

Технология настройки и компилирования своего ядра

В этой главе будут рассмотрены все этапы компилирования ядра, а также приведены рекомендации по повышению производительности системы.

Я использую версию ядра 2.4.18, но основные принципы останутся теми же и во всех следующих версиях ядра. Недавно появилось ядро 2.6, но оно еще не получило широкого распространения. О нем мы немного поговорим в конце этой главы.

31.1. Параметры ядра

Во время загрузки ядру ОС Linux могут быть переданы различные параметры. В этой главе будут рассмотрены не все параметры ядра (полное их описание занимает достаточно много места). За более подробным их описанием вам следует обратиться к BootPrompt-HOWTO. Передача параметров может быть осуществлена либо с помощью загрузчика LILO, либо с помощью любого другого загрузчика Linux (например, bootlin, bootact). В том случае, если вы решили использовать LILO, то в ответ на приглашение нужно ввести:

```
linux строка_параметров.
```

где **linux** — метка, указанная в файле `/etc/lilo.conf`.

Вторым способом указания ядру параметров является команда **append**, используемая в файле конфигурации LILO — `/etc/lilo/conf`. Параметры при этом следует указывать в следующем виде:

```
параметр [=значение1] [, значение2] ... [, значениеN]
```

Значения разделяются запятой без пробелов. Если нужно указать несколько параметров, используйте пробел для их разделения.

Пример строки параметров:

```
// правильное объявление параметров
root=/dev/hda1 ether=9,0x300,0xd0000,0xd4000,eth0
// неправильное объявление параметров
root=/dev/hda1 ether=9, 0x300, 0xd0000, 0xd4000, eth0
```

31.1.1. Параметры корневой файловой системы

Итак, начнем описание параметров с параметров корневой файловой системы:

root=устройство

Устанавливает корневую файловую систему.

Например, `root=/dev/hda1`. В качестве устройства допустимыми являются:

1. `/dev/hdaN` .. `/dev/hddN` — для IDE-дисков
2. `/dev/sdaN` .. `/dev/sdeN` — для SCSI-дисков
3. `/dev/xdaN` .. `/dev/xdbN` — для XT-совместимых дисков
4. `/dev/fdN` — дисковод для дискет. `N=0` — диск A, `N=1` — диск B
5. `/dev/nfs` — не является устройством, но указывает ядру, что нужно произвести загрузку по NFS.

ro

Этот параметр указывает монтирование корневой файловой системы в режиме «только чтение». Используется по умолчанию.

rw

Задаёт монтирование корневой файловой системы в режиме «чтение/запись». При использовании этого параметра нельзя запускать программы типа **fsck**. Перед запуском программы **fsck** нужно перемонтировать корневую файловую систему в режиме **ro**.

31.1.2. Управление RAMDISK

При создании загрузочных дискет для ОС Linux необходимо, чтобы на эти дискеты было помещено нужное программное обеспечение и чтобы для этого программного обеспечения хватило места. Обычно поступают следующим образом: создают сжатый архив всего необходимого программного обеспечения и помещают его на загрузочный диск.

При загрузке системы в памяти создается «электронный» диск, на который это программное обеспечение и записывается. Этот «электронный» диск называется **RAM-диск**. Описываемые далее параметры задают режимы работы с RAM-диск.

ramdisk_start=<смещение>

Разрешает ядру находиться на гибком диске вместе со сжатым образом RAM-диска.

Ядро не может быть включено в сжатый образ файловой системы RAM-диска, так как оно должно быть записано начиная с нулевого сектора, чтобы BIOS могла загрузить загрузочный сектор и ядро могло продолжить загрузку.

Если вы используете несжатый образ RAM-диска, то ядро может быть частью образа файловой системы. Такая дискета может быть загружена с помощью LILO.

В том случае, если вы для загрузки используете две дискеты (первая содержит ядро — **boot**, на второй находится образ файловой системы — **root**), образ файловой системы должен начинаться на нулевом секторе, а и смещение = 0.

load_ramdisk=

Этот аргумент заставляет ядро использовать RAM-диск. Значение **load_ramdisk=1** сообщает ядру, что нужно загрузить дискету с RAM-диска. Значение по умолчанию 0 (ядро не использует RAM-диск).

prompt_ramdisk=

Сообщает ядру, что нужно запросить дискету, которая содержит образ файловой системы (пример: **prompt_ramdisk=1**).

ramdisk_size=

Устанавливает размер RAM-диска в Кб.

ramdisk=

Определяет размер (в Кб) устройства RAM-диска. Например, для загрузочной дискеты 1.44 Мб нужно указать **ramdisk=1440**. Этот аргумент поддерживается ядрами, начиная с версии 1.3.47.

31.1.3. Управление памятью

Управление памятью осуществляется с помощью параметра **mem**:

mem=

Определяет объем памяти, установленной в компьютере.

Например: **mem=16384Kб** или **mem=16Мб**.

Иногда нужно указать объем ОЗУ, отличный от того, который имеется на самом деле. Например, у вас чипсет Intel 810 с интегрированной видеоплатой, тогда вам нужно указать объем ОЗУ на 1 Мб меньше (а иногда даже на

2 Мб). Это связано с аппаратной особенностью чипсета. Более подробно об этом вы можете узнать на сайте компании Intel (<http://www.intel.com>).

31.1.4. Другие параметры ядра

debug

Сообщения ядра (важные и не очень) передаются через функцию **printk()**. Если сообщение очень важно, то его копия будет передана на консоль, а также функции **klogd()** для его регистрации на жестком диске.

Сообщения передаются на консоль, потому что иногда невозможно за-протолировать сообщение на жестком диске (например, отказ самого диска). Предел того, что будет отображаться на консоли, задается переменной **console_loglevel**. По умолчанию на консоли отображается все, что выше уровня **DEBUG (7)**. Список уровней можно найти в файле `kernel.h`

init=

По умолчанию ядро пытается запустить программу `/sbin/init`, которая продолжит загрузку согласно стартовым сценариям (**rc**). Если программа **init** повреждена, вы можете использовать параметр **init=/bin/sh**. В оболочке вы сможете заменить поврежденную программу.

no-hlt

Процессоры 386 (и выше) имеют инструкцию **hlt**, которая сообщает процессору не производить никаких действий. При этом обычно процессор переводится в режим пониженного потребления энергии и ожидает прерывания от устройства. Параметр **no-hlt** отключает использование инструкции **hlt**. Существование этого параметра обусловлено тем, что некоторые чипы 486DX-100 имеют проблемы с этой инструкцией. Кроме того, параметр **no-hlt** позволяет использовать Linux на бракованных процессорах.

no387

Отключает использование математического сопроцессора.

no-scroll

Отключает функцию прокрутки экрана во время загрузки.

reboot=

Параметр, задающий режим перезагрузки. Возможные значения: **cold** и **warm**, то есть «холодная» или «горячая» перезагрузка. Поддерживается ядрами версии 2.0 и выше.

single

Устанавливает однопользовательский режим для администрирования системы, например, в случае отказа.

31.2. Конфигурирование ядра

В этом пункте будут рассмотрены все этапы компилирования ядра, а также приведены рекомендации по повышению производительности системы. Настройка параметров ядра будет приводиться на примере современного ядра 2.6. Если вы все еще используете ядро версии 2.4, вам следует купить второе издание этой книги. Ядро версии 2.2 рассматривалось в первом издании. Я понимаю, что сейчас купить второе издание сложно, а первое — вообще невозможно, поэтому если вам действительно нужно описание ядра 2.4 или 2.2, напишите мне — что-нибудь придумаем.

В отличие от первых двух изданий этой книги, в качестве основного конфигуриатора ядра будет рассмотренный графический конфигуриатор `qconf` (`make xconfig`), который более удобен, чем текстовая версия и версия, основанная на меню.

31.2.1. Code maturity level options

В данном разделе можно включить поддержку различных экспериментальных драйверов (читайте — модулей)

31.2.2. General setup

Support for paging of anonymous memory

Грозно звучит, не так ли? Я сначала даже не понял, что это. Оказывается это просто поддержка свопа — своп-устройств и своп-файлов. Настоятельно рекомендуется не отключать эту опцию — сколько бы ни было оперативной памяти, а своп все равно пригодится.

System V IPC

Поддержка средств межпроцессного взаимодействия System V (IPC — Inter Process Communication): очереди сообщений, семафоров, разделяемой памяти и т.д. Отключать не нужно, иначе программы не смогут «общаться» друг с другом.

BSD Process Accounting

Учет процессов. С помощью специального системного вызова пользовательская программа может попросить ядро записать системную информацию в специальный файл: время создания процесса, идентификатор владельца, командную строку, использование ресурсов, например, памяти и терминалов, и т.д. Чтобы все работало как нужно, не отключайте эту опцию.

Sysctl support

Включает поддержку **Sysctl**. Поддержка Sysctl позволяет изменять параметры ядра без перекомпилирования во время загрузки. Поддержка sysctl увели-

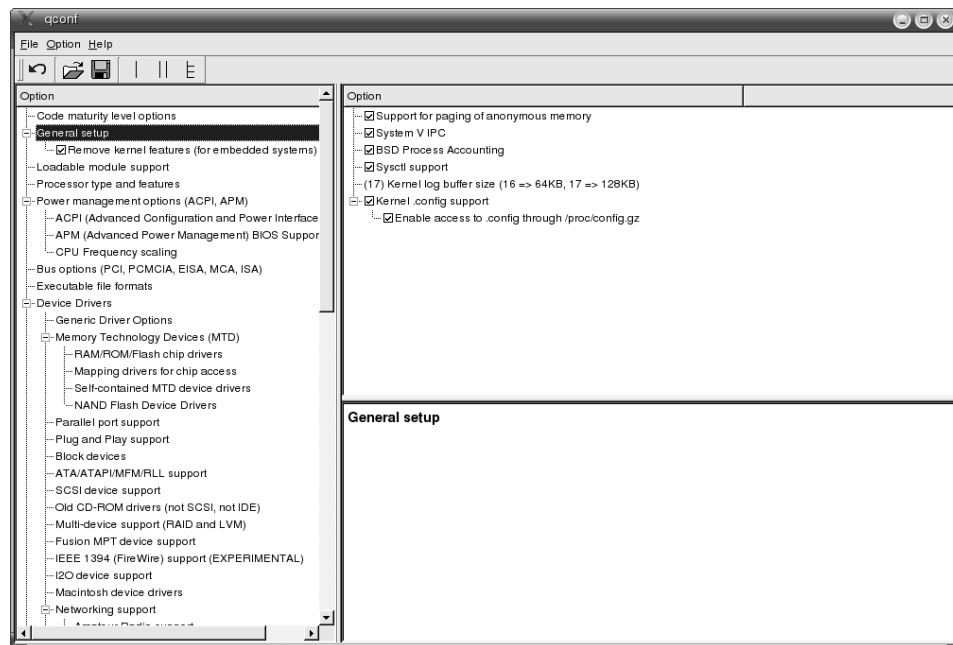


Рис. 31.1. Раздел General setup

чивает размер ядра на 8Кб. Если ядро, которое вы компилируете, не предназначено для дисков загрузки/восстановления, включите эту опцию.

Kernel log buffer size

Задаёт размер буфера протокола ядра в зависимости от значения, указанного в программе конфигурирования ядра:

- ♦ 17 — 128 Кб (по умолчанию)
- ♦ 16 — 64 Кб
- ♦ 15 — 32 Кб (рекомендуется для SMP)
- ♦ 14 — 16 Кб
- ♦ 13 — 8 Кб
- ♦ 12 — 4 Кб

Kernel .config support

Поддержка файлов `.config`, содержащих конфигурацию ядра.

31.2.3. Loadable module support

Если вы планируете использовать загружаемые модули, включите все функции. Можно создать компактную версию ядра, которая вообще не использует модули, а поддержка всех необходимых устройств будет включена непосредственно в ядро.

редственно в ядро. В этом случае можно отключить все функции в этой секции. Когда вы включаете ту или иную опцию в состав ядра, в программе `qconf` она отображается в виде галочки, если же опция не включается в состав ядра — в виде пустого квадратика, а если опция включается в виде модуля, то она представлена пиктограммой в виде кружочка.

Enable loadable module support

Включить поддержку загружаемых модулей. Рекомендуется не отключать эту опцию, если вы собираете обычное ядро для настольной системы или сервера. Если же вы собираете компактное ядро, можно эту опцию выключить, а все необходимые модули включить в состав ядра.

Module unloading

Разрешить удаление модулей из ядра. Если эта опция выключена, вы не сможете удалить модуль из ядра после того, как он был загружен.

Force module unloading

Принудительное удаление. Модуль будет удален из ядра, даже если какой-то процесс его использует. Такое удаление может быть опасным — ведь в результате удаления модуля устройства, которое в данный момент используется процессом, очень велика вероятность потери данных, поэтому лучше эту опцию не включать. Наоборот, когда вы занимаетесь программированием модулей, то есть созданием своих модулей, эта опция очень полезна, поскольку помогает сразу же выгрузить некорректно работающий модуль.

Module versioning support

Экспериментальная поддержка версий модулей. Обычно используются модули, которые откомпилированы для этой версии ядра. Включение данной опции позволяет использовать модули, откомпилированные для другой версии ядра. Нет никакой гарантии, что модуль, откомпилированный под другую версию ядра, будет работать стабильно с вашей версией, поэтому лучше выключите эту опцию.

Automatic kernel module loading

Обычно некоторые части ядра выполнены в виде модулей ядра. Когда ядру нужен тот или иной модуль, перед использованием модуля оно должно загрузить его (команда `modprobe`). При включенной этой опции ядро сможет автоматически загружать необходимые модули. Рекомендуется включить эту опцию.

31.2.4. Processor type and features

Здесь можно указать тип процессора и его функции, например, поддержка памяти более 1 Гб, MTRR, эмулирование математического сопроцессора.

Subarchitecture type

Тип архитектуры процессора:

- ♦ PC-compatible — PC-совместимый процессор, то есть процессор, использующий систему команд x86.
- ♦ Voyager (NCR) — SMP-архитектура, разработанная компанией NCR Corp.
- ♦ NUMAQ — позволяет запускать Linux на архитектуре NUMA (IBM/Sequent).
- ♦ SGI 320/540 — графические станции SGI.

Processor family

Очень важно правильно указать тип процессора: после того, как я правильно указал тип своего процессора, производительность системы повысилась примерно в 1,5 раза, особенно это стало ощутимо при загрузке системы. Данная функция используется для оптимизации работы процессора. Если вы укажете тип процессора, например 486, 586, Pentium, PPro, ядро не обязательно будет запускаться на более ранней архитектуре. Например, если вы укажете Pentium, ядро будет работать на PPro (хотя и медленнее), но нет никакой гарантии, что оно запустится на 486. В табл. 31.1 приведены типы процессоров, которые рекомендуются для получения наибольшей производительности.

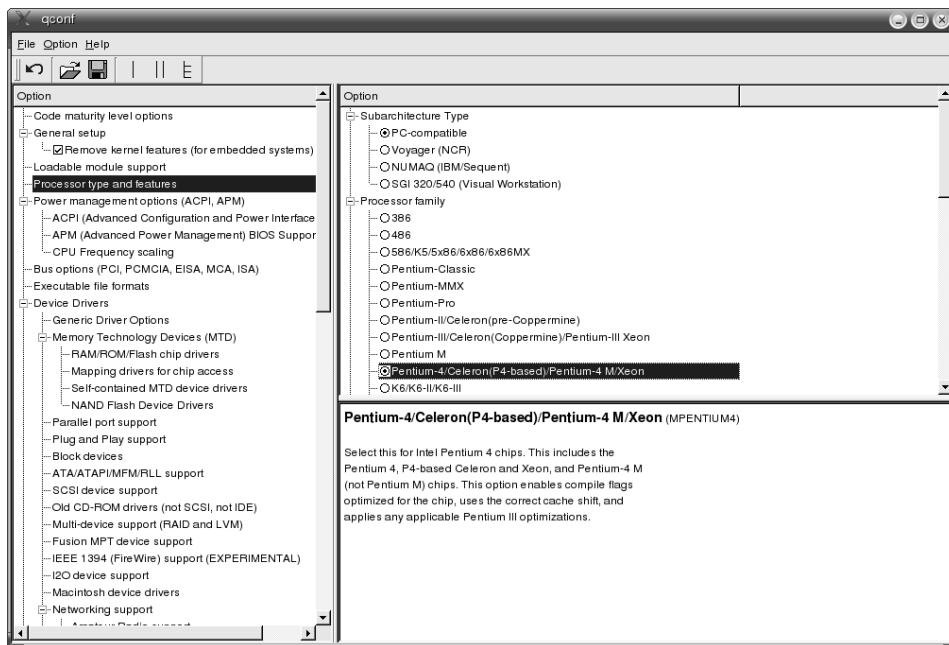


Рис. 31.2. Раздел **Processor type and features**

Типы процессоров

Таблица 31.1

Тип	Процессоры
386	Процессоры производства AMD/Cyrix/Intel 386DX/DXL/SL/SLC/SX, Cyrix 486DLC/DLC2, UMC 486SX-S
486/Cx486	AMD/Cyrix/Intel/IBM DX4, 486DX/DX2/SL/SX/SX2 AMD/Cyrix 5x86 NexGen Nx586, UMC U5D или U5S
586/K5/5x86/6x86/6x86MX	Обычные (самые первые) процессоры Pentium, AMD K5. Не используются инструкции RDTSC (Read Time Stamp Counter)
Pentium-Classic	Классические процессоры Pentium — без поддержки MMX. Используются инструкции RDTSC
Pentium-MMX	Процессоры Pentium с поддержкой MMX
Pentium Pro	Процессоры Pentium Pro/Celeron/Pentium II
Pentium II/Celeron (pre-Coopermine)	Процессоры Pentium II и Celeron (версия, которая была до Coopermine)
Pentium III/Celeron (Coopermine)/Pentium III Xeon	Процессоры Pentium III и Celeron (Coopermine)
Pentium M	Мобильная версия процессора Pentium с пониженным энергопотреблением для ноутбуков
Pentium 4 / Celeron (P4 based) / Pentium 4 M / Xeon	Процессоры Pentium 4, включая версию Pentium 4 M
K6/K6 II/K6 III	Процессоры AMD K6, K6-II, K6-III
Athlon/Duron/K7	Процессоры AMD Athlon/Duron
Opteron/Athlon64/Hammer/K8	64-х разрядные процессоры фирмы AMD
Elan/Crusoe	Процессоры Elan/Crusoe
WinChip-C6	Процессоры WinChip C6
WinChip 2A/Winchip-3	Процессоры WinChip 2A/Winchip-3
Cyrix III/C3	Процессоры IBM Cyrix III/C3

В моем случае ядро было оптимизировано под 586/K5. После того, как я установил Athlon/Duron/K7, Linux заработал быстрее (для справки: тогда у меня был AMD Duron 1,6 Ghz). Выбирайте именно ваш процессор — ваш классический Pentium не заработает быстрее и у него не появится поддержки MMX, если вы выберете Pentium MMX. Вообще система может зависнуть, поскольку она будет пытаться использовать MMX-инструкции, которые не поддерживаются процессором.

Generic x86 support

Базовая поддержка команд x86.

HPET Timer Support

HPET — это следующее поколение таймеров, пришедшее на смену классическому таймеру 8254. Просто включите эту опцию. Даже если ваша

BIOS не поддерживает HPET, будет активизирована поддержка классического таймера 8254.

Symmetric multi-processing support

Скорее всего, у вас установлен один процессор, и эту опцию вам нужно будет отключить — зачем включать лишний код в ядро? Если же вы счастливый обладатель мультипроцессорной машины, включите данную опцию. При включении SMP укажите правильный тип процессора. Вы должны указать хотя бы 586. Ядро не запустится, если у вас выбран тип процессора 486. Также ядро не будет работать, если ваш компьютер оснащен процессором Pentium, а вы установили тип процессора PPro. Если у вас мультипроцессорная машина, вы должны также включить опцию **Enhanced Real Time Clock Support**. Опция **Advanced Power Management** у вас будет отключена при использовании SMP.

Preemptible kernel

Данную опцию следует включить, если вам нужно ядро для RT-системы.

Local APIC support for uniprocessors

Поддержка внутреннего контроллера прерываний процессора (APIC — Advanced Programmable Interrupt Controller). Если у вас однопроцессорная система, единственный процессор которой оснащен APIC, включите эту опцию. Если ваш процессор не поддерживает APIC, включение данной опции существенно снизит производительность системы.

У вас мультипроцессорная система? Тогда APIC будет использоваться по умолчанию вне зависимости от значения этой опции.

Machine Check Exception

Позволяет процессору сообщать ядру о внутренних проблемах, например, о сбое.

Toshiba, DELL laptop support

Поддержка ноутбуков фирм Toshiba и DELL.

/dev/cpu/microcode

Включив эту опцию, вы сможете обновлять микрокод процессоров P Pro/ P II/ P III/ P4/ Xeon с помощью устройства `/dev/cpu`. Для работы этой опции нужно включить файловую систему `/dev` в разделе File systems. Информацию о микрокоде вы можете получить по адресу: <http://www.urbanmyth.org/microcode/>. Если вы откомпилировали эту опцию как модуль, для загрузки модуля нужно прописать в вашем файле `/etc/modules.conf` строку:

```
alias char-major-10-184 microcode
```

`/dev/cpu/*/msr`

Поддержка регистров MSR. Может понадобиться в некоторых случаях для SMP-систем. Вы можете ее со спокойной совестью выключить или, по крайней мере, откомпилировать как модуль

`/dev/cpu/*/cpuid`

Поддержка информации о процессоре. Рекомендуется включить эту опцию. Загляните в файл `/dev/cpu/0/cpuid` — вы узнаете много интересного о своем первом (0) процессоре.

High Memory Support

Поддержка памяти более 1GB.

Math emulation

Включите эту опцию, если вы используете один из следующих процессоров: 386SX/DX/SL/SLC без 80387, 486SL/SX/SX2.

MTRR

В семействе процессоров Intel P6 (Pentium Pro, Pentium II и выше) используются специальные регистры — **Memory Type Range Registers (MTRR)**. Эти регистры используются для управления доступом процессора к различным диапазонам памяти. Включение этой опции может существенно повысить производительность системы, особенно если вы используете видеокарту PCI или AGP. Данную возможность поддерживают процессоры и сторонних производителей: Cyrix 6x86, 6x86MX, MII, AMD K6-2 (stepping 8 и выше), K6-3, Centaur C6. Некоторые BIOS устанавливают MTRR для первого процессора, но отключают для второго. Активизация данной опции также решает и эту проблему. Если вы не уверены, поддерживает ли ваш процессор **MTRR**, все равно включите данную опцию. Поддержка MTRR увеличит объем ядра всего лишь на 3Кб.

31.2.5. Power Management Options

В этом разделе вы найдете все опции, касающиеся управления питанием. В принципе, с опциями Power Management разобраться не сложно и самому — в крайнем случае можно все оставить по умолчанию — опции вполне приемлемы. Но есть одна очень интересная экспериментальная (!) опция — Software Suspend. Если эта опция включена, вы можете приостановить машину, а при следующей загрузке — сразу же восстановить ее до того состояния, в котором она находилась до приостановки.

Работает это так: вы вводите команду **swsusp** или **shutdown -z <время>**, система записывает образ содержимого памяти на своп-раздел (или в своп-файл). Затем, при загрузке, вы вводите параметр ядра `resume=/dev/своп-`

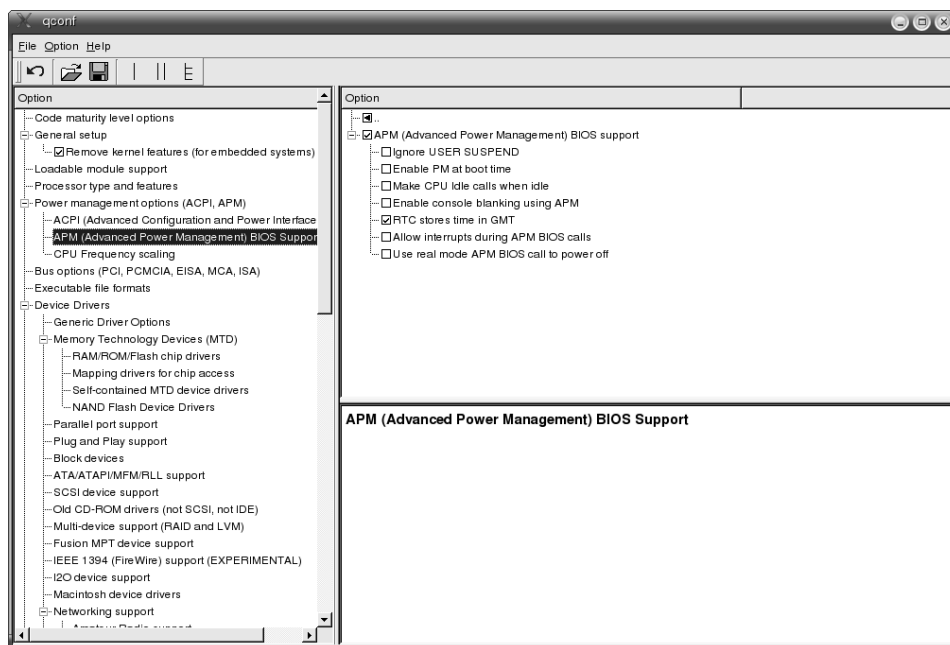


Рис. 31.3. Раздел Power Management Options / Advanced Power Management

раздел и система будет восстановлена до исходного состояния. Если вы не ввели этот параметр, то использование своп-раздела, который использовался для приостановки системы (создания образа памяти), будет невозможно. Опция **Suspend to disk** позволяет записывать образ памяти на диск, а не только на своп-раздел.

В этом же разделе можно включить опцию **Advanced Power Management**. Если вам нужно отключить функцию APM во время загрузки, введите в качестве параметра ядра **apm=off**.

При возникновении проблем (на старых компьютерах) проверьте следующее:

1. Наличие достаточного количества свопа (объема файла подкачки), а также убедитесь, что раздел подкачки включен.
2. Передайте ядру инструкцию **no-hlt**.
3. Попробуйте отключить поддержку сопроцессора (инструкция **no387**).
4. Передайте ядру инструкцию **floppy-nodma**.
5. Убедитесь, что процессор не «разогнан».
6. Установите новый вентилятор для процессора.

Что же касается APM, следует обратить внимание на следующие опции (см. табл. 31.2):

Опции APM

Таблица 31.2

Опция	Описание
Enable PM at boot time	Включает APM во время загрузки системы. Если эта опция отключена, BIOS не будет управлять питанием устройств, входить в режимы Standby и Suspend , а также не будет производить никаких действий в ответ на вызовы процессора CPU Idle . Если ваш компьютер зависает во время загрузки, выключите эту опцию
Make CPU Idle calls when idle	Во время цикла простоя ядра разрешает вызовы к APM. Включение данной опции может привести к зависанию компьютера во время загрузки! Если компьютер использует несколько процессоров, эта опция игнорируется. Заметьте, сколько процессоров именно <i>использует компьютер</i> , а не сколько в нем установлено. Если у вас два процессора, а вы используете только один и поддержка SMP у вас отключена, данная опция игнорироваться не будет!
Enable console blanking using APM	Включает мерцание консоли при использовании APM. Некоторые ноутбуки могут использовать эту опцию для того, чтобы отключить подсветку LCD-экрана, когда активизирован хранитель экрана на одной из виртуальных консолей Linux
RTC stores time in GMT	Если ваш аппаратный таймер сохраняет время в формате GMT, включите эту опцию, иначе она должна быть отключена. Если опция выключена, сохраняется локальное время. Рекомендуется сохранять время в формате GMT
Allow interrupts during APM BIOS calls	Обычно прерывания внешних устройств запрещены во время выполнения процедур APM. BIOS некоторых ноутбуков разрешает прерывания внешних устройств, например, IBM ThinkPad. По умолчанию данная опция выключена. Если вы не уверены, не включайте ее

31.2.6. Bus Options

В этом разделе находятся опции, касающиеся поддержки различных шин PCI, ISA, MCA. Данные опции вынесены из раздела **General Setup** (как это было в настройках ядра 2.4) в собственный раздел. Отключайте все, кроме шины PCI — не думаю, что у вас есть ISA или MCA-устройства.



Примечание

MCA — шина передачи данных, разработанная IBM — использовалась в системах PS1/PS2. Шина MCA давно снята с производства и не используется.

PCI access mode

Данная опция определяет режим доступа к PCI-устройствам. Если значение этой опции равно BIOS, значит, Linux будет использовать BIOS для определения PCI-устройств и их конфигураций. Однако на некоторых старых материнских платах BIOS не может корректно определить конфигурацию PCI-устройств, в этом случае нужно выбрать значение Direct, и Linux будет работать с PCI-устройствами напрямую — без BIOS. Если вы выберете Any, то Linux сначала попытается работать напрямую (так быстрее), а потом (если напрямую не получится) уже использовать BIOS.

31.2.7. Executable file formats

В этом разделе вы сможете включить поддержку различных исполнимых форматов. Обычно это нужно, если вы хотите запускать в эмуляторах программы других операционных систем, например, DOS- или Windows-программы.

31.2.8. Device drivers

В данном подразделе находятся опции, касающиеся драйверов устройств. Тут вы можете определить, какие устройства у вас установлены и какие хотите использовать в дальнейшем. По сравнению с программой настройки ядра 2.4, многие отдельные разделы «переехали» в этот раздел, например, Parallel port support, MTD, PnP support и т.д.

Memory Technology Devices (MTD)

В данном подразделе вы можете включить поддержку MTD-устройств. Самым ярким примером такого устройства может послужить flash-диск.

Parallel port support

Поддержка параллельного порта.

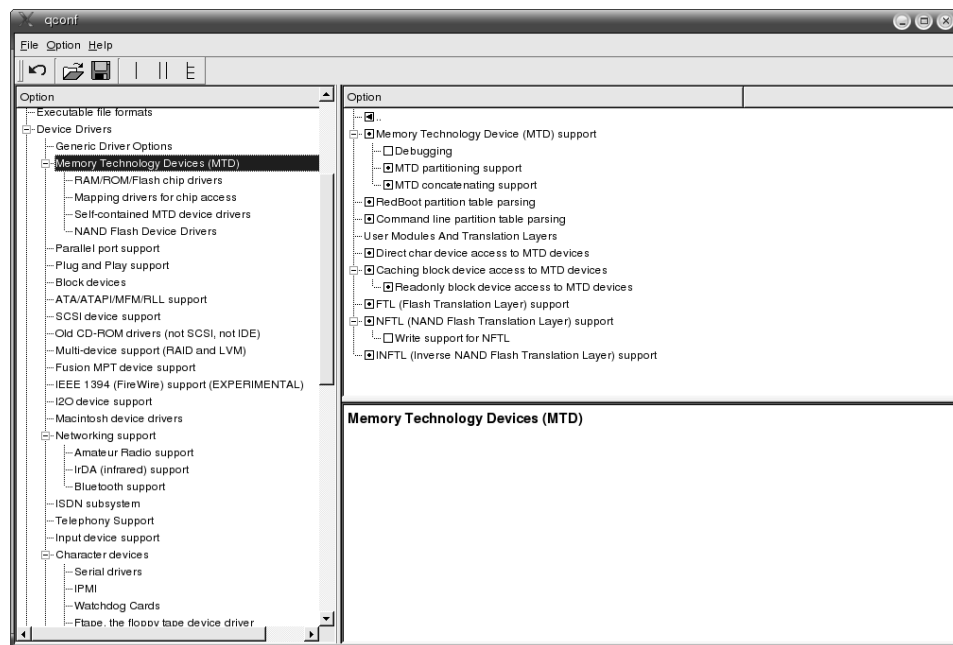


Рис. 31.4. Раздел Device Drivers

PnP support

В данной секции задается поддержка Plug and Play.

Block devices

В этом подразделе вы можете настроить блочные устройства, то есть такие устройства, обмен данными с которыми выполняется поблочно (передаются целые блоки информации), а не посимвольно, когда за одну операцию ввода/вывода передается 1 символ (байт) информации). Ярким примером блочных устройств выступают дисковые накопители — дисковод для гибких дисков, жесткие диски и т.д.

Normal floppy disk support

Поддержка обычного дисковода для гибких дисков. Обычно поддержка данного устройства требуется, но если вы собираете ядро для сервера, «флоппик» на нем не нужен, поэтому можете данную опцию выключить или хотя бы оставить в виде модуля. Модуль будет называться floppy.

XT hard disk support

Поддержка старых жестких дисков, которые устанавливались на компьютеры типа IBM XT. Отключите эту опцию — она вам не нужна, модуль тоже вам не нужен. Даже если вы где-то и найдете такой диск, вряд ли вы станете его подключать к современному компьютеру.

Parallel port IDE device support

Поддержка IDE-устройств, которые подключаются к компьютеру по параллельному порту. Очень часто к переносным компьютерам (ноутбукам, лэптопам) подключаются внешние IDE-устройства. Так как в то время шины USB не было, то был разработан интерфейс для подключения IDE-устройств по параллельному порту (по последовательному порту обмен информацией занимает много времени).

Compaq SMART2 support

Поддержка контроллеров SMART2 фирмы Compaq. Вряд ли она вам понадобится.

Compaq Smart Array 5xxx support

То же самое, что и предыдущая опция, но включается поддержка контроллеров 5xxx.

Mylex DAC960/DAC1100 PCI RAID Controller support

Поддержка RAID-контроллера фирмы Mylex.

Micro Memory MM5415 Battery Backed RAM support (EXPERIMENTAL)

Поддержка специальных карт памяти. Честно говоря, я эти карты и в глаза не видел, не то, чтобы в руках подержать. Если кому-то интересно, посетите сайт <http://www.umem.com/>.

Loopback device support

Вот эту опцию я отключать не советую — для многих задач нужны так называемые loopback-устройства. Поэтому если хотите сделать ядро компактнее, включите эту опцию хотя бы в виде модуля. Модуль будет называться `loop`.

Network block device support

Поддержка сетевых блочных устройств. Просто включите эту опцию в виде модуля. Модуль называется `nbd`.

RAM disk support

Это очень полезная опция, позволяющая часть информации, находящейся на жестком диске перенести в оперативную память для ускорения доступа к ней. Особенно эта опция полезна при создании загрузочных дисков, которые используются для восстановления системы. Если вы включите эту опцию в виде модуля, он будет называться `rd`.

Initial RAM disk (initrd) support

Инициализирующий RAM-диск — это RAM-диск, который загружается загрузчиком системы (`lilo`, `grub`, `loadlin`) и монтируется как корневая файловая система перед нормальной загрузкой. Он используется для загрузки модулей перед монтированием нормальной корневой системы. Включите эту опцию. А если вы создаете загрузочный диск, то ее отключение недопустимо вообще.

Support for Large Block Devices

Поддержка больших блочных устройств — с размером более 2Тб. Отключайте эту опцию — жестких дисков на 2Тб в ближайшее время в продаже не предвидится.

ATA/ATAPI/MFM/RLL support

В этом подразделе вы можете включить/выключить поддержку ATA-устройств. Тут уж смотрите сами — какие устройства у вас есть и какие вы планируете использовать. Ненужное — сразу отключать — нечего память забивать. Что не нужно, а что нужно? Обыкновенный CDROM есть у всех. Даже если у вас в данный момент его нет, вы рано или поздно его подключите к своему компьютеру. Не будете же вы из-за этого перекомпилировать ядро? Поэтому опция **Include IDE/ATAPI CDROM support (BLK_DEV_IDECD)** относится к категории нужных. А вот поддержка Silicon Image chipset — совершенно не нужна, да еще и встроена в ядро по умолчанию. Поддержку всех ATA-устройств следует отключить, только если у вас сервер и все устройства — SCSI (только не забудьте SCSI включить!!!)

SCSI device support

Поддержка SCSI-устройств. Все комментарии аналогичны предыдущему пункту — все ненужное отключаем, оставляем самое необходимое. Вот почему мне нравится Linux — можно явно указать, что мне нужно, а что — нет. Ни в одной Windows такое сделать нельзя. Можно, конечно, удалить всю базу с драйверами, но ...

Old CD-ROM drivers (not SCSI, not IDE)

В этом разделе вы можете включить поддержку старых (я бы сказал древних) приводов CDROM, но, скорее всего, это вам не нужно, поэтому смело отключайте целый подраздел — сэкономим место на диске.

Multi-device support (RAID and LVM)

Поддержка RAID-массивов и LVM-томов (Logical volume manager). Если планируете использовать RAID, включите некоторые опции из этого раздела. Вы можете указать, какие уровни RAID вам нужны, а какие — нет.

IEEE 1394 (FireWire) support (EXPERIMENTAL)

Поддержка последовательной высокоскоростной шины IEEE 1394. Если у вас есть IEEE-адаптер, включите эту опцию. Внимание: поддержка IEEE — экспериментальна (для ядра 2.6)!

31.2.9. Networking support

Это довольно большой подраздел раздела **Device Drivers**, в котором можно включить как саму поддержку сети, так и отдельных ее компонентов. Прежде всего нужно сказать, что поддержка сети нужна, даже если у вас нет сетевой платы или других сетевых устройств. Функции печати, а также графическая подсистема X Window требуют поддержки сети, а это значит, что если сеть у вас отключена, вы не сможете ни документ распечатать, ни KDE запустить. Мрачная перспектива, не правда ли?

Включив сеть (Network support), вы увидите два подраздела:

- ♦ **Networking options** — параметры сети, например, поддержка различных протоколов
- ♦ **Network device support** — поддержка различных сетевых устройств. Комментировать данный раздел не стану, поскольку вам и только вам лучше всех известно, какое оборудование вы используете.

Сетевых опций довольно много, поэтому для их установки воспользуемся таблицей 31.3.

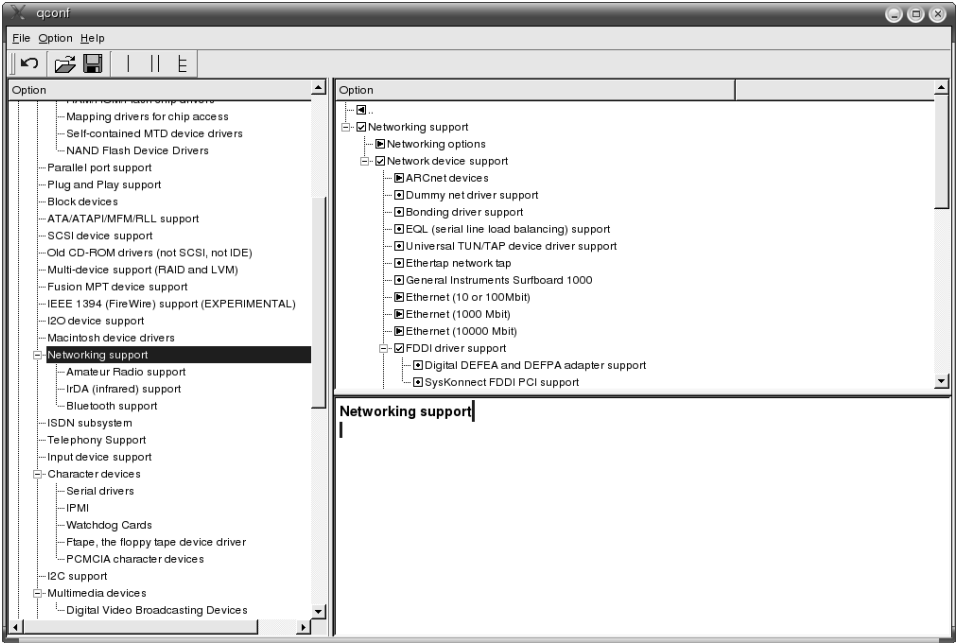


Рис. 31.5. Поддержка сети

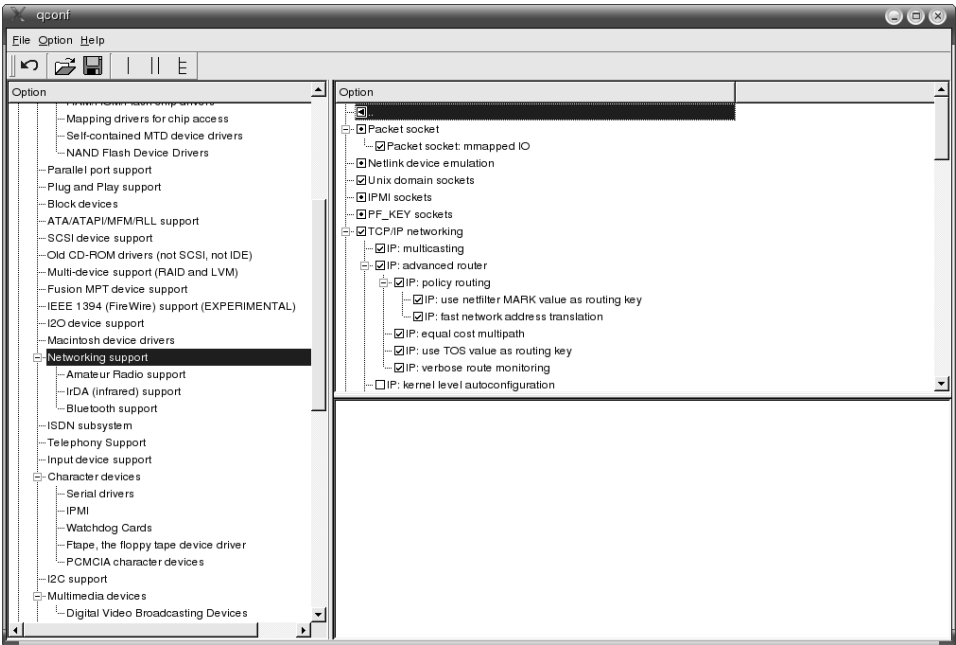


Рис. 31.6. Опции сети

Глава 31. Технология настройки и компилирования своего ядра

Опции сети

Таблица 31.3

Опция	Описание
Netlink device emulation	Опция обратной совместимости. Скоро будет удалена, но сейчас она нужна
Unix domain sockets	Поддержка UNIX-сокетов. Не отключайте эту опцию
IPMI sockets	Поддержка IPMI-сокетов. Обычно не нужна
PF_KEY sockets	Требуется для IPsec, поэтому лучше не отключать или включить в виде модуля
TCP/IP networking	Поддержка TCP/IP обязательно должна быть включена!!!
IP: multicasting	Позволяет адресовать сразу несколько компьютеров. Данная опция полезна, если вы используете MBONE или другую магистраль для широковещательной передачи аудио- и видеoinформации
IP: advanced router	Включите, если предполагаете использовать данный компьютер в виде маршрутизатора, а заодно нужно включить и все ее компоненты, например, IP: policy routing — довольно интересная функция маршрутизатора. В двух словах ее не опишешь, а прочитать о ней можно по адресу http://www.compendium.com.ar/policy-routing.txt
IP: kernel level autoconfiguration	Включает автоматическую конфигурацию IP-адреса сетевых устройств и таблицы маршрутизации во время загрузки ядра на основании информации, переданной ядру в командной строке или по протоколам BOOTP или RARP. Данную опцию имеет смысл включать только на бездисковых машинах, поскольку на обычных системах конфигурация сети задается в сценариях инициализации
IP: tunnelling	Данная опция понадобится вам, если вы будете настраивать виртуальную частную сеть — VPN (Virtual Private Network)
IP: GRE tunnels over IP	Данная опция полезна при использовании маршрутизаторов Cisco. Для ее работы необходимо включить предыдущую опцию
IP: multicast routing	Данная опция позволяет настроить работу маршрутизатора так, чтобы он отправлял IP-пакеты по нескольким адресам. Очень полезно для широковещания аудио/видео информации по Интернету
IP: ARP daemon support (EXPERIMENTAL)	Поддержка ARP-демона. Можно включить на маршрутизаторе/шлюзе небольшой сети
IP: TCP syncookie support (disabled per default)	Вот эту опцию я бы включил из соображений безопасности, она предотвратит вашу систему от так называемых SYN-наводнений, точнее, сообщит вам адрес атакующего хоста
IP: AH transformation	Поддержка AH-преобразования для IPsec. Если не уверены в том, что вы делаете, не отключайте эту опцию!
IP: ESP transformation	Поддержка ESP-преобразования для IPsec. Если не уверены в том, что вы делаете, не отключайте эту опцию!
IP: IPComp transformation	Поддержка IPComp-преобразования (сжатие данных, описано в RFC3173) для IPsec. Если не уверены в том, что вы делаете, не отключайте эту опцию!
IP virtual server support (EXPERIMENTAL)	Включение данной опции позволит вам построить виртуальный сервер, который будет использовать ресурсы нескольких физических серверов. Попросту говоря, данная опция позволяет собрать кластер. Раньше для создания кластеров использовались программные продукты сторонних разработчиков, а сейчас поддержка кластеров встроена в ядро. Если заинтересовались, посетите сайт http://www.linuxvirtualserver.org/
The IPv6 protocol (EXPERIMENTAL)	Поддержка протокола IPv6. Пока он практически не используется, поэтому можно эту опцию смело отключить
DECnet Support	Данную опцию на просторах бывшего СНГ вряд ли кто-то будет включать, точнее, вряд ли она кому-то нужна

Таблица 31.3 (продолжение)

Опция	Описание
802.1d Ethernet Bridging	Если вы включите эту опцию, ваш компьютер превратится в Ethernet-мост, который будет соединять различные сегменты вашей локальной сети
Network packet filtering (replaces ipchains)	Поддержка нового поколения бастiona Netfilter, пришедшего на замену IPChains. Если вы настраиваете маршрутизатор, включите эту опцию. Для обыкновенных систем ее нужно выключить
IPsec user configuration interface	Поддержка IPSec, если не уверены, просто включите эту опцию
Asynchronous Transfer Mode (ATM)	Поддержка ATM
The IPX protocol (IPX)	Поддержка протокола IPX (компания Novell)
Appletalk protocol support	Поддержка протокола компании Apple. Если в вашей сети есть хотя бы один Macintosh, включите эту опцию
CCITT X.25 Packet Layer (EXPERIMENTAL) (X25), LAPB Data Link Driver (EXPERIMENTAL) (LAPB)	Данные опции нужно включать, если вам это действительно необходимо. Если вы не знаете, что это такое, лучше их не трогать!
WAN router	Данная опция превращает ваш компьютер в маршрутизатор глобальной сети. Обычные маршрутизаторы не требуют включения этой опции. В случае включения данной опции в виде модуля модуль будет называться wanrouter
Fast switching	Перед включением этой опции настоятельно рекомендую прочитать документацию. Данная опция выбирает самый быстрый сетевой интерфейс для передачи данных. Внимание! Эта опция несовместима с опцией Network packet filtering
Forwarding between high speed interfaces	Не включайте эту опцию!!!

QoS and/or fair queueing

В этом подразделе можно включить поддержку QoS (Quality of Service).

IrDA (infrared) support

Поддержка IrDA-устройств.

Bluetooth support

Поддержка Bluetooth. Обычно требует включения на современных ноутбуках.

На этом раздел Network support закончился!

ISDN subsystem

Поддержка технологии ISDN (Integrated Services Digital Networks, во Франции — RNIS). Технология ISDN постепенно уходит в прошлое — вместо нее используется ADSL. Если у вас есть возможность перейти на ADSL, сделайте это.

Telephony Support

Если у вас есть специальная карта, позволяющая подключить обыкновенный телефон для использования голосовых IP-приложений, включите поддержку телефонии. Вам не нужно включать поддержку телефонии для использования обычного модема.

Input device support

Поддержка различных устройств ввода — джойстиков, мышей, тачскринов (touchscreens), клавиатур.

Character devices

Поддержка символьных устройств, например, стримеров.

Multimedia devices

Поддержка TV- и радио-тюнеров.

Graphics support

Поддержка графических адаптеров, выберите самое необходимое, а остальное — отключите.

Sound

Поддержка систем ALSA (Advanced Linux Sound Architecture) и OSS (Open Sound System). Тут же драйверы звуковых плат.

USB support

Поддержка USB.

31.2.10. Filesystems

В разделе Filesystems вы можете включить поддержку следующих файловых систем:

- ♦ **Second extended fs (ext2)** — до недавнего времени была основной файловой системой Linux.
- ♦ **Ext3 journalling file system** — журналируемая версия ext2, используется многими дистрибутивами как основная файловая система.
- ♦ **Reiserfs** — файловая система Reiser.
- ♦ **JFS filesystem** — файловая система JFS.
- ♦ **XFS** — файловая система XFS.
- ♦ **Minix fs** — файловая система Minix.
- ♦ **CD-ROM/DVD Filesystems ISO 9660** — файловая система, используемая для записи информации на CDROM.

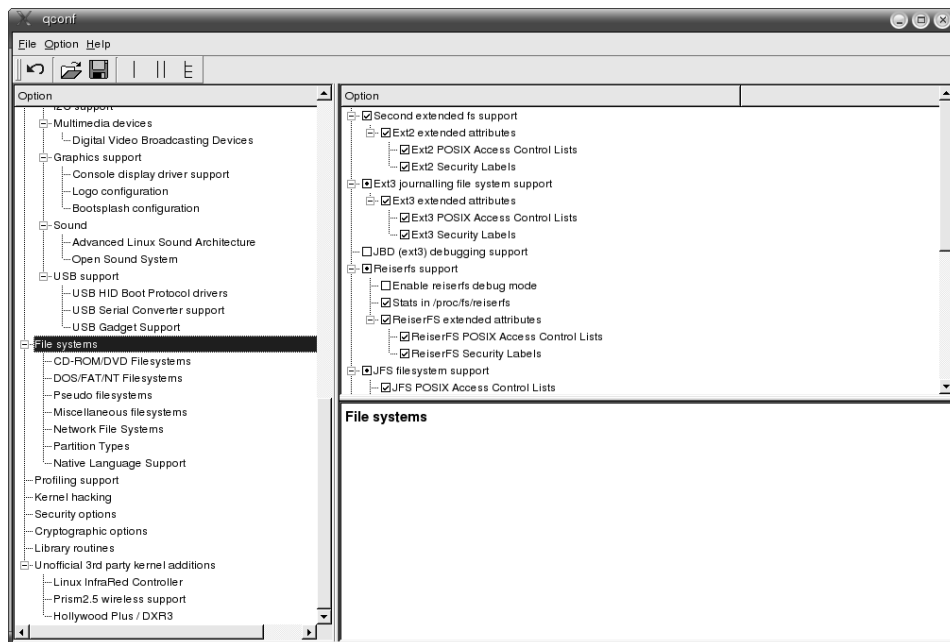


Рис. 31.7. Файловые системы

Что включить, а что выключить? Первые две файловые системы, а также файловую систему ISO 9660 нужно включить обязательно. Думаю, не нужно объяснять почему. Файловую систему Minix можно сразу отключить — она давно устарела и не используется. Файловые системы Reiser, JFS, XFS относятся к разряду новых, но редко используемых. В принципе, их нужно включить — вдруг кто-то принесет винчестер, на котором разделы будут содержать одну из этих файловых систем — ведь они же современные. Или просто вы захотите поэкспериментировать и отформатировать раздел в одной из этих систем.

Не забудьте включить средство автоматического монтирования сменных носителей — **automounter**, особенно если вы часто будете просто работать за этим компьютером! А вот поддержка квот (Quota support) окажется полезной, если вы настраиваете сервер.

В подразделе **DOS/FAT/NT Filesystems** вы можете включить поддержку следующих систем:

- ♦ Файловая система MS DOS — включить ее нужно обязательно — на просторах бывшего СНГ очень часто встречаются дискеты, записанные в этой файловой системе.
- ♦ VFAT (Windows-95) — это основная файловая система операционных систем Windows 95 и 98.

- ♦ NTFS — файловая система ОС Windows NT, 2000, XP. Здесь же можно включить поддержку записи на раздел NTFS, которая по умолчанию отключена.

В разделе **Pseudo filesystems** вы можете включить так называемые псевдо-системы — файловые системы `/proc` и `/dev`.

В разделе **Miscellaneous filesystems** находятся опции включения поддержки других, редко используемых файловых систем, например, HPFS (High Performance File System), которая используется по умолчанию ОС IBM OS/2.

Включить поддержку файловых систем NFS и SMB (используется для мониторинга удаленных Windows-разделов, читайте «общих дисков и папок») можно в разделе **Network File Systems**.

Раздел **Native Language Support** позволяет включить поддержку различных кодировок, в которых могут быть закодированы имена файлов. Например, если вы отключите кодировку `cp-1251`, при просмотре содержимого Windows-раздела вы увидите иероглифы вместо русских букв.

31.2.11. Kernel hacking

В этом разделе для вас найдутся две полезные опции, даже если вы не занимаетесь разработкой модулей ядра Linux. Опция `Prefer small over fast code` позволяет сделать ядро более маленьким, но более медленным. Маленький, но медленный код может пригодиться для создания загрузочной дискеты — там важен каждый байт. Вторую опцию `Kernel debugging` также можно отключить, если вы создаете системную дискету.

31.2.12. Cryptographic options

Различные опции, касающиеся криптографии.

31.2.13. Library routines

Вот этот раздел я так и не понял, для чего он предназначен. Он просто информирует о том, что ядро поддерживает некоторые библиотечные функции, но ничего включить, как и отключить — нельзя.

31.2.14. Unofficial 3rd party kernel additions

Различные вспомогательные опции. Не знаю, как вы, но для себя в этом разделе я не нашел ничего интересного.

31.3. Компилирование ядра

Теперь, когда все устройства сконфигурированы, нужно сохранить файл конфигурации ядра и перейти непосредственно к этапу компилирования ядра.

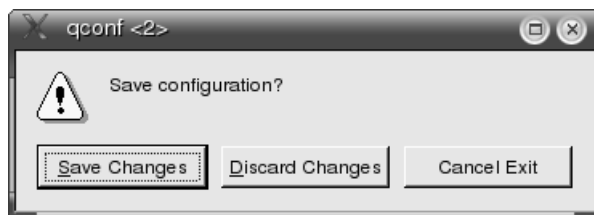


Рис. 31.8. Сохранение конфигурационного файла ядра

Введите команду:

```
# make dep
```

После завершения ее работы необходимо ввести команду:

```
# make bzImage
```

Если исходники ядра и компилятор установлены корректно, то примерно минут через 20 (это зависит от версии ядра и от быстродействия вашей системы) вы получите откомпилированное ядро. Обычно оно помещается в каталог `/usr/src/linux/arch/i386/boot`

Теперь следует откомпилировать модули, которые будут использоваться ядром:

```
# make modules
```

И установить их:

```
# make modules_install
```

Перед установкой модулей сделайте резервную копию модулей старого ядра (каталог `/lib/modules`). Теперь можно ввести команду:

```
# make install
```

Однако для установки только что созданного ядра я не рекомендую этого делать. Сначала нужно протестировать ваше ядро. С этой целью откройте в любом редакторе файл `/etc/lilo.conf` :

```
# vi /etc/lilo.conf
```

Добавьте в него следующие строки (более подробно файл `/etc/lilo.conf` обсуждался в одной из первых глав книги):

```
image=/usr/src/linux/arch/i386/boot/bzImage
label=my_linux
```

```
root=/dev/hda5
append=" mem=128M"
read-only
```

Потом введите команду:

```
# lilo
```

Теперь перезагрузите систему:

```
# reboot
```

Попробуйте загрузить ядро. В случае возникновения ошибок вы всегда сможете загрузить старую версию.

31.4. Пакет modutils

Ядро Linux содержит только самый необходимый код — чтобы только загрузиться. Поддержку остальных устройств обеспечивают модули, которые встраиваются в ядро при загрузке системы. В принципе, можно вкомпилировать в ядро все необходимые модули. Тогда у нас будет система, не использующая модули, но мы сейчас не будем говорить о таких системах. С помощью пакета **modutils**, который будет рассмотрен ниже, вы можете добавить нужный вам модуль в ядро во время работы системы. При этом перезагружать систему не нужно — устройство начнет работать сразу же после загрузки модуля.

Раньше, в первых версиях ядра Linux, механизм работы с модулями не был предусмотрен, и ядра тех времен содержали в себе код драйверов для ВСЕХ поддерживаемых устройств.

Такое решение нельзя назвать рациональным. Мы не можем предусмотреть, какие устройства будут установлены у конечного пользователя, даже если включить в состав ядра драйверы всех возможных устройств.

Предположим, что у нашего пользователя установлена звуковая плата Yamaha, а наше ядро «знает» еще с десятков звуковых плат. Получится, что один код будет работать всегда, а остальные десять драйверов будут просто использовать оперативную память. Кстати, об оперативной памяти: вы представляете размер ядра, если оно будет содержать драйверы *всех* устройств?

Исходя из всего этого, разработчики ядра Linux «изобрели» механизм динамически загружаемых модулей. Модули хранятся на диске в виде объектных файлов (*.o). При необходимости ядро загружает необходимый ему модуль. Откуда ядро знает, какой модуль нужно загружать, а какой — нет? Список модулей, а также передаваемые им параметры хранятся в файле `/etc/modules.conf` (или `/etc/conf.modules` — в зависимости от дистрибутива и версии ядра). Вот пример этого файла:

Листинг 31.1. Файл /etc/modules.conf (Linux Red Hat 7.3)

```
alias parport_lowlevel parport_pc
alias usb-controller usb-uhci
alias sound-slot-0 i810_audio
post-install sound-slot-0 /bin/aumix-minimal -f /etc/.aumixrc -L >/dev/null 2>&1
pre-remove sound-slot-0 /bin/aumix-minimal -f /etc/.aumixrc -S >/dev/null 2>&1
```

Подробнее о формате файла /etc/modules.conf вы можете прочитать в справочной системе, введя команду `man modules.conf`.

При загрузке система читает этот файл и загружает указанные в нем модули. В нашем случае загружается только модуль `i810_audio`, поскольку команда `alias` не загружает модуль, а только устанавливает для него псевдоним.



Примечание

Загрузка модулей из файла `modules.conf` обеспечивается программой `modprobe`, которая вызывается из сценария инициализации системы `/etc/rc.d/rc.sysinit`.

Во время работы системы мы сами можем загрузить нужный нам модуль. Для этого нужно использовать программу **insmod**, входящую в состав пакета **modutils**. Использовать программу очень просто:

```
insmod <имя_файла_модуля>
```

Чтобы использовать программу **insmod**, вы должны обладать привилегиями суперпользователя — пользователя `root`. Просмотреть список загруженных модулей можно с помощью команды **lsmod**. Вот вывод этой программы:

Module	Size	Used by	Not tainted
autofs	12164	0 (autoclean)	(unused)
nls_koi8-r	4576	2 (autoclean)	
nls_cp866	4576	2 (autoclean)	
vfat	12092	2 (autoclean)	
fat	37400	0 (autoclean)	[vfat]
usb-uhci	24484	0 (unused)	
usbcore	73152	1 [usb-uhci]	

Некоторые модули загружаются не из файла `/etc/modules.conf`. Например, модули файловых систем загружаются по мере необходимости — при монтировании определенной файловой системы загружается нужный модуль, если, конечно, он есть. Модули `nls_koi8-r` и `nls_cp866` за-

ружаются также при монтировании файловой системы, если указаны опции монтирования `iocharset=koi8-r,codepage=866`.

Выгрузить модуль предельно просто:

```
rmmod имя_модуля
```

Программа **modinfo** позволяет просмотреть информацию о модуле:

```
modinfo usbcore
filename:      /lib/modules/2.4.18-3/kernel/drivers/usb/usbcore.o
description:  <none>
author:       <none>
license:      "GPL"
```



Примечание

Программы **insmod**, **rmmod**, **lsmod** и **modinfo** входят в состав **modutils**. Для использования любой этой программы необходимы права пользователя **root**.

Полезные команды и программы для работы в Linux. Создание RPM-пакетов

Цель данной главы — вспомнить «хорошо забытое старое». Из современных пользователей ОС Linux мало кто пользовался или вообще знает о существовании таких программ как **mail**, **fastmail**, **locate**, **which**. Эти небольшие программы позволяют сделать многие операции намного быстрее, чем программы, обладающие графическим интерфейсом и предназначенные для работы в графической системе X Window. К тому же эти программы не так чувствительны к системным ресурсам компьютера. Если у вас, например, Intel Pentium 166 и 32 Мб ОЗУ, то после того, как вы раз попробуете поработать с программой **pine**, вам уже никогда не захочется запускать программу **netscape** с параметром «-mail» для чтения почты.

Тем не менее все это не имеет прямого отношения к теме книги, поэтому данная глава размещена на сайте издательства:

www.nit.com.ru

откуда вы ее можете скачать.

Интерпретатор команд **bash**. Написание сценариев

33.1. Введение в **bash**

Интерпретатор команд — это программа, выполняющая команды пользователя. Стандартным интерпретатором (или оболочкой) является **bash** (Bourne Again Shell). Достаточно распространенными также являются следующие интерпретаторы: **sh**, **ash**, **bsh**, **tcsh**, **csh**, **zsh**. Список установленных в вашей системе оболочек находится в файле `/etc/shells`. Команды оболочки можно вводить в командной строке, а можно оформить в виде сценария.

Сценарий — это файл, содержащий команды оболочки. Создайте обыкновенный текстовый файл и сделайте его исполнимым. Система выполнит указанную последовательность команд. Для того, чтобы система узнала, какую оболочку нужно использовать, первая строка сценария должна содержать полное имя сценария.

Например:

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

Между символами **#** и **!** не должно быть пробелов. Для обработки сценария вы можете использовать любую программу (естественно, она должна понимать синтаксис файла), а не только указанную в файле `/etc/shells`. Например, вы можете написать:

```
#!/usr/bin/my_proga
```

Оболочка при этом запустит программу `/usr/bin/my_proga` и передаст имя файла сценария в качестве параметра. Если вы напишете:

```
#!/usr/bin/my_proga -f
```

то оболочка выполнит команду:

```
#!/usr/bin/my_proga -f <имя_сценария>
```

Создадим небольшой сценарий, который будет очищать экран и выводить на него ваше имя. Имя можно передать как параметр. Назовем наш сценарий **test**:

```
#!/bin/bash
# Это комментарий
clear
echo $1
```

Теперь рассмотрим все по порядку. С первой строкой, я думаю, все ясно. Вторая строка очищает экран. Третья строка выводит информацию, которая передана сценарию как первый (1) параметр. Запустите сценарий следующим образом:

```
./test Vasya Pupkin
```

На экране будет напечатан первый параметр, то есть слово *Vasya*. Вы можете немного изменить сценарий, чтобы он выводил оба параметра:

```
echo $1 $2
```

Если вы хотите передать фамилию, имя и отчество, то нужно использовать следующую команду:

```
echo "$1"
```

При этом не нужно явно указывать три параметра, просто интерпретатор не будет использовать пропуск для разделения параметров и все, что вы передадите ему, будет считаться одним параметром. При этом если вы запустите сценарий с параметром *Vasya Pupkin*, на экране увидите *Vasya Pupkin*. А если укажете параметр *Ivanov Ivan Ivanovich*, то сценарий так и напечатает *Ivanov Ivan Ivanovich*.

33.2. Каналы и списки

Материал этого и следующего пункта дополняет гл. 5, в которой рассматривалось перенаправление ввода вывода. Поэтому я не буду подробно описывать сам механизм этих процессов, а ограничусь лишь несколькими примерами, чтобы напомнить вам гл. 5.

С помощью каналов вы можете перенаправить стандартный вывод одной программы на стандартный ввод другой. Например:

```
cat /var/log/secure | less
ps -ax | grep "$UID" | less
```

В первом случае стандартный вывод команды **cat** (содержимое файла */var/log/secure*) перенаправляется на стандартный ввод программы **less**, которая обеспечивает постраничный вывод информации.

Вторая строка выводит список всех процессов, принадлежащих данному пользователю. Первая команда `ps-ax` выводит список всех запущенных в системе процессов, вторая (`grep «$UID»`) ищет фрагмент текста, содержащий идентификатор пользователя (UID) и выводит результат на стандартный вывод, то есть на стандартный ввод команды `less`. `$UID` является переменной окружения, которая содержит идентификатор пользователя. О переменных окружения поговорим немного позже.

Интерпретатор позволяет указывать списки команд в командной строке. Например:

```
lpr file.txt; lpq
```

При этом сначала выполняется команда постановки задания в очередь печати, а потом проверяется состояние принтера. Теперь более сложный пример:

```
ps -ax | head -n 1; ps -ax | grep httpd
```

Сначала выполняется команда `ps` для печати заголовка таблицы, а потом — для вывода информации о демоне `httpd`.

Можно использовать операции конъюнкции и дизъюнкции, например:

```
command1 && command2  
command3 || command4
```

Команда `command2` будет выполнена в случае успешного завершения команды `command1` (возвратный код равен 0). Команда `command4` будет выполнена, если код возврата команды `command3` не равен 0. Самый простой пример — создание и изменение каталога:

```
mkdir mydir && cd mydir
```

Обычно интерпретатор `bash` выполняет команды в синхронном режиме, то есть после запуска программы ожидает ее завершения. Однако можно запустить программу и в асинхронном режиме, то есть без ожидания ее завершения. Для этого нужно использовать символ «&» в конце команды, например:

```
program &
```

33.3. Перенаправление ввода/вывода

Перенаправление ввода/вывода также уже рассматривалось в гл. 5, поэтому я лишь напомним общий формат команд:

```
команда > (>>) файл  
список > (>>) файл
```

Как вы уже знаете, при использовании одного знака **больше** файл, в который переназначен вывод, будет перезаписан, а при использовании двой-

ного **больше** информация будет добавлена в конец файла. При использовании списка команд список нужно взять в фигурные скобки:

```
{date; free; who; } > logfile
```

Перенаправление ввода/вывода может быть использовано и в обратном направлении. Например, для печати списка URL достаточно выполнить команды:

```
lpr << URL
http://www.linux.ru
http://linux.ru.net
http://www.linux.org
URL
```

33.4. Подболочки

При написании сценариев вы можете использовать **механизм подболочек**. Если вы знакомы с каким-нибудь языком программирования, то должны знать об области распространения действия переменных. Существуют глобальные и локальные переменные. Первые действуют во всей программе, а вторые только внутри определенного блока, например, функции. Если в локальной функции определена переменная с таким же именем, что и одна из глобальных переменных, в этом локальном блоке будет использоваться значение локальной переменной. Далее приведен простейший пример программы, использующей локальные и глобальные переменные на языке Pascal:

```
Program Test;
Var A : integer; {глобальная переменная}
procedure GetLocal;
var A : integer; {локальная переменная}
begin
  A:=10;
  writeln(A);
end;
begin
  A:=7;
  writeln(A);
  GetLocal(A);
  writeln(A);
end;
```

При запуске программа выведет на экран:

```
7
10
7
```

В **bash** наблюдается нечто подобное. Этот блок называется подболочкой. Если список команд заключен в фигурные скобки, то он выполняется в текущей оболочке, а если в обыкновенные, то в подболочке. Итак, рассмотрим выполнение почти аналогичных сценариев:

```
#!/bin/bash
# Сценарий 1
NUM="one"; (NUM="two"; echo $NUM);
echo $NUM
#!/bin/bash
# Сценарий 2
NUM="one"; {NUM="two"; echo $NUM;};
echo $NUM
```

Сценарий 1 выведет на экран следующую информацию:

```
two
one
```

А сценарий 2:

```
two
two
```

С помощью механизма подболочек вы можете создавать более гибкие сценарии. С его помощью, например, можно временно изменять рабочий каталог:

```
pwd; (cd /tmp; pwd;); pwd
```

Сценарий выведет на экран:

```
/home/user
/tmp
/home/user
```

33.5. Переменные и массивы

Пример простейшей переменной мы уже рассмотрели выше. Переменные в **bash** не нуждаются в предварительном описании, как в других языках, например, в том же Pascal. Все переменные в **bash** — текстовые. Имя переменной должно начинаться с буквы и может состоять из латинских букв, цифр, знака подчеркивания. Нельзя определять функцию и переменную с одинаковыми именами. Чтобы воспользоваться значением переменной, нужно использовать символ доллара перед именем переменной:

```
VAR="my var"
echo $VAR
```

Как я уже говорил, все переменные текстовые. Например, когда вы присваиваете переменной значение **VAR=13**, то это будет не числовое значение, а строка из двух символов. Если присваиваемое значение имеет пробелы, его нужно взять в кавычки:

```
VAR="value 1"
```

Присвоить значение переменной можно с помощью встроенной команды **read**:

```
echo -n "Enter value: "
read x
echo $x
```

Параметр **-n** команды **echo** не выводит символ новой строки в конце сообщения «*Enter value:*», то есть не переводит строку. Команда **read** читает значение, введенное пользователем с клавиатуры, и записывает его в переменную **x**. Последняя команда выводит только что введенное значение. При использовании команды **read** можно указывать несколько переменных:

```
read a b c
```

Пользователь должен ввести значения переменных, разделяя их пробелами. Для окончания ввода нужно нажать «Enter». Если введено меньше значений, чем нужно, оставшимся переменным будет присвоена пустая строка. А если больше, то лишние значения будут проигнорированы.

Интерпретатор **bash** использует следующие метасимволы, имеющие для него особое значение:

```
* ? ; & ( ) | ^ < > <возврат_каретки> <табуляция> <пробел>
```

Для того, чтобы использовать эти символы как они есть, нужно их цитировать с помощью символа ****. Например, символ перевода строки можно цитировать как **\n**, символ табуляции как **\t**, а символ вопроса как **\?**

Особое значение при присваивании переменным значений имеют кавычки. Все символы, заключенные в одинарные кавычки **' '** представляют самих себя. Между двойными кавычками **« »** выполняются команды и подстановки значений. Символы **«\»**, **«,»**, **« ' »**, **« \$ »** могут цитироваться с помощью обратной наклонной черты: ****, **\\\$**, **\'**

Для обозначения параметров командной строки используются специальные переменные, указанные в табл. 33.1.

Специальные переменные

Таблица 33.1

Название переменной	Значение
\$0	Имя выполняемой команды. Для сценария — путь, указанный при его вызове. Если вы знакомы с языком программирования Pascal, эта переменная должна вам напоминать вызов функции ParamStr(0)
\$n (где n — номер параметра, \$1, \$2, ...)	Обращение к параметру с номером n. Для Pascal — это вызов функции ParamStr(n)
\$#	Число параметров, которые были указаны при вызове сценария. Аналогична вызову функции ParamCount в Pascal

Таблица 33.1 (продолжение)

Название переменной	Значение
\$*	Все параметры, заключенные в кавычки: "\$1 \$2 ..."
\$@	Все параметры, каждый из которых заключен в кавычки "\$1" "\$2" ...
\$?	Код завершения последней команды
\$\$	Номер текущего процесса (PID)
\$_	Номер (PID) последнего асинхронного процесса (команды, выполненной в фоновом режиме)

Пример:

```
echo "Все параметры: "
echo $*
```

Интерпретатор **bash** поддерживает одномерные массивы с неограниченным числом элементов. В других оболочках существуют определенные ограничения на массивы, например, в **ksh** максимальное число элементов массива ограничено 1024-мя элементами.

Присвоить значение элементу массива можно с помощью такой конструкции:

```
Имя_массива[индекс]=значение
```

Например:

```
A[1]=23
A[3]=54
A[0]=77
```

Нумерация элементов начинается с нуля. Тип элементов массива, как и тип переменных, текстовый. Присвоить значение элементам массива можно также с помощью инструкции **set**. Например, выражение:

```
set -A array 3 56 77 12
```

аналогично выражениям:

```
array[0]=3
array[1]=56
array[2]=77
array[3]=12
```

Вышеприведенные способы инициализации массивов могут применяться как в **bash**, так и в **ksh**, а также и в других оболочках. Но, тем не менее, существует еще один способ, который работает только в **bash**:

```
array = ([0]=3 [3]=12 [2]=77 [1]=56)
```

Обратиться к значению элемента можно следующим образом:

```
${имя_массива[индекс]}
```

Например, вывести значение первого (нулевого) элемента массива можно так:

```
echo ${array[0]}
```

Обратиться ко всем элементам массива сразу можно с помощью одного из выражений:

```
${имя_массива[*]} или ${имя_массива[@]}
```

Если значение хотя бы одного элемента массива может содержать пробелы, используйте второе выражение:

```
echo ${array[@]}
```

Особое значение имеют переменные окружения. У каждого процесса есть своя среда — множество доступных ему переменных. Обыкновенные переменные доступны только из локального процесса. Чтобы использовать их значения в порождаемых процессах, нужно использовать встроенную команду **export**. После того, как вы экспортируете переменные, они будут доступны всем порожденным процессам.

Выражение **export**:

```
export имя
export имя=значение
export имя имя имя ...
export имя=значение имя имя=значение ...
```

Как видите, можно экспортировать уже инициализированную переменную (которой уже присвоено значение), а можно выполнить инициализацию переменной непосредственно при экспорте. Экспортировать можно как одну переменную, так и целый список.

Для деактивизации переменной используется команда **unset**:

```
unset имя
```

Каждому процессу доступны переменные оболочки, приведенные в табл. 33.2.

Пример:

```
echo "$HOME"
```

Переменные оболочки

Таблица 33.2

Название переменной	Значение
PWD	Текущий каталог
UID	Идентификатор пользователя, запустившего сценарий
REPLY	Последняя строка, введенная с помощью команды read
RANDOM	Случайное число в диапазоне от 0 до 32767
SECONDS	Число секунд, прошедшее с момента запуска оболочки

Таблица 33.2 (продолжение)

Название переменной	Значение
IFS (Internal Filed Separator)	Внутренний разделитель полей. Используется синтаксическим анализатором и командой <code>read</code> для разделения строки на слова. По умолчанию его значение равно " <code>\t\n</code> ", где: « <code> </code> » — пробел; « <code>\t</code> » — символ табуляции; « <code>\n</code> » — символ новой строки.
HOME	Домашний каталог
PATH	Путь вызова
LOGNAME	Имя пользователя, которое использовалось для входа в систему
MAIL	Имя файла, в который поступает электронная почта
SHELL	Имя интерпретатора команд
TERM	Тип терминала пользователя

33.6. Подстановка команд и арифметических выражений

В гл. 13 мы уже сталкивались с подстановкой команд. Тогда переменной `DT` присваивался результат выполнения команды `date`:

```
DT='date'
```

Как я уже писал, при подстановке команд нужно использовать обратные одинарные кавычки (они расположены под символом тильды на клавиатуре). Подставлять можно не только одну команду, а целые списки команд:

```
USERS='who | wd -l'
UP='date; uptime'
```

В первом случае мы получим количество пользователей, работающих в системе, а во втором — последовательно записанные результаты выполнения команд `date` и `uptime`.

Подставлять результаты выполнения можно не только в переменные, а и в другие команды, например

```
grep 'id -un' /etc/passwd
```

Данная команда ищет в файле `/etc/passwd` вхождение результата выполнения команды `id -un`.

Подстановка арифметических выражений осуществляется с помощью конструкции:

```
$ ( (выражение) )
```

Например:

```
A = $(((10+5)/2))  
echo $A
```

При этом на экране вы увидите число 7, а не 7.5, потому что используется целочисленное вычисление.

Пример. Количество часов, прошедшее с момента запуска оболочки:

```
hrs = $(($SECONDS/3600))
```

33.7. Управляющие структуры и циклы

33.7.1. Условный оператор. Конструкция if-fi

Синтаксис if-fi

Общий синтаксис конструкции **if-fi**:

```
if список1 then  
    список2  
elif список3 then  
    список4  
else  
    список5  
fi
```

Конструкция **if-fi** работает так же, как и в других языках программирования. Если *список1* (условие) истинный, выполняется *список2*, иначе выполняется *список3* и проверяется его истинность и т.д. Допускается неограниченная вложенность операторов **if**. Например:

```
if uuencode myfile myfile > myfile.uu; then  
    echo "Успешное конвертирование";  
else  
    echo "Ошибка";  
fi
```

Можно использовать сокращенный вариант:

```
if список1 then  
    список2  
fi
```

Например:

```
if [ $? -ne 0 ]; then echo "Ошибка. См. файл протокола"; fi;
```

Команда `test`

Вместо списка команд удобно использовать команду `test` или выражение [условие]. Например, следующие выражения аналогичны:

```
test -e /etc.passwd
[-e /etc/passwd]
```

И первое, и второе выражение проверяют существование файла `/etc/passwd`. Другие опции команды `test` представлены в табл. 33.3.

Опции команды `test`

Таблица 33.3

Опция	Возвращаемое значение и описание
-d файл	Истина, если файл существует и является каталогом
-e файл	Истина, если файл существует
-f файл	Истина, если файл существует и является простым файлом
-k файл	Истина, если файл существует и для него установлен бит односторонней операции
-L файл	Истина, если файл существует и является ссылкой
-r файл	Истина, если файл существует и доступен для чтения
-s файл	Истина, если файл существует и его размер больше 0
-x файл	Истина, если файл существует и является исполнимым
-w файл	Истина, если файл существует и доступен для записи
-o файл	Истина, если файл существует и принадлежит данному пользователю
-z строка	Истина, если длина строки равна 0
-n строка	Истина, если длина строки не равна 0

Команда `test` в случае успешного завершения возвращает значение истина, то есть 0 — успешное завершение. Если в скобках стоит непустое слово, `test` возвратит тоже 0, например:

```
[ word ]; echo $?
0
[]; echo $?
1
```

В первом случае возвращается истина (`true`), на экран выводится ноль — код удачного (безошибочного) завершения программы. Во втором случае на экран выводится единица — команда `test` возвратила значение ложь (`false`).

Сравнение строк

Сравнение строк осуществляется следующим образом: выражения `str1 = str2` или `str1 == str2` истинны, когда строки `str1` и `str2` равны. Обратите внимание: между двумя символами равно не должно быть пропусков!

Символ **!** инвертирует любое условие команды **test**, например выражение **str1 != str2** будет истинным, когда строки **str1** и **str2** не равны между собой. Символ **!** является символом логической операции NOT (отрицание). Кроме этого символа, можно использовать опции команды **-o** и **-a**, которые обозначают логические операции ИЛИ (OR) и И (AND). Например:

```
str="word"; export str; ["$str" -a -f /etc/passwd]; echo $?
0
str=""; export str; ["$str" -a -f /etc/passwd]; echo $?
1
```

В первом случае непустая строка **str** возвращает истину, опция **-f** возвращает также истину, потому что файл **/etc/passwd** существует всегда. Результат операции **И**: *истина И истина = истина*, поэтому на экране вы увидите 0.

Во втором случае пустая строка **str** возвратит ложь, а опция **-f** возвращает истину. Результат операции **И**: *ложь И истина = ложь*. Если вы забыли законы логики, освежите свои знания с помощью табл. 33.4.

Операция XOR — это **исключающее ИЛИ**. Данная операция не используется при создании сценариев с помощью интерпретатора **bash**.

Логические операции

Таблица 33.4

AND	True	False	OR	True	False	XOR	True	False
True	True	False	True	True	True	True	False	True
False	False	False	False	True	False	False	True	False

Сравнение целых чисел

Для сравнения целых чисел используются опции команды **test**, приведенные в табл. 33.5.

Интерпретатор **bash** воспринимает строки как целые числа. Если нужно обнулить строку, то это достигается таким присваиванием: **x=0**.

Пример.

```
x=124 ; export x ; [ 111 -lt "$x" ] ; echo $?
0
```

Поскольку 111 меньше, чем 124, на экране вы увидите 0 (истина).

Сравнение целых чисел Таблица 33.5

Опция	Описание
-eq	Равно
-ne	Не равно
-lt	Меньше
-gt	Больше
-le	Меньше или равно
-ge	Больше или равно

**Примечание**

Во всех примерах вы, наверное, заметили использование команды **export**. Это необходимо для того, чтобы порожденному процессу (не забывайте: **test** — это отдельная программа) переменная **x** была доступна.

Вложенные конструкции if-fi

Теперь, когда мы уже знакомы с конструкциями **test** и **if**, рассмотрим небольшой пример, демонстрирующий вложенность операторов **if** и использование команды **test**. Пример приведен в листинге 33.1.

Листинг 33.1. Пример вложенности операторов

```
echo -n "Какую оценку ты получил сегодня по программированию?"
read x
if [ $x = 5 ]
then echo "Отлично !"
elif [ $x = 4 ]
  then echo "Хорошо"
  elif [ $x = 3 ]
    then echo "Удовлетворительно"
    elif [ $x = 2 ]
      then echo "Надо бы пересдать"
      else echo "Как вообще можно было получить такую оцен-
ку???"
fi
```

Если вы введете 5, сценарий отобразит на экране слово «Отлично», при вводе 4 вы увидите слово «Хорошо» и так далее. Если вы введете 0, 1 или число больше пяти, вы увидите на экране последнюю фразу: «Как вообще можно было получить такую оценку???»

33.7.2. Конструкция выбора case-esac

Конструкция выбора (**case** — выбор) имеет следующий синтаксис:

```
case значение in
  шаблон1) список1 ;;
  ...
  шаблонN) списокN ;;
esac
```

Рассмотрим сценарий (см. листинг 33.2), аналогичный сценарию 33.1, но использующий конструкцию **case** вместо **if**.

Листинг 33.2. Пример использования оператора `case`

```
echo -n " Какую оценку ты получил сегодня по программированию? "
read x
case $x in
5) echo "Отлично !" ;;
4) echo "Хорошо" ;;
3) echo "Удовлетворительно" ;;
2) echo "Надо бы пересдать" ;;
*) echo "Как вообще можно было получить такую оценку???" ;;
esac
```

Работа сценария аналогична первому сценарию: при вводе оценок 2, 3, 4, 5 будут отображены соответствующие сообщения, а во всех остальных случаях — последнее сообщение.



Примечание

Структура оператора **case** больше напоминает структуру оператора **case** в языке Pascal, чем в языке C. Последняя строка выбора с шаблоном ***)** будет выбрана, когда не произойдет ни одного совпадения с ранее указанными шаблонами. Если же произошло совпадение с шаблоном **шаблонN**, то будет выполнен список **списокN**. После выполнения списка команд **списокN** будет произведен выход из структуры **case** — так же, как и в Pascal. В языке C наблюдается нечто другое: если будет обнаружено совпадение, скажем, с шаблоном 3, то будут выполнены последовательности операторов 3, 4, 5, ... N. Чтобы прервать выполнение блока **case** в языке C, нужно использовать оператор **break**. В **bash** же такого нет.

Если для одного списка команд нужно описать два или более шаблонов, используется символ **|** (OR).

```
case num in
1|2|3) echo "1 or 2 or 3" ;;
4|5) echo "4 or 5" ;;
*) echo "other num" ;;
esac
```

33.7.3. Циклы

Интерпретаторы **bash** и **ksh** поддерживают циклы **for**, **while**, **until**, **select**, а интерпретатор **sh** — только **for** и **while**.

Цикл **for**

Синтаксис цикла **for**:

```
for имя_переменной in список1
do
    список2
done
```

Простой пример:

```
for i in 1 2 3 4 5; do echo $i; done
```

На экране вы увидите:

```
1 2 3 4 5
```

Еще раз напомним, что любой список в **bash** должен заканчиваться точкой с запятой. Начинающие «программисты» делают много ошибок, связанных именно с этой особенностью списков. Пример использования: построчно вывести содержимое файла `/etc/passwd` мы можете с помощью такого цикла:

```
for str in `cat /etc/passwd`
do
    echo "$str";
done
```

Цикл **for** закончит свою работу, когда будет обработан последний элемент списка, в данном случае, когда на экран будет выведена последняя строка файла `/etc/passwd`

Цикл **while**

Синтаксис цикла **while**:

```
while список1
do
    список2
done
```

Цикл **while** будет выполняться, пока условие, заданное в списке *список1*, будет истинным. Поэтому цикл **while** иногда называют **циклом с истинным условием**. Например,

```
x=1
while [$x -lt 10]
do
    echo $x
    x=$((x + 1))
done
```

На экране вы увидите: 1 2 3 4 5 6 7 8 9

Когда переменная `x` примет значение 10, цикл завершит свою работу, так как программа **test** вернет значение *false* (`x` уже не меньше, а равен 10).

Цикл **until**

Цикл **until** (до) имеет похожую структуру, но выполняется несколько иначе:

```
until список1
do
    список2
done
```

Цикл **until** прекратит работу, когда условие, указанное в списке *список1*, станет истинным. Другими словами, он будет выполняться, пока это условие ложно. Цикл **while**, наоборот, выполняется, пока условие истинно. Лучше всего разница между этими циклами видна на примере (сравните листинги 33.3 и 33.4).

Листинг 33.3. Цикл while

```
x=1;
while ! [$x -ge 10]
do
echo $x
x = $(( $x + 1 ))
done
```

Листинг 33.4. Цикл until

```
x=1;
until [$x -ge 10]
do
echo $x
x = $(( $x + 1 ))
done
```

Циклы, приведенные в листингах 33.3 и 33.4, выведут одинаковую последовательность цифр на экран:

1 2 3 4 5 6 7 8 9 10

Цикл select

Рассмотрим еще один полезный цикл **select**, который позволяет создавать нумерованные пункты меню. Его конструкция такова:

```
select имя in список1
do
    список2
done
```

Пример:

```
echo "Выберите файл для просмотра"
select file in /home/den/temp/* QUIT
do
    if [-e $file]; then less $file
    else
        break
    done;
```

В моем временном каталоге `/home/den/temp` находится всего два файла — `file.txt`, `proto.txt`, поэтому на экране монитора будет отображено следующее:

Выберите файл для просмотра

- 1) `/home/den/temp/.`
- 2) `/home/den/temp/..`
- 3) `/home/den/temp/file.txt`
- 4) `/home/den/temp/proto.txt`
- 5) `QUIT`

Первые два файла — это ссылки на текущий и родительский каталоги. Пункты меню 3 и 4 — это файлы, которые можно выбрать для просмотра. `QUIT` — это последний элемент списка. При его выборе сработает оператор `break` и цикл завершится.

33.8. Подстановка переменных

Мы уже рассмотрели подстановку команд, сейчас рассмотрим подстановку переменных (см. табл. 33.6).

Подстановка переменных

Таблица 33.6

Конструкция	Описание
<code>\${переменная:-значение}</code>	Если переменная определена и не является пустой строкой, подставляется ее значение, иначе подставляется значение, указанное в конструкции. Реальное значение переменной при этом не изменяется
<code>\${переменная:=значение}</code>	Значение присваивается переменной, если она не определена или является пустой строкой
<code>\${переменная:?сообщение}</code>	Если переменная не определена или является пустой строкой, выводится указанное сообщение
<code>\${переменная:+значение}</code>	Если переменная инициализирована (определена), вместо нее используется указанное в конструкции значение. Реальное значение переменной не изменяется
<code>\${переменная}</code>	Если переменная определена, то подставляется ее значение. Скобки используются лишь тогда, когда после переменной стоит символ, который может «приклеиться» к имени переменной

Пример.

```
${1 :? "Не хватает параметра"}
```

Данное сообщение будет выведено, если сценарий будет запущен без параметров. Если указать хотя бы один параметр, сообщение не будет отображаться на экране.

33.9. Функции

Описание функции выглядит так:

```
имя() { список; }
```

Пример:

```
cdir()
{
# изменяем каталог
cd /
}
```

При выполнении функция не создает нового процесса, а выполняется в среде процесса, содержащего эту функцию. Аргументы функции можно передать ей как обыкновенные параметры при вызове сценария. Функции можно описывать в любом месте сценария, но вызов функции должен осуществляться только после ее описания. Возвращаясь к примеру, модифицируйте функцию:

```
#!/bin/bash
# файл fn
echo $$
cdir()
{
# изменяем каталог
echo "X=$X"
X=2
echo "Params $0 $# $1 $2"
echo "PID = $$"
return 0
cd $1
}
X=1
echo "X=$X"
cdir /etc # вызов функции "cd" с параметрами
echo $?
echo "X=$X"
```

На экране вы увидите примерно следующую информацию:

```
788
X=1
X=1
Params fn 1 /etc
788
0
X=2
```

Проанализируем полученную информацию. Как уже отмечалось, функция не порождает нового процесса, поэтому PID остался равным 788 как до вызова функции, так и во время ее выполнения. Переменная `X` доступна нашей функции, потому что описана до вызова функции. Функция «видит» значение переменной `X`, установленное в основном блоке сценария.

Затем функция изменяет значение переменной `X` и передает его в основной блок (`X=2`). Функции был передан только один параметр — `/etc`, вместо второго параметра была отображена пустая строка. Имя файла осталось прежним — `fn`.

Обратите внимание на важный момент: функция сообщила нам много полезной информации об устройстве функций в `bash`, но не оправдала своего названия — `cdir` (change dir). Реально изменения каталога не произошло, потому что перед выполнением команды `cd` была выполнена команда `return` с кодом завершения 0, которая прервала выполнение функции.

33.10. Обработка сигналов и протоколирование

Возможно, вы хотите обеспечить выполнение вашего сценария после выхода пользователя из интерпретатора или выполнить какие-нибудь действия при отключении удаленного пользователя от системы. «Перехватить» сигнал (прерывание) можно с помощью команды `trap`. Формат команды `trap` следующий:

`trap` имя сигналы

где:

- ♦ **имя** — это имя функции или набор команд, которые должны быть выполнены при получении сигнала;
- ♦ **сигналы** — наиболее часто используется перехват сигналов, описанных в табл. 33.7. Полный список сигналов вы найдете в гл. 5.

Сигналы

Таблица 33.7

Номер	Название	Описание
01	SIGHUP	Освобождение линии (hangup)
02	SIGINT	Прерывание (interrupt)
03	SIGQUIT	Выход (quit)
09	SIGKILL	Уничтожение процесса (kill). Не перехватывается и не игнорируется
15	SIGTERM	Программный сигнал завершения

Пример. Игнорирование сигналов 1, 2, 3, 15

```
trap : 1 2 3 15
```

: — это пустой оператор, не выполняющий никаких действий.

Рассмотрим, как можно протоколировать работу собственного сценария. Для этого существуют два способа — с помощью команды **tee** и команды **script**.

Способ 1:

```
$LOGFILE=my_log
if ["$LOGGING" != "true"]; then export LOGGING="true"; exec $0
| tee $LOGFILE; fi
```

Способ 2:

```
$LOGFILE=my_log
if ["$LOGGING" != "true"]; then export LOGGING="true"; exec
script $0 $LOGFILE; fi
```

В первом случае мы устанавливаем флаг протоколирования **LOGGING** и заново запускаем наш сценарий. При этом перенаправляем весь стандартный вывод команде **tee**, которая выполнит протоколирование. Второй способ аналогичен первому за исключением того, что мы не будем самостоятельно запускать сценарий — это за нас выполнит команда **script**. Оба способа можно использовать для протоколирования работы других программ:

```
script program ~/program.log
```