл просмотреть его (а лучше — распечатать) до начала конфигурирования, даже если вы не очень хорошо владеете английским.

На этом предварительные предупреждения сделаны, и я перехожу к изложению пошаговых инструкций по компиляции и установке нового ядра. В дальнейшем предполагается, что все описываемые действия выполняются с правами суперпользователя root, и что домашним каталогом является каталог /root.

## Семь шагов к новому ядру

### Получение и разархивация ядра

Исходные тексты новой версии ядра можно скачать с сайта [ftp.kernel.org](http://ftp.kernel.org). Как уже было сказано, bzip2-архив исходных кодов ядра версии 2.6.X имеет объем не менее 30 Мбайт, так что скачать его — тоже еще проблема. Но скачивать исходные тексты необходимо только в том случае, если вы желаете установить новую версию ядра. Если же вы просто хотите перекомпилировать существующее ядро (скажем, из-за того, что необходимо обеспечить поддержку какого-то протокола или нового оборудования), то можно обойтись и без

перекачки пакета из Интернета, поскольку с дистрибутивами обычно поставляются и исходные коды. Кроме того, даже если вы захотели скомпилировать ядро новой версии, существует возможность сократить объем информации, которую необходимо скачивать из Интернета. Что бы после выхода каждой новой версии ядра не скачивать полностью исходные коды новой версии ядра можно воспользоваться заплатками (patches) выпущенными для новой версии ядра. Текущей стабильной версией ядра является 2.6.26.1. Для того, что бы рассмотреть как применяются заплатки, мы скачем ядро версии 2.6.26 и применим к нему заплатку.

Итак, давайте выполним необходимые действия с исходными кодами ядра.

Поместите архив в каталог /usr/src

Распакуйте архив:

```
root@c1:~# cd /usr/src/
root@c1:/usr/src# ls
linux@          linux-2.6.26.tar.bz2*  rpm/
linux-2.6.24.5/ patch-2.6.26.1.bz2*  screen/
root@c1:/usr/src# tar -xjf linux-2.6.26.tar.bz2
root@c1:/usr/src# ls
linux@          linux-2.6.26/          patch-2.6.26.1.bz2*  screen/
linux-2.6.24.5/ linux-2.6.26.tar.bz2* rpm/
root@c1:/usr/src#
```

Переименуем директорию linux-2.6.26 с учетом будущей версии:

```
root@c1:/usr/src# mv linux-2.6.26 linux-2.6.26.1
root@c1:/usr/src# ls
linux@          linux-2.6.26.1/          patch-2.6.26.1.bz2*  screen/
linux-2.6.24.5/ linux-2.6.26.tar.bz2* rpm/
root@c1:/usr/src#
```

Исходные коды ядра должны располагаться в директории /usr/src/linux. Давайте посмотрим как обстоят дела сейчас:

```
root@c1:/usr/src# ls -l
итого 48312
lrwxrwxrwx  1 root root          23 2008-09-20 10:48 linux -> /usr/src/linux-
2.6.24.5/
drwxr-xr-x 20 root root      4096 2008-04-30 23:13 linux-2.6.24.5/
drwxr-xr-x 21 root root      4096 2008-07-14 02:51 linux-2.6.26.1/
-rwxr-xr-x  1 root root 49441874 2008-09-20 10:26 linux-2.6.26.tar.bz2*
-rwxr-xr-x  1 root root   20090 2008-09-20 10:47 patch-2.6.26.1.bz2*
drwxr-xr-x  7 root root        67 2003-10-29 10:08 rpm/
drwxr-xr-x  3 root root        93 2008-09-18 15:30 screen/
root@c1:/usr/src#
```

Как мы видим /usr/src/linux является символьной ссылкой. Это решение дает возможность манипулировать с исходными кодами нескольких ядер не теряя понимания над каким проектом вы сейчас работаете. Мы поступим точно также:

```
root@c1:/usr/src# rm linux
root@c1:/usr/src# ln -s /usr/src/linux-2.6.26.1 /usr/src/linux
root@c1:/usr/src# ls -l linux
lrwxrwxrwx 1 root root 23 2008-09-20 10:53 linux -> /usr/src/linux-2.6.26.1/
root@c1:/usr/src#
```

Все дальнейшие действия по отношению к ядру мы будем выполнять, находясь в директории с исходными текстами ядра:

```
root@c1:/usr/src# cd linux
root@c1:/usr/src/linux#
```

## Применение заплаток к исходникам ядра

Для того, что бы применить заплатку к исходникам ядра мы воспользуемся программами декомпрессии и применения патчей. Файл патча должен находиться на один уровень выше директории с исходниками.

```

root@cl:/usr/src/linux# bzip2 -dc ../patch-2.6.26.1.bz2 | patch -p1
patching file Documentation/networking/udplite.txt
patching file Documentation/video4linux/cx18.txt
patching file Makefile
patching file arch/ia64/kvm/kvm-ia64.c
.....
.....
patching file sound/pci/trident/trident_main.c
patching file virt/kvm/kvm_main.c
root@cl:/usr/src/linux#

```

Мы применили первую заплатку. Никаких сообщений об ошибках не появилось. Значит все прошло успешно.

## Конфигурирование будущего ядра

Следующий этап заключается в конфигурировании будущего ядра.

Собственно конфигурирование выполняется командой

```
make menuconfig
```

Конфигурация в этом случае будет производиться в режиме меню.

В более ранних версиях использовалась команда `make config`. Конфигурирование происходило в текстовом режиме. Однако она сильно устарела.

Еще один вариант конфигурирования ядра состоит в использовании графических программ, например `xconfig` или `gconfig`. Но мы не будем рассматривать эти программы так как на экзаменах LPI требуется знание варианта с использованием программы `make`.

Процедура конфигурации будет заключаться в том, что вы должны будете последовательно ответить на серию вопросов о том, какое значение присвоить определенному параметру. На каждый вопрос предлагается обычно несколько вариантов ответа. Допустимые варианты ответа предлагаются в виде символов, заключенных в квадратные скобки. Я думаю, что смысл символов "y" и "n" пояснять не требуется. Пояснения требуют символы "?" и "m".

Символ "?" присутствует среди возможных вариантов ответа на любой вопрос и позволяет получить подсказку (конечно, по-английски). Эти подсказки и составляют содержание файла `Configure.help`, который упоминался выше.

Если вы выбираете вариант "m", это означает, что драйвер соответствующего устройства будет сконфигурирован (и впоследствии скомпилирован) в виде отдельного подключаемого модуля.

Один из вариантов ответа на каждый вопрос представлен большой буквой, что означает, что он выбирается по умолчанию (когда вы не задаете явно свой вариант выбора, а просто нажимаете клавишу <Enter>).

В процессе конфигурации ядра следует иметь в виду следующее:

включение в ядро драйвера любого устройства делает его больше и, более того, добавление ненужных драйверов в ядро может привести к проблемам, когда ядро будет пытаться обратиться к несуществующему устройству;

если задать тип процессора ("Processor type") выше, чем 386, то ядро не будет работать на 386-х процессорах. Ядро обнаружит это при загрузке и откажется работать;

ядро, скомпилированное с опцией эмуляции математического сопроцессора, все равно будет использовать сопроцессор, если он имеется. Функция эмуляции никогда не будет использоваться в этом случае. Правда, ядро в этом случае будет чуть больше, но зато будет работать на любых компьютерах, независимо от того, имеется ли на них сопроцессор;

некоторые вопросы помечены указаниями "development", "experimental" или "debugging", которые указывают на то, что соответствующая функция или драйвер включены в ядро в виде эксперимента. Включение этих функций в ядро может сделать ядро не только больше, но и нанести ущерб стабильности его работы. Так что на такие вопросы лучше ответить "n", если, конечно, вы не ставите своей целью провести тестирование этих новых доработок в ядре;

если вы компилируете ядро для использования на своем персональном компьютере, то поддержка мультипроцессорной обработки (Symmetric multi-processing support) вам, скорее всего, не нужна. Исключая случаи использования многоядерных процессоров или процессоров поддерживающих функцию HyperThreading

Впрочем, я начал уже давать пояснения по тому, как задавать значения отдельных параметров. Но их, во-первых, очень много, так что эта глава грозит распухнуть до неприличия, а, во-вторых, самое приемлемое решение определяется конкретной конфигурацией вашего компьютера и вашими личными потребностями и пожеланиями, так что обратитесь лучше к упоминавшемуся выше файлу `Configure.help`.

Так же необходимо каким-то образом обозначить суб-версию ядра вашей сборки. Для этого давайте посмотрим первые строки файла `Makefile`:

```
VERSION = 2
PATCHLEVEL = 6
SUBLEVEL = 26
EXTRAVERSION = .1
```

Замените четвертую строку на что-то вроде этого:

```
EXTRAVERSION = .1-my
```

Это приведет к тому, что сделанное нами ядро будет называться `linux-2.6.26.1-my`. Если нам нужно будет несколько вариантов ядра 2.6.26.1, то можно снова изменить строчку `EXTRAVERSION=.1-my2`, скомпилировать ядро еще раз и получить ядро `linux-2.6.26.1-my2`, что позволит при загрузке выбирать вариант ядра.

```
root@cl:/usr/src/linux# make menuconfig
HOSTCC  scripts/basic/fixdep
HOSTCC  scripts/basic/docproc
.....
```

Как конфигурировать новое ядро, какие параметры нужно использовать, а что исключить? В большинстве случаев только Вы можете ответить на эти вопросы. Перед началом конфигурирования ядра Вам нужно точно знать из каких комплектующих собран Ваш компьютер, их производитель и модель. Далее, если Вы не понимаете о чем идет речь в данном пункте меню — посмотрите помощь (кнопка «HELP»). Помощь есть по каждому пункту меню. Однако, если прочитав справку Вы все равно не уверены что выбрать, почти везде есть предложение выбора, например, `If unsure say Y`.

Рекомендации по конфигурированию ядра Вы можете найти в сети Интернет. Перевод конфигурационного меню можете посмотреть здесь: [http://wiki.kryukov.biz/wiki/Параметры\\_ядра\\_Linux](http://wiki.kryukov.biz/wiki/Параметры_ядра_Linux).

Итак, настройка завершена. В завершении конфигурирования просто выходите из программы настройки. При этом ваши настройки автоматически сохранятся. Все настройки сохраняются в файле `.config` (скрытый).

```
#
# configuration written to .config
#

*** End of Linux kernel configuration.
*** Execute 'make' to build the kernel or try 'make help'.
```

```
root@cl:/usr/src/linux#
```

Еще одним обязательным действием является выполнение программы `make clean`, которая очищает все временные папки и файлы.

```
root@cl:/usr/src/linux# make clean
root@cl:/usr/src/linux#
```

Далее, если ваш вариант конфигурирования окажется неудачным, вы, выполнив команду `make menuconfig`, увидите свой вариант конфигурации. Исправьте его и продолжите.

Я думаю, что сказанного достаточно для того, чтобы вы смогли сконфигурировать ядро, поэтому переходим к дальнейшим шагам.

## Компиляция ядра

Вот мы и добрались до основного этапа — собственно компиляции ядра. На этом этапе Ваше участие сводится только к запуску команды `make bzImage`, которая служит для создания сжатого образа ядра.

```
root@cl:/usr/src/linux# make bzImage
```

Снова по экрану бегут непонятные сообщения (которые просто не успеваешь воспринять). О том, что процесс компиляции завершился без ошибок, вы можете судить по появлению сообщений примерно такого вида (естественно, я привожу те сообщения, которые были выданы в процессе компиляции ядра на моем компьютере):

```
root@c1:/usr/src/linux# make bzImage
.....
.....
LD      arch/x86/boot/setup.elf
OBJCOPY arch/x86/boot/setup.bin
OBJCOPY arch/x86/boot/vmlinux.bin
HOSTCC  arch/x86/boot/tools/build
BUILD   arch/x86/boot/bzImage
Root device is (3, 5)
Setup is 10764 bytes (padded to 11264 bytes).
System is 2608 kB
CRC 68014c0a
Kernel: arch/x86/boot/bzImage is ready  (#1)
root@c1:/usr/src/linux#
```

И в текущем каталоге должны появиться файлы System.map и vmlinux. Кроме того, в подкаталогах каталога linux тоже появляется масса новых файлов (в том числе и .o-файлы, которые упоминались выше, как мешающие повторному проведению компиляции ядра в том же каталоге).

На этом собственно компиляция ядра и заканчивается.

## Компиляция модулей

Если вы сконфигурировали какие-то драйверы как отдельные модули (выбирали при конфигурации вариант "m" при ответе на некоторые вопросы), то вы теперь должны еще выполнить команду `make modules`, а затем еще команду `make modules_install`. В файле Documentation/modules.txt можно найти дополнительную информацию по этому поводу, а также объяснение того, как использовать модули.

```
root@c1:/usr/src/linux# make modules
CHK      include/linux/version.h
CHK      include/linux/utsrelease.h
CALL     scripts/checksyscalls.sh
CC [M]   drivers/input/mouse/sermouse.o
CC [M]   drivers/scsi/scsi_wait_scan.o
Building modules, stage 2.
MODPOST 2 modules
CC       drivers/input/mouse/sermouse.mod.o
LD [M]   drivers/input/mouse/sermouse.ko
CC       drivers/scsi/scsi_wait_scan.mod.o
LD [M]   drivers/scsi/scsi_wait_scan.ko
root@c1:/usr/src/linux# make modules_install
INSTALL drivers/input/mouse/sermouse.ko
INSTALL drivers/scsi/scsi_wait_scan.ko
DEPMOD  2.6.26.1-my
root@c1:/usr/src/linux#
```

## Установка ядра

После этого остается сделать последний шаг — установить ядро настроить загрузчик и перезагрузиться. Для установки ядра вы должны иметь права суперпользователя. (Хотя в начале главы и было сказано, что для компиляции ядра надо иметь права суперпользователя, однако все предыдущие шаги можно было выполнить и от имени обычного пользователя. Не стоит работать от имени суперпользователя без необходимости!)

Разработчики рекомендуют вначале сохранить где-нибудь копию старого ядра на случай, если что-то пойдет не так, как задумано. Эта рекомендация особенно актуальна для случая, если вы ставите ядро из нестабильной ветки, поскольку такие версии могут содержать неотлаженный код. Кроме самого ядра надо сделать backup-копии модулей, соответствующих ядру. Если вы устанавливаете новое ядро с тем же самым номером версии, что и у работающего в Вашей системе ядра, сделайте резервную копию всего каталога с модулями перед выполнением команды `make modules_install`. Путь, по которому вы можете найти установленные модули ядра

## /lib/modules

Для того, чтобы иметь возможность загружать новое ядро, копию образа ядра (которая после компиляции создана в виде файла `.../linux/arch/i386/boot/bzImage`) необходимо поместить туда, где у вас расположены загружаемые ядра (обычно это каталог `/boot`).

```
root@c1:/usr/src/linux# cp arch/i386/boot/bzImage /boot/vmlinuz-my
root@c1:/usr/src/linux#
```

Я при копировании добавил к его имени суффикс `-my`, превратив его в `vmlinuz-my`, чтобы не путать с теми ядрами, которые уже были в системе ранее. Переименовывать `bzImage` в `vmlinuz` в принципе не обязательно, потому что образ ядра может иметь как то, так и другое имя, и обычно располагается либо в корневом каталоге (`/`), либо в каталоге `/boot`.

## Настройка загрузчика ядра

Далее нам осталось только обеспечить загрузку нового ядра с помощью `lilo`. `Lilo` — `Linux LOader` — простой и достаточно функциональный загрузчик ядра Linux. Программа предназначена для загрузки Linux, а также других операционных систем.

Загрузчик `LILO` состоит из двух частей. Задача первой части — найти вторую часть загрузчика и передать ей управление. Первую часть загрузчика можно размещать в `MBR` или начале раздела.

Конфигурационный файл загрузчика: `/etc/lilo.conf`

При желании вы можете использовать иные загрузчики, например `GRUB`.

Итак, надо вначале подкорректировать файл `/etc/lilo.conf`. Ваш файл `/etc/lilo.conf` может иметь примерно такой вид:

```
# cat /etc/lilo.conf
# LILO configuration file
# generated by 'liloconfig'
#
# Start LILO global section
# Append any additional kernel parameters:
append=" vt.default_utf8=0"
boot = /dev/hda

# Boot BMP Image.
# Bitmap in BMP format: 640x480x8
# bitmap = /boot/slack.bmp
# Menu colors (foreground, background, shadow, highlighted
# foreground, highlighted background, highlighted shadow):
# bmp-colors = 255,0,255,0,255,0
# Location of the option table: location x, location y, number of
# columns, lines per column (max 15), "spill" (this is how many
# entries must be in the first column before the next begins to
# be used. We don't specify it here, as there's just one column.
# bmp-table = 60,6,1,16
# Timer location x, timer location y, foreground color,
# background color, shadow color.
# bmp-timer = 65,27,0,255

# Standard menu.
# Or, you can comment out the bitmap menu above and
# use a boot message with the standard menu:
#message = /boot/boot_message.txt

# Wait until the timeout to boot (if commented out, boot the
# first entry immediately):
prompt
# Timeout before the first entry boots.
# This is given in tenths of a second, so 600 for every minute:
timeout = 1200
# Override dangerous defaults that rewrite the partition table:
```

```

change-rules
  reset
# VESA framebuffer console @ 1024x768x256
vga = 773
# Normal VGA console
# vga = normal
# VESA framebuffer console @ 1024x768x64k
# vga=791
# VESA framebuffer console @ 1024x768x32k
# vga=790
# VESA framebuffer console @ 1024x768x256
# vga=773
# VESA framebuffer console @ 800x600x64k
# vga=788
# VESA framebuffer console @ 800x600x32k
# vga=787
# VESA framebuffer console @ 800x600x256
# vga=771
# VESA framebuffer console @ 640x480x64k
# vga=785
# VESA framebuffer console @ 640x480x32k
# vga=784
# VESA framebuffer console @ 640x480x256
# vga=769
# End LILO global section
# Windows bootable partition config begins
other = /dev/hda1
  label = Windows
  table = /dev/hda
# Windows bootable partition config ends
# Linux bootable partition config begins
image = /boot/vmlinuz
  root = /dev/hda5
  label = Linux
  read-only
# Linux bootable partition config ends
root@cl:/usr/src/linux#

```

Параметры конфигурационного файла /etc/lilo.conf:

- boot — определяет, куда будет помещена первая часть загрузчика.
- prompt — заставляет загрузчик выводить список загружаемых ядер Linux и других операционных систем.
- timeout — определяет количество десятых долей секунды, задержка выбора. Если выбор не будет сделан, загрузится операционная система по умолчанию.
- lba32 — параметр, который рекомендуется использовать со всеми винчестерами, выпускающимися после 1998 года.
- restricted — при вводе дополнительных параметров ядра в командной строке будет спрашиваться пароль.
- password — определяет пароль загрузчика. Пароль указывается в открытом виде.
- vga — определяет режим консоли. Если ядро было собрано без поддержки framebuffer, следует устанавливать значение normal.
- image — определяет месторасположение файла ядра.
- root — определяет месторасположение корневой файловой системы.
- label — определяет название, которое будет показано в окне загрузчика.
- read-only — заставляет ядро монтировать корневую файловую систему в режиме ro.
- append — определяет дополнительные параметры, которые передаются ядру при его загрузке.

- `other` — определяет другую операционную систему.
- `table` — определяет устройство, таблица разделов которого передается другой ОС.
- `default` — определяет операционную систему, загружаемую по умолчанию.

Начиная со строки `image`, идут секции конфигурационного файла, соответствующие разным операционным системам, которые должны загружаться по выбору пользователя. В каждой такой секции имеется строка `label`. В этой строке записывается имя, которое вводится в ответ на приглашение LILO и служит для выбора пользователем загружаемой ОС.

Скопируйте секцию `image` и замените в новой секции название образа ядра и метку. Файл `/etc/lilo.conf` примет следующий вид:

```
other = /dev/hda1
    label = Windows
    table = /dev/hda
# Windows bootable partition config ends
# Linux bootable partition config begins
image = /boot/vmlinuz
    root = /dev/hda5
    label = Linux
    read-only
image = /boot/vmlinuz-my
    root = /dev/hda5
    label = Linux-my
    read-only
```

После того, как вы откорректировали файл `/etc/lilo.conf`, необходимо выполнить команду `/sbin/lilo` или просто `lilo`, чтобы изменения вступили в силу. Эта команда (которая в руководстве называется `map-installer`) обновляет карту загрузки системы. Прежде, чем запускать `/sbin/lilo` для модификации загрузочных процедур, выполните эту команду

```
# chmod 600 /etc/lilo.conf
```

Эта команда имеет особенный смысл, если вы решили поставить пароль на доступ к опциям ядра во время загрузки.

Не стоит полностью убирать возможность загрузки предыдущей, работающей версии ядра, тем более, что это никак не мешает загрузке нового ядра, ведь LILO обеспечивает многовариантную загрузку.

```
# lilo
Added Windows *
Added Linux
Added Linux-my
#
```

Все! Вы можете перезагрузить компьютер и выбрать при загрузке новое ядро.

В завершение этапа компиляции, в случае успешной загрузки нового ядра стоит выполнить еще команду `make clean`, чтобы удалить образующиеся в процессе компиляции промежуточные объектные файлы (файлы с расширением `.o`).

```
root@c1:/usr/src/linux# make clean
root@c1:/usr/src/linux#
```



## Глава 21. Система инициализации Linux

Система инициализации Linux — это набор скриптов, выполняющихся при старте системы. Скрипты написаны на языке shell script классического Bourne Shell ("sh").

Исторически сложилось так, что существует две системы инициализации:

- System V
- BSD

Они отличаются друг от друга организацией стартовых скриптов: как они называются, в каких директориях располагаются, последовательность вызова и т.д.

В Linux наибольшее распространение получила система инициализации System V. Ее используют такие дистрибутивы как:

- RedHat Linux
- SuSE Linux
- Mandriva
- и многие другие.

Система инициализации BSD используется в дистрибутиве Slackware Linux и его производных.

Если говорить о SystemV — это очень строгая система инициализации, где шаг вправо или влево — расстрел на месте. Правда, некоторые дистрибутивы (не буду тыкать в них пальцем, хотя это был Red Hat) даже из такой строгой системы могут сделать черте что и сбоку бантик.

В системе инициализации BSD не наблюдается строгих правил, но соблюдаются определенные принципы ее построения. Поэтому если рассматривать дистрибутивы и операционные системы, использующие ее, можно сказать, что это организованный бардак. То есть, нет двух похожих систем в которых совпадало бы именование файлов и порядок их вызова. Но все прекрасно понимают как это работает.

При запуске PC совместимого компьютера происходит следующая последовательность действий:

1. выполняется BIOS компьютера;
2. запускается загрузчик операционной системы — LILO, grub или любой другой;
3. загружается ядро Linux.

Ядро стартует и кроме основных действий подключает корневую файловую систему в режиме только для чтения, а так же оно запускает самую первую программу в системе — `init`.

То есть, в результате мы имеем:

- подключенную в режиме только для чтения корневую файловую систему
- работающую программу `init`

Дальнейшие действия, которые будут выполняться при старте системы, во многом зависят от конфигурации программы `init`.

Так же хочется обратить ваше внимание на то, что до этого момента еще не важно какая система инициализации используется. Не зависимо от нее во всех Linux сначала запускается `init`. А вот какая система инициализации будет использоваться в дальнейшем, зависит от того, как сконфигурирован `init`.

### Программа `init`

Как уже говорилось, самым первым процессом является программа `init`. И от того, каким образом она сконфигурирована, зависит дальнейшая загрузка системы.

Ядро Linux при старте обязательно монтирует корневую файловую систему (обычно в режиме только для чтения). Поэтому `init` может быть запущен (для этого необходимо получить доступ к файлу программы) и может прочитать свой конфигурационный файл — `/etc/inittab`.

Процесс запуска `init` каждый знакомый с языком программирования C может посмотреть в исходных кодах ядра, в файле `init/main.c`, для остальных скажу, что:

- когда наступает время породить первый процесс, то первым делом ядро пытается запустить программу,

указанную опцией `rdinit=`;

- если таковой опции ядру передано не было, то будет запущен `/init`;
- если ничего не вышло с `/init`, то ядро пытается запустить программу, указанную опцией `init=`;
- если это не получилось, то ядро пытается запустить `/sbin/init`;
- если возникли проблемы с `/sbin/init`, то ядро пытается запустить `/etc/init`;
- если не удалось запустить `/etc/init`, то ядро пытается запустить `/bin/init`;
- если невозможно запустить `/bin/init`, то ядро пытается запустить `/bin/sh`;
- ну а если даже и `/bin/sh` подвел, то ядро паникует с сообщением

***No init found. Try passing init= option to kernel.***

**Внимание!** Никогда не выносите в отдельный раздел директории `/bin`, `/sbin`, `/etc`, `/dev`. Файлы, находящиеся в них могут потребоваться при старте системы. Ярким примером являются программа `init` и ее конфигурационный файл `/etc/inittab`.

В Linux существует такое понятие как уровень выполнения (*run level*). Уровень выполнения обозначается числами от 0 до 6.

Система в определенный момент времени находится на соответствующем уровне выполнения. Вы, как администратор системы, можете переводить ее с одного уровня выполнения на другой. Это делается при помощи программы `init` (или `telinit`). Для этого программе в качестве аргумента передается число, соответствующее уровню выполнения. Например, что бы перевести систему на 3-й уровень выполнения, необходимо запустить `init` следующим образом:

```
# init 3
```

В различных дистрибутивах Linux уровни выполнения используются для различных целей.

Современная версия программы `init` может использовать десять уровней выполнения, но обычно используются только семь:

- 0 — выполняются действия по выключению системы.
- 1 — однопользовательский режим (*single user mode*). Предназначен для различных административных действий по восстановлению системы. По своему смыслу аналогичен *Safe Mode Windows*, но полностью его не повторяет. На этом уровне выполнения система полностью сконфигурирована, но не запущен ни один сервис, а из пользователей может работать только один `root`.
- 2 — не используется, но сконфигурирован как уровень выполнения 3. В RedHat и SuSE Linux, сконфигурирован, как уровень выполнения 3, но без поддержки сетевых файловых систем. В Debian используется как многопользовательский режим.
- 3 — многопользовательский режим (*multiuser mode*). Нормальный режим работы сервера.
- 4 — В Slackware Linux используется для графического входа в систему. В RedHat и SuSE Linux не сконфигурирован.
- 5 — В RedHat и SuSE Linux используется для графического входа в систему. В Slackware Linux не сконфигурирован.
- 6 — выполняются действия по перезагрузке системы.

Суперпользователь может остановить систему, переведя ее на нулевой уровень:

```
# init 0
```

Или перезагрузить систему:

```
# init 6
```

## Формат файла `/etc/inittab`

Конфигурационный файл программы `init` — это текстовый файл. Символ комментария — `#`. Файл состоит из строк следующего формата:

`id:run_level:action:process`

- **id** — идентификатор, один или два символа. Поле должно быть уникальным. Никак не влияет на работу

современной версии программы `init`.

- **run\_level** — список уровней выполнения, на которых будет выполняться программа. Поле может быть пустым.
- **action** — определяет особенности выполнения программы. В этом поле можно писать только заранее определенные слова.
- **process** — программа, которая будет выполняться.

`Init` рассматривает строки в том порядке, в котором они написаны в файле. Предположим, что мы хотим перевести систему на третий уровень выполнения, и запускаем `init` следующим образом:

```
# init 3
```

Программа начнет выполнять только те строки, в поле `run_level` которых будет присутствовать цифра 3. То есть, будут запускаться программы, описанные в поле `process`, но с некоторыми оговорками, определяемые полем `action`.

Поскольку программу писали люди, родным языком для которых является английский, они внесли некоторые особенности английской грамматики в программу. В английском языке есть правила, но так же существует большое количество исключений из этих правил. То, что я написал в предыдущем абзаце — это правило, но у него есть несколько исключений.

Давайте рассмотрим конфигурационный файл `inittab` на примере дистрибутива `Slackware Linux`. Ниже показано содержимое этого файла, за исключением комментариев и пустых строк:

```
id:3:initdefault:
si:S:sysinit:/etc/rc.d/rc.S
su:1S:wait:/etc/rc.d/rc.K
rc:2345:wait:/etc/rc.d/rc.M
ca::ctrlaltdel:/sbin/shutdown -t5 -r now
10:0:wait:/etc/rc.d/rc.0
16:6:wait:/etc/rc.d/rc.6
pf::powerfail:/sbin/genpowerfail start
pg::powerokwait:/sbin/genpowerfail stop
c1:1235:respawn:/sbin/agetty 38400 tty1 linux
c2:1235:respawn:/sbin/agetty 38400 tty2 linux
c3:1235:respawn:/sbin/agetty 38400 tty3 linux
c4:1235:respawn:/sbin/agetty 38400 tty4 linux
c5:1235:respawn:/sbin/agetty 38400 tty5 linux
c6:12345:respawn:/sbin/agetty 38400 tty6 linux
x1:4:respawn:/etc/rc.d/rc.4
```

При старте системы ядро запускает программу `init` без указания уровня выполнения. То есть, просто

```
init
```

Программе, каким-то образом, необходимо узнать, какие строки в этом случае выполнять. Какой уровень выполнения использовать? Для указания уровня выполнения используется строка

```
id:3:initdefault:
```

Это первое исключение из общего правила. Как видите поле `process` пустое. В поле `action` написано ключевое слово `initdefault`, которое означает, что в поле `run_level` написан уровень выполнения, используемый по умолчанию.

Дальше мы будем рассматривать файл с учетом того, что система стартует на третьем уровне выполнения. В следующей строке нас ждет сразу два исключения из общего правила!

```
si:S:sysinit:/etc/rc.d/rc.S
```

Во-первых, в поле `run_level` стоит буква `S`, да и `action` `sysinit`, не просто так тут написана.

Начнем с буквы `S`. Когда ядру при загрузке передается параметр `single`, ядро запускает программу `init` таким образом, что она перестает реагировать на все строки, кроме тех, в поле `run_level` которых стоит буква `S`. Обычно дистрибутивы сконфигурированы таким образом, что бы в этом случае система запускалась на уровне выполнения 1 (`single user mode`).

Действие `sysinit` означает, что программа, описанная в поле `process`, будет выполняться только при старте системы, и `init` будет ожидать ее завершения, прежде чем начнет выполнять следующие строки. Таким образом, программа `/etc/rc.d/rc.S` будет выполняться только при старте системы. При переходе с одного уровня

выполнения на другой она запускаться не будет.

```
su:1S:wait:/etc/rc.d/rc.K
```

Строка на третьем уровне выполнения работать не будет, мы ее пропустим.

```
rc:2345:wait:/etc/rc.d/rc.M
```

В поле `run_level` присутствует цифра три, поэтому программа `/etc/rc.d/rc.M` будет запущена. Действие `wait` означает, что `init` будет ожидать завершения выполнения программы, прежде, чем начнет выполнять следующие строки из файла `inittab`. Таким образом, второй при старте системы будет запущена программа `rc.M`.

```
ca::ctrlaltdel:/sbin/shutdown -t5 -r now
```

В поле `run_level` этой строки нет ни одной цифры, значит она не будет выполняться при старте системы. Как-то странно при старте сразу запускать программу `shutdown`. Но в поле действия присутствует интересный параметр: `ctrlaltdel`. Он заставляет `init` следить за нажатием комбинации клавиш `Ctrl(левый)+Alt(левый)+Del`. И если кто-то их нажмет, выполнит программу, указанную в поле `process`.

**Внимание!** Обратите внимание на то, что не имеет значения, залогинился пользователь в систему или нет. Любой проходящий мимо человек может нажать эту комбинацию клавиш, и система будет перезагружена.

```
10:0:wait:/etc/rc.d/rc.0
```

```
16:6:wait:/etc/rc.d/rc.6
```

Эти две строки никакого отношения к третьему уровню выполнения не имеют, поэтому мы их пропускаем.

```
pf::powerfail:/sbin/genpowerfail start
```

```
pg::powerokwait:/sbin/genpowerfail stop
```

Поле `run_level` в этих строках пустое, поэтому при старте системы они работать не будут. Но, `init` их запомнит. Для того, что бы эти строки заработали, необходимо, что бы у вас был интеллектуальный UPS, подключенный к вашей машине. А так же запущена программа, которая понимает команды UPS (обычно используют программу `genpowerd`).

Если питание пропадет, то UPS сообщит об этом программе, а та в свою очередь передаст информацию `init`, который обработает действие `powerfail`. Если питание восстановится, будет обработано действие `powerokwait`.

Программа `genpowerfail` — это обыкновенный скрипт. Если его запустить с параметром `start`, то будет запущена программа `shutdown`. В зависимости от состояния UPS, процедура остановки системы будет произведена сразу или с задержкой на одну или две минуты. Если скрипт запустить с параметром `stop`, то процедура остановки системы будет отменена (`shutdown -c`).

В таблице приведены некоторые ключевые слова, которые можно использовать в поле `action`.

Поле	Описание
<code>initdefault</code>	Определяет уровень выполнения по умолчанию во время загрузки системы.
<code>sysinit</code>	Программа будет выполняться при запуске системы самой первой. <code>init</code> будет ожидать ее завершения, прежде чем начнет выполнение следующих в списке программ.
<code>wait</code>	Программа запускается один раз. <code>init</code> будет ожидать ее завершения, прежде чем начнет выполнение следующих в списке программ.
<code>once</code>	Программа запускается один раз. <code>init</code> не ждет ее завершения.
<code>ctrlaltdel</code>	Определяет программу, которая будет запущена при нажатии на клавиши “ <code>Ctrl+Alt+Del</code> ”.
<code>powerfail</code>	Определяет программу, которая будет запущена при

	получении процессом <code>init</code> сигнала сбоя питания.
<code>powerokwait</code>	Определяет программу, которая будет запущена при получении процессом <code>init</code> сигнала восстановления питания.
<code>respawn</code>	Процесс будет запущен. <code>init</code> не будет ожидать окончания процесса, и начнет обрабатывать следующие в списке строки. Если процесс завершит свою работу – <code>init</code> запустит его снова.

Из конфигурационного файла можно сделать вывод, что при уровне выполнения 3, `init` запускает следующие программы:

- `/etc/rc.d/rc.S`
- `/etc/rc.d/rc.M`
- `/sbin/agetty` для каждой виртуальной консоли.

## Система инициализации BSD

В Slackware Linux используется система инициализации, очень похожая на систему инициализации BSD, которая отличается простотой стартовых скриптов. И эту простоту можно проследить на примере стартовых скриптов Slackware Linux.

Все скрипты располагаются в директории `/etc/rc.d`.

## Файл `/etc/rc.d/rc.S`

Согласно файла `/etc/inittab`, самой первой, независимо от уровня выполнения, запускается программа `/etc/rc.d/rc.S`. Программа написана на языке shell script и по ее коду легко проследить какие действия выполняются при старте системы.

Ниже, на примере скрипта `rc.S` из дистрибутива Slackware Linux, будут подробно рассмотрены этапы запуска системы.

```
#!/bin/sh
```

В первой строке определяется интерпретатор, используемый для выполнения программы.

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

Определяется переменная `PATH`. Переменная содержит пути к директориям, в которых будут искаться выполняемые программы.

**Внимание!** Эта переменная локальная для данного скрипта и ее значение не будет передаваться остальным программам системы. Первоначальное значение переменной `PATH` определяется в специальном конфигурационном файле, который будет рассмотрен позже.

```
/sbin/mount -v proc /proc -n -t proc
```

Затем происходит подключение файловой системы `proc` к директории `/proc`. После этого будут доступны все возможности предоставляемые данной файловой системой.

Обратите внимание на опцию «-n», которая отменяет запись строки в файл `/etc/mtab`.

```
if [ -w /proc/sys/kernel/hotplug ]; then
    if grep -w nohotplug /proc/cmdline 1> /dev/null 2> /dev/null ; then
        echo "/dev/null" > /proc/sys/kernel/hotplug
    elif [ ! -x /etc/rc.d/rc.hotplug ]; then
        echo "/dev/null" > /proc/sys/kernel/hotplug
    fi
fi
```

В этих строках происходит проверка возможности запуска системы `hotplug`.

Основное назначение системы hotplug — загрузка драйверов для устройств, подключаемых без выключения питания компьютера. К таким устройствам относятся устройства, подключаемые к шинам PCMCIA, PCI и USB разъемов.

Сначала проверяется существование и возможность записи в файл `/proc/sys/kernel/hotplug`. В нем файле записана строка запуска программы, которая будет выполняться ядром Linux, если последнее обнаружит подключение нового устройства. Если этого файла нет, значит ядро собрано без поддержки возможности «горячего» подключения устройств, и тогда нет необходимости запускать автоматическую загрузку драйверов при старте системы.

Если ядро Linux поддерживает возможность «горячего» подключения устройств, но вам по каким либо причинам не хочется ее использовать. Тогда hotplug, в Slackware Linux, можно отключить двумя способами:

1. При старте ядра, передать ему в командной строке параметр `nohotplug`. Первый оператор `if` проверяет наличие этого параметра в файле `/proc/cmdline`. Если параметр присутствует, содержимое файла `/proc/sys/kernel/hotplug` обнуляется и ядро не будет знать какую программу запустить в случае обнаружения нового устройства.
2. Сделать не исполняемым файл `/etc/rc.d/rc.hotplug`. Оператор `elif` проверяет этот файла. Если он не исполняемый, содержимое файла `/proc/sys/kernel/hotplug` обнуляется.

```
if [ -x /etc/rc.d/rc.devfsd ]; then
    /etc/rc.d/rc.devfsd start
fi
```

Затем проверяется наличие файла `/etc/rc.d/rc.devfsd`. Этот файл является скриптом, позволяющим включить поддержку виртуальной файловой системы `devfsd`. В современных версиях Linux эта система не рекомендуется к дальнейшему применению.

```
if [ -d /sys ]; then
    if cat /proc/filesystems | grep -w sysfs 1> /dev/null 2> /dev/null
    then
        if ! cat /proc/mounts | grep -w sysfs 1> /dev/null 2> /dev/null
        then
            /sbin/mount -v sysfs /sys -n -t sysfs
        fi
    fi
fi
```

В следующих строках происходит проверка возможности подключения виртуальной файловой системы `sysfs`. Ее поддержка появилась в ядрах Linux, начиная с версии 2.6.x. В этой файловой системе отображаются параметры всех устройств обнаруженных в системе.

Подключение файловой системы `sysfs` возможно:

- если существует директория `/sys`,
- если `sysfs` поддерживается ядром (список поддерживаемых файловых систем находится в файле `/proc/filesystems`),
- если ее нет в списке подключенных файловых систем (список подключенных файловых систем находится в файле `/proc/mounts`).

```
if [ -x /etc/rc.d/rc.udev ]; then
    if ! grep -w nohotplug /proc/cmdline 1> /dev/null 2> /dev/null; then
        /etc/rc.d/rc.udev
    fi
fi
```

Проверяется возможность запуска программы `udev`. Эта программа предназначена для динамического формирования файлов устройств. Файлы устройств будут созданы только для тех устройств, которые реально присутствуют в системе. В качестве источника информации об устройствах используется файловая система `sysfs`.

Для запуска программы `udev` используется специальный стартовый скрипт `/etc/rc.d/rc.udev`. Поэтому в первую очередь проверяется наличие этого скрипта. Скрипт обязательно должен быть исполняемым.

Если Linux запускается без поддержки системы `hotplug`, то нет необходимости запускать `udev`, так как работа `udev` напрямую связана с `hotplug`.

```
/sbin/swapon -a
```

Подключаются все файловые системы типа swap и swap файлы, описанные в файле /etc/fstab.

```
READWRITE=no
if touch /fsrwtestfile 2>/dev/null; then
    rm -f /fsrwtestfile
    READWRITE=yes
else
    echo "Testing root filesystem status: read-only filesystem"
fi
```

После загрузки ядра, Linux подключает корневую файловую систему. В этих строках происходит проверка, в каком режиме была подключена эта файловая система: только для чтения или в режиме полного доступа.

Проверяется, удалось ли создать файл /fsrwtestfile. Если удалось, значит система подключена в режиме rw и переменной READWRITE присваивается значение yes. Если не удалось создать, на экран выводится сообщение, а у переменной остается старое значение no.

```
if [ -r /etc/forcefsck ]; then
    FORCEFSCK="-f"
fi
```

Проверяется наличие файла /etc/forcefsck. Если файл существует, переменной FORCEFSCK присваивается значение «-f».

Если вы хотите, что бы при старте системы происходила обязательная проверка файловых систем, достаточно создать файл /etc/forcefsck. В дальнейшем, в файле rc.S, для проверки файловых систем будет вызываться программа fsck, которой будет передаваться опция, находящаяся в переменной FORCEFSCK.

Ниже описан большой участок кода, в котором происходит проверка корневой файловой системы.

```
if [ ! $READWRITE = yes ]; then
    RETVAL=0
```

Если корневая файловая система подключена в режиме только для чтения (проверяется содержимое переменной READWRITE), то выполняется первая часть оператора if.

```
if [ ! -r /etc/fastboot ]; then
    echo "Checking root filesystem:"
    /sbin/fsck $FORCEFSCK -C -a /
    RETVAL=$?
fi
```

Проверяется отсутствие файла /etc/fastboot. Если файл не существует, выполняется проверка корневой файловой системы.

Файл fastboot используется администратором для быстрой загрузки Linux, без проверки файловых систем. То есть, достаточно его создать и при старте системы не будет происходить проверка файловых систем. Файл fastboot имеет приоритет перед файлом forcefsck.

```
if [ $RETVAL -ge 2 ]; then
    if [ $RETVAL -ge 4 ]; then
        echo
        PS1="(Repair filesystem) \#"; export PS1
        sulogin
    else
        echo
    fi
    echo "Unmounting file systems."
    /sbin/umount -a -r
    /sbin/mount -n -o remount,ro /
    echo "Rebooting system."
    sleep 2
    reboot -f
fi
```

Затем проверяется код возврата программы fsck.

Если при проверке файловой системы программа `fsck` не нашла никаких ошибок, она возвращает 0. Если были обнаружены ошибки и программа их исправила, она возвращает 1. Если после исправления ошибок требуется перезагрузить компьютер, код возврата равен 2. Если программа не может автоматически исправить ошибки и требуется вмешательство администратора, она возвращает 4 или более.

Первый оператор `if` проверяется условие: код возврата больше или равен 4. Если условие верно, на экран выводится большое сообщение (в примере кода оно не показано). Переменной `PS1` присваивается соответствующее значение, и она экспортируется. Затем запускается программа `sulogin`, которая спрашивает пароль пользователя `root` и запускает `shell`. В этом `shell` администратор производит все необходимые действия по восстановлению корневой файловой системы и выходит из `shell`.

Если код возврата больше либо равен 2, но меньше 4, значит требуется только перезагрузка системы. Срабатывают операторы после `else` и на экран выводится сообщение о необходимости перезагрузки системы (в примере сообщение не показано).

Все строки, начиная с «`echo "Unmounting file systems."`», выполняются всегда, при условии, что код возврата больше либо равен 2. Отключаются все файловые системы кроме корневой. Корневая файловая система переводится в режим только для чтения. Происходит задержка на 2 секунды и система перезагружается.

```
/sbin/mount -w -v -n -o remount /
if [ $? -gt 0 ] ; then
    echo
    read junk;
fi
```

Если после проверки корневой файловой системы не требуется перезагружать компьютер, тогда запускается программа `mount`, которая переводит файловую систему в режим полного доступа.

Затем проверяется код возврата программы `mount`. Если возникли ошибки, на экран выводится сообщение (в примере текст сообщения отсутствует) и система ждет когда пользователь нажмет на клавишу `Enter`.

Если корневая файловая система изначально была подключена в режиме полного доступа, то будут выполняться операторы после `else`.

```
else
    echo "Testing root filesystem status: read-write filesystem"
    if cat /etc/fstab | grep ' / ' | grep umsdos \
        1> /dev/null 2> /dev/null ; then
        ROOTTYPE="umsdos"
    fi
    if [ ! "$ROOTTYPE" = "umsdos" ]; then
        echo
        echo -n "Press ENTER to continue. "
        read junk;
    fi
fi
```

В первую очередь проверяется тип корневой файловой системы. Если используется файловая система `umsdos`, то переменной `ROOTTYPE` присваивается значение `umsdos`. Если Корневая файловая система не типа `umsdos`, тогда на экран выводится сообщение об ошибке и система ожидает когда пользователь нажмет кнопку `Enter`.

**Внимание!** Файловая система `umsdos` применялась для установки Linux поверх файловой системы `FAT`. В настоящее время, учитывая размеры накопителей на жестких дисках, эта файловая система практически не используется.

```
/bin/rm -f /etc/mtab*
/sbin/mount -w -o remount /
```

При подключении файловой системы, программа `mount` добавляет строку в файл `mtab`. При отключении файловой системы, программа `umount` удаляет соответствующую строку. Если система была выключена не корректно, в этом файле могла остаться неверная информация. Поэтому при старте системы `mtab` удаляется.

Но файл `mtab` необходим для работы некоторых программ. Поэтому на программа `mount` запускается еще раз, и перемонтирует уже подключенную корневую файловую систему только для того, что бы описание этой файловой системы было добавлено в файл `mtab`. Заодно этот файл создается.

```
if [ -d /proc/sys ]; then
    /sbin/mount -f proc /proc -t proc
```



```
fi
if [ -d /sys/bus ]; then
    /sbin/mount -f sysfs /sys -t sysfs
fi
```

Описание виртуальных файловых систем proc и sysfs, также должно присутствовать в файле /etc/mtab. Несмотря на то, что эти файловые системы уже могли быть подключены, их повторное подключение происходит только для того, чтобы соответствующие записи появились в файле mtab.

```
if [ -x /sbin/hwclock ]; then
    if grep "^UTC" /etc/hardwareclock 1> /dev/null 2> /dev/null ; then
        echo "Setting system time from the hardware clock (UTC)."
        /sbin/hwclock --utc --hctosys
    else
        echo "Setting system time from the hardware clock (localtime)."
        /sbin/hwclock --localtime --hctosys
    fi
fi
```

Если установлена программа hwclock, происходит попытка прочитать системное время из часов, установленных на материнской плате. И устанавливается время системы.

```
if [ -r /etc/isapnp.conf ]; then
    if [ -x /sbin/isapnp ]; then
        /sbin/isapnp /etc/isapnp.conf
    fi
fi
```

Если существует конфигурационный файл isapnp.conf и программа isapnp, последняя запускается.

```
if [ -x /etc/rc.d/rc.modules -a -r /proc/modules ]; then
    . /etc/rc.d/rc.modules
fi
```

Проверяется наличие файла /proc/modules. Если этот файл существует, значит ядро Linux собрано с поддержкой загружаемых модулей. Там же проверяется наличие исполняемого файла rc.modules. Если оба условия верны, тогда файл rc.modules будет выполняться тем же экземпляром bash что и основной файл rc.S.

В Slackware Linux файл rc.modules предназначен для загрузки модулей при старте системы. Содержимое файла будет рассмотрено ниже.

```
if [ -x /sbin/sysctl -a -r /etc/sysctl.conf ]; then
    /sbin/sysctl -e -p /etc/sysctl.conf
fi
```

Проверяется наличие программы sysctl и ее конфигурационного файла sysctl.conf. Если они существуют, запускается программа sysctl.

Программа sysctl применяется для записи параметров в файлы в директории /proc/sys. Вместо этой программы значения в файлы можно записывать напрямую. Например, так:

```
echo 1 > /proc/sys/net/ipv4/ip_forward
```

Строки, запускающие LVM пропущены.

```
if [ ! -r /etc/fastboot ]; then
    /sbin/fsck $FORCEFSCK -C -R -A -a
fi
```

Если отсутствует файл /etc/fastboot, включается проверка остальных файловых систем, описанных в файле /etc/fstab.

```
/sbin/mount -a -v -t nonfs,nosmbfs,noproc
```

Подключаются все файловые системы, описанные в файле /etc/fstab, за исключением сетевых (smbfs и nfs) и файловой системы proc.

```
/sbin/swapon -a
```

На всякий случай, еще раз подключается swap.

```
( cd /var/log/setup/tmp && rm -rf * )
/bin/rm -f /var/run/utmp /var/run/*pid /etc/nologin /var/run/lpd* \
    /var/run/ppp* /etc/dhpcpc/*.pid /etc/forcefsck /etc/fastboot
```

Происходит удаление временных файлов, оставшихся после процесса установки системы. А также удаление различных временных файлов. В том числе /etc/forcefsck и /etc/fastboot.

```
if [ -d /initrd ]; then
    /sbin/umount /initrd 2> /dev/null
    rmdir /initrd 2> /dev/null
    blockdev --flushbufs /dev/ram0 2> /dev/null
fi
```

Если при загрузке ядра использовалась технология inird (Initial RAM disk), RAM диск отключается, удаляется временная директория initrd и освобождается память, отведенная под RAM диск.

```
touch /var/run/utmp
chown root.utmp /var/run/utmp
chmod 664 /var/run/utmp
```

Создается файл utmp, который был удален ранее. Файл содержит список пользователей, которые в данный момент находятся в системе. Затем он передается пользователю root и группе utmp и у него изменяются права доступа.

Следующие строки пропущены, так как файловая система umsdos практически не используется.

```
echo "$(/bin/uname -sr)." > /etc/motd
```

В файл /etc/motd помещается информация о системе и номер ядра Linux. Содержимое файла выводится после входа пользователя в систему и о его применении мы поговорим позже.

**Внимание!** Обратите внимание на то, что файл /etc/motd создается после каждой загрузки системы.

```
if [ -x /etc/rc.d/rc.sysvinit ]; then
    . /etc/rc.d/rc.sysvinit
fi
```

Slackware Linux позволяет использовать стартовые скрипты SystemV. Для их запуска используется скрипт /etc/rc.d/rc.sysvinit. Он эмулирует систему инициализации SystemV.

```
if [ -f /etc/random-seed ]; then
    echo "Using /etc/random-seed to initialize /dev/urandom."
    cat /etc/random-seed > /dev/urandom
fi
if [ -r /proc/sys/kernel/random/poolsize ]; then
    dd if=/dev/urandom of=/etc/random-seed count=1 \
        bs=$(cat /proc/sys/kernel/random/poolsize) 2> /dev/null
else
    dd if=/dev/urandom of=/etc/random-seed count=1 bs=512 2> /dev/null
fi
chmod 600 /etc/random-seed
```

В последних строках файла создается все необходимое для устройства генератора псевдослучайных чисел.

## Заключение

Файл /etc/rc.d/rc.S запускается самым первым в системе, не зависимо от уровня выполнения.

Он предназначен для:

- Подключения swap пространства.
- Проверки файловых систем.
- Подключения файловых систем.
- Загрузки драйверов устройств, чья загрузка описана в файле rc.modules.

- Некоторых других действий, необходимых при старте системы.

Таким образом, можно сказать, что после выполнения скрипта `rc.S` все готово для запуска сервисов.

## Файлы `/etc/rc.d/rc.modules` и `/etc/rc.d/rc.netdevice`

Файл `/etc/rc.d/rc.modules` служит для загрузки модулей при старте системы. Загрузка модулей происходит «в ручную», то есть, путем явного вызова программы `modprobe`.

Ниже приведен пример строки из файла `rc.modules`.

```
#!/sbin/modprobe 3c503
```

Почти все строки в этом файле закомментированы. Для того, что бы интересующий вас модуль загружался при старте системы следует найти соответствующую строку, убрать символ комментария и, если необходимо, дописать параметры загрузки.

В файле `rc.modules` происходит подключение файла `rc.netdevice`:

```
if [ -x /etc/rc.d/rc.netdevice ]; then
    . /etc/rc.d/rc.netdevice
fi
```

Файл `rc.netdevices` используется для загрузки модулей сетевых карт при старте системы. Обычно он создается программой установки.

## Файл `/etc/rc.d/rc.M`

Файл `/etc/rc.d/rc.M` выполняется при переходе системы на второй, третий и четвертый уровни выполнения.

Ниже подробно рассмотрено содержимое файла `rc.M`.

```
echo "Going multiuser..."
```

На экран выводится информационное сообщение о переходе на многопользовательский уровень выполнения.

```
/bin/setterm -blank 15 -powersave powerdown -powerdown 60
```

Вызывается программа `setterm`, устанавливающая параметры терминалов.

```
if [ -r /etc/HOSTNAME ]; then
    /bin/hostname $(cat /etc/HOSTNAME | cut -f1 -d .)
else
    echo "darkstar.example.net" > /etc/HOSTNAME
    /bin/hostname darkstar
fi
```

Проверяется наличие файла `/etc/HOSTNAME`. Если файл существует, выполняется программа `hostname`, которая устанавливает имя машины. Имя берется из файла `HOSTNAME`. Если файл не существует, он создается. В него заносится имя машины `darkstar.example.net`. Самой машине присваивается имя `darkstar`.

Файл `/etc/HOSTNAME` играет важную роль при инициализации системы. В нем содержится полностью квалифицированное доменное имя машины (FQDN). Если файл случайно удалить, при старте машине будет присвоено неправильное имя. Поэтому, в файле `rc.M` рекомендуется заменить `darkstar.example.net` на реальное имя машины.

```
/bin/dmesg -s 65536 > /var/log/dmesg
```

Сообщения, которые выводились ядром при старте системы, помещаются в файл `/var/log/dmesg`.

**Внимание!** Обратите внимание на то, что программа показывает только сообщения ядра. Сообщения стартовых скриптов показаны не будут.

```
if [ -x /etc/rc.d/rc.syslog -a -x /usr/sbin/syslogd -a -d /var/log ]
then
    . /etc/rc.d/rc.syslog start
fi
```

Если существуют исполняемые файлы `/etc/rc.d/rc.syslog` и `/usr/sbin/syslogd`, а так же существует директория `/var/log`, выполняется стартовый скрипт `/etc/rc.d/rc.syslog`. Этот скрипт запускает систему журнальной регистрации `SYSLOG`.

```
if [ -x /etc/rc.d/rc.pcmcia ] ; then
    . /etc/rc.d/rc.pcmcia start
    if [ -r /var/run/cardmgr.pid ]; then
        sleep 5
    fi
fi
```

Если существует исполняемый файл `/etc/rc.d/rc.pcmcia`, то он выполняется. Этот стартовый скрипт, предназначен для включения поддержки PCMCIA контроллеров.

```
if [ -x /etc/rc.d/rc.inet1 ]; then
    . /etc/rc.d/rc.inet1
fi
```

Проверяется наличие исполняемого файла `/etc/rc.d/rc.inet1`. Если он есть, то скрипт выполняется. `rc.inet1` предназначен для конфигурации сетевых устройств и таблицы маршрутизации. Его использование будет рассматриваться в следующей главе.

```
if [ -x /etc/rc.d/rc.hotplug -a -r /proc/modules ]; then
    if ! grep nohotplug /proc/cmdline 1> /dev/null 2> /dev/null ; then
        echo "Activating hardware detection: /etc/rc.d/rc.hotplug start"
        . /etc/rc.d/rc.hotplug start
    fi
fi
```

Проверяется наличие исполняемого файла `/etc/rc.d/rc.hotplug` и файла `/proc/modules`. Также, рассматривается содержимое файла `cmdline` (в нем находится командная строка, переданная ядру при запуске) на отсутствие параметра `nohotplug`. Если все условия верны, запускается скрипт `rc.hotplug`, предназначенный для поиска установленных в системе устройств и загрузки соответствующих драйверов.

```
if [ -x /etc/rc.d/rc.inet2 ]; then
    . /etc/rc.d/rc.inet2
fi
```

Если существует исполняемый файл `/etc/rc.d/rc.inet2`, он выполняется.

Стартовый скрипт `rc.inet2` применяется для запуска различных сетевых сервисов и для подключения сетевых файловых систем.

```
/bin/rm -f /var/lock/* /var/spool/uucp/LCK.* \
/tmp/.X*lock /tmp/core /core 2> /dev/null
```

Удаляются различные временные файлы.

```
chmod 755 / 2> /dev/null
chmod 1777 /tmp /var/tmp
```

Изменяются права на корневую директорию, а также на директории `/tmp` и `/var/tmp` устанавливается sticky бит.

```
if [ -x /sbin/ldconfig ]; then
    echo "Updating shared library links: /sbin/ldconfig"
    /sbin/ldconfig
fi
```

Проверяется наличие исполняемого файла `ldconfig`. Если он есть, программа выполняется.

`Ldconfig` предназначена для создания кеша динамических библиотек. Более подробно об этой программе будет рассказано ниже.

```
if [ -x /usr/X11R6/bin/fc-cache ]; then
    echo "Updating X font indexes: /usr/X11R6/bin/fc-cache"
    /usr/X11R6/bin/fc-cache
fi
```

Если существует исполняемый файл `fc-cache`, программа выполняется. `Fc-cache` применяется для обновления индексов шрифтов системы X Window.

```
if [ -x /etc/rc.d/rc.dnsmasq ]; then
    /etc/rc.d/rc.dnsmasq start
fi
```

Если существует исполняемый файл `rc.dnsmasq`, он выполняется и ему передается параметр `start`. Скрипт предназначен для запуска простого, кеширующего DNS сервера.

```
if [ -x /etc/rc.d/rc.cups ]; then
    /etc/rc.d/rc.cups start
elif [ -x /etc/rc.d/rc.lprng ]; then
    . /etc/rc.d/rc.lprng start
fi
```

Проверяется, какая система печати установлена, `cups` или `LPRng`. В зависимости от результатов проверки, запускается соответствующий стартовый скрипт.

```
#if [ -x /usr/sbin/smardt ]; then
#    /usr/sbin/smardt
#fi
```

Если необходимо включить поддержку технологии S.M.A.R.T., удалите комментарии со строк показанных выше. Тогда при старте системы будет запускаться соответствующий демон.

```
#if [ -x /sbin/genpowerd ]; then
#    echo "Starting genpowerd daemon..."
#    /sbin/genpowerd
#fi
```

Демон `genpowered` позволяет системе работать с «интеллектуальными» UPS. Если в нем возникнет потребность, уберите комментарии со строк показанных выше.

```
if [ -x /sbin/accton -a -r /var/log/pacct ]; then
    /sbin/accton /var/log/pacct
    chmod 640 /var/log/pacct
    echo "Process accounting turned on."
fi
```

Проверяется возможность запуска программы `accton`. Это так называемая система «BSD process accounting», предназначенная для контроля за процессами в системе. Для ее включения необходима соответствующая поддержка в ядре Linux и наличие журнального файла `/var/log/pacct`.

**Внимание!** В свой журнальный файл система «BSD process accounting» помещает огромное количество информации. Следует следить за тем, что бы он не переполнил файловую систему.

```
if [ -x /usr/sbin/crond ]; then
    /usr/sbin/crond -l10 >>/var/log/cron 2>&1
fi
```

Происходит проверка возможности запуска программы `crond`. Если программа установлена, то она запускается. Система CRON будет подробно рассмотрена на следующих занятиях.

```
if [ -x /usr/sbin/atd ]; then
    /usr/sbin/atd -b 15 -l 1
fi
```

Если существует исполняемый файл `atd`, то он запускается при старте системы. Демон `atd` позволяет выполнять программы в заранее определенное время.

```
if grep -q quota /etc/fstab ; then
    if [ -x /sbin/quotacheck ]; then
        echo "Checking filesystem quotas: /sbin/quotacheck -avugm"
        /sbin/quotacheck -avugm
    fi
```

```
if [ -x /sbin/quotaon ]; then
    echo "Activating filesystem quotas: /sbin/quotaon -avug"
    /sbin/quotaon -avug
fi
fi
```

Приведенные выше строки позволяют включить механизм квотирования дискового пространства.

Квоты в Linux можно применять только к разделам. При использовании квот, раздел необходимо подключать с параметрами монтирования `usquota` или `grquota`. Первый оператор `if` проверяется наличие разделов с указанными параметрами монтирования.

Перед включением квот, необходимо запустить программу проверки текущего состояния файловой системы — `quotacheck`. Она проверит сколько места занимают файлы пользователя и сохранит эту информацию в специальных файлах.

Затем запускается механизм поддержки квот. Для этого используют программу `quotaon` с соответствующими параметрами.

О том, как настраивать квотирование дискового пространства будет рассказано позднее.

```
if [ -x /etc/rc.d/rc.sendmail ]; then
    . /etc/rc.d/rc.sendmail start
fi
```

Если существует исполняемый файл `/etc/rc.d/rc.sendmail`, он запускается.

Запуск всех остальных программ в файле `rc.M` выполнен по тому же принципу, что и запуск `sendmail`. Сначала проверяется наличие соответствующего стартового скрипта, и если он есть, то программа запускается.

Самым последним, при старте системы запускается файл `rc.local`. Он предназначен для запуска приложений, у которых нет стандартных стартовых скриптов в системе инициализации Slackware Linux.

## Файлы `/etc/rc.d/rc.inet1` и `/etc/rc.d/inet2`

### Файл `/etc/rc.d/rc.inet1`

Файл `/etc/rc.d/rc.inet1` предназначен для конфигурации сетевых интерфейсов и таблицы маршрутизации.

Для конфигурации сетевых интерфейсов используется дополнительный конфигурационный файл `/etc/rc.d/rc.inet1.conf`. Содержимое этого файла достаточно простое, в нем присутствуют четыре группы переменных, предназначенных для конфигурации четырех сетевых интерфейсов:

```
IPADDR[0]="192.168.0.1"
NETMASK[0]="255.255.255.0"
USE_DHCP[0]=""
DHCP_HOSTNAME[0]=""
```

Так же используется переменная для определения маршрута по умолчанию:

```
GATEWAY="192.168.0.100"
```

В этом файле так же находятся переменные, которые позволяют сконфигурировать беспроводные интерфейсы. Но, по умолчанию, они закомментированы.

Недостатком скрипта `rc.inet1` является его ограниченность. Например, при помощи `rc.inet1` нельзя определить дополнительные IP адреса на сетевом интерфейсе. Невозможно сконфигурировать больше, чем четыре сетевых интерфейса. Для добавления дополнительных маршрутов в таблицу маршрутизации их приходится добавлять вручную. Поэтому можно полностью удалять все содержимое файла `rc.inet1` и переписывать его по новой. При написании скрипта необходимо учесть следующее:

- Конфигурация всех сетевых интерфейсов происходит при помощи программы `ifconfig`. Следовательно, для конфигурации каждого интерфейса необходимо написать одну строку, содержащую вызов программы `ifconfig` со всеми необходимыми параметрами.
- Для добавления маршрута в таблицу маршрутизации используют программу `route`. Следовательно, для каждого добавляемого маршрута необходимо написать строку, содержащую вызов программы `route` со всеми необходимыми параметрами.

- При конфигурации сетевых интерфейсов, маршрут к сети, подключенной к ним, в таблицу маршрутизации добавляется автоматически. Что не требует явного вызова программы `route`.
- При конфигурации сетевого интерфейса `lo`, сеть `127.0.0.0` в таблицу маршрутизации автоматически не добавляется. Поэтому эту сеть необходимо добавлять явно:

```
route add -net 127.0.0.0/8 lo
```

Предположим, что у Вас есть один сетевой интерфейс `eth0` с IP адресом `192.168.0.1`. Маршрут по умолчанию — `192.168.0.100`. Тогда файл `rc.inet1` будет выглядеть следующим образом:

```
#!/bin/bash
/sbin/ifconfig lo 127.0.0.1
/sbin/ifconfig eth0 192.168.0.1
/sbin/route add -net 127.0.0.0/8 lo
/sbin/route add default gw 192.168.0.100
```

Если необходимо добавить маршруты к каким либо сетям, эти маршруты добавляются в файле `rc.inet1` путем вызова программы `route`, в любом месте этого файла, сразу после конфигурации всех сетевых интерфейсов.

### Файл `/etc/rc.d/rc.inet2`

Файл `/etc/rc.d/rc.inet2` применяется для запуска сетевых приложений и подключения сетевых файловых систем при старте Linux. В этом файле происходит попытка подключения сетевых файловых систем `nfs` и `smbfs`.

Дальше запускаются (если это необходимо) следующие стартовые скрипты:

- `/etc/rc.d/rc.firewall` — настройка `firewall`.

**Внимание!** На самом деле, запуск этого скрипта необходимо осуществлять в начале файла `rc.inet1`, перед конфигурацией сетевых интерфейсов. А в файле `rc.inet2` вызов скрипта следует удалить.

- `/etc/rc.d/rc.ip_forward` — этот скрипт разрешает пересылку пакетов с одного сетевого интерфейса на другой.
- `/etc/rc.d/rc.inetd` — запуск сетевого суперсервера.
- `/etc/rc.d/rc.sshd` — запуск демона `sshd`.
- `/etc/rc.d/rc.bind` — запуск DNS сервера.
- И другие программы.

**Внимание!** Некоторые программы, которые можно отнести к сетевым сервисам, запускаются из файла `rc.M`. Например, `sendmail` и `samba`.

## Заключение

Система инициализации Slackware Linux построена на использовании стартовых скриптов, размещенных в директории `/etc/rc.d`.

Если необходимо, что бы какой-либо сервис запускался при старте системы, соответствующий скрипт необходимо сделать исполняемым. И наоборот, если сервис не надо запускать при старте системы, стартовый скрипт не должен быть исполняемым.

Все стартовые скрипты представляют из себя простые программы, написанные на shell script. Некоторые из них понимают параметры командной строки: `start` — запуск и `stop` — останов сервиса.

Если для запуска какой-либо программы, в системе инициализации Slackware Linux не предусмотрено стартового скрипта, то можно поступить двумя способами:

1. Запуск программы описать в файле `/etc/rc.d/rc.local`.
2. Написать свой собственный стартовый скрипт и дописать его запуск в файле `/etc/rc.d/rc.M`.

Во втором случае стартовый скрипт может выглядеть следующим образом:

```
#!/bin/bash
start()
{
    echo "Program started"
```

```
        program_start
    }
    stop()
{
    echo "Program stoped"
    killall program
}
case $1 in
    start)
        start ;;
    stop)
        stop ;;
    restart)
        stop
        sleep 2
        start ;;
    *)
        echo "Usage: Program start|stop|restart"
esac
```

Вызов скрипта в файле /etc/rc.d/rc.М организуется следующим образом:

```
if [ -x /etc/rc.d/rc.script ]; then
    . /etc/rc.d/rc.script start
fi
```

**Внимание!** При помощи оператора точка можно выполнять только программы, написанные на языке shell script. Другие программы необходимо вызывать без оператора точка.

Ну и, пожалуй, самое главное — **вы можете вносить изменения в любые стартовые скрипты**. При апдейте системы, стартовые скрипты не будут изменяться, в отличие от дистрибутивов, использующих систему инициализации SystemV. Рядом с текущим стартовым скриптом появляется новый, но с расширением .new.

## Система инициализации SystemV

Система инициализации SystemV, в отличие от системы инициализации BSD, построена по другому принципу.

Среди дистрибутивов Linux, систему инициализации SystemV используют такие дистрибутивы как: RedHat Linux, SuSE Linux, Debian, ASPLinux. В этих дистрибутивах самым первым тоже загружается программа init. Но в конфигурационном файле /etc/inittab вызываются другие файлы.

Система инициализации SystemV будет рассмотрена на примере дистрибутива ASPLinux.

Файл /etc/rc.d/rc.sysinit выполняется не зависимо от уровня выполнения самым первым. Это программа, написанная на языке shell-scripts, и администратор может самостоятельно вносить изменения в файл.

В файле выполняется следующие действия:

- Если использовалась технология initrd, отключают директорию, используемую initrd и освобождается память, отводимая под RAM диск.
- Выполняется программа sysctl, конфигурирующая некоторые особенности ядра.
- Устанавливает системное время.
- Устанавливает имя машины.
- Загружаются все модули, необходимые для работы USB и IEEE1394 устройств.
- Проверяет файловые системы.
- Подключает файловые системы.
- Накладывает ограничения на файловые системы (квоты).
- Включает поддержку swp пространства.
- Если в системе поддерживается программный RAID – включается его поддержка.
- Удаляет некоторые файлы, оставшиеся после перезагрузки системы.



- Запускает программу `hdparm`, для конфигурации IDE интерфейсов.
- А так же некоторые другие действия.

Как видно из приведенного списка выполняется первоначальная загрузка системы.

Поскольку ASPLinux является прямым потомком RedHat Linux, в нем используются особенности создания стартовых скриптов RedHat — конфигурация загрузки системы происходит при помощи внешних конфигурационных файлов, находящихся в директории `/etc/sysconfig`.

Конфигурационные файлы подключаются в тело основного скрипта при помощи оператора «.». В этих файлах определяются разнообразные переменные, значение которых проверяется и используется в основном скрипте.

Например, имя машины берется из файла `/etc/sysconfig/network`. В этом файле можно определить переменную `HOSTNAME` и присвоить ей FQDN имя компьютера. Сам файл `network` подключается в основном скрипте следующим образом:

```
if [ -f /etc/sysconfig/network ]; then
    . /etc/sysconfig/network
else
    NETWORKING=no
fi
```

Как видно из примера, сначала проверяется наличие файла и только если он существует, файл подключается. После подключения в скрипте можно использовать переменные, находящиеся в файле `network`. Ну а само определение имени хоста происходит путем вызова функции `action`:

```
action $"Setting hostname ${HOSTNAME}: " hostname ${HOSTNAME}
```

Все функции, которые Вам встретятся в стартовых скриптах находятся в файле `/etc/rc.d/init.d/functions` (реальный путь к файлу, без использования символьных ссылок). Этот файл подключается таким же образом, как и файл `network`.

```
. /etc/init.d/functions
```

**Внимание!** В стартовых скриптах наличие файла `/etc/rc.d/init.d/functions` не проверяется, поскольку этот файл должен ОБЯЗАТЕЛЬНО существовать.

## Файл `/etc/rc.d/rc`

Файл `/etc/rc.d/rc.sysinit` выполняет только первоначальную загрузку системы. В дальнейшем, в зависимости от уровня выполнения, запускается программа `rc`, которой в качестве параметра передается номер уровня выполнения.

Задача программы `rc` — остановить (если они были запущены) и запустить программы, которые должны выполняться на указанном уровне выполнения.

Если в системе инициализации BSD может присутствовать скрипт аналогичный `rc.sysinit`, то аналога файла `rc` там нет. Именно наличием файла, подобного файлу `rc`, организации директорий стартовых скриптов и особенности написания стартовых скриптов для запуска служб, определяют систему инициализации System V.

Если посмотреть содержимое директории `/etc/rc.d`, где находятся все стартовые скрипты, можно увидеть следующие файлы и директории:

```
init.d/  rc0.d/  rc2.d/  rc4.d/  rc6.d/      rc.sysinit
rc       rc1.d/  rc3.d/  rc5.d/  rc.local
```

**Внимание!** В дистрибутиве SuSE Linux стартовые скрипты находятся в директории `/etc/init.d`. В RedHat Linux тоже существует такая директория, но она является символьной ссылкой на директорию, которая находится в `/etc/rc.d`.

В директории присутствует несколько похожих директорий, начиная с `rc0.d` и заканчивая `rc6.d`.

Ниже приводится содержимое директории `rc3.d`.

**Внимание!** Какие файлы будут находиться в этой директории, определяет администратор системы.

```
K05saslauthd K15httpd K20nfs K24irda K34dhcrelay
```

```
K35dhcpcd K35smb K35vncserver K35winbind K36lisa
K45named K50snmpd K50snmptrapd K50vsftpd K54pxe
K65kadmin K65kprop K65krb524 K65krb5kdc K70aep1000
K70bcm5820 K74ntpd K84bgpd K84ospf6d K84ospfd
K84ripd K84ripngd K85zebra K95firstboot
S05kudzu S08iptables S10network S12syslog S13portmap
S14nfslock S17keytable S20random S24pcmcia S25netfs
S26apmd S28autofs S55sshd S56rawdevices S56xinetd
S80postfix S85gpm S90crond S90cups S90xfs S91dictd
S95anacron S95atd S97rhnsd S99local
```

Файлы, находящиеся в этой директории имеют некоторые общие свойства:

- Их имена начинаются с “K” или “S”.
- После первой буквы идет две цифры.
- Все файлы — это либо символичные ссылки на файлы в директории /etc/rc.d/init.d, либо файлы в формате shell script.

Задача программы rc заключается в следующем:

- Она переходит в директорию rc<номер уровня выполнения>.d.
- Сначала запускает все файлы, начинающиеся на K. (Им передается параметр stop).
- Затем запускает все файлы, начинающиеся на S. (Им передается параметр start).

Все файлы, находящиеся в директории /etc/rc.d/init.d — это стартовые скрипты определенных программ.

Стартовые скрипты системы инициализации System V обязательно должны поддерживать два параметра: start и stop. По названию этих параметров можно понять, что start — это запуск программы, а stop — это завершение работы программы.

**Внимание!** В стартовых скриптах могут поддерживаться дополнительные параметры: restart, status и другие. Но они не являются обязательными.

Таким образом, видно, что rc сначала пытается завершить выполнения программ, выполняя стартовые скрипты с параметром stop. А затем запускает программы, выполняя соответствующие стартовые скрипты с параметром start.

Цифра в имени файла определяет порядок выполнения скрипта.

Администратор системы может изменить набор программ, запускаемых при старте системы, добавляя в директорию соответствующую уровню выполнения ссылки на стартовые скрипты программ или удаляя их.

## Программа chkconfig

В ASPLinux, как и в RedHat, SuSE и Mandriva Linux, для облегчения работы со стартовыми скриптами поставляется программа chkconfig.

**Внимание!** Программа chkconfig также поставляется с дистрибутивами RedHat Linux, Fedora Core, SuSE Linux. В SuSE Linux, параметры программы отличаются.

Программа chkconfig позволяет осуществлять следующие действия:

- показывает, какие сервисы, на каких уровнях выполнения будут запускаться.
- позволяет добавлять или удалять сервисы, которыми может управлять программа chkconfig.
- позволяет включать или выключать выполнение стартовых скриптов на определенных уровнях выполнения.

Стартовый скрипт должен быть специально подготовлен для того, чтобы программа chkconfig могла с ним работать. Для этого в нем необходимо написать две строки, они должны быть закомментированы.

**Внимание!** В SuSE Linux применяются другие параметры, для указания параметров программе chkconfig.

Первая строка говорит программе chkconfig на каких уровнях выполнения должен стартовать скрипт, а так же число – минимальный приоритет, и число – максимальный приоритет.

```
# chkconfig: 2345 20 80
```

Вторая строка – это краткое описание программы.

```
# description: Saves and restore configuration files of service ...
```

Для того, чтобы посмотреть на каком уровне выполнения какие скрипты будут выполняться, следует воспользоваться опцией --list, с указанием или без указания имени стартового скрипта:

```
chkconfig --list
```

```
chkconfig --list httpd
```

**Внимание!** В SuSE Linux у программы chkconfig используются другие параметры. Смотрите документацию к программе, поставляемую с вашим дистрибутивом.

Для включения выполнения стартового скрипта на определенном уровне выполнения, используют опцию --level.

```
chkconfig --level 35 httpd on
```

Если вместо ключевого слова «on» использовать «off», то указанный скрипт не будет выполняться на заданных уровнях выполнения.

В подавляющем большинстве случаев Вам не придется самостоятельно писать стартовые скрипты для сервисов, поскольку большинство из них поставляется с уже готовыми скриптами. Вашей основной задачей будет включение запуска этих скриптов на необходимых уровнях выполнения.

**Внимание!** С ASPLinux поставляется программа serviceconf. Это графическая оболочка, позволяющая управлять стартовыми скриптами. Так же управлять стартовыми скриптами можно при помощи WEB интерфейса и сервиса webmin.

## Запуск и останов сервисов вручную

В ASPLinux администратор может вручную запускать и останавливать сервисы. Для этого можно использовать:

- Стандартные стартовые скрипты системы инициализации System V.
- Программу service.

В первом случае необходимо в командной строке написать запуск стартового скрипта с указанием полного пути к нему. А в качестве параметра передавать start, stop или другие параметры, поддерживаемые стартовым скриптом.

Например, для запуска поддержки сети в ASPLinux необходимо ввести следующую команду:

```
/etc/rc.d/init.d/network start
```

Для останова:

```
/etc/rc.d/init.d/network stop
```

Для перезапуска:

```
/etc/rc.d/init.d/network restart
```

Если использовать программу service, то в качестве параметров ей передаются имя стартового скрипта службы и действие.

Для того, чтобы выполнить все описанные выше действия при помощи программы service, необходимо использовать следующие команды:

```
service network start
```

```
service network stop
```

```
service network restart
```

На самом деле программа service вызывает указанный скрипт и передает ему указанный параметр.

В SuSE Linux программы service нет. Там используется другой принцип. В директории /sbin или /usr/sbin

создается ссылка на стартовые скрипты сервиса. Ссылка начинается с букв rc, дальнейшее имя совпадает с именем стартового скрипта. Например, rcnetwork.

```
rcnetwork start
```

```
rcnetwork stop
```

## Конфигурационные файлы в директории /etc/sysconfig

В директории /etc/sysconfig содержатся конфигурационные файлы стартовых скриптов. В этих файлах определяются различные переменные. Сами файлы подключаются в скрипты при помощи оператора «.».

**Внимание!** Не следует путать конфигурационные файлы программ. Например: WEB сервера Apache и конфигурационный файл стартового скрипта /etc/rc.d/init.d/httpd. Первый — /etc/httpd/httpd.conf, предназначен для конфигурации самого WEB сервера. Второй — /etc/sysconfig/httpd, предназначен для указания опций, которые будут передаваться при запуске демона httpd.

Существует документация по файлам, находящимся в директории /etc/sysconfig. Найти ее можно в директории /usr/share/doc/itscripts-<версия>. В файле sysconfig.txt описаны все возможные переменные и их значения. В SuSE Linux документации по параметрам конфигурационных файлов нет, но сами параметры неплохо описаны в комментариях к ним.

В файле sysvinitfiles рассказывается о принципе построения стартовых скриптов. А так же о функциях, которые можно использовать в стартовых скриптах.

## Приложение 1. Восстановление пароля пользователя root

В старых версиях дистрибутивов Linux, при потере пароля суперпользователя проблема решалась просто, достаточно было загрузиться в однопользовательском режиме (уровень выполнения 1 или S). В этом случае система предоставляла командную строку с правами суперпользователя без запроса пароля. В современных дистрибутивах при загрузке в однопользовательском режиме сначала требуется ввести пароль суперпользователя (запускается программа `sulogin`). Поэтому следует воспользоваться другими методами смены пароля пользователя root.

Основная задача при смене пароля пользователя root каким либо образом получить возможность выполнять программы с привилегиями суперпользователя. Существуют два способа получить такую возможность:

Первый способ предусматривает знание опций ядра Linux. Ядру, как и любой другой программе, при запуске можно передать опции командной строки. Одна из опций позволяет заменить стандартную программу `init` на какую либо другую. В нашем случае необходимо чтобы после загрузки ядра сразу запускался интерпретатор `bash` или любой другой. В этом случае не будут выполняться стартовые скрипты, запускающие механизмы аутентификации пользователя и мы сразу получим командную строку с привилегиями пользователя root.

Для изменения стартовой программы, ядру необходимо указать опцию:

`init=/путь/к/программе.`

После старта, поскольку не выполнялись инициализационные скрипты, будет смонтирована только корневая директория в режиме только для чтения. Ваша задача перед попыткой смены пароля, перемонтировать корневую директорию в режиме `rw`.

```
mount -o rw,remount /
```

Затем достаточно вызвать программу `passwd` для смены пароля пользователя root.

```
passwd
```

И обязательно сбросить данные из буферов файловых систем на диск:

```
sync
```

Затем достаточно перезагрузить компьютер.

Этот способ изменения пароля суперпользователя возможен только в том случае, если на загрузчик операционной системы не установлен пароль.

Второй способ. Необходимо загрузиться с внешнего носителя: загрузочная дискета или CD-ROM. Подключить файловую систему, в которой находится директория `/etc`. Затем необходимо изменить пароль пользователя в файле `shadow`.

Например, корневая директория находится в разделе `/dev/hda3`. Мы загружаемся с первого CD-ROM дистрибутива Slackware Linux. В результате получаем полноценную систему. Корневая директория которой находится в RAM диске. Теперь необходимо подключить раздел где находится корень системы, которую мы восстанавливаем:

```
mount /dev/hda5 /mnt
```

Интересующий нас файл `shadow` будет находится в директории `/mnt/etc`. Но редактировать его вручную не потребуется. Мы запустим стандартную программу, предназначенную для смены паролей — `passwd`, но с использованием другой программы — `chroot`:

```
chroot /mnt passwd
```

Программа `chroot` ограничивает доступ запускаемой программы к файловой системе. Она делает так, что программе «кажется», что `/mnt` — это корень файловой системы и за пределами этого корня она ничего не «видит». `Passwd` изменяет пароль в файле `/etc/shadow`, но поскольку для нее корень файловой системы находится в `/mnt`, она изменяет содержимое файла `/mnt/etc/shadow`.

## Методические рекомендации

Диск с дистрибутивом можно получить через ЛинуксЦентр или загрузить с сайта проекта Slackware Linux.

На компьютерах учащихся должно быть установлено по две сетевые карты. Одна из которых соединена с сетью класса, а вторая, с помощью crossover cable, с компьютером соседа. Это необходимо для лабораторной работы по настройке маршрутизации. Во время выполнения этой лабораторной работы, на компьютере с четным номером отключается кабель, соединяющий с сетью класса. Остальные действия описаны в самой лабораторной работе.

Продолжительность данного курса предполагается 40 ак./часов.

Вот примерная тематическая схема, расписанная по часам:

тема	ак/ч
1 Исторические сведения. Файловая система. Стандартный набор утилит UNIX.	8
2 Безопасность системы: права доступа, процессы. Текстовые редакторы.	4
3 Основы shell-script.	4
4 Инсталляция Linux.	4
5 Основные обязанности администратора. Русификация.	4
6 Конфигурация сети в Linux.	4
7 Резервное копирование, как один из основных элементов системы безопасности. Система печати CUPS	4
8 Ядро Linux. Настройка загрузчика.	4
9 Система инициализации BSD и SystemV.	4

Модуль 1. Исторические сведения. Файловая система. Стандартный набор утилит UNIX.

- История создания Unix и Linux
- Дистрибутивы Linux
- Файловая система
- Стандартный набор утилит

Модуль 2. Безопасность системы: права доступа, процессы. Текстовые редакторы.

- Права доступа, как основа системы безопасности Unix
- Запуск и мониторинг процессов
- Взаимодействие процессов и управление процессами при помощи сигналов
- Текстовый редактор vi

Модуль 3. Основы shell-script

- Основы программирования на языке shell-script

Модуль 4. Инсталляция Linux

- Варианты установки Linux
- Создание разделов жесткого диска
- Выбор устанавливаемого программного обеспечения

Модуль 5. Основные обязанности администратора. Русификация

- Создание, проверка и монтирование файловых систем

- Работа с пользователями
- Создание пользовательского окружения
- Русификация консоли и консольных приложений
- Управление программным обеспечением

Модуль 6. Конфигурация сети в Linux.

- Работа с модулями (драйверами)
- Настройка сети
- Статическая маршрутизация
- Сетевые утилиты
- Протокол ssh (удаленный доступ, безопасное копирование данных, создание туннелей)

Модуль 7. Резервное копирование, как один из основных элементов системы безопасности. Система печати CUPS

- Программы для архивирования и компрессии данных
- Классическая система печати
- Система печати CUPS
- Сетевая печать
- Стратегии осуществления резервного копирования

Модуль 8. Ядро Linux. Настройка загрузчика

- Конфигурирование и сборка ядра
- Настройка загрузчика операционной системы

Модуль 9. Система инициализации BSD и System V

- Инициализация системы. BSD стиль
- Инициализация системы. Стиль System V
- Конфигурация login