Федеральное агентство по образованию
Государственное образовательное учреждение
высшего профессионального образования
ПЕТРОЗАВОДСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

С. А. Кипрушкин, А. В. Соловьев

Основы работы в Linux

Учебное пособие

Петрозаводск
Издательство ПетрГУ
2009

УДК 004
ББК 32.973–018.2
К427

Печатается по решению редакционно-издательского совета
Петрозаводского государственного университета

Рецензенты:
кандидат физ.-мат. наук *В. Б. Ефлов*,
кандидат физ.-мат. наук *А. П. Мощевикин*

Кипрушкин С. А.

К427 Основы работы в Linux : учебное пособие / С. А. Кипрушкин, А. В. Соловьев. – Петрозаводск : Изд-во ПетрГУ, 2009. – 104 с.

ISBN 978–5–8021–0997–7

В пособии представлены лабораторные работы по освоению операционных систем на основе ядра Linux или аналогичных. Большое внимание уделено вопросам организации многозадачности, многопользовательского режима, файловой системы, механизмов контроля доступа, а также сетевым возможностям операционных систем типа UNIX. В ходе выполнения лабораторных работ студенты знакомятся с необходимым теоретическим материалом, а также осваивают наиболее распространённые утилиты, обеспечивающие описанные возможности или управляющие этими возможностями, приобретая, таким образом, навыки администрирования операционных систем.

Учебное пособие может быть рекомендовано студентам техникумов и вузов, обучающимся по специальностям «Автоматизированные системы организации и управления», «Информационно-измерительная техника и технологии», «Физическая электроника», «Прикладная математика» и др., а также аспирантам и научно-техническим работникам.

ISBN 978–5–8021–0997–7

УДК 004
ББК 32.973–018.2

СОДЕРЖАНИЕ

ПРЕДИСЛОВИЕ

Материал, представленный в данном пособии, является переработкой большого количества информации как по теоретическим аспектам, так и по практическим приёмам работы в системах UNIX. Особое внимание уделялось не полноте описания отдельных экзотических функций или особенностей операционных систем, а знаниям, позволяющим понять сущность работы ОС. Все практические примеры и задания отбирались как наиболее употребительные в практической работе профессиональных пользователей и администраторов компьютерных систем. Целью было сформировать умения и практические навыки, которые максимально могли бы пригодиться обучаемому, независимо от типов и версий ОС UNIX, которые он будет использовать. В частности, выполнение предлагаемых работ активно стимулирует использовать электронные руководства (Help, Man, HowTo) для выяснения особенностей работы команд и технологии их применения, при том, что суть выполняемой работы и методы её выполнения подробно излагаются.

Представленное пособие направлено на развитие практических навыков работы пользователей систем UNIX:

- системных администраторов;
- программистов, переносящих или устанавливающих свои программные продукты на UNIX-систему;
- профессиональных пользователей UNIX-систем.

Широкий круг целевой аудитории при небольшом объёме пособия потребовал от авторов объяснения только наиболее принципиальных возможностей и приёмов работы. Для читателей это хорошая возможность при ограниченных затратах времени и сил получить максимальное количество навыков работы в операционных системах UNIX. Стандартный курс рассчитан на 30 часов занятий.

Описанные лабораторные работы выполняются студентами Петрозаводского госуниверситета в течение почти 10 лет, что позволяет считать методику их выполнения отработанной. Качественность получаемых знаний подтверждается откликами бывших студентов, работающих по специальности.

Для дополнительной проверки знаний авторы рекомендуют использовать публичную систему тестирования знаний http://iq.karelia.ru, тест «Основы работы с Linux». Вопросы этого теста сформированы на основе данного пособия.

ВВЕДЕНИЕ

Прежде всего, UNIX – это многопользовательская многозадачная сетевая операционная система. *Многозадачная* означает, что система обеспечивает одновременное выполнение многих приложений, *многопользовательская* – обеспечивает одновременную работу нескольких пользователей, управляет корректным разделением ресурсов между пользователями, *сетевая* – может быть включена в локальную вычислительную сеть и может предоставлять другим компьютерам в сети определённые ресурсы.

Первая система UNIX была разработана в 1969 г. в лаборатории Белла компании AT&T. В настоящее время существует множество разновидностей UNIX-систем. Следует отметить, что юридически лишь немногие из них имеют право именоваться «UNIX», остальные же используют сходные концепции и технологии, поэтому их следует именовать «UNIX-подобные» системы.

В настоящее время UNIX является торговой маркой, принадлежащей консорциуму The Open Group (www.opengroup.org). Права на лицензии и дальнейшую разработку UNIX System V (одна из базовых разновидностей UNIX) принадлежат компании SCO Group (бывшая Caldera). Права на исходный код UNIX System V принадлежат Novell. В то же время существует большое количество UNIX-подобных систем (как свободных, так и собственнических), не связанных лицензионными соглашениями с AT&T и её преемниками.

С точки зрения The Open Group, название UNIX могут носить только системы, прошедшие сертификацию на соответствие стандарту Single UNIX Specification. В данном пособии авторы решили пренебречь юридическими тонкостями и использовать название UNIX как для аутентичных UNIX-систем, так и для UNIX-подобных.

С одной стороны, UNIX-системы имеют большую историческую важность, поскольку благодаря им распространились некоторые популярные сегодня концепции и подходы в области операционных систем (ОС) и программного обеспечения (ПО). С другой стороны, современные UNIX-системы являются неотъемлемым атрибутом передовых информационных технологий. Для UNIX-систем характерно большое разнообразие поддерживаемых аппаратных платформ. В списке 500 наиболее высокопроизводительных суперкомпьютеров планеты (www.top500.org) 439 позиций занимают системы на основе Linux (ядро свободно распространяемой UNIX-подобной ОС), 25 позиций – другие UNIX-системы, 5 позиций – системы на основе Win-

dows, 31 позицию – прочие системы (по состоянию на ноябрь 2008 г.) На рынке персональных компьютеров позиции UNIX-систем скромнее. По информации компании SpyLOG, специализирующейся на предоставлении сервисов Интернет-статистики, по состоянию на IV квартал 2008 г. 95 % персональных компьютеров работают под управлением ОС семейства Windows. Лишь 3 % пользователей используют MacOS X (UNIX-система компании Apple). На долю GNU/Linux приходится менее 1,5 % пользователей.

Производство ПО в настоящий момент является серьёзным бизнесом, управляемым уже не столько техническими соображениями, сколько законами конкуренции. Разработчики программ начали пользоваться своими авторскими правами, чтобы заставить пользователей придерживаться многочисленных правил. Витиеватым юридическим языком в условиях лицензионных соглашений собственнического[1] ПО указывается, что вам запрещено распространять или изменять программу. Идея, что общество, основанное на таких принципах, неэтично и попросту неправильно, может удивить некоторых читателей. Многие воспринимают систему собственнического ПО как данность либо судят о ней в терминах, принятых в программном бизнесе. Однако в середине 90-х годов XX века появилось движение за свободное ПО (Проект GNU – www.gnu.org). Целью движения стало наделить программы принципами свободы:

- пользователь волен запускать программу с любыми целями;
- пользователь волен модифицировать программу под свои нужды (чтобы сделать эту свободу практически эффективной, он должен иметь доступ к исходным текстам, поскольку изменять программу без них очень сложно);
- пользователь имеет право распространять копии как бесплатно, так и за деньги;
- пользователь волен распространять свои модификации, чтобы общество могло извлечь из них пользу.

За годы существования Проекта GNU в его рамках (или под влиянием его идей) были разработаны десятки тысяч свободных программ различного назначения. Особо следует отметить весьма популярные в настоящее время UNIX-подобные полностью свободные системы GNU/Linux. Существует большое количество дистрибьютивов этого вида UNIX-систем, сильно отличающихся один от другого. Общее для них – это использование ядра Linux, разработанного финским про-

[1] *Proprietary* software – *собственническое* или *проприетарное* ПО.

граммистом Линусом Торвальдсом (в настоящее время в развитии ядра Linux принимает участие большое количество программистов), и набора системных библиотек и утилит, разработанных в рамках проекта GNU под эгидой Фонда свободного программного обеспечения. Наиболее распространёнными дистрибутивами GNU/Linux являются: Debian GNU/Linux, Slackware (один из самых старых дистрибьютивов), RedHat Linux, Fedora, Novell SuSE Linux, Mandriva Linux, Gentoo, Ubuntu и др. Среди российских дистрибьютивов можно упомянуть ASP Linux, ALT Linux, а также специализированные дистрибьютивы, например, для школ, образовательных учреждений, армии (например, МСВС – Мобильная система Вооружённых сил, разработанная на основе Red Hat Linux в 2002 г.)

Другой популярной разновидностью UNIX-систем являются системы на основе разработок университета Беркли (BSD): FreeBSD, OpenBSD, NetBSD и др. Для них, как и для GNU/Linux, характерна политика свободного распространения ПО и открытых исходных текстов. Компания Apple предлагает достаточно популярные персональные компьютеры под управлением несвободной ОС MacOS X, разработанной на основе FreeBSD. Ещё одна частично свободная ОС, берущая свои истоки от аутентичных версий UNIX – это ОС OpenSolaris компании Sun. Среди коммерческих версий UNIX можно упомянуть HP/UX компании Hewlett Packard, AIX компании IBM, QNX (в настоящее время принадлежит компании Harman International Industries). Следует отметить, что для многих компаний-владельцев коммерческих версий UNIX-систем характерна тенденция отказа от собственнического ПО в пользу свободных систем GNU/Linux. Например, с 2006 г. компания SGI проводит политику перевода своих клиентов с собственной UNIX-системы IRIX на различные варианты GNU/Linux: RedHat Enterprise Linux или SUSE Linux Enterprise Server. Компания IBM также активно поддерживает разработчиков GNU/Linux, несмотря на наличие собственной версии UNIX – AIX.

Перечисленные выше факты позволяют сделать вывод об актуальности изучения современных UNIX-систем. В данном пособии пристальное внимание уделено именно системам GNU/Linux, однако, представленный материал в большинстве своём применим и к другим UNIX-системам как к подобным, так и аутентичным. Там, где отличия между системами несущественны, система, за которой читатель работает, называется просто «UNIX». Если же акцент делается на особенности систем GNU/Linux, исследуемая читателем система явно названа «GNU/Linux».

1. РЕГИСТРАЦИЯ И ПОДКЛЮЧЕНИЕ К СИСТЕМЕ UNIX. ОБЩИЕ ПРИНЦИПЫ РАБОТЫ

1.1 Общие сведения

Каждый пользователь, работающий с системой UNIX, взаимодействует с ней при помощи *терминала*. Под терминалом подразумевается не только устройство ввода-вывода данных, подсоединённое локально или удалённо к процессорной системе, исполняющей код UNIX, но и различные программы-эмуляторы терминалов, позволяющие подключиться к UNIX и использовать устройства ввода-вывода компьютера (клавиатура и экран), на котором запущена эта программа, как удалённый терминал UNIX. Устройство ввода-вывода данных, являющееся частью процессорной системы, исполняющей код UNIX, называется *системной консолью*.

Различают *графические* и *текстовые терминалы*.

Графический терминал предоставляет пользователю возможность запускать программы с графическим интерфейсом и работать с ними с помощью мыши и различных объектов экрана (кнопки, окна, полосы прокрутки, выпадающие меню и другие атрибуты GUI). Наиболее распространена реализация графического терминала X Window System. (Терминалы, поддерживающие эту спецификацию, называют *X-терминалами*.) Графический терминал может быть реализован как на системной консоли, так и удалённо. В данной работе режим графического терминала не рассматривается.

Текстовый терминал крайне нетребователен к аппаратным средствам, поэтому и аппаратные реализации текстового терминала, и программы-эмуляторы для других компьютеров очень просты. Реализации текстовых терминалов различаются по своим возможностям: одни терминалы поддерживают цвета, мышь и псевдографические примитивы, на других остаётся довольствоваться монохромным режимом с маленьким разрешением. Для программ-эмуляторов текстового терминала распространён стандарт VT100.

Основная особенность UNIX состоит в том, что пользователь может использовать ресурсы компьютера под управлением UNIX, находясь территориально в любом месте (при наличии канала связи). Более того, один и тот же пользователь может параллельно использовать ресурсы нескольких компьютеров с UNIX. И наоборот, ресурсы одного и того же компьютера могут одновременно использовать несколько пользователей. Коммерческие версии UNIX могут накладывать огра-

ничения на число одновременно работающих пользователей, однако, для каждого пользователя количество одновременно работающих задач практически не ограничено.

Промежуток времени между подключением пользователя к системе и отключением от неё называется сеансом (session). В течение сеанса пользователь может выполнять любые доступные в системе команды, в том числе и открывать сеансы с другими системами UNIX.

Условием возможности подключения пользователя является наличие учётной записи пользователя (account) в системе. Добавление новой учётной записи осуществляет администратор. В UNIX ему сопоставляется учётная запись с именем root. При добавлении новой учётной записи в базу данных о пользователях заносится регистрационное имя нового пользователя (login) и устанавливается пароль для его входа в систему, а также некоторые дополнительные системные настройки.

Таким образом, пользователь должен знать определённое для него имя (login) и пароль, а в случае удалённого подключения – имя или IP-адрес нужной системы в сети Internet/Intranet. После запуска программы-эмулятора терминала и выбора системы для подключения происходит установление соединения и появляется приглашение для входа в систему:

```
login:
```

При наборе имени пользователя следует иметь в виду, что UNIX различает строчные и прописные буквы. После ввода регистрационного имени, завершающегося нажатием на [Enter], система предлагает ввести пароль:

```
password:
```

При вводе пароля следует быть особенно внимательным, поскольку он не отображается. Если при регистрации произошла ошибка, система выдаст сообщение:

```
Login incorrect.
login:
```

Как правило, после определённого числа неудачных попыток регистрации система разрывает соединение и работа эмулятора терминала завершается.

Системная консоль современных версий UNIX предоставляет возможность разделять клавиатуру и экран между несколькими *«виртуальными» терминалами*. Для системной консоли не требуется процедуры установки соединения с системой, поэтому каждый виртуаль-

ный терминал уже содержит приглашение входа в систему (и, как правило, некоторые сведения о системе, такие как идентификатор терминала). Переключение между виртуальными терминалами осуществляется при помощи комбинаций [Alt]+[F1], [Alt]+[F2][2], … и т. д.

После успешной регистрации система выводит некоторую актуальную информацию, например, кратко сообщает свою версию, имя компьютера и идентификатор назначенного пользователю терминала (TTY), а также сообщение о наличии новой почты. После этого появляется приглашение командного интерпретатора системы (оболочки) – система готова принимать команды пользователя:

```
[pupkin@somehost homedir] $
```

Вид приглашения определяется настройками пользователя. Обычно приглашение содержит имя пользователя, название системы и текущий каталог. Завершается приглашение знаком $ или # (для администратора). В последующих примерах знак $ указан для обозначения командной строки и не должен набираться.

В командной строке текстовой оболочки пользователь набирает команды. **Строчные и заглавные буквы в командной строке РАЗЛИЧАЮТСЯ!** Команды, их опции и имена файлов следует указывать в правильном регистре (обычно все строчные). Команды могут быть внешние (исполняемые файлы) или внутренние (встроенные команды оболочки). При выполнении внешних команд надо учитывать, что система должна знать путь к файлу, либо он должен находиться в одной из директорий, указанной в переменной окружения PATH. UNIX не ищет исполняемый файл в текущем каталоге (текущий каталог обозначается точкой), если только путь "./" не присутствует в PATH. Если требуется запустить программу, находящуюся в каталоге, не указанном в PATH, обязательно надо указать путь к исполняемому файлу.

Ввод команды **`exit`** или **`logout`** закончит сеанс работы с системой и приведёт к разрыву связи с выдачей соответствующего сообщения. После этого можно либо снова соединяться с удалённой системой, либо вернуться в исходную операционную среду.

Оболочка хранит историю последних набранных в ней команд. Перемещение по истории команд осуществляется при помощи клавиш вверх [↑] и вниз [↓]. Поиск в истории команд – [Ctrl]+[r]. Нажав эту комбинацию, можно набрать ключевой фрагмент искомой команды,

[2] Запись [Alt]+[F1] означает, что клавиши [Alt] и [F1] должны быть нажаты одновременно. Можно сначала нажать [Alt], а затем, не отпуская [Alt], нажать [F1], после чего отпустить обе клавиши.

тогда из истории будет подставлена команда, сопоставляющаяся с данным фрагментом. Чтобы просмотреть историю команд целиком, можно воспользоваться встроенной командой **history**.

Оболочка также предоставляет средства автозавершения имён (автоматический донабор имени). Если в командной строке набрать часть имени команды или файла и нажать [Tab], оболочка закончит имя или выведет все возможные варианты завершения имени. Если вариантов очень много, пользователю будет выдан запрос:

```
Display all 2515 possibilities? (y or n)
```

Например, если в командной строке набрать **his** и нажать [Tab], оболочка дополнит это имя до команды **history**. Если же набрать только **hi**, то система выдаст несколько вариантов завершения имени.

Очень многие команды работают с данными, поступающими на стандартный ввод программы (набираются на клавиатуре). Если требуется завершить ввод таких данных, следует нажать [Ctrl]+[d] в пустой строке. Например, конец стандартного ввода для оболочки приводит к завершению её работы. Если оболочка была начальной, то это эквивалентно завершению сеанса (**exit** или **logout**).

Если какая-то программа выводит большой фрагмент текста на экран, можно приостановить её вывод, нажав [Ctrl]+[s]. Возобновление вывода продолжится после нажатия [Ctrl]+[q]. Если система не реагирует на нажатые вами клавиши, проверьте, не нажали ли вы случайно [Ctrl]+[s] (нажмите [Ctrl]+[q]). Пролистать уже выведенную на текстовый терминал информацию можно при помощи комбинаций [Shift]+[PgUp] и [Shift]+[PgDn].

Прервать программу можно при помощи комбинаций [Ctrl]+[c] или [Ctrl]+[\ ]. Завершение всех программ и перезагрузка системы – [Ctrl]+[Alt]+[Del].

Если строка команды длинная, её можно разбить на несколько строк. Для этого в конце каждой строки, кроме последней, следует поместить символ обратного слэша (\ ).

Некоторые символы имеют для интерпретатора специальное значение:

| & ; < > ( ) $ ' \ " ` пробел табуляция перевод строки

Если эти символы необходимо использовать в каком-либо слове командной строки и заблокировать их специальное значение, следует применить экранирование (*quoting*). Есть три способа экранирования: обратный слэш (\ ), апостроф (') и кавычки ("). Неэкранированный обратный слэш блокирует специальное значение следующего за ним

символа. Все символы, заключённые между апострофами, экранируются. Внутри такой последовательности символ апострофа недопустим. Заключение символов в кавычки экранирует их, за исключением символов: $ ' \. Первые два символа сохраняют своё специальное значение. Обратный слэш сохраняет специальное значение, только если после него находится один из символов: $ ' " \ или перевод строки.

При обработке командной строки командный интерпретатор выполняет определённые преобразования. После того, как все преобразования выполнены и все специальные символы интерпретированы, символы экранирования из командной строки удаляются.

В приведённых примерах используется встроенная команда **echo**, которая выводит на стандартный вывод (экран) указанные ей аргументы:

```
$ echo 1\&2\
> 345
1&2345
$ echo '123
> 45'
123
45
$ echo  1  2  3  4  5
1 2 3 4 5
$ echo " 1   2"\ \ 3'  4'   5
 1   2  3  4 5
```

Если при работе с терминалом доступна мышь, то выделение мышью фрагмента текста вызывает копирование этого текста в буфер обмена. Текст может быть вставлен в текущей позиции курсора (не курсора мыши!!!) при нажатии на правую кнопку мыши. Некоторые программы используют мышь по-своему.

1.2 Справочная система

В UNIX используются несколько видов справочных систем.

Справочная система **man** содержит информацию о программах, установленных в системе, о форматах конфигурационных файлов, о библиотечных функциях языка Си, а также о некоторых понятиях, используемых в UNIX. Формат команды **man**:

man [*опции*] [*секция*] *имя*

Файлы этой справочной системы разбиты на несколько секций. Если не указана секция, **man** ищет справочный файл с указанным именем по очереди во всех секциях, начиная с первой. В первой сек-

ции хранятся справочные файлы команд общего назначения. Во второй, третьей и девятой секциях хранятся справочные файлы для программистов (описание библиотечных функций Си, системных вызовов, функций ядра и др.). Пятая секция хранит описания конфигурационных файлов различных программ. Седьмая секция содержит справку о некоторых понятиях. В восьмой секции описаны специальные системные программы. Примеры:

- **man man** – получение подсказки по ключам команды **man**;
- **man bash** – описание командного интерпретатора bash и его встроенных команд;
- **man 1 printf** – получение подсказки по команде **printf**;
- **man 3 printf** – получение информации о функции printf из стандартной библиотеки языка Си;
- **man 7 ascii** – получение справки о коде ASCII.

Программа допускает прокрутку информации вперёд и назад при помощи клавиш управления курсором. Другие клавиши: [<] – начало файла, [>] – конец файла, [/] – поиск вперёд по шаблону, [?] – поиск назад по шаблону, [n] – продолжить поиск, [N] – продолжить поиск в обратном направлении, [h] – вывести подсказку, [q] – выход.

Чтобы получить подробную подсказку по встроенным командам оболочки (bash), можно воспользоваться **man bash**. Для получения быстрой и короткой подсказки удобнее использовать команду **help**. Вызов **help** без параметров приводит к отображению на экране всех встроенных команд оболочки. Чтобы разобраться с конкретной командой, надо указать её в качестве параметра команды:

- **help help** – получение подсказки по команде **help**;
- **help type** – получение подсказки по команде **type**.

Третий способ получения справочной информации в UNIX – использование гипертекстовой системы **info**. Эта программа предоставляет возможность навигации по ссылкам между связанными документами: переход по ссылке – [Enter], переход на уровень вверх – [u], переход на последнюю просмотренную страницу – [*l*], переход на следующую страницу в разделе – [n] и др.

Кроме того, большинство команд выводят краткую справку, если в их командной строке указать ключ *--help*. Например:

```
man --help
```

При описании синтаксиса команд используются определённые правила. Например, команда **date** выдаёт системную дату и время. Используя различные опции (ключи), можно добиться вывода даты

в различных форматах. Администратору эта команда позволяет изменять системную дату и время. Синтаксис команды **date** описан так:

date [*OPTION*]… [**+***FORMAT*]
date [**-u**|**--utc**|**--universal**] [*MMDDhhmm*[[*CC*]*YY*][.*ss*]]

Первый вариант используется для вывода даты и времени в различных форматах, второй – для изменения даты и времени. При описании синтаксиса квадратные скобки [], многоточие, и вертикальная палочка (|) имеют специальный смысл и не должны набираться в командной строке.

Символы, заключённые в квадратные скобки, обозначают необязательное использование. Если применены вложенные квадратные скобки, то символы внутри вложенных квадратных скобок могут быть указаны только наряду с внешними, окружающими эти вложенные скобки символами. Например, формат даты указан в виде [MMDDhhmm[[CC]YY][.ss]] (MM – месяц, DD – день, hh – часы, mm – минуты, CC – столетие, YY – год, ss – секунды). Это означает, что столетие, год и секунды указывать необязательно. То есть дату можно задать в виде: 06181020, что будет означать 18 июня сего года 10 часов 20 минут. Можно дополнительно указать секунды: 061820.34, при этом синтаксис команды требует их отделения от остальной части при помощи точки. Можно указать год: 0618102094 (18 июня 1994 года 10 часов 20 минут). Если указывается столетие (2 цифры), то оно обязательно должно сопровождаться последующими двумя цифрами года: 061810201994. И наконец, полный вариант: 061810201994.34 – 18 июня 1994 года 10 часов 20 минут 34 секунды. Причём весь код даты заключён в квадратные скобки – это означает, что команда может быть запущена вообще без этого параметра (тогда она просто выведет текущую системную дату на экран).

Вертикальная палочка разделяет эквивалентные или взаимно исключающие последовательности в синтаксисе. То есть команда **date** может быть запущена в одном из трёх вариантов:

```
date -u 06181020
date --utc 06181020
date --universal 06181020
```

В данном случае вертикальная палочка разделяет эквивалентные варианты. Опция *-u* (*--utc*, *--universal*) используется для задания времени по UTC (всеобщее скоординированное время, т. е. время на нулевом меридиане).

Многоточие в описании синтаксиса команды **date** показывает, что в командной строке может быть использовано несколько опций.

Как правило, команды используют опции двух типов: короткие (обозначаются одним символом, начинаются с одиночного минуса) и длинные (обозначаются словами или фразами, начинаются с двух минусов; если использовано несколько слов, слова разделяются одним минусом). Обычно одна и та же функция команды имеет обозначение и в виде короткой опции, и в виде длинной. Например, для команды **`date`** для короткой опции *-u* имеется два синонима – длинные опции *--utc* и *--universal*. Для опции *-R* (вывод даты в формате стандарта RFC 2822 для электронных сообщений) имеется синоним *--rfc-2822*.

Некоторые опции могут требовать дополнительных параметров. В таком случае для короткой опции её параметр отделяется пробелом, для длинной – знаком присваивания. Например, команде **`date`** можно указать вывести на экран не текущую дату, а заданную в командной строке (не изменяя при этом системную дату). Дату можно указать при помощи опции *-d* или *--date* (в этом случае она задаётся либо по стандарту ISO 8601 ГГГГ-ММ-ДД, либо по общепринятому в США формату ММ/ДД/ГГГГ):

```
date -d 2007-03-13
date --date=03/13/2007
```

При необходимости опции можно комбинировать, при этом порядок следования опций, как правило, роли не играет:

```
date --rfc-2822 -d 03/13/2007
date --date=2007-03-13 -R
```

Особенностью коротких опций является то, что их можно объединять за общим знаком минуса (однако, с учётом необходимых им параметров):

```
date -Rd 2007-03-13\ 13:22
```

В данном примере последовательность *-dR* будет недопустимой, поскольку опция *-d* требует обязательного параметра.

Обратите внимание, в последнем примере использован символ экранирования \, поскольку параметр для опции *-d* содержит пробел, являющийся для оболочки специальным символом (символом-разделителем параметров).

1.3 Информационные команды

Команды **`who`** и **`w`** выводят список подключённых к системе пользователей.

Чтобы получить более подробную информацию о конкретном пользователе, следует использовать команду:

finger [username[@hostname]]

Команда **uname** выводит информацию о системе. Ключ *-a* используется, чтобы получить тип системы, версию ядра, дату его последней компиляции, архитектуру процессора и т. п.

```
$ uname
FreeBSD
$ uname -a
FreeBSD students.soros.karelia.ru 2.2.6-RELEASE FreeBSD
2.2.6-RELEASE #0: Thu Jul 6 15:12:06 MSD 2000
oleg@students.soros.karelia.ru:/usr/src/sys/compile/STUD
ENTS i386
```

Встроенная команда **type** сообщает тип команды, указанной в качестве параметра: builtin (встроенная) или внешняя (выводит путь к исполняемому файлу):

```
$ type help
help is a shell builtin
$ type man
man is hashed (/usr/bin/man)
```

hostname – вывод доменного имени компьютера. Имя компьютера состоит из нескольких частей, отделённых точками. Слово до первой точки является собственно именем данной системы, а остальная часть – именем *домена*, к которому данный компьютер относится. Упрощённо доменом называют некоторую совокупность компьютеров. Домены выстраивают в некоторую иерархию, так что домены более верхнего уровня включают в себя домены нижних уровней. Эта иерархическая зависимость отражается в имени домена.

```
$ hostname
dims.karelia.ru
```

В приведённом примере собственное имя компьютера – *dims*. Этот компьютер входит в домен второго уровня *karelia.ru*. В свою очередь, имя этого домена говорит о том, что он входит в структуру домена верхнего уровня *ru*.

Команда **tty** выдаёт идентификатор терминала, за которым работает пользователь.

1.4 Многопользовательская защита

Пользователи, которым разрешено входить в систему (регистрироваться), перечислены в учётной базе пользователей. Пользователи объединяются в группы; последние перечислены в учётной базе групп. Каждому пользователю и каждой группе пользователей назначается целочисленный идентификатор (*UID – user identifier, GID – group identifier*). Администратору соответствует UID=0 и GID=0.

Чтобы определить UID и GID пользователя, можно воспользоваться командой **id**. Команды **whoami** и **groups** являются аналогами команды **id** с различными ключами.

Команда **chfn** позволяет пользователю изменить информацию, выдаваемую о нём командой **finger**.

Чтобы сменить пароль, пользователю следует воспользоваться командой **passwd**. При этом если пользователь не является администратором, система сначала запросит у него старый пароль, а лишь потом предложит ввести новый. При наборе пароля знаки на экране не отображаются. Чтобы уменьшить вероятность ошибки, новый пароль набирается два раза. Если слова не совпадают, система оставляет старый пароль.

Если в системе вам доступно несколько учётных записей, возможно временное изменение пользовательского идентификатора при помощи команды:

```
su [ключи] [пользователь]
```

При этом запускается оболочка с правами указанного пользователя (если вы не администратор, система запросит пароль для указанного пользователя). Если от другого пользователя надо выполнить лишь одну команду, эту команду можно указать при помощи опции *-c*. Если имя пользователя опущено, подразумевается root.

1.5 Редактор vi

В UNIX много различных текстовых редакторов, таких как **pico**, **emacs**, **ed** и т. п. Одним из самых распространённых является **vi** (visual editor) или **vim** (VI Improved – улучшенный VI). Достаточно часто **vi** используется для редактирования конфигурационных файлов операционной системы. При вызове **vi** можно в качестве аргумента указать имя редактируемого файла. В редакторе **vi** имеются три режима работы: командный, режим вставки и режим последней строки. По умолчанию редактор запускается в командном режиме.

Этот режим позволяет использовать определённые команды для редактирования файлов или перехода в другие режимы. Вставка или редактирование текста осуществляется в режиме вставки. Переход из командного режима в режим вставки осуществляется с помощью клавиши [i] или клавиши [Insert]. Для завершения режима вставки и возврата в командный режим – клавиша [Esc]. В режим последней строки программа переходит по клавише [:] из командного режима. В этом режиме в последней строке экрана появляется поле для ввода многосимвольных команд. Чтобы отличать эти команды от обычных команд, перед ними записывают двоеточие.

Начните работу с редактором, вызвав команду **`vi`**. Клавишей [i] перейдите в режим вставки. Напечатайте краткую информацию о себе. При вставке текста можно напечатать сколько угодно строк, нажимая [Enter] после каждой строки. Можно также корректировать ошибки, используя клавишу [Backspace]. Завершение работы в режиме вставки и возврат в командный режим осуществляется клавишей [Esc]. В командном режиме можно использовать клавиши со стрелками для перемещения по файлу или для тех же целей – [h], [j], [k], [*l*], соответственно, влево, вниз, вверх, вправо.

Есть несколько способов вставки текста, отличных от использования [i]. Нажав на [o], вы начинаете вставку текста в строку ниже текущей. В командном режиме клавиша [x] удаляет символ перед курсором. Вы можете удалять целые строки, набирая команду `dd` (т. е. нажимая [d] дважды). Чтобы удалить слово, на котором находится курсор, используйте команду `dw`. Вы можете заменить фрагменты текста, используя клавишу [R]. Использование [R] для редактирования текста очень походит на клавиши [i] и [o], но [R] заменяет прежний текст вместо вставки в него. Клавиша [r] позволяет заменить один символ, отмеченный курсором. Клавиша [~] (знак тильда) изменяет регистр буквы, отмеченной курсором: заглавную делает строчной и наоборот. Клавиша [w] перемещает курсор на начало следующего слова, [b] перемещает на начало предыдущего слова. Клавиша [0] (ноль) передвигает курсор на начало текущей строки, а клавиша [$] перемещает на конец строки. Нажатием [Ctrl]+[f] курсор перемещается на экран вперёд, с помощью [Ctrl]+[b] – на экран назад. Чтобы переместить курсор в конец файла – клавиша [G]. Можно переместиться также на любую строку; например, напечатав команду `10G`, вы переместите курсор на десятую строку файла. Чтобы встать на начало (на первую строку), используйте `1G`. Можно сочетать команды перемещения с другими командами, такими, как удаление. Например, ко-

манда `d$` удалит от местоположения курсора до конца строки, `dG` удалит всё от курсора до конца файла и т. д.

Для выхода из **vi** без внесения изменений в ранее существовавший файл используйте команду `:q!`. Когда вы нажмёте [:], курсор переместится на последнюю строку экрана, поскольку вы перейдёте в режим последней строки. В режиме последней строки вы должны нажать [Enter] после набора команды. Команда `:wq` сохраняет файл, а затем выходит из **vi**. Команда `ZZ` (в режиме команд) эквивалентна `:wq`. Для записи файла без выхода из **vi** используется просто `:w`, при этом через пробел можно указать имя файла. Для перехода к редактированию другого файла используется команда `:e` (через пробел можно указать имя). Если вы используете `:e` без предварительного сохранения файла, то сначала вы получите соответствующее предупреждение. В этот момент вы можете использовать `:w`, чтобы сохранить исходный файл, а затем использовать `:e` или сразу нажать [!] – редактировать новый файл без сохранения изменений в первом файле. Если вы используете команду `:r`, можно включить содержимое другого файла в текущий файл. Например, команда `:r foo.txt` вставит содержимое файла foo.txt в данное место текста.

Вы можете также выполнять команды прямо из **vi**. Команда `:r!` работает, как `:r`, но вместо чтения файла она вставляет результат выполнения данной команды в место, где находится курсор. Например:

```
:r! who
```

Также вы можете выполнить команду, находясь в редакторе **vi,** и вернуться в редактор после её завершения. Например:

```
:! who
```

В данном примере будет выполнена команда **who**, её результат выдан на экран, а не вставлен в редактируемый файл.

Можно временно запустить ещё одну копию оболочки для выполнения других команд системы при помощи команды `:shell`. Редактор запустит командный интерпретатор, который позволит временно «отложить» **vi** и выполнить команды. После выхода из оболочки (используя команду **exit**) вы вернётесь в **vi**.

Для получения более подробной информации следует обратиться к справочному руководству **man vi**, а также специальному справочнику **vimtutor**.

1.6 Процессы, сигналы, задания

Единицей управления и потребления ресурсов в системе служит *процесс*. Вызывая команды, пользователь тем самым порождает процессы. При создании процесс получает уникальный ненулевой целочисленный идентификатор (*PID – process identifier*), по которому система отличает данный процесс от других. Права доступа процесса определяются атрибутами UID и GID пользователя, запустившего процесс. Если эти процессы порождают другие процессы, то те наследуют права родителей. При этом для каждого процесса запоминается PID его родителя (*PPID – parent process identifier*).

Информацию о запущенных процессах можно получить, используя команду **ps**. Ключ *-A* предназначен для вывода всех процессов (иначе выводятся только те процессы, которые связаны с текущим терминалом). Ключ *-l* предназначен для вывода полной информации о процессе: F (флаги), S (состояние), UID, PID, PPID, C (процент использования процессора), колонки PRI и NI определяют приоритет, SZ (размер в блоках), WCHAN (состояние), TTY (терминал), CMD (строка запуска процесса).

```
$ ps
  PID TTY          TIME CMD
 4156 pts/2    00:00:00 bash
 5023 pts/2    00:00:00 ps
$ ps -l
 F  S UID  PID PPID C PRI  NI SZ WCHAN  TTY    TIME  CMD
000 S 501 4156 4154 0 70   0 720 wait4 pts/2 0:00:00 bash
000 R 501 5024 4156 0 74   0 762     - pts/2 0:00:00 ps
```

Чтобы отследить процессы, интенсивно использующие процессор, удобно воспользоваться утилитой **top**. Она выводит информацию, аналогичную той, что выводит команда **ps**, но при этом работает интерактивно, периодически обновляя выводимую информацию; при этом процессы сортируются по процентному использованию процессорного времени. Клавиши управления: [пробел] – обновить информацию, [q] – выход из программы.

Сигнал – логическое взаимодействие процесса с окружающей средой, позволяющее сообщить процессу о наступлении некоторого события в системе. Сигналы различаются своими номерами – целыми числами от 1 до 32 или 64 (зависит от системы). Сигнал генерируется (т. е. посылается процессу), когда происходит определённое событие: аппаратная ошибка, результат измерения времени (сработал таймер), прерывание с терминала (пользователь нажал [Ctrl]+[c]), выполнение

команды **kill** и т. д. На каждый сигнал, определённый в системе, процесс должен иметь реакцию – действие, которое он выполняет при получении сигнала. Например, сигнал SIGKILL всегда завершает процесс, которому он послан.

Некоторые сигналы:

- SIGHUP (*hang up – «разрыв линии»*) – Сигнал генерируется, в частности, когда терминал, подсоединённый через последовательный порт (модем), отсоединился; часто используется для сообщения процессу о том, что его конфигурационные файлы изменились.
- SIGINT (*interrupt*) – Прерывание с терминала (пользователь нажал [Ctrl]+[c]).
- SIGKILL (*kill*) – Безусловное завершение.
- SIGSEGV (*segment violation*) – Нарушение защиты памяти (программа обратилась к области памяти, которая ей не принадлежит).
- SIGALRM (*alarm*) – Таймер процесса (заказанное процессом время истекло).
- SIGTERM (*termination*) – Условное завершение (посылается командой **kill**, если не указан другой сигнал).
- SIGTSTP (*terminal stop*) – Остановка с терминала (пользователь нажал [Ctrl]+[z] или [Ctrl]+[y]).
- SIGCONT (*continue*) – Продолжение после остановки.
- SIGTTIN (*trying terminal input*) – Попытка чтения фоновым процессом с активного терминала.
- SIGTTOU (*trying terminal output*) – Попытка записи фоновым процессом на активный терминал.

Пользователь может послать любой сигнал процессу, UID которого совпадает с UID пользователя, при помощи команды **kill**. Обычно у пользователя возникает необходимость послать процессу сигнал SIGTERM или SIGKILL, когда какой-либо процесс перестаёт реагировать (в том числе на [Ctrl]+[c]). Формат команды:

kill [-s *сигнал*] *PID*

Примеры:

- **kill -l** – получить список используемых сигналов;
- **kill 1234** – послать сигнал SIGTERM процессу 1234;
- **kill -s SIGHUP 1234**;
- **kill -HUP 1234**;
- **kill -1 1234** (сигналу SIGHUP соответствует номер 1).

Последние три команды посылают сигнал SIGHUP процессу с PID=1234.

Аналогом команды **kill** является команда **killall**. Вместо PID процесса её аргументом является имя процесса. Пользуйтесь ей осторожно, т. к. указанный сигнал посылается всем процессам с указанным именем:

```
killall httpd
killall -HUP httpd
```

Большинство систем UNIX обеспечивают механизм управления заданиями. При интерактивной работе интерпретатора каждая простая команда или конвейер (см. раздел 3) называются *заданием*. Задания, как правило, имеют отдельную нумерацию. Посмотреть список заданий можно при помощи встроенной команды **jobs**. Пока оболочка выполняет команды синхронно (одну за другой), механизм управления заданиями не активен и список заданий пуст. Этот механизм активируется, когда текущее задание приостанавливается или запускается в фоновом режиме. Приостановка задания – [Ctrl]+[z]:

```
$ jobs
$ vi
^Z
[1]+  Stopped                 vi
$ jobs
[1]+  Stopped                 vi
```

Для ссылок на то или иное задание можно использовать не только PID процессов, но и номера заданий в виде **%num**. Кроме того, на текущее задание можно сослаться комбинацией **%%** или **%+**, а на предыдущее **%-**. Эти обозначения можно также использовать в команде **kill**.

Чтобы возобновить выполнение приостановленного задания или перевести фоновое задание в приоритетный (foreground) режим, используется команда **fg**. Если аргумент не указан, команда оперирует текущим заданием. Примеры:

- **fg** или **fg %%** – возобновить выполнение текущего задания;
- **fg %-** – возобновить выполнение предыдущего задания;
- **fg %5** – возобновить выполнение задания № 5.

Если процессу не требуется активное взаимодействие с терминалом (вывод данных непосредственно на экран, ввод данных непосредственно с клавиатуры), то процесс может быть переведён в фоновый (background) режим. В фоновом режиме процесс выполняется парал-

лельно с другими процессами до тех пор, пока ему не потребуется выполнить операцию ввода-вывода с терминалом, при этом он приостанавливается до тех пор, пока снова не будет переведён в приоритетный режим.

Чтобы запустить задание в фоновом режиме, в его командной строке следует использовать ограничитель **&**. Приостановленное задание можно перевести в фоновый режим при помощи команды **bg**. Пример:

```
$ while true; do date >>1.txt; sleep 5; done &
[2] 21491
$ jobs
[1]+ Stopped  vi
[2]- Running while true; do date >>1.txt; sleep 5;done &
$ kill -SIGTSTP %2
$ jobs
[1]- Stopped  vi
[2]+ Stopped while true; do date >>1.txt; sleep 5; done
$ bg %-
[1]- vi &
$ bg %2
[2]- while true; do date >>1.txt; sleep 5; done &
$ jobs
[1]+ Stopped  vi
[2]- Running while true; do date >>1.txt; sleep 5;done &
$ kill %2
[2]+ Terminated while true; do date >>1.txt;sleep 5;done
$ jobs
[1]+ Stopped  vi
```

В этом примере в фоновом режиме запускается команда, которая каждые 5 секунд сохраняет в файле 1.txt текущее время. Обратите внимание на амперсанд в конце командной строки. Система присвоила номер 2 этому заданию. Затем при помощи команды **kill** ему посылается сигнал SIGTSTP, который эквивалентен нажатию [Ctrl]+[z] для активного задания, однако, это задание не является приоритетным, т. е. не взаимодействует с терминалом, поэтому нажатие [Ctrl]+[z] никак не повлияет. В результате задание приостанавливается. Следующая команда – попытка перевести в фоновый режим остановленный ранее редактор **vi**, а затем задание № 2. Очевидно, что работа задания № 2 возобновилась в фоновом режиме, а редактор так и остался приостановленным, т. к. при возобновлении он сразу же пытается читать с терминала. Для завершения задания № 2 использовалась команда **kill**.

Следует отметить, что после выполнения команд **bg**, **fg** и **kill** оболочка не всегда сразу отображает изменение состояния задания. Иногда такое сообщение появляется только после выдачи очередного командного приглашения (нужно нажать [Enter]).

О других особенностях механизма управления заданиями можно узнать из **man bash**, раздел JOB CONTROL.

1.7 Перенаправление ввода-вывода

Каждый запущенный процесс имеет три открытых файла с дескрипторами 0, 1 и 2. Файл с дескриптором 0 соответствует стандартному вводу. Файл с дескриптором 1 соответствует стандартному выводу. Файл с дескриптором 2 соответствует стандартному выводу ошибок. По умолчанию все три файла связаны с терминалом, т. е. чтение из файла стандартного ввода приводит к вводу данных с клавиатуры, а запись в файл стандартного вывода или стандартного вывода ошибок приводит к отображению текста на экране. Обычно в файл стандартного вывода отправляется полезный результат выполнения программы, а в файл стандартного вывода ошибок – диагностические сообщения. Командный интерпретатор позволяет переназначить эти файлы, т. е. вместо терминала указанные стандартные файловые дескрипторы будут связаны с определённым файлом.

Перенаправление вывода (*n* – необязательный аргумент, номер дескриптора, по умолчанию – 1):

команда [*n*]**>***файл*

Если файл не существует, он создаётся. Если файл существует, он усекается до нулевого размера. Для дозаписи в указанный файл используется следующий синтаксис:

команда [*n*]**>>***файл*

Перенаправление ввода (*n* – необязательный аргумент, номер дескриптора, по умолчанию – 0):

команда [*n*]**<***файл*

```
$ echo :1,10j >2.txt
$ echo :wq >>2.txt
$ vi -e 1.txt <2.txt
```

В этих примерах первая команда выводит текст «:1,10j» в файл 2.txt. Вторая команда дописывает текст «:wq» в конец файла 2.txt. Третья команда запускает **vi** в пакетном режиме (ключ *-e*) – на стан-

дартный ввод подаются команды из файла 2.txt. В результате в файле 1.txt будут объединены строки 1–10.

Если сообщения программы об ошибках неважны, стандартный вывод ошибок (дескриптор 2) можно перенаправить в специальное устройство /dev/null, как показано в нижеследующем примере:

```
$ date -s 12:30
date: cannot set date: Operation not permitted
Fri Jan 30 12:30:00 MSK 2009
$ date -s 12:30 2>/dev/null
Fri Jan 30 12:30:00 MSK 2009
```

Особый тип перенаправления стандартного ввода – встраиваемые в командную строку документы (here documents). Оболочка передаёт на стандартный ввод команды текст встроенного документа, размещаемого в том же источнике, что и команда. Встроенный документ должен оканчиваться словом-ограничителем, совпадающим с указанным после знака перенаправления.

```
команда <<ограничитель
встроенный документ
ограничитель
```

В следующем примере **vi** запускается в пакетном режиме. На стандартный ввод **vi** подаётся текст из командной строки: команды «:2,5j» (объединение строк 2–5) и «:wq» (выход с сохранением). В качестве ограничителя использовано слово «BNDR».

```
$ vi -e 1.txt <<BNDR
> :2,5j
> :wq
> BNDR
```

Специальный случай перенаправления – конвейер. *Конвейер* – это последовательность команд, соединённых управляющей операцией “|”. При этом стандартный вывод каждой команды конвейера, кроме последней, направляется в стандартный ввод следующей команды. Команды конвейера исполняются не последовательно, а параллельно. Соединение соседних команд в конвейере основано на использовании программных каналов (*pipes*). Оболочка ожидает завершения всех процессов, составляющих конвейер, если только он не выполняется в фоновом режиме. Пример:

```
who | tr a-z A-Z | vi -
```

В этом конвейере использовано три команды. Стандартный вывод команды **who** подаётся на стандартный ввод команды **tr**. Команда **tr**

предназначена для замены указанного набора символов на другой набор символов, т. е. первый символ первого набора всегда заменяется на первый символ второго набора и т. д. Наборы символов задаются как аргументы командной строки, причём можно использовать минус (–) как знак диапазона. Обрабатываемый текст берётся со стандартного ввода, а результат выводится на стандартный вывод. В нашем случае первый набор задан как «a–z», что соответствует всем строчным латинским буквам. Второй набор соответствует всем заглавным латинским буквам. Результат выполнения этой команды – текст, в котором все строчные латинские буквы заменены на заглавные. Этот результат передаётся на стандартный ввод редактора **vi**. Редактор запущен с ключом "–", означающем, что редактироваться будет текст, взятый со стандартного ввода программы.

Контрольные вопросы и задания

1. Выясните необходимую информацию для подключения и осуществите регистрацию в системе. Воспользуйтесь справочной системой UNIX и изучите синтаксис команд, упомянутых в этом разделе.
2. Используя описанные команды, получите информацию о работающих пользователях, о системе, данные о конкретном пользователе, о своей группе.
3. Создайте несколько текстовых файлов, содержащих информацию о вас и месте вашей работе (учёбы). Отредактируйте файлы, используя приведённые команды.
4. Запустите редактор **vi**, приостановите его работу. Запустите в фоновом режиме предложенную в примере команду периодического сохранения времени в файле 1.txt. Возобновите работу **vi** и загрузите в него файл 1.txt. Снова приостановите **vi**. Приостановите работу вашей команды, как показано в примере. С помощью **vi** убедитесь, что новые данные в файл не записываются, пока команда приостановлена. Завершите работу команды, вернитесь в **vi** и штатным образом выйдите из него.
5. Составьте конвейер из команд **ps** и **tr** так, чтобы информация выводилась только заглавными буквами. Вывод перенаправьте в файл ps.txt.

2. ФАЙЛОВАЯ СИСТЕМА ОС GNU/LINUX. РАБОТА С ФАЙЛАМИ И КАТАЛОГАМИ

2.1 Файловая система EXT2

Файловая система – это метод организации файлов на устройствах хранения данных. В мире UNIX существует несколько видов различных файловых систем со своей структурой. Наиболее известны традиционная файловая система S5 (для System V) и файловая система UFS, разработанная университетом Беркли для своих операционных систем BSD. Многие файловые системы других UNIX-подобных операционных систем используют модифицированные варианты UFS. В частности, для систем на основе ядра Linux была разработана файловая система EXT2 (Second Extended File System), во многом наследующая основные черты UFS. В настоящее время применяется файловая система EXT3, представляющая собой обычную систему EXT2, дополненную функцией журналирования.

Файловая система EXT2 состоит из последовательности секций (т. н. *групп цилиндров*), каждая из которых содержит не только общую информацию, но и информацию о размещении файлов в этой группе (рис. 2.1).

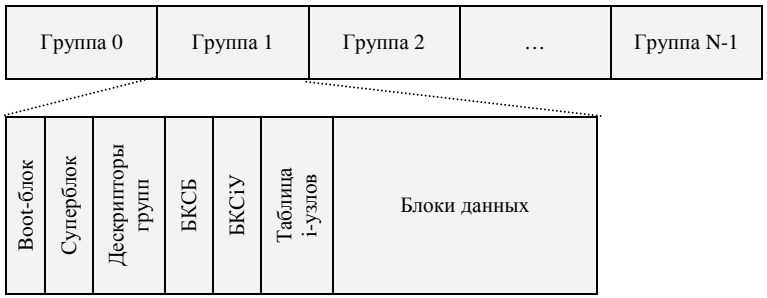

Рис. 2.1. Структура EXT2

Примечания. Boot-блок размещается в группе 0, его копии есть также в некоторых других группах. БКСБ – битовая карта свободных блоков. БКСіУ – битовая карта свободных і-узлов.

Файловая система обычно размещается на дисках или других устройствах внешней памяти, имеющих блочную структуру. Выделение места под данные на диске производится порциями, обычно больши-

ми, чем один физический блок (сектор). В UNIX такая порция также называется *блоком*. Далее, если не будет оговорено специально, термин *блок* будет использоваться во втором значении. Размер логического блока определяется при форматировании и может быть 1, 2, 4, 8 или 16 физических блоков (секторов).

Кроме блоков, содержащих файлы и каталоги, в файловой системе организовано ещё несколько служебных областей. *Суперблок* – это наиболее важная часть файловой системы, т. к. содержит сведения, необходимые для работы с файловой системой в целом (число блоков, число i-узлов в системе и в каждой группе цилиндров и др.).

Основной структурой в файловых системах для UNIX является *i-узел* (*i-node*). Каждому файлу соответствует один и только один i-узел. В i-узле содержится информация, описывающая файл: размер файла, время создания и модификации, режимы доступа и права доступа к файлу, информация о местоположении блоков, составляющих файл, и пр. I-узлы содержатся в таблицах фиксированного размера, поэтому число i-узлов (а значит, и файлов) ограничено. Нумерация i-узлов (как и блоков) единая во всём разделе. То есть если в каждой группе по 1976 i-узлов, то i-узлы с 1 по 1976 находятся в группе № 0, i-узлы с 1977 по 3952 – в группе № 1 и т. д.

I-узел определяет блоки, в которых содержится файл. При этом номера первых 12 блоков содержатся непосредственно в i-узле, а остальные блоки адресуются косвенно через специальные таблицы. Каждая таблица занимает ровно один блок. Например, в случае блоков размером 1 Кбайт таблица косвенно адресуемых блоков (ТКАБ) 1-го уровня содержит номера следующих 256 блоков файла. Если файл размером больше 268 блоков, то заполняется таблица косвенно адресуемых блоков 2-го уровня. Она хранит номера блоков, в которых содержатся таблицы 1-го уровня косвенности, адресуя 256×256 блоков. Если же размер файла превышает 65804 блоков, то задействуется таблица косвенно адресуемых блоков 3-го уровня (рис. 2.2). Если бы для размещения файла требовалось целиком заполнить эту таблицу, то файл занимал бы на диске более 16 Гбайт. В традиционной версии EXT2 под размер файла в i-узле отводится 32 бита (макс. размер файла – 4 Гбайт). Существуют 64-битные версии этой файловой системы.

Другая важная часть i-узла – права доступа к файлу. I-узел содержит *идентификатор пользователя-владельца файла* (UID) и *идентификатор группы владельца файла* (GID). Процесс может получить доступ к файлу только в том случае, если хранящееся в слове атрибутов значение позволяет сделать это. Кроме того, в слове атрибутов

хранятся некоторые параметры управления запуском исполняемого файла, а также тип i-узла.

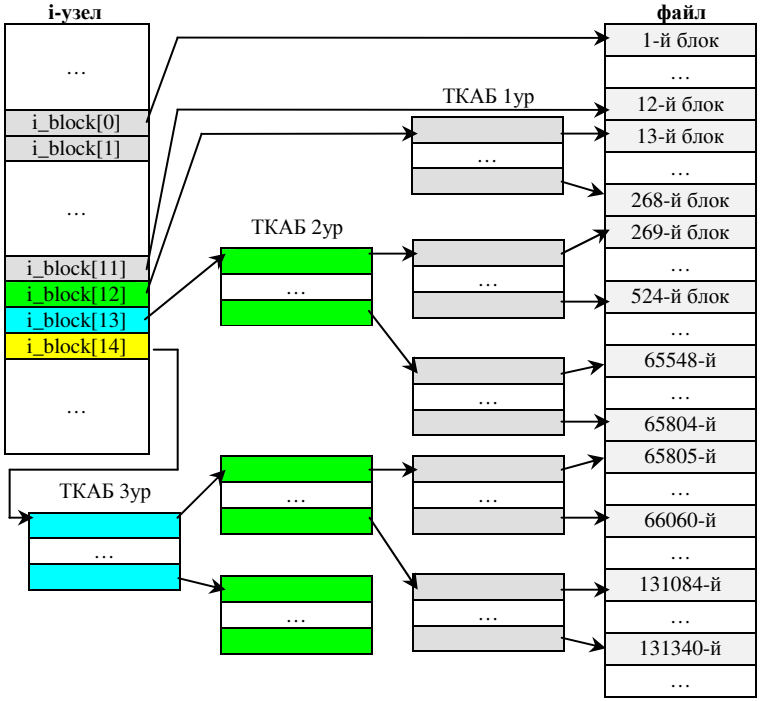

Рис. 2.2. Схема адресации блоков файла (блок=1К)

I-узел не содержит имени файла. Имя файла задаётся элементом каталога. Элемент каталога в UNIX называется *связью*, он содержит («связывает») только имя файла и его номер i-узла. Первые два элемента каталога имеют специальное назначение: элемент “.” связан с собственным i-узлом каталога, а элемент “..” связан с i-узлом родительского каталога. С каждым несвободным i-узлом должен быть связан, по крайней мере, один элемент какого-либо каталога. Такой элемент называют ещё *«жёсткой» связью*. Иначе говоря, у каждого файла может быть одно и более имён (жёстких связей). В одном из полей i-узла файла запоминается количество жёстких связей. Как только это значение становится равным нулю, i-узел и все соответствующие ему блоки данных помечаются как свободные (файл «удаляется»).

Кроме имён, которые задаются жёсткими связями, пользователь для доступа к файлу может использовать ещё так называемые *симво-*

лические связи. *Символическая связь* – специальный файл, содержащий одно из имён файла, на который эта связь ссылается. Для символической связи выделяется отдельный i-узел, поэтому символическая связь может ссылаться на файл, который не существует или расположен на другом носителе.

2.2 Логическая организация файловой системы

Один из самых простых и наиболее элегантных аспектов конструкции системы UNIX в целом – это представление всего в виде файла. Даже устройства, на которых хранятся файлы, представляются как файлы. Таким образом, файловая система UNIX организована в виде древовидной структуры каталогов, внутри которых находятся файлы и подкаталоги. Физические или логические устройства (диски, ленты) в файловой системе UNIX не указываются. При загрузке система определяет, какое запоминающее устройство будет предоставлять корневой каталог. Остальные устройства внешней памяти монтируются (подключаются) как каталоги внутри корневой файловой системы.

В UNIX можно выделить несколько классов файлов:

- *Обычный дисковый файл* – к этому классу относят обычные файлы любого формата с пользовательской информацией, в том числе исполняемые файлы и объектные модули. Структурой этих файлов управляет пользователь.
- *Каталог* – класс файлов, представляющий собой список жёстких связей, сопоставляющих имя тому или иному i-узлу.
- Кроме того, существует несколько классов *специальных файлов* – это файлы символических связей, файлы отображения внешних устройств и др. Обращение к файлу символической связи система переадресует файлу, на который эта связь ссылается. При обращении к файлу устройства система вызывает ту или иную функцию драйвера соответствующего устройства (выполняется ввод-вывод).

Корневой каталог обозначается, в отличие от DOS и Windows, косой чертой (/). Обычно корневой каталог системы GNU/Linux содержит следующие подкаталоги:

- bin содержит основные программы общего назначения;
- boot содержит файлы, использующиеся при загрузке системы;
- dev содержит файлы отображения внешних устройств;
- etc содержит конфигурационные файлы системы;
- home содержит домашние каталоги пользователей;

- lib содержит служебные файлы (системные библиотеки);
- mnt или media содержит каталоги для подключения других файловых систем;
- sbin содержит программы специального назначения;
- tmp – каталог для временных файлов;
- usr содержит прикладные программы (/usr/bin) и их данные;
- var используется системой в процессе работы для различных целей (ведение журналов, временных баз данных, кэширование данных программ и т. д.)

Максимальный размер имени файла в EXT2 – 65535 символов. Строчные и прописные буквы различаются (my.txt и my.TXT – разные имена). Имя файла может содержать любые символы, кроме слэша[3]. Файл, имя которого начинается с точки (.), считается скрытым и не отображается обычными командами просмотра каталогов.

Для пользователей DOS и Windows привычным является выделение в имени файла «расширения». В UNIX имя файла монолитно. Вместо термина «расширение» используется термин «суффикс». Использование или неиспользование суффиксов остаётся на усмотрение пользователя. При необходимости определить тип документа и связанное с ним приложение оболочка и прикладные программы зачастую делают это не по суффиксу, а по содержимому файла (по нескольким первым байтам, если это возможно).

Когда какой-либо процесс пытается получить доступ к файлу, UID и GID процесса сравниваются с UID и GID файла. В зависимости от результатов такого сравнения выбираются ограничения режима доступа. При доступе к файлу через символическую связь права самой связи игнорируются, эффективны лишь права файла, на который она ссылается. UNIX различает три режима доступа к файлу:

Таблица 2.1. Режимы доступа к файлам

Режим	Для обычных файлов	Для каталогов
r	Чтение	Получение списка файлов
w	Запись	Удаление, создание, переименование файлов в каталоге
x	Исполнение	Переход в каталог, получение атрибутов файлов

[3] Символы \ | $! & % * ( ) { } ~ ` ' " ; < > ? и пробел при использовании их в командной строке оболочки имеют специальное назначение, в таком случае их надо «экранировать» обратной косой чертой (\) или заключать имя файла в кавычки.

Процессам, запущенным от имени администратора системы (пользователя root), все файлы доступны для любого типа доступа вне зависимости от атрибутов файла. Администратор может также поменять для любого файла UID владельца, GID владельца и атрибуты доступа. Различаются три класса прав доступа к файлу. Для каждого класса доступа в атрибутах файла хранится свой набор разрешённых режимов доступа.

1. *Владелец файла* ($UID_{процесса}=UID_{файла}$). Процессы владельца файла получают доступ к файлу в соответствии с заданными для него ограничениями. Владелец файла не имеет права сменить UID владельца. Владелец может сменить GID владельца файла только на идентификатор той группы, к которой сам принадлежит. Владелец файла может менять атрибуты доступа для всех трёх классов доступа.
2. *Группа владельца файла*. Если UID процесса не совпадает с UID файла, но идентификаторы группы совпадают, процесс имеет право на доступ с заданными для этого класса ограничениями. Пользователи, входящие в эту категорию, не могут изменить UID владельца, GID владельца и атрибуты файла.
3. *Все остальные*. Если UID процесса, пытающегося получить доступ к файлу, не совпадает с UID файла и не равен 0, GID процесса и файла также различаются, то применяются соответствующие ограничения типа доступа. Пользователи, входящие в эту категорию, не могут изменить UID владельца, GID владельца и атрибуты файла.

Процесс не получает доступ к файлу, если система отказывает в доступе «*x*» в одном из родительских для данного файла каталогов.

Атрибуты доступа обозначаются следующим образом. Если для определённого класса какой-либо из режимов доступа запрещён, ставится прочерк.

r	**w**	**x**	**r**	**w**	**x**	**r**	**w**	**x**
права доступа для владельца			права доступа для группы			права доступа для остальных		

Строка «`rw-r--r--`» означает, что файл не является исполняемым, доступен владельцу для чтения и записи, остальные (в т. ч. группа владельца) имеют доступ только для чтения.

Строка «`r-xr-xr-x`» для обычного файла означает, что этот файл исполняемый и может быть запущен на исполнение любым пользователем, но не может быть модифицирован.

Строка «rwxr-x--x» для каталога означает, что владелец имеет право просматривать каталог, создавать, удалять и переименовывать файлы в каталоге, а также получать доступ к файлам в этом каталоге; пользователи из группы владельца файла могут просматривать каталог и получать доступ к файлам в этом каталоге; все остальные не имеют права просматривать каталог, но могут сделать его текущим и получать доступ к файлам в этом каталоге.

Если пользователю для каталога разрешён тип доступа «r» и «x», пользователь имеет право осуществлять в нём поиск файла (например, при помощи команды **find**), в противном случае, поиск невозможен.

2.3 Команды для работы с файлами и каталогами

pwd – вывод имени текущего каталога.

cd – смена текущего каталога. При использовании без аргументов команда переходит в домашний каталог пользователя (обычно /home/*имя_пользователя*). Альтернативное обозначение домашнего каталога пользователя ~ (тильда). Примеры:

- **cd** или **cd ~** – переход в домашний каталог пользователя;
- **cd /** – переход в корневой каталог;
- **cd ..** – переход в родительский каталог.

ls – просмотр содержимого каталога. Будучи запущенной без параметров, команда выдаёт лишь список файлов в текущем каталоге без каких-либо атрибутов. Ключ *-l* обеспечивает вывод основных атрибутов файла.

```
$ ls -l
итого 65180
drwxr-xr-x 2 pupkin pupkin     4096 Май 9 14:16 bmk/
-rw-r--r-- 1 pupkin pupkin  1033427 Фев 22 2003 etc.tgz
lrwxrwxrwx 1 pupkin pupkin       13 Сен 1 21:36 home ->
/home/pupkin/
-rw-r--r-- 1 pupkin pupkin 65626430 Фев 22 2003 home.rar
```

В первой строке выводится число блоков, занятых файлами из этого каталога. Затем идут строки с подробной информацией о файле. Первый знак описывает тип файла (табл. 2.2). Затем следуют атрибуты доступа. Следующий столбец содержит количество жёстких связей для данного файла. Третий столбец содержит имя владельца файла. Четвёртый столбец содержит группу владельца. Пятый столбец содержит размер файла. В разных системах по умолчанию размер файла

может выводиться в блоках (тогда следует добавить ключ *-h*) или байтах. Затем следует время создания или последней модификации файла. Последний столбец содержит имя файла. Для символических связей в последнем столбце (после знака –>) также указывается имя файла, на который данная связь ссылается.

Таблица 2.2. Расшифровка класса файла

–	Обычный файл
d	Каталог
l	Символическая ссылка
b	Файл блочного устройства
c	Файл символьного устройства

du – оценка пространства, занимаемого файлом или каталогом. Команда **du** выдаёт отчёт об использовании дискового пространства заданными файлами, а также каждым каталогом иерархии подкаталогов каждого указанного каталога. Опция *-s* заставляет du выводить только суммарный размер для указанных аргументов, не расписывая их по подкаталогам. Опция *-h* определяет вывод значений в наиболее удобных единицах.

```
$ du -hs mplayer
25M     mplayer
$ du -h mplayer
968K    mplayer/skins
14M     mplayer/codecs
25M     mplayer
```

quota отображает лимит дискового пространства для пользователя, а также текущий объём, занимаемый файлами данного пользователя.

cp – копирование файлов. Для рекурсивного копирования каталогов следует использовать ключ *-R*. Если указано несколько файлов или каталогов, последнее имя является именем каталога назначения. Пример:

```
cp a1.txt a2.txt a3.txt /home/pupkin
cp *.txt /home/pupkin
cp –R /home/pupkin /mnt/floppy
```

rm – удаление файлов. Для рекурсивного удаления используется ключ *-R*.

```
rm a1.txt a2.txt a3.txt
rm –i *.txt
```

mv – перемещение (переименование) файла.

mkdir – создание каталога.

rmdir – удаление каталога. Удаляет подкаталог, имя которого задаётся аргументом. Удаляемый каталог должен быть пустым.

chown – смена владельца файла. Синтаксис:

chown [*ключи*] *пользователь*[:*группа*] *файлы*

Например:

```
chown -R pupkin:pupkin /home/pupkin
```

Приведённая команда рекурсивно (ключ *-R*) устанавливает для всех файлов и подкаталогов папки /home/pupkin владельца pupkin и группу владельца pupkin.

chgrp – смена группы владельца файла. Синтаксис:

chgrp [*ключи*] *группа файлы*

chmod – смена атрибутов доступа файла. Синтаксис:

chmod [*ключи*] *атрибуты файлы*

Атрибуты могут задаваться в символьном или цифровом виде. Формат элемента символьного определения атрибутов доступа:

[**u g o a**][**+ – =**][**rwx**]

Первая часть элемента определяет класс доступа: **u** – владелец (user), **g** – группа (group), **o** – остальные (other), **a** – все (all, эквивалентно комбинации **ugo**). Если первая часть отсутствует, используется **a**, но с учётом значения **umask** (см. ниже). Вторая часть определяет, как меняется режим доступа: **+** – добавляется режим доступа, **–** – удаляется режим доступа, **=** – режим доступа устанавливается точно, как описано третьей частью. Третья часть определяет, с какими видами доступа производится операция. В команде можно использовать несколько таких элементов, разделяя их запятыми без пробелов.

Примеры:

- **chmod a+rw my.txt** – разрешить всем любой доступ к файлу my.txt;
- **chmod ug=r,o-rwx my.txt** – владельцу и группе устанавливается доступ только для чтения, всем остальным доступ запрещается;
- **chmod -R +r ./** – разрешить всем (с учётом **umask**) чтение из файлов и подкаталогов текущего каталога (рекурсивная операция: ключ *-R*).

Атрибуты можно задать в виде восьмеричного числа. Биты этого числа сопоставляются режимам доступа следующим образом:

8	7	6	5	4	3	2	1	0
$\mathbf{r_u}$	$\mathbf{w_u}$	$\mathbf{x_u}$	$\mathbf{r_g}$	$\mathbf{w_g}$	$\mathbf{x_g}$	$\mathbf{r_o}$	$\mathbf{w_o}$	$\mathbf{x_o}$

Примеры:

- **chmod 0666 my.txt** – файлу my.txt назначаются атрибуты «rw-rw-rw-»;
- **chmod 0751 mydir** – каталогу mydir назначаются атрибуты «rwxr-x--x»;
- **chmod 0600 private.txt** – файлу private.txt назначаются атрибуты «rw-------».

При создании нового файла система назначает ему атрибуты доступа в соответствии с маской, задаваемой встроенной командой **umask**. При вызове без параметров команда выводит текущую маску. Маска может быть задана в символьном виде (как для команды **chmod**) либо в виде восьмеричного числа. В последнем случае устанавливаются биты, которых не должно быть в атрибутах создаваемого файла.

```
$ umask
0002
$ umask -S
u=rwx,g=rwx,o=rx
$ touch 1.txt
$ ls -l 1.txt
-rw-rw-r-- 1  pupkin  pupkin     0  Сен 17 21:33  1.txt
$ umask 0022
$ ls > 2.txt
$ ls -l 2.txt
-rw-r--r-- 1  pupkin  pupkin   223  Сен 17 21:34  2.txt
```

cat – вывод файла на стандартный вывод. Если указано несколько файлов, они выводятся последовательно. Если в командной строке файлы не указаны, данные берутся со стандартного ввода.

more выдаёт содержимое указанных файлов поэкранно. После заполнения экрана программа приостанавливается и ожидает нажатия клавиши: [пробел] – показать следующий экран, [Enter] – показать следующую строку, [q] – выход из программы и др.

less – улучшенный вариант **more**. Допускает прокрутку не только вперёд, но и назад. Другие команды: [<] – начало файла, [>] – конец файла, [/] – поиск вперёд по шаблону, [?] – поиск назад по шаб-

лону, [n] – продолжить поиск, [N] – продолжить поиск в обратном направлении, [h] – вывести подсказку.

grep – поиск по шаблону. Если файл не указан, поиск осуществляется в данных, поступающих на стандартный ввод программы. Строки, содержащие указанный шаблон, выводятся на экран. Синтаксис команды:

grep [*ключи*] *шаблон* [*файлы*]

Например, чтобы вывести все строки, содержащие «student» в файле /etc/passwd, следует выполнить следующую команду:

```
$ grep student /etc/passwd
student:x:500:500::/home/student:/bin/bash
```

Ключ –E позволяет задавать шаблон в виде расширенных регулярных выражений. В данном случае поиск осуществлялся в данных, поступивших на стандартный ввод. Ввод данных завершается при нажатии на [Ctrl]+[d]. *Разберитесь, почему шаблону соответствуют только две строки?*

```
$ grep -E '[^[:alnum:]]{2,3}ex+' >1.txt
23ex+
--exxx
???ex
#exx
...e
^D
$ cat 1.txt
--exxx
???ex
```

find – поиск файлов по различным критериям. Синтаксис:

find [*путь поиска*] [*выражение*]

Параметр *путь поиска* задаёт начальную точку дерева каталогов, с которой осуществляется поиск (т. е. поиск будет проходить только в указанных подкаталогах). *Выражение* составляется из различных опций, определяющих критерий поиска:

- `-name` *шаблон* – поиск файла по шаблону имени файла;
- `-user` *пользователь* – поиск файлов, принадлежащих указанному пользователю;
- `-size` *тип число* – поиск по размеру и т. д.

По умолчанию имена файлов, удовлетворяющих критериям поиска, выводятся на экран. Можно также указать опции для осуществле-

ния определённых операций над найденными файлами. Для получения более подробной информации используйте **man**.

Примеры:

- **find / -name '*conf*'** – поиск всех файлов и каталогов, в названиях которых встречается последовательность символов conf;
- **find /usr -user pupkin** – поиск всех файлов в подкаталогах каталога /usr, владельцем которых является пользователь pupkin.

2.4 Файловый менеджер GNU Midnight Commander

Удобный интерфейс для навигации по файловой системе и управления файлами и каталогами предоставляет файловый менеджер GNU Midnight Commander[4] (запускается командой **mc** или **midc**). Его интерфейс аналогичен программам типа FAR, Norton Commander и т. п. (рис. 2.3). Перемещение по папкам осуществляется стрелочками [↓], [↑] и кнопкой [Enter]. Выбор активной панели – [Tab]. Клавиши [F3], [F4], [F5], [F6], [F8] используются для просмотра, редактирования, копирования, переименования и удаления выбранного файла. [F7] – создание подкаталога, [F9] – вход в меню (рис. 2.3, поз. 1), [F10] – выход из Midnight Commander.

Файлы разных классов подсвечиваются различным цветом, кроме того, Midnight Commander добавляет перед именем специального файла определённый символ (который не является частью имени), характеризующий его класс: каталог – имя белого цвета, начинается со слэша (/) (рис. 2.3, поз. 2); исполняемый файл – имя зелёного цвета, начинается с астериска (*) (рис. 2.3, поз. 5); символическая связь – начинается с тильды (~), если ссылается на каталог, или с «эт» (@), если ссылается на обычный файл (рис. 2.3, поз. 3); обычный файл – имя серого цвета. Если имя файла слишком длинное, на панели выводится начало и конец имени файла, соединённые тильдой (~) (рис. 2.3, поз. 4), а полное имя может быть отображено в информационном поле (рис. 2.3, поз. 6).

Midnight Commander поддерживает работу с архивами, как с каталогами, т. е. для просмотра архива надо в него «войти» [Enter]. Извлечение файлов из архива достигается «копированием» [F5]. Стандартные для Linux типы архивов: .tgz (.tar.gz), .tar.bz2, а также мультип-

[4] В системах BSD популярен файловый менеджер Demos Commander (**deco**), который обладает сходными возможностями.

латформенные .zip, .rar, .arj, .lhz и др. при условии, что соответствующие команды установлены в системе.

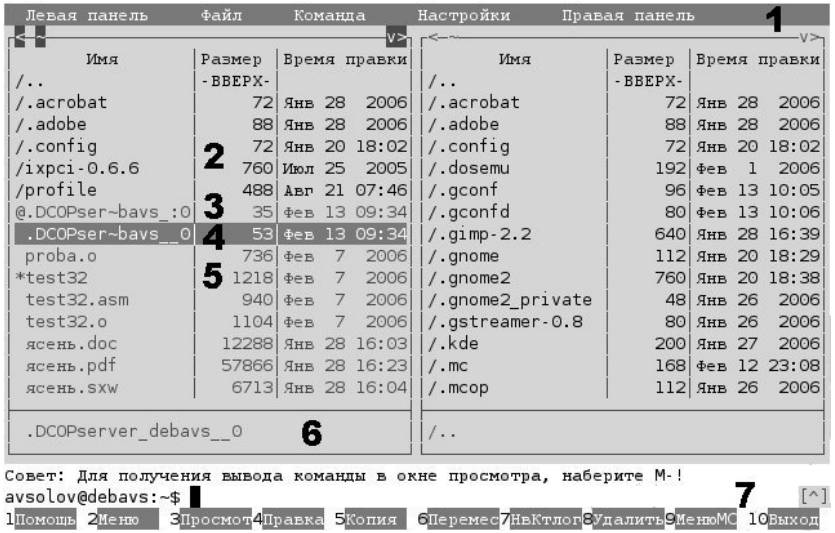

Рис. 2.3. Интерфейс программы Midnight Commander

Клавиша [Esc] в текстовых терминалах используется для задания управляющих последовательностей. Например, если вы работаете с терминалом, который не поддерживает функциональные клавиши [F1], [F2], …, то соответствующие команды для Midnight Commander можно подать при помощи [Esc]: [Esc],[1][5] соответствует [F1], [Esc],[2] соответствует [F2], … [Esc],[0] соответствует [F10]. Чтобы выйти из диалоговых окон **mc**, приходится нажимать [Esc] дважды.

Множество функций в Midnight Commander доступно через комбинации клавиш, начинающихся с клавиши [Meta]. На клавиатуре IBM PC-совместимого компьютера её роль выполняет [Alt]. Т. е. для вызова функции меню «Файл», «Quick CD» (смена каталога) в подсказке написано «M-c», что соответствует комбинации [Alt]+[c]. Для выполнения команды с отображением её стандартного вывода в окне просмотра («Файл», «Filtered View») указана комбинация «M-!», что соответствует [Alt]+[Shift]+[1].

Команды можно набирать в командной строке (рис. 2.3, поз. 7). Чтобы вставить в эту командную строку имя выделенного на панели

[5] Это означает, что клавиши нажимаются последовательно: сначала [Esc], затем [1].

файла, надо нажать [Esc],[Enter]. Midnight Commander имеет собственные средства хранения истории команд, которая вызывается комбинацией [Alt]+[h]. Выполнив команду, Midnight Commander может вывести приглашение нажать любую клавишу, прежде чем вновь отобразить панели, тем самым закрыв вывод только что завершившейся команды. Однако эта возможность определяется типом терминала и настройкой заданной через меню «Настройки», «Конфигурация», пункт «Пауза после выполнения…».

```
[pupkin@somehost homedir]$ ls
Desktop      tmp          my.txt
Для продолжения нажмите любую клавишу...
```

Другой способ скрыть панели и просмотреть вывод команд – при помощи комбинации [Ctrl]+[o]. Обратное включение панели – также [Ctrl]+[o]. В режиме, когда панели скрыты, пользователю доступна командная строка оболочки, в которой действуют стандартные для неё комбинации клавиш (история – [↑] [↓], автозавершение – [Tab] и др.). Если пользователь начнёт ввод команды в этой командной строке, то запуск команд из командной строки режима панелей становится недоступен – Midnight Commander в таком случае выдаст предупреждение (рис. 2.4). Проблема устраняется, если переключится на второй командный интерпретатор (скрыть панели при помощи [Ctrl]+[o]) и завершить начатую там команду, нажав [Enter].

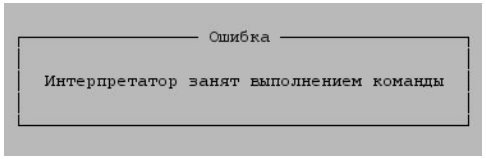

Рис. 2.4. Попытка запустить команду из режима панелей, если ввод команды во второй оболочке не завершён

Midnight Commander не обновляет информацию на панелях автоматически, если содержимое текущего каталога было изменено в процессе выполнения такой команды. Чтобы перечитать каталог, надо нажать [Ctrl]+[r].

2.5 Контроль доступа к файлам на основе POSIX ACL

Кроме традиционной схемы управления доступом к файлам на основе трёх классов доступа (владелец, группа владельца, остальные) в UNIX может быть использована схема на основе Access Control

Lists (ACL) – списков контроля доступа. Стандартизацией этой схемы занимаются группы POSIX 1003.1e и 1003.2c. Поддержка POSIX ACL внедрена в Linux, начиная с ядра 2.5.46. Существуют патчи для ядер 2.4.x, дополняющие Linux этой функциональностью. В отличие от традиционной схемы управления доступом, где режимы доступа задаются только для трёх классов, в ACL каждый элемент списка содержит набор прав доступа. Количество элементов в ACL ограничено только технической реализаций конкретной ФС (в EXT2/EXT3 размер ACL для каждого файла не должен превышать 1 блок (1–8 Кб), в других ФС – до 64 Кб). В ACL могут быть элементы следующих типов:

- *владелец* (только один элемент, содержит права доступа владельца из традиционной схемы),
- *названный пользователь* (множество элементов),
- *группа-владелец* (только один элемент, содержит права доступа группы владельца из традиционной схемы),
- *названная группа* (множество элементов),
- *маска* (только один элемент),
- *прочие* (только один элемент, содержит права доступа прочих из традиционной схемы).

Расширением традиционной схемы являются элементы «названный пользователь» и «названная группа», которые дополняют класс доступа «группа владельца»[6]. При использовании ACL традиционный класс доступа «группа владельца» отражает максимально допустимый набор режимов доступа для элементов ACL типов «названный пользователь», «группа-владелец» и «названная группа». Такой максимально допустимый набор задаётся элементом «маска». Если какой-либо из элементов этого класса содержит режим доступа, не указанный в маске, этот режим игнорируется (не предоставляется). Маска не действует на другие классы доступа («владелец» и «прочие»).

Алгоритм контроля доступа POSIX ACL:

1. Если UID процесса совпадает с UID файла, применяется элемент ACL «владелец».
2. Если UID процесса совпадает с одним из UID элементов «названный пользователь», применяется этот элемент с учётом маски.
3. Если GID процесса совпадает с GID файла, применяется эле-

[6] Обратите внимание, что в класс доступа «группа владельца» в терминологии управления доступом POSIX ACL включаются элементы «названные пользователи» и «названные группы», которые в терминах учётных записей UNIX могут не входить в группу владельца файла.

мент ACL «группа-владелец» с учётом маски.

4. Если GID процесса совпадает с одним из GID элементов «названная группа», применяется этот элемент с учётом маски.
5. В противном случае применяется элемент ACL «прочие».

Выбранный по указанному алгоритму элемент ACL определяет, разрешить запрашиваемый доступ к файлу или запретить. Таким образом, элементы ACL можно использовать как для явного разрешения доступа выбранным пользователям или группам, так и для явного запрещения.

Следующие схемы поясняют действие ACL.

Минимальный набор элементов ACL, полное соответствие традиционной схеме:

ACL	традиционная схема

`user::rw-`
`group::r--`
`other::---`
	`rw- r-- ---`

Расширенный набор элементов ACL и его сопоставление традиционной схеме:

ACL	традиционная схема

`user::rw-`
`user:pupkin:rw-`
`group::r--`
`mask::rw-`
`other::---`
	`rw-` **`rw-`** `---`

Если в примере справа добавить в ACL ещё один элемент:

`user:ivanov:rwx`,

то эффективными правами для него будут только «r» и «w», т. к. «x» для данного класса запрещён элементом «маска».

Для каталогов в ACL могут быть ещё включены элементы с пометкой default (назначаемые по умолчанию). Эти элементы не участвуют в алгоритме контроля доступа, а используются для назначения ACL дочерним элементам данного каталога при их создании.

Для манипулирования ACL используются команды:

- **`getfacl`** *`имя_файла`* – получить ACL указанного файла;
- **`setfacl -m`** *`acl_spec имя_файла`* – изменить ACL указанного файла;
- **`setfacl -x`** *`acl_spec имя_файла`* – удалить какой-либо элемент ACL.

Элементы ACL (*acl_spec*) задаются в следующем виде:

- **user:**[*uid*]**:**rwx – элемент «названный пользователь» или (если не указан uid) «владелец»,
- **group:**[*gid*]**:**rwx – элемент «названная группа» или (если не указан gid) «группа-владелец»,

- **mask**[**:**]**:**rwx – элемент «маска»,
- **other**[**:**]**:**rwx – элемент «прочие».

Примеры:

```
$ umask 0027
$ mkdir dir
$ ls -dl dir
drwxr-x---  2 student student 6 Sep  6 22:23 dir/
$ getfacl --omit-header dir
user::rwx
group::r-x
other::---
```

Этот пример показывает, что с каждым файлом всегда можно сопоставить минимальный ACL из трёх элементов.

```
$ setfacl -m user:netuser:rwx dir
$ getfacl --omit-header dir
user::rwx
user:netuser:rwx
group::r-x
mask::rwx
other::---
$ ls -dl dir
drwxrwx---+ 2 student student 6 Sep  6 22:23 dir/
```

В последнем примере разрешается полный доступ пользователю netuser. Элемент «маска» отражает все возможные режимы доступа для класса «группа владельца». При появлении расширенных элементов в списке ACL стандартные команды начинают показывать именно маску для класса доступа «группа владельца», кроме того, некоторые системы добавляют знак “+” в конце поля прав для индикации наличия расширенных элементов ACL.

Контрольные вопросы и задания

1. Просмотрите содержимое вашего каталога. Создайте внутри ещё один с любым именем. Скопируйте туда какие-нибудь текстовые файлы из общих папок и опробуйте команды перемещения и удаления. Просмотрите файлы командами **`cat`**, **`more`**, **`less`** и сравните результаты. Просмотрите информацию из справочного руководства по используемым командам.
2. Скрытые файлы (имя начинается с точки) при просмотре командой **`ls -l`** не выдаются на экран. Выясните, как можно просмотреть всё содержимое каталога.

3. Создайте пробный файл любого формата с именем test_file и поэкспериментируйте на нём с применением команды **chmod**. Каков результат команд:

```
chmod u+w,og+rx test_file
chmod 744 test_file
chmod 640 test_file
```

4. При помощи команды **grep** найдите в файле /etc/passwd всех пользователей с домашним каталогом в папке /home.
5. Объясните, что делают команды, и что будет результатом их выполнения:

```
find /tmp -type d
find /var -name log -ls
find /usr/src/linux -follow -name '*.c' -exec grep -l \
      foo '{}' ';'
```

6. С помощью команды **find** найдите в каталоге /usr все файлы, размер которых не превышает 4 Кб. На экран выведите имя файла и его размер в байтах.
7. Ознакомьтесь с работой файлового менеджера Midnight Commander. Выясните, как в нём выполняются операции копирования, перемещения файлов, создания каталогов, изменения атрибутов файлов. Изучите встроенный редактор.
8. Выясните, на какой элемент ACL влияет команда **chmod** при изменении прав доступа для класса «группа владельца» (например, **chmod g+w dir**). Сравните результат этой команды при применении к файлу с минимальным ACL и при применении к файлу с расширенными элементами ACL.
9. Выясните, какой параметр требуется указать в командной строке **setfacl**, чтобы она не изменяла элемент «маска» при добавлении новых элементов в ACL.
10. В приведённом справа примере выделите элементы, для которых эффективный режим доступа будет отличаться от указанного в элементе. Как в этом случае будут отображаться атрибуты доступа командой **ls**? Какой тип доступа могут получить пользователи petrov и ivanov, входящие в группу users?

```
# owner: student
# group: student
user::rwx
user:petrov:rwx
group::rwx
group:users:---
mask:r--
other:rwx
```

3. ПРОГРАММИРОВАНИЕ В SHELL

3.1 Общие сведения

Синтаксис, возможности и сам набор команд в системах UNIX определяется интерпретатором команд (shell, оболочка). Существует несколько разновидностей интерпретаторов, входящих в любую поставку UNIX: Bourne shell (**sh**), Korn shell (**ksh**), C shell (**csh**), Bourne again shell (**bash**), Touch shell (**tcsh**) и др. Каждый из них имеет свои функциональные возможности. В процессе работы можно переключиться с одного интерпретатора на другой, запустив соответствующую программу. Оболочка по умолчанию задаётся пользователю при создании его учётной записи в системе. Командный интерпретатор, используемый по умолчанию в GNU/Linux – **bash**.

bash – совместимый с **sh** командный интерпретатор, выполняющий команды, поступающие на стандартный ввод или из файла. Файлы, содержащие команды для интерпретатора, называют *сценариями* (script). **bash** реализует многие полезные возможности интерпретаторов **ksh** и **csh**. **bash** соответствует спецификации POSIX 1003.2 для командных интерпретаторов и системных утилит.

Все встроенные команды **bash** описаны в разделе SHELL BUILTIN COMMANDS в конце справочного руководства **man bash**. Быстрый доступ к справочной информации по той или иной встроенной команде **bash** осуществляется при помощи встроенной команды **help**.

При входе пользователя в систему **bash** выполняет сначала общесистемный сценарий /etc/profile, а затем один из пользовательских сценариев настройки[7] ~/.bash_profile, ~/.bash_login или ~/.profile (какой найдёт первым). Экземпляр оболочки, запускаемый при входе пользователя в систему, называется *начальной оболочкой* (login shell). Когда начальная оболочка завершает работу (по [Ctrl]+[d] закончился стандартный ввод или пользователь выполнил встроенные команды **exit** или **logout**[8]), сеанс работы пользователя закрывается, при этом выполняется сценарий ~/.bash_logout. Если запускаются другие экземпляры оболочки, не являющиеся начальными (но интерактивными), выполняется сценарий ~/.bashrc. Когда оболочка запускается

[7] Знаком ~ (тильда) обозначается домашний каталог пользователя (обычно /home/*user_login_name*).

[8] Команда **logout** не работает, если оболочка не является начальной.

в неинтерактивном режиме (например, для выполнения пользовательского сценария), выполняется конфигурационный сценарий, имя которого хранится в переменной окружения BASH_ENV (если только эта переменная непустая).

В конфигурационном сценарии устанавливаются значения переменных окружения. Большинство из них принимает определённые значения при старте системы, однако, пользователь может по своему желанию переопределять эти значения. Кроме того, пользователь может переопределять команды, присваивая каким-либо вариантам команд или их комбинациям псевдонимы (aliases), тем самым создавая свою систему команд по вкусу.

3.2 Переменные окружения

Переменной окружения (параметром) называется некоторый объект, имеющий значение. Параметры образуют *среду окружения* (environment), переменные которой доступны каждой запущенной из оболочки команде. Особенности функционирования многих команд зависят от тех или иных переменных окружения. Переменная имеет имя, значение и опционально атрибуты. Переменная становится доступной после присвоения ей значения. Атрибуты можно назначить переменной при объявлении её во встроенной команде **`declare`** (или **`typeset`**, **`export`**, **`readonly`**, **`local`**). Значение переменной может быть получено при помощи специального преобразования: $NAME (NAME – имя переменной):

```
$ echo $SHELL - $BASH_VERSION
/bin/bash - 2.05b.0(1)-release
$ A=123
$ echo $A
123
$ A=A+1
$ echo $A
A+1
$ declare -i A=123
$ echo $A
123
$ A=A+1
$ echo $A
124
```

Просмотреть значения всех определённых переменных можно с помощью встроенных команд **`set`** или **`declare`** без параметров.

Удалить переменную из среды окружения можно при помощи встроенной команды **unset**.

Другой способ задать значение какой-либо переменной – использовать встроенную команду **read**. В этом случае значение переменной считывается со стандартного ввода. Если команде **read** не указать никакого параметра, считанное значение присваивается переменной REPLY.

Специальный тип переменных окружения – *позиционные параметры* (*аргументы командной строки*). Позиционные параметры обозначаются цифрами (1, 2, 3 и т. д.) Значения позиционным параметрам присваиваются при запуске оболочки либо при помощи команды **set**. Непосредственным присваиванием изменить значение позиционных параметров нельзя. При получении значения позиционного параметра, обозначенного двумя и более цифрами, его обозначение следует заключать в фигурные скобки.

Некоторые параметры имеют специальное назначение в оболочке. На них можно только ссылаться, но непосредственно их значение изменить нельзя. Такие специальные параметры обозначаются различными знаками, перечисленными в таблице 3.1.

```
$ set a b c d e f g h i j k l m n
$ echo $10 ${10}
a0 j
$ set "$@"
$ echo $#
14
$ set "$*"
$ echo $#
1
```

Таблица 3.1. Обозначение специальных параметров **bash**

#	Соответствует количеству позиционных параметров
* @	Оба параметра соответствуют всему множеству аргументов командной строки, т. е. при подстановке значения этих специальных параметров подставляются все заданные аргументы командной строки. Различие наблюдается при использовании этих параметров в кавычках (см. пример выше). В этом случае значение параметра * подставляется как одно слово (т. е. как "$1 $2 $3 …"), а при использовании параметра @ позиционные параметры подставляются как отдельные слова (т. е. как "$1" "$2" "$3"…)
?	Содержит статус выхода последней выполненной команды. Нулевой статус выхода соответствует успешному завершению программы
$	Содержит PID выполняющегося командного интерпретатора
0	Содержит имя выполняющейся оболочки или сценария

Интерпретатор **bash** версии 2.0 и новее поддерживает *параметры-массивы*. Массивы в **bash** являются одномерными. Для массивов нет явного ограничения на размер, также не является обязательным требование непрерывности нумерации элементов. Нет необходимости специально объявлять массив (хотя это можно сделать при помощи **declare**) – массив создаётся при задании значения любому его элементу. Элементы массивов нумеруются с нуля. Элементу массива можно присвоить значение при помощи команды:

имя`[`*индекс*`]=`*значение*.

Чтобы задать значения нескольким элементам массива, можно использовать следующий синтаксис:

имя`=(`*значение1 значение2* …`)`,

где вместо *значения-i* может быть использовано выражение:

`[`*индекс*`]=`*значение*.

`A=(a b c)`	эквивалентно	`A[0]=a; A[1]=b; A[2]=c;`
`B=([3]=z [5]=q [13]=w)`	эквивалентно	`B[3]=z; B[5]=q; B[13]=w;`

Для получения значения элемента массива используется следующая запись: **${***имя***[***индекс***]}**. Если в качестве индекса использован символ * или @, вместо этого выражения подставляются все элементы массива. Различие в использовании * или @ наблюдается только тогда, когда это выражение заключено в кавычки. В таком случае вместо **"${***имя***[*]}"** подставляется **"${***имя***[0]} ${***имя***[1]}** …", а вместо **"${***имя***[@]}"** – **"${***имя***[0]}" "${***имя***[1]}"** … Вместо выражения **${#***имя***[***индекс***]}** подставляется длина элемента с указанным *индексом*. Если в качестве *индекса* указан символ * или @, подставляется количество элементов в массиве.

```
$ echo $BASH_VERSINFO
2
$ for i in "${BASH_VERSINFO[@]}"; do echo = $i =; done
= 2 =
= 05b =
= 0 =
= 1 =
= release =
= i586-mandrake-linux-gnu =
$ for i in "${BASH_VERSINFO[*]}"; do echo = $i =; done
= 2 05b 0 1 release i586-mandrake-linux-gnu =
```

Обратите внимание, что тело первого цикла выполняется 6 раз, а тело второго цикла – только 1 раз.

Использование имени массива без указания индекса подразумевает обращение к элементу с индексом 0.

Для удаления массива или его отдельных элементов можно использовать встроенную команду **unset**.

3.3 Псевдонимы

Псевдоним – это строка символов, подставляемая оболочкой вместо первого слова командной строки. Псевдонимы определяются при помощи встроенной команды **alias** и удаляются при помощи **unalias**. Чтобы просмотреть список определённых псевдонимов, запустите **alias** без параметров. Синтаксис команды:

alias name=value

Например:

```
$ alias DIR='ls -la'
$ DIR
итого 65188
drwxr-xr-x 3 pupkin pupkin     4096 Сен 6 21:36 ./
drwx------ 7 pupkin pupkin     4096 Сен 8 12:59 ../
drwxr-xr-x 2 pupkin pupkin     4096 Май 9 14:16 bmk/
-rw-r--r-- 1 pupkin pupkin  1033427 Фев 22 2003 etc.tgz
lrwxrwxrwx 1 pupkin pupkin       13 Сен 6 21:36 home ->
/home/pupkin/
-rw-r--r-- 1 pupkin pupkin 65626430 Фев 22 2003 home.tgz
```

Установки переменных окружения и псевдонимы определены только для текущего экземпляра оболочки. Порождённые процессы и дочерние оболочки получают в свою среду окружения параметры, для которых задано свойство экспортирования (при помощи ключа *-x* в команде **declare** или команды **export**). Однако дочерние процессы не могут изменить среду окружения родительской оболочки, и все изменения переменных и определения псевдонимов теряются при выходе из дочернего процесса.

3.4 Шаблоны

В различных конструкциях **bash** могут использоваться *шаблоны*. В шаблоне любой символ, кроме перечисленных в таблице 3.2 специальных символов, сопоставляется с самим собой. Если требуется со-

поставить себе один из специальных символов, следует применять экранирование.

Таблица 3.2. Специальные символы в шаблонах

*	Сопоставляется с любой подстрокой, в т. ч. с пустой
?	Сопоставляется с любым ровно одним символом
[…]	Сопоставляется с одним из символов, перечисленных в квадратных скобках. Можно использовать минус (–) для определения диапазона символов. Если первым символом внутри скобок является ! или ^, то шаблон сопоставляется с любым из символов, НЕ перечисленных в скобках. Внутри скобок могут использоваться идентификаторы классов символов: [:alnum:] [:alpha:] [:ascii:] [:blank:] [:cntrl:] [:digit:] [:graph:] [:lower:] [:print:] [:punct:] [:space:] [:upper:] [:word:] [:xdigit:]

3.5 Простая команда, конвейер, список, составные команды

Простая команда (simple command) может содержать следующие элементы (в порядке следования, разделённые пробелами):

- определения переменных окружения

  `[Var1=Val2] [Var2=Val2] …`

- имя команды (или путь к исполняемому файлу)
- аргументы команды (позиционные параметры)
- операции перенаправления

  `[<file1] [>file2] …`

Конец простой команды определяется одним из ограничителей:

`||  |  &  &&  ;  ;;  (  )` конец строки

Примеры:

```
ls -l >test.ls
MANWIDTH=40 man bash
GREP_COLOR=\001 GREP_OPTIONS="--color=always" grep bash \
</etc/passwd
```

Переменные, значения которым заданы в составе простой команды, попадают только в её среду окружения и никак не влияют на среду окружения оболочки.

Синтаксис конвейера (pipeline):

ПростаяКоманда1 **|** *ПростаяКоманда2* [**|** …]

При этом стандартный вывод *ПростойКоманды1* подаётся на стандартный ввод *ПростойКоманды2* и т.д. Пример:

```
cat /etc/passwd | grep bash | tr : " "
```

Список (list) – это последовательность одного или нескольких конвейеров (или простых команд), разделённых знаками:

```
;  &  &&  ||
```

Эта последовательность может быть завершена точкой с запятой (;), амперсандом (&) или ограничителем «конец строки». Ограничители «точка с запятой» и «конец строки» предназначены для простого разделения команд в списке, такие команды выполняются последовательно. Там где стоит точка с запятой, может быть конец строки и наоборот. Если команда завершается амперсандом, она выполняется в фоновом режиме. Оболочка не ожидает её завершения и приступает к выполнению следующей команды в списке.

Управляющие операции списка “&&” и “||” используются следующим образом:

команда1 **&&** *команда2*
команда1 **||** *команда2*

В первом случае *команда2* выполняется, только если *команда1* выполнилась успешно (вернула нулевой статус выхода). Во втором случае *команда2* выполняется, только если *команда1* выполнилась неуспешно (вернула ненулевой статус выхода). В примерах ниже использованы команды **true** и **false**, которые ничего не делают, а всегда возвращают определённый статус выхода: **true** возвращает статус 0 (успешный), а **false** возвращает статус 1 (неуспешный).

```
$ true && uname
Linux
$ false && uname
$ true || uname
$ false || uname
Linux
```

Составные команды (compound commands):

for *NAME* **in** *WORDS* **;** **do** *LIST* **;** **done**

В цикле выполняются команды из списка *LIST*. В каждой итерации цикла параметру *NAME* присваивается очередное слово из списка *WORDS* (параметры, разделённые пробелами). Цикл завершается, когда все слова из списка *WORDS* исчерпаны.

```
$ for A in 1 2 3 ; do echo $A ; done
1
2
3
```

for ((*EXPR1* **;** *EXPR2* **;** *EXPR3* **)) ; do** *LIST* **; done**

Перед выполнением цикла по правилам арифметических выражений вычисляется *EXPR1*. Перед выполнением очередной итерации цикла проверяется значение арифметического выражения *EXPR2*. Если его значение отлично от нуля, выполняется список *LIST* и вычисляется выражение *EXPR3*. Когда значение выражения *EXPR2* становится равным 0, цикл завершается.

```
$ for ((A=100; A>=96; A--)); do echo $A; done
100
99
98
97
96
```

if *LIST* **; then** *LIST* **;** [**elif** *LIST* **; then** *LIST* **;**] …
[**else** *LIST* **;**] **fi**

Выполняется список, стоящий после **if**. Если список выполняется успешно (статус выхода 0), то выполняется список, стоящий после **then**. В противном случае выполняется список, стоящий после **elif** или после **else**.

select *NAME* **in** *WORDS* **; do** *LIST* **; done**

На стандартный вывод ошибок выдаётся пронумерованный список слов *WORDS*. Со стандартного ввода в параметр REPLY читается строка, а затем выполняется список *LIST*. При этом если введённая строка содержит номер одного из указанных слов, то параметру *NAME* присваивается это слово, в противном случае *NAME* остаётся пустой. Эта последовательность действий выполняется до тех пор, пока не будет выполнена встроенная команда **break** или не закончится стандартный ввод.

```
$ select R in a b c ; do echo Reply:[$REPLY] R:[$R]
> if [ "$R" == a ]; then break; fi
> done
1) a
2) b
3) c
#? q
Reply:[q] R:[]
#? 2
Reply:[2] R:[b]
#? 1
Reply:[1] R:[a]
```

case *WORD* **in** *PATTERN* [**|** *PATTERN*] **)** *LIST* **;;** … **esac**

Слово *WORD* последовательно сравнивается со всеми шаблонами *PATTERN*. При нахождении соответствия, выполняется список *LIST*, стоящий после подходящего шаблона. Дальнейшие сопоставления не производятся.

```
$ case $OSTYPE in
> linux* | freebsd* ) uname;;
> *) echo Unknown system;;
> esac
Linux
```

while *LIST1* **;** **do** *LIST2* **;** **done**
until *LIST1* **;** **do** *LIST2* **;** **done**

Список *LIST2* выполняется до тех пор, пока статус выхода *LIST1* не станет отличен от нуля (для **while**) или не станет равным нулю (для **until**).

Внутри циклических команд могут быть использованы встроенные команды **break** и **continue**. Команда **break** немедленно завершает цикл, а команда **continue** вызывает немедленный переход к очередной итерации.

3.6 Преобразования

При обработке командной строки **bash**, встретив неэкранированные специальные символы, выполняет определённые преобразования.

При *преобразовании скобок* (brace expansion) выражение вида «a**{**d,c,b**}**e» заменяется на «ade ace abe». Это преобразование может быть вложенным. Сортировка по алфавиту не выполняется – сохраняется порядок слева направо.

Примеры:

```
$ echo test{1,2,3,4}
test1 test2 test3 test4
$ ls /{bin,sbin}/*sh
/bin/bash /bin/rbash /bin/tcsh /bin/csh /bin/sh
```

Если слово начинается с тильды (~), выполняется *преобразование тильды* (tilde expansion): вместо одиночной тильды подставляется путь к домашнему каталогу текущего пользователя; вместо выражения ~*username* – путь к домашнему каталогу указанного пользователя; вместо ~+ подставляется значение переменной PWD (текущий каталог); вместо ~– подставляется значение переменной OLDPWD (каталог, который был текущим до выполнения последней команды **cd**).

Преобразование параметров (parameter expansion):

Таблица 3.3. Преобразование параметров

$*NAME* или **${***NAME***}**	Вместо выражения подставляется значение параметра *NAME*. Фигурные скобки обязательны, только если параметр является позиционным с номером больше 9 или после имени параметра следует символ, который может быть интерпретирован как часть имени, но ею не является
${!*NAME***}**	*Косвенная подстановка.* Подставляется значение параметра, имя которого хранится в *NAME*
${!***PREFIX****}** или **${!*PREFIX***@}**	Подставляются все имена переменных, начинающиеся с указанного префикса
${*NAME***:-***WORD***}**	Если параметр *NAME* не установлен либо пуст, вместо выражения подставляется *WORD*, в противном случае подставляется значение параметра
${***NAME***:***OFS***}** **${*NAME***:***OFS***:***LEN***}**	*Подстановка подстроки.* Вместо выражения подставляется не более *LEN* символов из значения параметра *NAME*, начиная с символа со смещением *OFS* (смещения от 0). Если смещение отрицательно, символы отсчитываются от конца строки. Вместо *NAME* может быть * или @, тогда подставляются позиционные параметры, начиная с параметра с индексом *OFS* в количестве не более *LEN* штук. То же самое касается массивов с индексом * или @
${#*NAME***}**	Подставляется размер значения параметра *NAME* в символах. Если в качестве *NAME* использован массив, см. комментарии в п. 3.2 Переменные окружения
${***NAME***#***PATTERN***}** **${*NAME***##***PATTERN***}**	Вместо выражения подставляется значение параметра *NAME*, при этом из начала значения параметра вырезается минимальная (для #) или максимальная (для ##) подстрока, соответствующая шаблону *PATTERN*
${***NAME***%***PATTERN***}** **${*NAME***%%***PATTERN***}**	Вместо выражения подставляется значение параметра *NAME*, при этом из конца значения параметра вырезается минимальная (для %) или максимальная (для %%) подстрока, соответствующая шаблону *PATTERN*
${***NAME***/***PTRN***/***STR***}** **${*NAME***//***PTRN***/***STR***}**	Вместо выражения подставляется значение параметра *NAME*, в котором первая подстрока (для /) или все подстроки (для //) максимальной длины, соответствующие шаблону *PTRN*, заменены на *STR*. Если *STR* отсутствует, подстроки удаляются

Следующие примеры позволяют наблюдать разницу между различными вариантами преобразований:

```
$ ZZZ=1qwerty2qwerty3qwerty
$ echo ${ZZZ:14}
3qwerty
$ echo ${ZZZ:7:7}
2qwerty
$ echo ${ZZZ:(-3):2}
rt
$ echo ${ZZZ#1q*rt}
y2qwerty3qwerty
$ echo ${ZZZ##1q*rt}
y
$ echo ${ZZZ%qw*ty}
1qwerty2qwerty3
$ echo ${ZZZ%%qw*ty}
1
$ echo ${ZZZ/q/\'}
1'werty2qwerty3qwerty
$ echo ${ZZZ//q/\'}
1'werty2'werty3'werty
```

Подстановка команды (command substitution) подразумевает, что вместо выражения будет подставлен текст, выданный на стандартный вывод командой, содержащейся в выражении. Есть две равнозначные формы синтаксиса подстановки команды[9]:

`` `команда` `` $(команда)

Специальный случай подстановки команды: вместо **$(cat** *файл***)** можно использовать **$(<***файл***)**, т. е. вместо выражения будет подставлено содержимое указанного *файла*.

Арифметическая подстановка (arithmetic expansion) предназначена для вычисления арифметических выражений и подстановки результата: **$((***выражение***))**. Например:

```
$ echo 5+6=$((5+6))
5+6=11
```

Преобразование пути (pathname expansion) выполняется, если после всех преобразований слово содержит астериск (*), знак вопроса (?) или квадратные скобки [] – специальные символы шаблонов (см. п. 3.4). В таком случае слово считается шаблоном имени файла

[9] Обратите внимание, в первом случае используются не прямые апострофы (одинарные кавычки), а обратные (на клавиатуре располагаются, как правило, рядом с [Tab]).

и заменяется на все имена файлов, удовлетворяющие этому шаблону. Имена файлов подставляются в алфавитном порядке. Если не один файл этому шаблону не соответствует, то в зависимости от настроек **bash** слово может быть заменено на пустую строку или оставлено в неизменном виде. Примеры:

```
$ echo ~/.bash*
/home/pupkin/.bash_history /home/pupkin/.bash_logout
/home/pupkin/.bash_profile /home/pupkin/.bashrc
$ ls /etc/[knqwy]*
/etc/krb5.conf     /etc/nsswitch.conf
/etc/warnquota.conf  /etc/ytalkrc
/etc/ksysguarddrc  /etc/qt.fontguess   /etc/wgetrc
```

3.7 Арифметические выражения

Арифметические выражения используются в арифметических подстановках, а также во встроенной команде **let**. Вместо команды **let** "*выражение*" можно использовать следующий синтаксис:

((*выражение***))**

В любом случае такая команда вычисляет выражение по правилам для арифметических выражений и возвращает успешный статус выхода (0), если результат выражения отличен от нуля, и неуспешный статус выхода (1), если результат арифметического выражения – нуль. При вычислении арифметических выражений **bash** оперирует только целыми числами с разрядностью, соответствующей архитектуре машины (32 или 64 бита). Воспринимаются те же арифметические операции, что и в языке Си, с таким же порядком приоритета:

`++ -- ! ~ **` (возведение в степень) `* / % + - << >> <= >= < > == != & ^ | && || ?: =` (и др. виды присваивания).

В качестве операндов могут стоять переменные окружения без использования преобразования параметров. Константы, начинающиеся с 0, считаются восьмеричными. Константы, начинающиеся с 0x или 0X, считаются шестнадцатеричными. Константы, начинающиеся с *num*# (где num=2…64), считаются заданными в системе исчисления по основанию *num*. В противном случае константа считается десятичной.

Примеры:

```
$ echo $((5+6))
11
$ ((10>20))
$ echo $?
1
```

```
$ ((10<20))
$ echo $?
0
$ A=2#111
$ let A++
$ printf %o\\n $A
10
```

3.8 Условные выражения

Условные выражения указываются в составном операторе **[[**…**]]** и во встроенной команде **test** (или в её синониме **[…]**). Условные выражения используются для проверки атрибутов файлов, а также сравнения строк и чисел. Если условное выражение истинно, команда **[[…]]** или **test** даёт успешный статус выхода (0). Ложному условному выражению соответствует неуспешный статус выхода (1).

Полностью с возможностями условных выражений можно ознакомиться в **man bash**. Некоторые возможные операции перечислены в таблице 3.4.

Таблица 3.4. Опции и операнды команды **test**

-e *файл*	Истинно, если *файл* существует
-f *файл*	Истинно, если *файл* существует и является обычным файлом
-d *файл*	Истинно, если *файл* существует и является директорией
-r *файл*	Истинно, если *файл* существует и доступен для чтения
-w *файл*	Истинно, если *файл* существует и доступен для записи
-x *файл*	Истинно, если *файл* существует и является исполняемым
-s *файл*	Истинно, если *файл* существует и имеет ненулевой размер
-z *строка*	Истинно, если длина *строки* равна нулю (строка пустая)
-n *строка*	Истинно, если длина *строки* не равна нулю (строка непустая)
строка1 == *строка2* *строка1* != *строка2*	Сравнение строк на равенство или неравенство
арг1 ОР арг2	Сравнение численных аргументов

ОР может быть: **-eq** (равно), **-ne** (не равно), **-lt** (меньше), **-le** (меньше или равно), **-gt** (больше), **-ge** (больше или равно).

Обратите внимание, что в примерах вокруг квадратных скобок стоят пробелы. Они необходимы, т. к. символы “[” и “]” не являются *метасимволами* (символами, разделяющими слова).

```
$ [[ 10 -gt 20 ]]
$ echo $?
1
$ [[ 10 -lt 20 ]]
$ echo $?
0
$ if test $RANDOM -gt 15000
> then echo big
> else echo small ; fi
ok
$ if test $RANDOM -gt 15000
> then echo big
> else echo small ; fi
try again
$ test -r /etc/passwd && echo file is readable
file is readable
$ [ -r /etc/shadow ] && echo file is readable
$ [ -r /etc/shadow ] || echo file is NOT readable
file is NOT readable
```

3.9 Сценарии командного интерпретатора

Командный интерпретатор может обрабатывать команды не только со стандартного ввода (из командной строки), но и из текстового файла – *сценария*. Принято помещать в первую строку сценария путь к интерпретатору в виде:

```
#!путь опции
```

Это правило справедливо не только для **bash**, но и для других языков описания сценариев, например, Perl. Строки, начинающиеся с октоторпа (#), считаются комментариями и игнорируются (как и пустые строки). Запуск сценария на выполнение может осуществляться несколькими способами. Если файл сценария является исполняемым, то сценарий можно запустить, просто указав его имя в качестве команды. Обратите внимание, что система ищет команды только в каталогах, указанных в параметре PATH, в противном случае надо писать путь к исполняемому файлу. Для запуска сценария можно также использовать встроенные команды **source** или **.** (точка). В таком случае имя сценария передаётся как аргумент командной строки.

Пример сценария:

```
#!/bin/bash
num=${1:-0}
a='#'
for ((i=0; i<num; i++))
do
       echo $a
       a="#$a"
done
```

Если сценарий назвать myscript.sh, то в работе он будет выглядеть следующим образом:

```
$ source myscript.sh 3
#
##
###
```

Контрольные вопросы и задания

1. Выясните назначение следующих переменных: BASH, BASH_VERSINFO, BASH_VERSION, GROUPS, HOSTNAME, HOSTTYPE, MACHTYPE, OSTYPE, PPID, PWD, RANDOM, REPLY, SHLVL, UID, HOME, LANG, PATH, PS1, PS2, PS3. Запишите значение перечисленных переменных для того экземпляра оболочки, в котором вы работаете.
2. Просмотрите файлы системных настроек в вашем домашнем каталоге. Измените вид строки приглашения для вашей среды так, чтобы она содержала время.
3. Назначьте командам **cp**, **mv** и **rm** в качестве псевдонимов их аналоги из DOS.
4. В разделе «Выполнение команды» (COMMAND EXECUTION) справочного руководства по **bash** выясните приоритет, в котором выполняется поиск соответствия имени команды. Выясните назначение встроенных команд **builtin**, **command**, **enable** и **type**.
5. При помощи преобразования **${!***PREFIX****}** и цикла **for** выведите значения всех переменных, имена которых начинаются на BASH.
6. Что помещается в переменную A при присваивании `A=`ls`` ? Что помещается в переменную A при присваивании `A=`<~/.bash_history`` ?
7. Выясните назначение встроенной команды **ulimit**.

4. СЕТЕВЫЕ ВОЗМОЖНОСТИ UNIX

4.1 Служба DNS

В основе сетевых возможностей UNIX лежит стек протоколов TCP/IP, образующих основу Интернет. Протокол IP обеспечивает однозначную идентификацию сетевого интерфейса в глобальной сети. Значение IP-адреса состоит из четырех чисел в пределах от 0 до 255, разделённых точками (например, 192.168.12.134). Чаще всего компьютер (хост в сети Интернет) имеет всего один сетевой интерфейс, и поэтому говорят про IP-адрес хоста. Кроме IP-адресов используются имена хостов (доменные имена). Для большинства хостов установлено специальное отображение между именем хоста и его IP-адресом. Такое соответствие возможно благодаря работе DNS (Domain Name Service – служба имён доменов). Если DNS содержит запись о том, что *www.domain.com* соответствует 192.168.42.7, то при обращении к сервисам этого хоста можно использовать данное доменное имя.

Для получения информации из DNS используются команды **`dig`** и **`host`**. Эти команды имеют множество параметров. В самом простейшем случае в качестве параметра указывается имя хоста, DNS-запись о котором вы хотите получить. Например, получить DNS-запись о хосте *ya.ru*:

```
$ dig ya.ru

; <<>> DiG 9.2.4 <<>> ya.ru
;; global options:  printcmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 453
;; flags: qr rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 2

;; QUESTION SECTION:
;ya.ru.                 IN  A

;; ANSWER SECTION:
ya.ru.             306  IN  A   213.180.204.8

;; AUTHORITY SECTION:
ya.ru.             306  IN  NS  ns1.yandex.ru.
ya.ru.             306  IN  NS  ns5.yandex.ru.

;; Query time: 5 msec
;; SERVER: 194.85.172.133#53(ns.karelia.ru)
;; WHEN: Mon Jan 28 23:47:04 2008
;; MSG SIZE  rcvd: 82
```

Ответ команды **dig** состоит из нескольких секций. Сначала идёт секция с общей информацией. Затем идёт секция, описывающая запрос к службе DNS (QUERY SECTION). Третья секция (ANSWER SECTION), собственно, и содержит требуемую информацию. Наибольший интерес представляют первая колонка (доменное имя) и последняя колонка (IP-адрес). В следующей секции (AUTHORITY SECTION) сообщается, какие DNS-хосты предоставили эту информацию. В отчёте программы может присутствовать секция с дополнительными DNS-записями, касающимися данного запроса.

По умолчанию результат работы команды **host** не содержит такого обилия информации:

```
$ host ya.ru
 ya.ru has address 213.180.204.8
```

С помощью **dig** и **host** можно выполнять обратный поиск (reverse mapping) – определение доменного имени по IP-адресу.

```
$ dig -x 213.180.204.8
 ; <<>> DiG 9.2.4 <<>> -x 213.180.204.8
 ;; global options:  printcmd
 ;; Got answer:
 ;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 197
 ;; flags: qr rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 2
 ;; QUESTION SECTION:
 ;8.204.180.213.in-addr.arpa.      IN  PTR
 ;; ANSWER SECTION:
 8.204.180.213.in-addr.arpa. 14384 IN  PTR  ya.ru.
 ;; AUTHORITY SECTION:
 204.180.213.in-addr.arpa.   85632 IN NS ns2.yandex.net.
 204.180.213.in-addr.arpa.   85632 IN NS ns1.yandex.net.
 ;; Query time: 4 msec
 ;; SERVER: 194.85.172.133#53(ns.karelia.ru)
 ;; WHEN: Sat Jan 31 23:56:56 2009
 ;; MSG SIZE  rcvd: 109 ;; MSG SIZE  rcvd: 91
$ host 213.180.204.8
 8.204.180.213.in-addr.arpa domain name pointer ya.ru.
```

4.2 Команда `ping`

С помощью команды **ping** можно проверить наличие соединения с тем или иным хостом. Работа команды состоит в том, что по указанному адресу посылаются небольшие порции данных (пакеты) и реги-

стрируется их благополучный приход обратно. Если связи нет (линия отсутствует или система назначения выключена), то пакеты не возвращаются. Работа команды прерывается комбинацией клавиш [Ctrl]+[c]. При этом выдаётся статистика:

```
$ ping ns.karelia.ru
PING ns.karelia.ru(194.85.172.133) 56(84) bytes of data.
64 bytes from ns.karelia.ru (194.85.172.133): icmp_seq=0
ttl=62 time=16.5 ms
64 bytes from ns.karelia.ru (194.85.172.133): icmp_seq=1
ttl=62 time=8.72 ms
64 bytes from ns.karelia.ru (194.85.172.133): icmp_seq=2
ttl=62 time=20.2 ms
^C
--- ns.karelia.ru ping statistics ---
3 packets transmitted, 3 received, 0% loss, time 2004ms
rtt min/avg/max/mdev = 8.724/15.154/20.217/4.791 ms
```

Следует отметить, что не всякая система отвечает на запросы команды **ping**, что, тем не менее не является показателем её неработоспособности. Возможность ответа системы на запросы команды **ping** определяется её конфигурацией (например, настройками сетевого экрана – брэндмауэра).

4.3 Обмен сообщениями между пользователями

Обмен краткими сообщениями обеспечивает команда **write.** Текст сообщения берётся со стандартного ввода команды (для завершения ввода можно использовать [Ctrl]+[d]). Общий синтаксис:

write *имя_пользователя*

При использовании **write** адресат получает уведомление:

```
Write: Message from имя_пользователя@имя_хоста
```

после которого следует текст сообщения. О завершении текста сигнализирует появление «EOF». Возникающие сообщения могут смешаться с набираемыми пользователем командами и результатом их работы, поэтому опытные пользователи открывают одновременно несколько сеансов, один из которых используется только для обмена сообщениями. Кроме того, пользователь может заблокировать обмен сообщениями при помощи команды **mesg n**. В таком случае он не сможет ни сам отправить сообщения, ни другой пользователь не сможет побеспокоить его своими сообщениями. Отмена блокировки производится командой **mesg y**.

Возможность интерактивного режима обмена информацией предоставляет команда

talk *имя_пользователя@имя_хоста*

Эта команда выводит отправителя и адресата из режима командной строки. При вызове **talk** у отправителя экран делится на две части по горизонтали и в верхнем левом углу появляется надпись «Waiting for respond». Одновременно с этим у адресата на экране появляется сообщение:

```
talk: Message from talk daemon on имя_хоста ...
talk: Answer to имя_пользователя@имя_хоста ...
```

сопровождаемое звуковым сигналом. Если адресат не реагирует, то это сообщение будет появляться каждые 3 секунды до тех пор, пока отправитель не прекратит работу программы **talk** нажатием [Ctrl]+[c]. Если адресат хочет установить соединение, он должен при появлении вышеуказанного сообщения также вызвать **talk** с указанием абонента. При этом у обоих участников связи экран делится на две части по горизонтали и в верхнем левом углу появляется сообщение «Connection established», после чего оба участника могут набирать текст своих сообщений на экране одновременно. Для каждого из участников текст, набираемый им, отображается в верхней половине экране, а текст, набираемый его абонентом – в нижней половине. Выход из программы **talk** осуществляется по [Ctrl]+[c]. Если один из участников выходит из программы, другой видит в верхнем левом углу экрана сообщение «Connection closed. Exiting» и программа **talk** с его стороны прекращает работу.

Командами **write** и **talk** можно пользоваться, если существует уверенность, что те пользователи, с которыми есть необходимость пообщаться, в данный момент подключены к системе (см. команды **who** и **finger**).

4.4 Средства удалённого доступа

TELNET – это прикладной протокол, позволяющей любой машине работать в качестве удалённого *терминала* UNIX-хоста. Программа, реализующая этот протокол (обычно она называется **telnet**), посылает коды нажатых пользователем клавиш удалённой машине под управлением UNIX и выводит на экран символы, получаемые от удалённого хоста. Это позволяет войти на другую машину и работать на ней, как будто вы сидите за её системной консолью.

В большинстве операционных систем синтаксис команды таков: **`telnet`** *`сервер`*. Чтобы завершить работу **`telnet`**, используются команды **`exit`** или **`logout`**.

В настоящее время протокол TELNET используется всё реже и реже. Это объясняется остро стоящими вопросами безопасности. Все данные передаются в открытом виде (рис. 4.1). Серверу передаются символы, соответствующие нажатым на клавиатуре клиента клавишам, а клиент получает от сервера символы для вывода на экран. Информация уязвима для перехвата. Если злоумышленник перехватывает сетевые пакеты, ему могут достаться пароли и другая персональная информация пользователей. На смену TELNET приходит протокол SSH, обеспечивающий шифрование передаваемых данных (рис. 4.2).

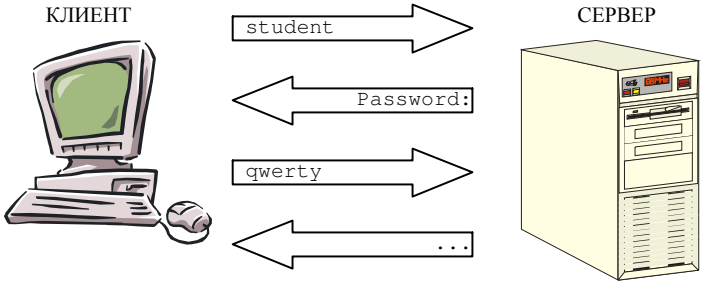

Рис. 4.1. Взаимодействие клиента и сервера по протоколу TELNET

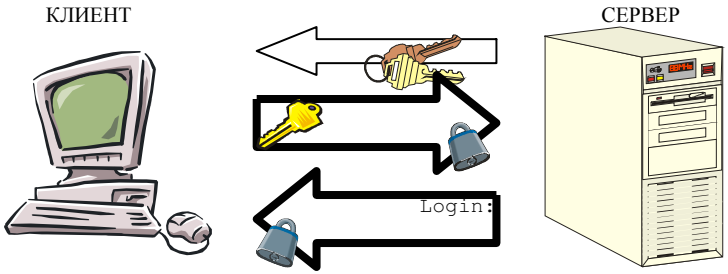

Рис. 4.2. Взаимодействие клиента и сервера по протоколу SSH

Работа SSH основана на использовании симметричных и асимметричных алгоритмов шифрования. Класс симметричных алгоритмов характеризуется тем, что для шифрования и расшифровки используется один и тот же ключ. Он секретен для стороннего наблюдателя, но его должны знать и отправитель, и получатель сообщения. Симметричные алгоритмы используются для быстрого шифрования.

В асимметричном шифровании используется два ключа – один для шифрования, другой для расшифровки. В этом случае один из ключей является секретным, а другой общедоступным (открытым), поэтому такой класс алгоритмов ещё называют шифрованием с открытым ключом. При этом не должно существовать эффективной процедуры для получения секретного ключа по известному открытому ключу. В симметричной криптографии есть принципиально не раскрываемые шифры (так называемые теоретически стойкие или совершенно секретные системы Клода Шеннона), правда, для этого нужно, чтобы размер ключа был равен объёму текста. Асимметричные шифры раскрываемы в принципе, а знание секретного ключа лишь ускоряет дешифрование. Суть технологии асимметричного шифрования заключается в использовании такого алгоритма, вычисление которого со знанием некоторого секрета имело бы полиномиальную сложность ($\sim n^a$), а без этого знания – экспоненциальную ($\sim e^n$). Увеличивая число разрядов (n), можно добиться того, что решение задачи на современных вычислительных устройствах без знания секрета потребует нескольких сотен или даже тысяч лет, а знание секрета позволяет решить эту же задачу достаточно быстро. Асимметричные криптоалгоритмы обычно применяют к небольшим объёмам данных.

Схематично описать процедуру установления соединения по протоколу SSH можно следующим образом (рис. 4.2):

1. Сервер передаёт клиенту свой открытый ключ.
2. а) Клиент генерирует ключ сессии для симметричного криптоалгоритма.
   б) Клиент шифрует ключ сессии открытым ключом сервера по асимметричному криптоалгоритму (RSA) и пересылает это зашифрованное сообщение серверу.
   в) Сервер получает и расшифровывает ключ сессии при помощи своего секретного ключа.
3. Аутентификация пользователя (передача имени, пароля) и дальнейший обмен данными происходит в виде сообщений, зашифрованных при помощи ключа сессии по симметричному криптоалгоритму (AES, IDEA, 3DES, RC4 и др.)

4.5 Передача файлов

Протокол FTP (File Transfer Protocol) предназначен для передачи файлов в глобальной сети Интернет. Сами программы, реализующие данный протокол для различных систем, как правило, носят то же самое имя – **`ftp`**. Используя **`ftp`**, вы можете копировать файлы с уда-

лённой системы (на профессиональном жаргоне – «скачивать») либо на удалённую систему (т. е. загружать их туда). Набор команд **ftp** в различных реализациях программы не совсем одинаков. Принципы сохранены, и основные команды одни, но некоторые тонкости в зависимости от системы всегда есть.

Для получения помощи по конкретной команде предназначена команда **help** или **?**. При использовании без аргумента выводится полный список всех команд.

Открыть соединение с хостом – команда **open** *адрес хоста*. Чтобы продолжить работу вам необходимо либо зарегистрироваться в системе под своим именем, либо использовать анонимное подключение. В первом варианте вы вводите имя своей учётной записи (login) и пароль. Во втором варианте вместо имени вы вводите «anonymous» (или «ftp»), а в качестве пароля – свой электронный адрес (некоторые системы разрешают в этом случае оставлять пароль пустым). Не все системы разрешают анонимное подключение.

Команды для просмотра содержимого и смены текущего каталога идентичны уже знакомым командам – **ls** и **cd** (в некоторых FTP-клиентах – **dir** и **cdup**, соответственно).

Перед загрузкой файлов следует обратить внимание на режим загрузки файлов. FTP различает два режима загрузки файлов: бинарный (image, binary) и текстовый (ascii). При использовании текстового режима передаваемые данные могут подвергаться перекодированию между текстовыми форматами разных систем. Например, в UNIX строки текстовых файлов завершаются только символом «перевод строки» (ASCII 10), а в MS-DOS и Windows строки завершаются парой символов: «возврат каретки» (ASCII 13) и «перевод строки» (ASCII 10). FTP позволяет автоматически выполнять требуемое преобразование. При передаче в бинарном режиме никаких преобразований не осуществляется – файл просто передаётся байт за байтом. Если попытаться передать в текстовом режиме файл нетекстового формата (например, архив, исполняемый файл и т. п.), он будет испорчен. Если возникают сомнения – используйте бинарный режим. Команды **binary** и **ascii** используются для перевода, соответственно, в бинарный режим передачи файлов и в текстовый.

Командой **hash** можно включить вывод меток. Метки выводятся на экран при передаче очередного блока информации и отражают ход передачи информации.

Ваш локальный каталог – это каталог вашей системы, куда вы хотите в конечном счёте сохранить файлы. В то время как команда **cd**

меняет каталог удалённой машины (машины, на которую вы вошли по FTP), командой **lcd** можно сменить локальный каталог. Кроме **lcd** есть и другие команды для работы с локальным диском.

Процесс загрузки файла с удалённой системы на локальную осуществляется командой

get *имя_файла_на_удалённой_машине* [*локальное_имя*]

Чтобы загрузить файл на удалённую систему, используется команда **put**.

Для прекращения FTP-сессии используется команда **quit** или **bye**. Команда **close** может использоваться для закрытия связи с данным FTP-сервером, но без выхода из программы **ftp**. Затем команда **open** может быть использована для начала новой FTP-сессии.

Протокол FTP так же, как и TELNET, передаёт данные (в том числе при регистрации пользователя) в открытом виде. Если это неприемлемо, используется протокол SFTP (Secure FTP), реализованный как подсистема протокола SSH. Для работы с протоколом SFTP используются программы **sftp**, **scp** и др.

4.6 Электронная почта (E-mail)

Электронная почта (e-mail) – один из самых распространённых методов обмена информацией в сетевых системах. Принципы работы с электронной почтой для пользователя напоминают традиционную бумажную почту. Получив учётную запись в системе UNIX, зачастую вы автоматически получаете почтовый ящик с именем *имя@хост* и возможность обмена электронной почтой. Ваш электронный адрес складывается из вашего регистрационного имени и имени хоста. И то и другое нетрудно узнать посредством команд **whoami** и **hostname**. Каждый раз система, получая почту на ваш адрес, будет помещать её в ваш почтовый ящик. Почтовый ящик – это обыкновенный файл.

Основную роль в работе почты играет программа **sendmail**. Она осуществляет приём почты для всех пользователей системы, «раскладывает» почту по почтовым ящикам, а также принимает от вашего почтового клиента исходящую почту. Самые простые почтовые клиенты для UNIX – программы **mail** или **mailx**. Доступ к вашему почтовому ящику можно получить с удалённой машины при помощи протоколов POP3 или IMAP4. Наиболее популярные почтовые клиенты для Windows, реализующие эти протоколы: The Bat!, Mozilla Thunderbird, Outlook Express и др. В текстовом терминале UNIX обычно для этих целей используют программы **pine**, **elm** или **mutt**.

Контрольные вопросы и задания

1. Просмотрите справочное руководство по командам **dig** и **host**. Определите IP-адреса указанных преподавателем хостов. Выполните обратный поиск.
2. Посредством команд **write** и **talk** произведите обмен сообщениями с пользователями, работающими в сети.
3. Изучите работу команды **ftp**. Загрузите несколько небольших файлов с одного из анонимных FTP-серверов. По протоколу **sftp** осуществите обмен файлами с указанной преподавателем машиной.
4. Воспользуйтесь командой **ssh** и произведите подключение к указанному преподавателем удалённому серверу UNIX. По справочному руководству **ssh** изучите альтернативные способы аутентификации в системе (без пароля при помощи ключей асимметричных криптоалгоритмов – SSH version 2, the public key method). Настройте указанную преподавателем систему на такой способ аутентификации.
5. Воспользуйтесь любой почтовой программой (например, **pine**) для чтения собственной почты и её отправки. Воспользуйтесь справочным руководством и встроенной помощью.
6. Изучите работу текстовых web-браузеров **links** или **lynx**. Выясните, как в них задаётся прокси-сервер, как выбирается кодировка страниц.

5. МОНТИРОВАНИЕ ФАЙЛОВЫХ СИСТЕМ. СЕТЕВЫЕ ФАЙЛОВЫЕ СИСТЕМЫ

5.1 Общие сведения

UNIX предоставляет пользовательским программам интерфейс Virtual File System, который скрывает от программ физические реализации файловых систем (ФС) и позволяет одновременно использовать в системе различные ФС. При загрузке основная ФС (в GNU/Linux это обычно система типа EXT2) объявляется корневой. Если при работе возникнет необходимость в использовании другой ФС (другого типа или на другом физическом носителе), то такая ФС может быть присоединена к основному дереву каталогов при помощи операции, называемой *монтированием*. Для выполнения монтирования необходимо указать устройство (специальный файл, соответствующий физическому носителю), точку монтирования (каталог основной ФС, на который будет отображено дерево каталогов монтируемой ФС) и тип монтируемой ФС. Таким образом, для пользовательских программ работа с файлами и каталогами происходит единообразно в рамках единой древовидной структуры независимо от количества и типов присутствующих запоминающих устройств и ФС.

В качестве устройства может быть указан один из специальных файлов, соответствующих блочным устройствам ввода-вывода, сетевой путь или обычный файл, содержащий образ какой-либо ФС. В GNU/Linux накопителям, присоединённым по интерфейсу IDE, соответствуют специальные файлы /dev/hda (primary master), /dev/hdb (primary slave), /dev/hdc (secondary master) и /dev/hdd (secondary slave). Отдельным разделам на таких накопителях соответствуют имена /dev/hda1, /dev/hda2 и т. д. Первичным разделам присваиваются номера 1–4, а расширенным – начиная с 5. Таким образом, для монтирования первого раздела первичного устройства на первом канале IDE (primary master) надо указать файл /dev/hda1, а для монтирования компакт-диска в CD-приводе, подключенном первичным устройством на втором канале IDE (secondary master), надо указать файл /dev/hdc (номера разделов к компакт-диску неприменимы). Для флоппи-дисков используются специальные файлы /dev/fd0 (первый дисковод), /dev/fd1 и т. д. Накопители, подключенные через интерфейс SCSI, соответствуют файлам /dev/sda, /dev/sdb,... (/dev/sda1, /dev/sda2,... – разделы первого диска) и /dev/scd0, /dev/scd1,… (CD-приводы). Драйверы

специальных устройств (CD-RW привод с интерфейсом ATAPI, картридеры и др.), как правило, эмулируют программный интерфейс SCSI и поэтому доступны в GNU/Linux под именами /dev/sd* и /dev/scd*. В других системах типа UNIX используются иные имена устройств.

В качестве точки монтирования должен быть указан реально существующий в основной файловой системе каталог. На время, пока будет примонтирована внешняя ФС, файлы, находящиеся в этом каталоге, будут недоступны. Поэтому желательно в качестве точки монтирования указывать пустой каталог.

Некоторые типы ФС перечислены в таблице 5.1

Таблица 5.1. Обозначения файловых систем

ext2	Second Extended File System (ФС GNU/Linux)
ext3	Journaling Extended File System (журналируемая ФС GNU/Linux)
ufs	Unix File System (ФС для операционных систем на основе BSD)
iso9660	Файловая система компакт-дисков
udf	Пакетная ФС для компакт-дисков
msdos	Файловая система FAT для MS-DOS
vfat	ФС VFAT для Windows с поддержкой длинных имён и FAT32
ntfs	Файловая система NTFS для Windows NT
hpfs	Файловая система HPFS для OS/2
nfs	Сетевая ФС NFS
smbfs	Сетевая ФС SMB для сетей Microsoft Windows
ncpfs	Сетевая ФС NCP для сетей Novell Netware

Монтирование осуществляется командой **mount**.

mount [*-t тип ФС*] [*-o опции*] *устройство точка_монтир-я*

Монтировать файловые системы обычно может только привилегированный пользователь (root), но при необходимости тем или иным способом права для выполнения этой операции могут быть переданы любому пользователю. Команда **mount**, выполняемая без параметров, выводит список всех ФС, смонтированных в данный момент.

Постоянно используемые ФС можно монтировать при каждом запуске автоматически. Для этого параметры монтирования надо указать в файле /etc/fstab. Если в этом файле в настройках какой-либо ФС установлена опция *user*, то такую ФС может монтировать и демонтировать обычный пользователь.

Пример монтирования первого раздела диска, подключенного вторичным устройством на первом канале IDE (primary slave), с файловой системой NTFS (система NTFS поддерживается в режиме «только чтение»):

```
mount -t ntfs -o ro /dev/hdb1 /mnt/win2k
```

Аналогичным образом монтируются съёмные носители: дискеты, компакт-диски и т. д.

Монтирование дискеты:

```
mount -t vfat /dev/fd0 /mnt/floppy
```

Для съёмных носителей необходимо помнить, что пока система считается примонтированной, часть данных может находиться в дисковых кэшах, а не на самом носителе. Чтобы обеспечить сброс данных в кэше на носитель и отсоединить ФС от точки монтирования, используется команда **umount**:

umount *точка_монтирования*

Очень важно перед извлечением съёмного носителя демонтировать находящуюся на нём ФС, в противном случае может быть нарушена целостность ФС. Все накопители автоматически демонтируются при завершении работы системы (перезагрузке или выключении компьютера).

Ручное монтирование и демонтирование съёмных носителей осложняет жизнь обычных пользователей. В GNU/Linux для решения этой проблемы предложено несколько технологий.

- Драйвер специальной файловой системы *supermount* позволяет автоматически монтировать носитель при обращении программ к соответствующему каталогу. Все операции со смонтированным ресурсом выполняются синхронно, поэтому носитель можно извлекать, как только операция закончена. Отпадает необходимость в выполнении специальных действий при установке и извлечении съёмного носителя (дискеты, компакт-диска). Надо лишь в файле /etc/fstab указать необходимые параметры монтирования. Особенно активно эта технология использовалась в дистрибьютивах Mandrake (Mandriva) Linux на основе ядер 2.4.x. Для её использования необходимо специально модифицированное ядро Linux (патч supermount).
- Технология automount/autofs подразумевает запуск специального сервиса (демона), который следит за обращениями пользователя к заданным в настройках демона файловым системам. Если файловая система не смонтирована, демон пытается её примонтировать. Демонтирование происходит по истечении заданного таймаута бездействия.
- Технология submount представляет собой композицию первых двух методов. Для файловых систем съёмных носителей

используется специальная ФС – *subfs*, реализованная в виде модуля ядра. Кроме того, для работы необходим демон submoutd. Преимуществом по сравнению с supermount является то, что ядро не надо модифицировать. Данная технология применима для ядер Linux 2.6.x.

- При работе в графической среде KDE или GNOME используется другой подход к монтированию съёмных носителей. Специальный демон HAL (hardware abstraction layer) взаимодействует с системой управления устройствами udev (ядро Linux 2.6.x), которая информирует его о вставке или извлечении съёмных носителей. Этот демон через механизм межпроцессных коммуникаций D-BUS информирует об этих событиях графическую среду. В ответ на эти события графическая среда может запустить команду mount с необходимыми параметрами, запросив подтверждение у пользователя или автоматически.

5.2 Прочие команды работы с дисками

fdisk или **cfdisk** – просмотр и изменение разделов на винчестере.

mkfs – создание файловой системы (форматирование раздела).

Например, форматирование дискеты в формате ext2:

```
mkfs -t ext2 -m0 /dev/fd0
```

df – отчёт об использовании дискового пространства на примонтированных файловых системах.

fsck – проверка целостности файловой системы.

5.3 Сетевые файловые системы

В UNIX традиционной сетевой ФС считается NFS (Network File System). Этот протокол позволяет монтировать ФС с удалённого компьютера так, как будто она локальная и находится в вашей системе. После такого монтирования вы можете непосредственно обращаться к файлам этой удалённой ФС. Преимущество состоит в том, что различные компьютеры могут получать прямой доступ к одним и тем же файлам без необходимости создания их копий. Существует только один экземпляр файла, находящийся в удалённой файловой системе, и к нему может обращаться любой компьютер.

Система NFS работает в сети TCP/IP. Удалённый компьютер, на котором находится файловая система, предоставляет её другим машинам в сети. Эта возможность называется *экспортированием* файловой системы. Её осуществляют два процесса-демона, обслуживающие запросы удалённых компьютеров (это программы rpc.mountd и rpc.nfsd). Параметры экспортирования хранятся в файле конфигурации NFS (/etc/exports). В каждой строке файла /etc/exports указывается экспортируемый каталог и сетевые компьютеры, которые имеют право доступа к ней.

Прежде чем начать пользоваться удалённой ФС, её нужно смонтировать на локальном компьютере. Удалённую файловую систему можно смонтировать при загрузке компьютера, если имеется соответствующая запись в файле /etc/fstab, либо явно командой **mount**. В качестве типа файловой системы нужно указать *nfs*. В качестве устройства монтирования указывается сетевой путь, состоящий из имени удалённого компьютера и полного пути каталога, экспортируемого удалённым компьютером. Эти имена разделяются двоеточием. Например, имя `rose.berkeley.edu:/horce/project` относится к экспортируемому каталогу /home/project на компьютере *rose.berkeley.edu*.

Существует несколько специальных опций монтирования NFS, которые можно указать в файле /etc/fstab. Допускается, в частности, указание размера передаваемых и принимаемых дейтаграмм, а также периода, в течение которого компьютер будет ждать ответа от удалённой системы. Можно также указать режим монтирования файловой системы – *hard* или *soft*. Если система смонтирована в режиме *hard*, то в случае, если удалённая система не отвечает, ОС будет непрерывно пытаться установить соединение с нею. При монтировании в режиме *soft* ОС прекращает попытки и выдаёт сообщение об ошибке. По умолчанию осуществляется монтирование в режиме *hard*.

Ниже приведён пример для явного монтирования сетевой ФС. Удалённая система (*pluto*) экспортирует каталог /home/work. В локальной системе эта ФС будет смонтирована в каталог /mnt/nfs.

```
mount –t nfs –o timeo=20 pluto:/home/work /mnt/nfs
```

Кроме сетевой ФС NFS GNU/Linux позволяет работать с сетевыми ресурсами сетей Microsoft Windows и Novell Netware.

Для GNU/Linux доступ к сетевым ресурсам Microsoft Wndows реализован посредством сетевой ФС *smbfs*[10] (*cifs*). При вызове команды **mount** с типом файловой системы *smbfs* запускается программа

[10] С версии ядра Linux 2.6.20 драйвер ФС *smbfs* заменён на *cifs*.

smbmount. Имя пользователя и пароль можно передавать в опциях этой файловой системы. При демонтировании вызывается команда **smbumount**. Кроме того, можно использовать программу **smbclient** для просмотра ресурсов, предоставляемых тем или иным компьютером, или для обмена сообщениями WinPopup. В сетевых путях, принятых в MS Windows (UNC), следует обратные слэши заменять обычнымии. Примеры:

```
mount -t smbfs -o ro,username=Admin //MYWIN/C\$ /mnt/smb
```

Ресурс \\MYWIN\C$ будет примонтирован в каталоге /mnt/smb в режиме «только чтение». Если удалённая машина потребует аутентификации, в качестве имени пользователя будет передано Admin. GNU/Linux будет пытаться определить адрес удалённой машины по её NetBIOS-имени (MYWIN).

```
smbmount //LABWIN/SHARE /mnt/share -o \
    ip=192.168.1.1,uid=root,gid=samba
```

Ресурс \\LABWIN\SHARE будет примонтирован в каталоге /mnt/share. Для файлов в примонтированной системе владельцем будет считаться пользователь root, группой владельца – samba. GNU/Linux не будет пытаться определять адрес удалённой машины по её NetBIOS-имени (LABWIN), а будет считать адресом этого хоста IP-адрес 192.168.1.1.

```
umount /mnt/share
smbumount /mnt/smb
```

Демонтирование файловых систем SMB можно делать как при помощи команды **umount**, так и при помощи **smbumount**.

```
smbclient -M JUPITER
```

Программа будет пытаться отправить сообщение WinPopup компьютеру JUPITER.

При помощи пакета программ Samba компьютер под управлением GNU/Linux может предоставлять свои ресурсы в сеть Microsoft Windows и выполнять функции контроллера домена Windows NT.

Сетевые ресурсы Novell Netware в GNU/Linux представлены как сетевая ФС *ncpfs*. При вызове команды **mount** с типом ФС ncpfs запускается программа **ncpmount**. Имя пользователя и пароль можно передавать в опциях монтирования этой ФС либо поместить в конфигурационный файл .nwclient в домашнем каталоге пользователя. При демонтировании вызывается команда **ncpumount**. Кроме того, суще-

ствует ряд утилит для работы с сетью Novell Netware: **slist**, **nwfsinfo**, **nwuserlist**, **nwrights**, **ncopy**, **nprint**, **nsend** и др. Сетевой путь в команде **mount** обозначается следующим образом: *сервер*/*пользователь*. При этом в указанной точке монтирования монтируются все тома файлового сервера как отдельные каталоги. Если необходимо монтировать отдельный том, его имя можно указать в опции *volume*. Пример:

```
mount -t ncpfs -o volume=SYS PSUFS131/GUEST /mnt/psu
```

Том SYS файлового сервера PSUFS131 будет примонтирован в каталоге /mnt/psu, причём будет осуществлена аутентификация под именем GUEST.

```
ncpmount -S PSUFS131 -U DFE /mnt/share
```

Все тома файлового сервера PSUFS131 будут примонтированы в каталоге /mnt/share, причём будет осуществлена аутентификация под именем DFE.

При помощи пакета программ Mars компьютер под управлением GNU/Linux может выполнять функции файл-сервера и принт-сервера сети Novell Netware.

Контрольные вопросы и задания

Перед выполнением задания осведомитесь у преподавателя, даны ли вам права на монтирование.

1. Смонтируйте флоппи-диск, CD-диск, раздел FAT или NTFS.
2. Смонтируйте различные удалённые файловые системы по протоколам NFS, SMB или NCP.
3. Ознакомьтесь с командой **df**. Примените команды копирования, удаления, изменения атрибутов файлов для проверки функциональности смонтированных устройств.
4. По справочному руководству изучите формат файла /etc/fstab. Запишите использовавшиеся команды монтирования и соответствующие им строки для файла /etc/fstab.

Демонтируйте файловые системы перед завершением работы.

6. РАСПРЕДЕЛЁННАЯ СЕТЕВАЯ ФАЙЛОВАЯ СИСТЕМА AFS

6.1 Общие сведения

AFS – это распределённая файловая система, способная объединять файловые системы множества файловых серверов, обладающая следующими преимуществами по сравнению с централизованными системами:

- *расширенная доступность*: копии часто используемых файлов (например, файлы приложений) могут храниться на множестве серверов, тогда при выходе из строя одной или даже нескольких машин такой файл может оказаться по-прежнему доступным, потому что запросы пользователей будут направлены на исправные машины;
- *улучшенная эффективность*: нагрузка может быть распределена между несколькими небольшими файловыми серверами, что зачастую оказывается выгоднее одного дорогого высокопроизводительного сервера.

При этом распределённость структуры AFS скрыта от пользователя. Работа с AFS мало чем отличается от работы с локальными файлами, хранящимися на машине пользователя. На машинах пользователя AFS представляется в виде иерархической файловой структуры, монтируемой, как правило, в каталог /afs.

В структуре AFS выделяют элементы, называемые *ячейками* (cell). Каждая ячейка представляет собой домен администрирования, в котором задаётся, как конфигурировать клиентские машины, какие пользователи имеют доступ к файловому пространству ячейки и т. п. Обычно в ячейку AFS объединяются компьютеры одной фирмы, факультета университета или определённой группы пользователей. Связанные между собой ячейки объединяются в AFS-*сайт* (site). Принято сопоставлять подкаталоги директории /afs файловым пространствам отдельных ячеек (например, каталоги и файлы ячейки фирмы Foo Inc. можно поместить в /afs/fooinc.com). Таким образом, хотя каждая ячейка AFS управляет и обслуживает собственное файловое пространство, существует возможность подключаться к файловым пространствам других ячеек. Та ячейка, к которой относится ваша клиентская машина, в файловом пространстве AFS называется *локальной ячейкой* (local cell), все остальные – *чужими ячейками* (foreign cells).

Обычно запоминающие устройства в компьютерах разделены на порции, называемые разделами. AFS производит дальнейшее деление предоставленного ей раздела на единицы, называемые *томами* (volumes). Том представляет собой удобный контейнер для хранения взаимосвязанных файлов и каталогов. Системный администратор может перемещать тома с одного сервера на другой, причём такое перемещение будет совершенно прозрачным для пользователя, т. к. AFS автоматически отслеживает место расположения тома. Пользователь получает доступ к содержимому тома через его точку монтирования в файловом пространстве AFS. Обычно для домашней директории каждого пользователя выделяется отдельный том, который монтируется в одну из папок локальной ячейки. Например, том пользователя pupkin в ячейке fooinc.com может быть смонтирован в каталог /afs/fooinc.com/home/pupkin. Для каждого тома администратор устанавливает предельный размер занимаемого им файлового пространства – *квоту*. При превышении этого предела возникает ошибка.

Для работы с AFS необходимо не только примонтировать файловое пространство AFS в выделенный каталог, но и запустить специальное клиентское программное обеспечение – *кэш-менеджер*. Когда пользователь обращается к файлу в AFS, кэш-менеджер определяет необходимый сервер и загружает с него копию этого файла на локальную машину. Прикладные программы используют локальную кэшированную копию файла, поэтому внесённые изменения не сразу отражаются на главной версии файла на сервере. Как правило, изменения переносятся, когда файл закрывается. Если во время работы с каким-либо файлом хранящий его сервер становится недоступным, можно продолжить работу с локальной копией, но изменения не будут сохранены до тех пор, пока сервер снова не станет доступным.

Когда главная копия файла на сервере изменяется, файловый сервер AFS сообщает всем кэш-менеджерам о недостоверности их локальных копий этого файла. Это происходит следующим образом. Файловый сервер вместе с копией файла передаёт кэш-менеджеру механизм обратного вызова (callback). При изменении главной копии файла файловый сервер разрушает обратный вызов. Когда программа запрашивает очередную порцию данных из изменённого файла, кэш-менеджер обнаруживает разорванный обратный вызов и загружает обновлённую копию файла. AFS сохраняет только изменения, сделанные в самую последнюю очередь. Поэтому необходимо предпринять определённые меры, чтобы не потерять свои изменения, если с файлом работают одновременно несколько пользователей. Один из возможных способов – воспользоваться средствами ограничения доступа AFS.

6.2 Аутентификация в AFS

Для обеспечения авторизованного доступа к файловому пространству ячейки используются пароли и взаимная аутентификация на основе протокола Kerberos. Во время аутентификации кэш-менеджер получает от сервера *токен* – порцию информации, зашифрованную при помощи вашего пароля. Если пользователь предоставил корректный пароль, кэш-менеджер может расшифровать токен. В дальнейшем токен служит доказательством вашей аутентичности для серверных программ AFS. Более того, когда кэш-менеджер обращается к файловому серверу, он шифрует полученный пользователем токен при помощи ключа AFS-сервера, так что только истинный файловый сервер может его расшифровать, что обеспечивает взаимную аутентификацию.

Чтобы получить доступ к файловому пространству AFS, пользователь должен пройти процедуру регистрации на локальной машине и аутентифицировать себя в AFS. При использовании команды регистрации, модифицированной для AFS, эти две процедуры могут быть совмещены в одну. В этом случае регистрационная информация UNIX и AFS должна совпадать. Пользователь вводит в приглашении **login** своё имя и пароль и при успешной аутентификации автоматически получает AFS-токен. Проверить наличие токена можно при помощи команды **tokens**:

```
$ tokens
Tokens held by the Cache Manager:
User's (AFS ID 1022) tokens for afs@fooinc.com [Expires
Aug 3 14:35]
--End of list--
```

В данном примере сообщается о наличии одного токена пользователя с идентификатором 1022 для ячейки fooinc.com, действительного до 3 августа 14:35.

Если процесс регистрации в системе не подразумевает аутентификацию в AFS, необходимо сделать это явно при помощи команды **klog**. При успешной аутентификации данная программа передаёт кэш-менеджеру выделенный пользователю токен. Следует иметь ввиду, что можно получить только по одному токену для каждой ячейки AFS. Если вы запрашиваете ещё один токен для какой-то ячейки, то он заменит собой предыдущий выданный вам для этой ячейки токен. Более того, при создании токена для него задаётся определённое время жизни. По истечении этого времени токен перестаёт распознаваться как действительный, так что пользователю может быть отказа-

но в доступе к AFS, поэтому необходимо вовремя запрашивать при помощи **klog** новый токен.

klog [**-setpag**] [**-cell** *ячейка*] [*пользователь*]

Если имя пользователя отсутствует, используется регистрационное имя в UNIX. Если не указывать ключ *-cell*, выдаётся токен для локальной ячейки. Ключ *-setpag* используется для привязки токена к определённой группе процессов (PAG – process authentication group), так что им могут воспользоваться только процессы из данного сеанса пользователя. В противном случае токен привязывается к UID пользователя, что позволяет воспользоваться токеном любому процессу с таким же UID.

Выход из системы не означает уничтожение токенов. Их следует удалять явно при помощи команды **unlog**.

unlog [**-cell** *ячейка*]

Если не указан ключ *–cell*, удаляются все токены, иначе удаляется только токен для указанной ячейки.

Для смены пароля в AFS используется команда **kpasswd**. Если используется модифицированная для AFS система регистрации, эта команда изменяет также системный пароль.

6.3 Информационные команды AFS

Большинство команд AFS выполняется при помощи сервиса **fs**:

fs *команда* [*параметры*]

Чтобы получить список доступных команд и краткое их описание, следует выполнить **fs help**. Подсказку по конкретной команде можно получить одним из следующих способов:

fs help *команда*
fs *команда* **-help**

Команда **fs quota** выводит информацию о проценте использованной квоты на томе, к которому относится указанный каталог (или текущий, если не указан). Более подробную информацию о томе можно получить при помощи команд **fs listquota** и **fs examine**.

```
$ fs quota /afs/fooinc.com/home/pupkin
   34% of quota used.
$ fs listquota /afs/fooinc.com/home/pupkin
   Volume Name      Quota    Used    % Used   Partition
   user.pupkin      10000    3400       34%         86%
```

```
$ fs examine /afs/fooinc.com/home/pupkin
   Volume status for vid = 536871122 named user.pupkin
   Current disk quota is 10000
   Current blocks used are 5745
   The partition has 1593 blocks available out of 99162
```

Приведённые команды сообщают имя тома, на котором расположен указанный каталог (или текущий, если не указан), величину квоты в Кбайтах, текущий размер файлового пространства в Кбайтах, а также информацию о разделе, на котором расположен этот том (общий размер и размер занятого файлового пространства).

Обычно пользователю нет необходимости знать, на каком именно файловом сервере AFS расположен тот или иной файл, тем не менее это можно определить при помощи команды **`fs whereis`**. Если файл вдруг станет недоступен, может оказаться полезным знать, где он на самом деле расположен, чтобы выяснить, всё ли в порядке с той машиной. Если в выводе команды упоминаются несколько серверов, значит, соответствующий том реплицирован на указанные сервера.

```
$ fs whereis /afs/fooinc.com/home/pupkin
File /afs/fooinc.com/home/pupkin is on host fs.fooinc.com
```

Иногда из-за программных или аппаратных проблем, а также во время технического обслуживания какие-либо сервера, составляющие ячейку AFS могут оказаться недоступны. Чтобы определить статус серверов в ячейке, используется команда:

`fs checkservers` [**`-cell`** *ячейка*] [**`-all`**]

Кэш-менеджер поддерживает список ячеек, к которым он может предоставить доступ. При помощи команды **`fs listcells`** можно получить этот список и имена серверов, составляющих ячейки.

6.4 Контроль доступа

Для управления доступом к файлам и папкам AFS использует *списки контроля доступа* (access control lists). Список контроля доступа (ACL) может содержать примерно до 20 элементов, разрешающих или запрещающих доступ для тех или иных пользователей или групп. Владелец каталога и системный администратор всегда имеют право на управление ACL данного каталога. Прочие пользователи могут менять ACL, только если для них в ACL специально определён такой тип доступа («*a*» – administer). Следует иметь в виду, что каждая ячейка AFS определяет свои группы и своих пользователей. Если пользователь принадлежит какой-либо группе, которая указана

в ACL, пользователь получает все указанные типы доступа, как если бы он сам непосредственно был указан в ACL.

Все пользователи, которые имеют доступ к файловому пространству некоторой ячейки, вне зависимости от того, аутентифицированы они или нет, автоматически сопоставляются с предопределённой группой system:anyuser. Все аутентифицированные в данный момент в локальной ячейке пользователи сопоставляются в ней с предопределённой группой system:authuser.

AFS сопоставляет ACL каждому каталогу и применяет его ко всем файлам в данном каталоге. Обычные файлы не имеют отдельных ACL. При перемещении файла из одного каталога в другой изменяются атрибуты доступа к данному файлу. Таким образом, изменяя ACL каталога, пользователь изменяет атрибуты доступа ко всем файлам в данном каталоге. Создаваемый подкаталог наследует текущий ACL своего родительского каталога, но для этого подкаталога ACL может быть изменён позднее. Следует иметь в виду, что пользователь получит доступ к содержимому каталога, только если для этого каталога и всех его родительских каталогов установлен режим доступа ***l*** (lookup).

Таблица 6.1. Режимы доступа, определённые в AFS

Обозначение	Описание
l (lookup)	Право на поиск в каталоге. Позволяет пользователю просмотреть ACL данного каталога (**`fs listacl`**), информацию о данном каталоге (**`ls -ld`**), а также получить список файлов в данном каталоге (**`ls`**)
i (insert)	Право на добавление новых файлов и подкаталогов в данном каталоге. Не распространяется на подкаталоги, т. к. они имеют собственные ACL
d (delete)	Право на удаление файлов и подкаталогов в данном каталоге
a (administer)	Право на администрирование, т. е. на изменение ACL данного каталога
r (read)	Право на чтение файлов и просмотр информации о них
w (write)	Право на изменение самих файлов и изменение их UNIX-атрибутов доступа (**`chmod`**)
k (lock)	Право на блокирование файлов (на монопольный доступ)
all = rlidwka	Обозначает группу из всех семи режимов доступа
none	Используется для удаления записи из ACL, лишая пользователя или группу каких-либо полномочий
read = rl	Право на чтение файлов
write = rlidwk	Право на полный доступ к каталогу, за исключением изменения ACL

В команде изменения ACL (**`fs setacl`**) можно использовать комбинации режимов доступа, перечисленные в таблице 6.1, и псевдонимы для групп режимов доступа (*all*, *none*, *read*, *write*).

ACL может содержать *разрешающие элементы* (normal rights) и *запрещающие элементы* (negative rights). Запрещающие элементы имеют больший приоритет и используются для явного запрещения того или иного типа доступа для пользователя или группы.

AFS лишь частично использует атрибуты доступа UNIX, а именно AFS использует у обычных файлов биты, соответствующие классу доступа «владелец», игнорируя остальные биты прав доступа. Для каталогов AFS вообще не использует атрибуты доступа UNIX, контролируя доступ лишь с помощью ACL. Биты класса доступа «владелец» используются следующим образом:

- Если право «*r*» не установлено, никто (даже владелец) не имеет право читать файл вне зависимости от разрешений ACL. Если право «*r*» установлено, файл имеют право просматривать те, для кого в ACL заданы права «*r*» и «*l*».
- Если право «*w*» не установлено, никто (даже владелец) не имеет право изменять файл вне зависимости от разрешений ACL. Если право «*w*» установлено, файл может изменяться теми, для кого в ACL заданы права «*w*» и «*l*».
- Право «*x*» определяет, является ли файл исполняемым. Чтобы запустить файл на выполнение, пользователь должен иметь право читать его («*r*» и «*l*»).

Чтобы просмотреть ACL того или иного каталога, используется команда **`fs listacl`** (без параметров – ACL текущего каталога).

```
$ fs listacl /afs/fooinc.com/home/pupkin
   Access list for /afs/fooinc.com/home/pupkin is
   Normal rights:
      system:anyuser rl
      ivanov rlw
      petrov rlidwka
   Negative rights:
      ivanov:other-dept rl
      sidorov rl
```

В данном примере всем пользователям, кроме sidorov и тех, кто входит в группу ivanov:other-dept, разрешён доступ на чтение. Кроме того, пользователю ivanov разрешён доступ на запись, а пользователю petrov – полный доступ, если только эти пользователи не входят в группу ivanov:other-dept.

Для изменения ACL используется команда:

```
fs setacl [[-dir] каталог] [[-acl] acl_spec] \
          [-negative] [-clear]
```

Если указана опция *-negative*, *acl_spec* задаёт запрещающие элементы. Если не указана – разрешающие. В качестве *acl_spec* указываются пары пользователь (группа) и атрибуты, разделённые пробелами. Внутри пары имя пользователя (группы) отделяется от атрибутов также пробелом. Если указана опция *-clear*, то перед установкой заданных элементов ACL будет очищен (все элементы удалены).

```
$ pwd
/afs/fooinc.com/home/pupkin
$ fs setacl sidorov none -negative
$ fs setacl -dir . -acl system:anyuser none \
   ivanov:colleagues write system:authuser rl
$ fs listacl
   Access list for /afs/fooinc.com/home/pupkin is
   Normal rights:
      system:authuser rl
      ivanov rlw
......ivanov:colleagues rlidwk
      petrov rlidwka
   Negative rights:
      ivanov:other-dept rl
```

В данном примере для каталога /afs/fooinc.com/home/pupkin удаляется запрещающий элемент ACL sidorov. Затем удаляется разрешающий элемент system:anyuser и добавляется доступ на запись для членов группы ivanov:colleagues, а также доступ на чтение для всех аутентифицированных пользователей (system:authuser).

Контрольные вопросы и задания

1. Аутентифицируйтесь в AFS. Выясните свой ID в AFS. Определите время жизни выданного вам токена (сколько часов?).
2. Определите название локальной ячейки AFS. Какие ячейки формируют доступное вам файловое пространство AFS?
3. Выясните название вашего персонального тома, его точку монтирования, квоту и файловый сервер, на котором он расположен.
4. Разрешите вашему соседу доступ в один из подкаталогов в вашей домашней папке на AFS-сервере. Убедитесь в возможности доступа. Отмените доступ.

5. Сравните системы контроля доступа AFS ACL и POSIX ACL по следующим критериям:
   а) возможность задать ACL для каждого файла,
   б) возможность явного запрещения доступа для пользователя или группы,
   в) возможность раздельного контроля доступа на добавление файлов и на удаление файлов в каталоге,
   г) кто имеет право изменять ACL,
   д) взаимодействие ACL со стандартными атрибутами доступа UNIX,
   е) как организовано наследование прав.

7. СРЕДСТВА РАЗРАБОТКИ ПРОГРАММ НА ЯЗЫКЕ ПРОГРАММИРОВАНИЯ СИ

7.1 Компиляция и связь программных модулей

Система разработки программ на языках программирования Си/Си++ состоит из препроцессора, компилятора, ассемблера и редактора связей. Также к средствам разработки можно отнести библиотеки функций с сопутствующими заголовочными файлами и справочную информацию по ним.

Наиболее популярным средством для транслирования текстов программ в исполняемый файл является GNU Compiler Collection (**`gcc`**) – это коллекция компиляторов различных языков программирования (C, C++, Objective-C, Java, Fortran, Ada) и сопутствующие средства: ассемблер, препроцессор, редактор связей и т. п.

Схематично процесс трансляции можно разбить на следующие стадии (рис. 7.1):

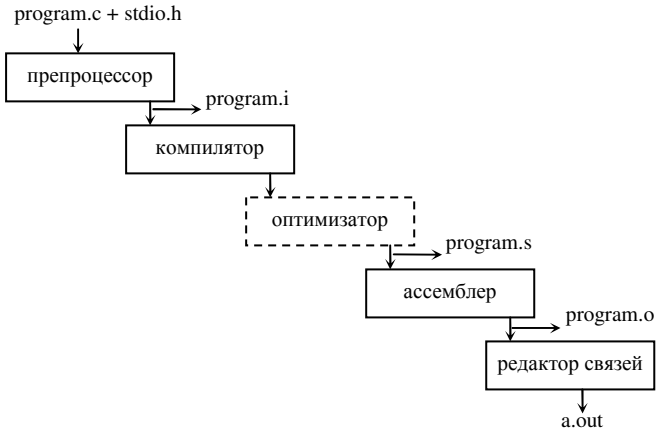

Рис. 7.1. Процесс генерации исполняемого файла по исходному тексту

Первая стадия. Исходный текст программы на языке Си (суффикс «.c») или Си++ (суффикс «.C», «.cpp»), а также заголовочные файлы (суффикс «.h») обрабатываются *препроцессором*. На этой стадии в тексте программы интерпретируются такие директивы, как *#define*, *#include* и т. п. Полученный результат (текст программы, не требующий препроцессорной обработки) обычно имеет суффикс «.i» и без

явного указания не создаётся.

Вторая стадия. *Компилятор* Си/Си++ производит синтаксический разбор текста и формирует соответствующий ему ассемблерный код. После чего возможна обработка текста *оптимизатором*, позволяющим сократить размер получаемого кода и увеличить скорость его работы. Полученный результат (текст программы на языке ассемблера) обычно имеет суффикс «.s» и без явного указания не создаётся. Ассемблерные файлы генерируются не в семантике Intel (принятой, например, в DOS или Windows), а в семантике AT&T.

Третья стадия. *Ассемблер* транслирует программу в так называемый объектный модуль (суффикс «.o»). Объектный файл представляет собой блоки машинного кода и данных с неопределёнными (импортируемыми) адресами ссылок на данные и процедуры в других объектных модулях и содержит список своих (определённых, экспортируемых) процедур и данных.

Четвёртая стадия. *Редактор связей (компоновщик)* связывает один или несколько объектных модулей, полученных в результате компиляции, в исполняемый файл (без суффикса) или динамически загружаемую библиотеку (суффикс «.so»). Для каждого импортируемого имени находится его определение в других модулях, упоминаемое имя заменяется его адресом (этот процесс называется «разрешением» – resolve). При этом в процессе компоновки происходит связывание программы с динамическими или статическими библиотеками (последние являются архивами объектных файлов) и добавляется код, обеспечивающий инициализацию экземпляра программы при её запуске. Если не указано иное, результирующий файл именуется a.out.

Описанная последовательность может быть начата или прервана на любой стадии, что задаётся опциями в командной строке. Например, опция *-E* указывает завершить обработку файла после стадии препроцессора (результат выдаётся на стандартный вывод); опция *-S* указывает завершить обработку файла после стадии компилятора (перед ассемблером); опция *-c* указывает завершить обработку файла перед компоновкой редактором связей. Оптимизатор **gcc** задействуется при использовании одного из вариантов опции *-O*.

Программа **gcc** определяет, какие действия предпринять с тем или иным файлом, по его суффиксу, поэтому необходимо правильно именовать файлы, в противном случае в командной строке придётся указывать дополнительные опции.

В командной строке **gcc** можно указывать несколько исходных файлов (не обязательно от одной и той же стадии), тогда в результате

они будут собраны в один исполняемый файл. Это бывает удобно при разработке многомодульных приложений, когда отдельный модуль описан в отдельном исходном файле. Пример:

```
$ cat hello.c
#include <stdio.h>
main() {
       printf("hello, world\n");
}
$ gcc hello.c
$ ls -l
a.out hello.c
$ ./a.out
hello, world
```

Можно указать в командной строке ключ *-o*, чтобы для исполняемого файла задать имя, отличное от a.out.

```
$ gcc -o hello hello.c
$ ./hello
hello, world
```

Опция *-lимя* подключает соответствующую библиотеку. Например, *-lx* подключает библиотеку libx. Необходимо помнить, что включение в исходный текст программы того или иного заголовочного файла (например, math.h) недостаточно для успешной генерации исполняемого файла, поскольку заголовочный файл содержит информацию лишь для синтаксического анализатора, позволяющую компилятору сгенерировать корректный код вызова тех или иных библиотечных функций. А вот собственно интеграция с кодом библиотечных функций происходит на стадии компоновки, для которой необходим объектный код этих функций (например, файл libm.a), что достигается при помощи этого ключа (для libm – ключ *-lm)*.

Весьма полезно использовать опцию *-Wall*, которая указывает компилятору выводить все предупреждения, благодаря чему становится проще обнаружить семантические ошибки в программе.

Подробности обо всех опциях компилятора Си/Си++ можно узнать по команде **`man gcc`**, а редактора связей – по команде **`man ld`**.

7.2 Программа `make`

Программа **`make`** в соответствии со специальным сценарием отслеживает зависимости между заданными файлами, так что изменение одного из них влечёт выполнение определённых действий с другими

файлами. Чаще всего **make** используется для компиляции группы программных модулей и сборки их в один проект.

Сценарий для программы **make** обычно хранится в файле с именем Makefile, и в этом случае можно запустить **make** без параметров. В противном случае имя сценария нужно передать программе **make** при помощи ключа *-f*:

```
make –f my_makefile
```

Рассмотрим работу с **make** на примере проекта baseline. Предположим, проект baseline состоит из четырёх основных модулей: main.c, create.c, update.c и delete.c. Предположим также, что файл update.c использует файл определений change.h. Взаимосвязи файлов схематично представлены на рисунке 7.2.

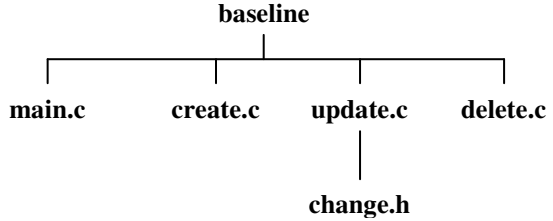

Рис. 7.2. Пример взаимосвязи файлов (проект baseline)

Для сборки данного проекта можно использовать возможность **gcc** указывать в командной строке несколько исходных файлов, тогда, указав все «.c»-файлы проекта, мы получим исполняемый файл, содержащий все указанные программные модули. Однако при изменении любого из модулей при таком способе сборки придётся перекомпилировать все модули.

Более эффективное решение предоставляет программа **make**. Можно компилировать перечисленные модули по отдельности, доходя до стадии сборки, но не вызывая редактор связей, а когда получены объектные файлы всех модулей, собрать их воедино при помощи редактора связей. Сценарий Makefile, описывающий необходимые взаимосвязи, для данного случая выглядит следующим образом:

```
# Makefile для построения программы baseline
CFLAGS=-g –Wall

baseline: main.o create.o update.o delete.o
    gcc –o baseline main.o create.o update.o delete.o

main.o: main.c
```

```
create.o: create.c
update.o: update.c change.h
delete.o: delete.c
```

Строки, начинающиеся с октоторпа (#), являются комментариями и игнорируются. Далее следуют определения некоторых переменных окружения. В данном примере объявлена переменная CFLAGS, которая хранит опции компилятора по умолчанию. Сценарий содержит несколько блоков вида:

```
<цель>: <зависимости>
       <команда>
       <команда>
       ...
```

В данном примере результат компилирования проекта (первая цель в сценарии Makefile) – файл baseline. Файл зависит от объектных модулей main.o, create.o, update.o и delete.o. Чтобы получить этот файл, должна быть выполнена команда, указанная в этом блоке. Команды в блоке записываются не в начале блока, а после символа табуляции (важно, чтобы это был именно символ табуляции, а не пробелы). Остальные цели описывают зависимости этих объектных модулей. Например, файл main.o зависит от файла main.c. Если для какой-то цели не указаны команды, как получить целевой файл из его зависимостей, срабатывает механизм умолчаний. Так, файлы с суффиксом «.c» обрабатываются Си-компилятором; файлы с суффиксом «.cpp» обрабатываются Си++-компилятором и т. п. Таким образом, программа **make** определяет, что для получения перечисленных объектных модулей надо скомпилировать соответствующие файлы.

```
$ make
gcc -c main.c
gcc -c create.c
gcc -c update.c
gcc -c delete.c
gcc -o baseline main.o create.o update.o delete.o
```

Если теперь внести изменения в файл change.h, то при запуске программа **make** по времени модификации определит, что он новее, чем зависящий от него файл update.o и перекомпилирует этот объектный модуль. После этого в соответствии со сценарием потребуется перекомпиляция основной цели – файла baseline, тогда как остальные объектные модули останутся неизменными.

```
$ make
gcc -O -c update.c
gcc -o baseline main.o create.o update.o delete.o
```

Если выполнить make в то время, когда не было никаких изменений, то она выводит предупреждение:

```
$ make
'baseline' is up to date
```

Таким образом, программа **make** значительно облегчает компиляцию сложных многомодульных проектов. Более того, набор утилит GNU содержит также средства для автоматической генерации файлов сценариев программы **make** при компиляции проекта на разных платформах. При помощи утилит **autoconf** и **automake** генерируется специальный сценарий для командного интерпретатора (обычно этот сценарий называется **configure**). Это сценарий распространяется вместе с исходным текстом проекта и при запуске определяет тип платформы и необходимые для компиляции проекта средства. В результате своей работы такой сценарий создаёт Makefile с необходимыми опциями для компилятора, редактора связей и т. п., а также заголовочный файл (обычно config.h), в котором указываются директивы препроцессора, управляющие особенностями компиляции на данной платформе.

7.3 Отладчик

Для отладки программ в GNU/Linux предназначена программа GNU Debugger (**gdb**), которая предоставляет средства выполнения программы по шагам, просмотра переменных программы, регистров процессора, стека и т. п. Программа имеет командный интерфейс – при запуске выдаётся приглашение, после которого вводятся команды **gdb** и их параметры. Отладчик позволяет пошагово выполнять программу, расставлять в ней контрольные точки, наблюдать значения переменных, отслеживать стек вызовов процедур и проч.

В GNU/Linux при аварийном завершении программы может быть сформирован файл образа памяти программы в момент аварии. Этот файл можно использовать в отладчике, чтобы найти место аварийного завершения программы и определить вероятную причину.

Отладочная информация – это дополнительные данные, добавляемые в исполняемый файл, представляющие собой информацию об именах переменных и функций, номера строк исходного текста программы, позволяющие связать машинные инструкции с операторами

языка высокого уровня. Отладочная информация существенно увеличивает размер получаемого исполняемого файла и отменяет некоторые способы оптимизации, поэтому по умолчанию отладочная информация в исполняемый файл не добавляется. Однако эффективная работа с отладчиком без неё невозможна. Поэтому на стадии разработки и тестирования программисты обычно включают её в исполняемый файл, а финальная версия программы для эксплуатации широким кругом пользователей уже формируется без отладочной информации. **`Gcc`** включает отладочную информацию в исполняемый файл, если указана опция *-g*.

Некоторые команды отладчика:

- **`help`** – помощь,
- **`run`** – запустить программу,
- **`print`** – вывести значение выражения (можно использовать переменные программы),
- **`backtrace`** – вывести стек вызовов процедур,
- **`list`** – вывод фрагмента листинга программы,
- **`start`** – загрузить программу и начать пошаговое выполнение,
- **`step`** – выполнить очередной шаг в программе,
- **`break`** – установить контрольную точку по коду,
- **`watch`** – установить контрольную точку по данным,
- **`continue`** – продолжить выполнение программы.

Контрольные вопросы и задания

1. Компиляция готового программного продукта.
   а) С указанного преподавателем ресурса загрузите два каких-либо архива с исходными текстами (вместо астериска в имени файла указывается версия): bc*.tar.bz2, file*.tar.bz2, links*.tar.bz2, mp3info*.tar.bz2, wget*.tar.bz2, recode*.tar.bz2, units*.tar.bz2, stat*.tar.bz2 или др.
   б) Распакуйте архив в своём домашнем каталоге (воспользуйтесь командой **`tar`**).
   в) Изучите порядок компиляции и установки программного продукта (внутри архива найдите файлы README, INSTALL и т. п.). Обычно это включает выполнение следующих действий:

```
./configure --help
./configure <необходимые параметры>
make
make install
```

г) Последняя команда должна выполняться от имени привилегированного пользователя, если программный продукт устанавливается в общесистемный каталог (/usr или /usr/local). Вам же надо скомпилировать и установить программный продукт в своём домашнем каталоге, в подкаталоге soft (~/soft).

д) Убедитесь в работоспособности программного продукта.

2. Изучение отладчика **gdb**.

а) Скомпилируйте программу с нарушением доступа к памяти. Например:

```
#include <stdio.h>
int main() {
 float f;
 scanf("%f",f);
 printf("f=%f",f);
 return 0;
}
```

б) Запустите её. Если создание образа памяти (core dump) отключено, включите его при помощи команды **ulimit**.

в) Откройте программу и её дамп памяти в отладчике.

```
gdb ./a.out ./core
```

г) Проследите стек вызовов функций (**backtrace**).

д) Перекомпилируйте программу с поддержкой отладочной информации и сравните результат.

8. X WINDOW SYSTEM – ГРАФИЧЕСКАЯ ОКОННАЯ СИСТЕМА

8.1 Общие сведения

Современные операционные системы не обходятся без подсистемы графического интерфейса пользователя (GUI). UNIX с самого начала позиционировалась как гибкая система, пусть даже в ущерб лёгкости использования. Тем не менее есть несколько причин использования GUI и в системах типа UNIX. Например, графический интерфейс облегчает управление традиционными для UNIX функциями многозадачности. Кроме того, некоторые виды информации удобнее представляются в графической форме, а некоторые виды только в графической форме и могут быть представлены. Исторически так сложилось, что различные элементы операционных систем типа UNIX разрабатывались в университетах (например, дистрибьютивы BSD). В свою очередь X Window System (или просто «X») – основа графического интерфейса большинства операционных систем типа UNIX – также является университетской разработкой, а именно частью проекта Athena, разработанного в 1984 году в Массачусетском институте технологий (MIT).

UNIX изначально была многопользовательской сетевой операционной системой, позволяющей пользователям работать с ней с удалённого терминала. Подобные принципы легли в основу при создании GUI для UNIX. Фактически X имеет достаточно сложную структуру, что нередко упоминают как недостаток. Однако при всей своей сложности X является чрезвычайно гибкой системой, что позволило ей распространиться на громадном количестве платформ.

С 1997 года разработка X координируется The Open Group. Спецификация интерфейса X свободно доступна (последняя версия протокола, выпущенная 23.09.2008, обозначается X11R7.4). В GNU/Linux и в некоторых других ОС до 2004 г. использовалась модификация X под названием XFree86. В 2004 г. после выпуска очередной версии XFree86 (версия 4.4) под ограниченной лицензией, которую многие проекты, полагавшиеся на X, посчитали неприемлемой, представители X.Org и freedesktop.org основали фонд X.Org Foundation для разработки собственной реализации X. Именно эта реализация (X.Org Server) используется в настоящее время в большинстве дистрибьютивов GNU/Linux.

Система X реализована по архитектуре «клиент–сервер». Графические приложения являются клиентами. Они соединяются с X-сервером, выполняют запросы и получают информацию от него. Клиентам надо лишь знать протокол общения с сервером, но совершенно не требуется информация о реальной аппаратной реализации графической подсистемы. Клиенты выполняют запросы типа «нарисуй линию с указанными координатами», «выведи такой-то текст определённым шрифтом в указанной позиции» и т. п. Такая архитектура позволяет разнести клиентов и сервер на разные машины. Например, можно запустить какую-нибудь вычислительную задачу на ЭВМ типа Cray, систему управления базами данных – на сервере под управлением Solaris, почтовую программу – на небольшом сервере BSD, а программу графического моделирования – на SGI-сервере, причём все эти программы могут выводить результаты своей работы у вас дома на рабочей станции под управлением GNU/Linux.

Следовательно, X-сервер обслуживает реальный графический дисплей. X-сервер запускается на том компьютере, за которым работает оператор. Именно обязанностью X-сервера является фактическое взаимодействие с пользователем, получение событий от мыши и клавиатуры. Однако, хотя X-сервер обеспечивает механизмы манипулирования областью отображения клиента (окном), но он не предоставляет средств для этого. Где отобразить окно клиента, средства изменения окна клиента, средства оформления окна клиента (заголовок, рамка и т. п.) – все эти задачи решаются специальной программой – «диспетчером окон». Диспетчер окон – это специальный X-клиент, не являющийся по своей сути частью X Window System, но наделённый определёнными привилегиями. Существует огромное количество диспетчеров окон для X: twm, fvwm, icewm, afterstep, enlightenment и sawfish (используются в GNOME), kwm (используется в KDE) и многие другие.

8.2 Запуск X-сервера и графических приложений

X может стартовать при запуске операционной системы. В таком случае для регистрации пользователя в системе используется программа **xdm** или аналогичная ей. Если X не стартует автоматически, то графическую подсистему можно запустить с помощью специального сценария **startx**:

startx [[*клиент*] *опции*] [**--** [*сервер*] *опции*]

Этот сценарий выполняет команду **xinit**, запуская указанный X-сервер и соединяя с ним первоначального клиента. Кроме того, в этом сценарии выполняются различные настройки X-сервера. Если клиент или сервер не указаны, то используются настройки из файлов конфигурации ~/.xinitrc и ~/.xserverrc. Обычно эта команда вызывается без параметров, что обеспечивает запуск подходящего X-сервера и стандартного диспетчера окон в качестве начального клиента на дисплее под номером 0. Как правило, X-сервер запускается на свободном виртуальном терминале системной консоли. Переключение между виртуальными терминалами в графическом режиме осуществляется при помощи комбинаций [Ctrl]+[Alt]+[F*n*] (обычно *n*=1…6 – для текстовых терминалов и *n*=7,… – для графических; комбинации [Alt]+[F*n*] зарезервированы для использования X-клиентами).

Графическим программам (X-клиентам) необходимо знать адрес X-сервера, который задаётся в виде:

hostname:displayname.screenname

- *hostname* – имя компьютера, на котором запущен X-сервер (в случае локального запуска может быть пустым);
- *displayname* – номер дисплея (под «дисплеем» понимается группа мониторов, разделяющая одну клавиатуру и мышь);
- *screenname* – номер экрана (номер монитора в группе).

Типичное значение адреса X-сервера – «:0.0». Адрес X-сервера обычно передаётся X-клиенту диспетчером окон через среду окружения в переменной DISPLAY. Другой способ передачи адреса – в параметре командной строки *-display*. Если переменная окружения DISPLAY не определена и параметр *-display* не задан, то X-клиент откажется запускаться. Примеры:

```
DISPLAY=":0.0" xcalc
xclock -display huba.buba.com:10.0
```

Допускается разделение несколькими X-серверами одного физического дисплея, т. е. запуск нескольких серверов на одном компьютере, но при этом им надо назначать различные номера логических дисплеев. Это достигается при помощи опции **:***N* (*N*=0,1,…) при старте сервера:

```
startx -- :1
```

При этом каждый логический дисплей занимает отдельный виртуальный терминал системной консоли. Для логических дисплеев используется своя нумерация, не совпадающая с нумерацией виртуаль-

ных терминалов. Так, в этом примере X-серверу присваивается логический дисплей 1, который занимает первый свободный виртуальный терминал (например, № 7).

Среди прочих параметров сервера пользователь может также указать с помощью параметра *-depth* глубину цвета (количество бит на пиксел) в графическом режиме:

```
startx -- -depth 8
startx -- :2 -depth 32
```

Разрешение экрана можно поменять в процессе работы при помощи комбинаций[11] [Ctrl]+[Alt]+[+] и [Ctrl]+[Alt]+[−].

Во многих диспетчерах окон комбинация [Ctrl]+[Alt]+[Esc] запускает программу **`xkill`**. Эта программа по щелчку мыши на окне того или иного приложения определяет процесс, которому принадлежит это окно, и вынуждает сервер разорвать соединение с этим клиентом, обеспечивая таким образом принудительное завершение работы того или иного приложения.

Другая полезная комбинация – [Ctrl]+[Alt]+[BackSpace]. Эта комбинация вынуждает X-сервер закрыть соединения со всеми клиентами и завершиться. Будьте очень осторожны: все несохраненные данные в программах-клиентах будут потеряны! Администратор системы может сконфигурировать X-сервер так, чтобы эта комбинация игнорировалась.

8.3 Ресурсы графических приложений

Все клиенты, использующие стандартную библиотеку Xlib, поддерживают параметры командной строки, влияющие на отображение их окна. Кроме уже упомянутого параметра *-display* распознаются следующие параметры:

- *-title имя* – задать заголовок окна;

```
xlogo -title "X rulezzz 4ever!!!"
```

- *-iconic* – запускать свёрнутым;

```
xload -iconic
```

- *-geometry* [*ширина***x***высота*][**±***xpos***±***ypos*] – позиция и размеры окна программы (“+” соответствует отсчётам от левой или верхней грани экрана, а “–” от правой или нижней), единица измерения (символы или пикселы) зависит от приложения;

[11] Клавиши [+] и [–] на дополнительной клавиатуре («серые»).

```
xlogo -geometry 100x200+100+200
xcalc -geometry -50+50
xterm -geometry 80x43
```

- *-background цвет* или *-bg цвет* – установка цвета фона;
- *-foreground цвет* или *-fg цвет* – установка основного цвета;

```
xcalc -background orange -fg blue
xclock -bg rgb:0/FF/0 -foreground rgb:FF/80/80
xfontsel -background '#0080FF' -fg '#FF0000'
```

- *-font шрифт* или *-fn шрифт* – установка основного шрифта (как описывается тот или иной шрифт можно определить при помощи программы **xfontsel**)

```
xclock -digital -font '-cronyx-helvetica-medium-r-*-*-*-
    *-*-*-*-*-koi8-r'
xclock -digital -font '-cronyx-courier-bold-o-*-*-*-*-*-
    *-*-*-koi8-r'
xcalc -font '-adobe-times-*-*-*-*-*-*-*-*-*-*-*-*'
```

Настройки такого рода называются *описанием ресурсов* приложения. Значения по умолчанию хранятся в базе данных ресурсов X-сервера. При запуске X-сервера из сценария **startx** эта база данных заполняется из общесистемных файлов описания ресурсов (/usr/lib/X11/app-defaults/*) и пользовательских файлов настроек (~/.Xdefaults или ~/.Xresources). Эта база данных может изменяться интерактивно при помощи команды **xrdb**. Например, чтобы добавить в базу данных ресурсов описания из файла appres, надо выполнить команду:

```
xrdb -merge appres
```

Если имя файла не указано, описания ресурсов читаются со стандартного ввода. Просмотреть текущую базу ресурсов можно, задав ключ *-query:*

```
$ xrdb -query
Emacs*menubar*background: #DCDCDC
XTerm*font: -adobe-courier-*-*-*-*-*-*-*-*-*-*-iso8859-1
XTerm*background: #FFFFFF
XTerm*foreground: #000000
$ xrdb -merge
XTerm*font: -misc-fixed-*-*-*-*-*-*-*-*-*-*-koi8-r
```

В обозначении атрибутов графических элементов приложения строчные и прописные буквы различаются! (Например, для **xterm** должно быть «XTerm…», а для **xcalc** – «xcalc…»).

8.4 Контроль доступа (авторизация клиентов)

X-сервер может контролировать доступ клиентов при помощи одного из пяти механизмов.

Host Access Control – простой контроль по имени хоста. X-сервер хранит список хостов, доступ клиентам с которых безусловно разрешается (причём владелец процесса X-клиента не проверяется). Такая система авторизации применима в том случае, если все пользователи доверяют друг другу. Изменение списка хостов осуществляется при помощи команды **`xhost`**. При использовании более безопасных средств контроля доступа список хостов должен быть пуст, тогда доступ к X-серверу будут получать только авторизованные клиенты.

MIT-MAGIC-COOKIE-1 – при использовании этого метода авторизации клиент обязан послать X-серверу 128-битный секретный код (*magic cookie*). Если код, посланный клиентом, совпадает с тем, который использует сервер, доступ клиенту разрешается. Пользовательская копия кода доступа обычно хранится в бинарном файле ~/.Xauthority. Операции с этим файлом производятся при помощи команды **`xauth`**. Программа **`xdm`** автоматически сохраняет *magic cookie* в каталоге пользователя и передаёт X-серверу. При локальном запуске X-сервера при помощи сценария **`startx`** генерируется новый *magic cookie*, помещается в пользовательскую базу кодов, а X-серверу для обеспечения авторизации в опции *-auth* передаётся пользовательский файл с кодом. *Magic cookie* посылается по сети в открытом виде, поэтому такую схему авторизации не следует применять в сетях, где вероятен перехват сетевых пакетов.

XDM-AUTHORIZATION-1 – этот механизм аналогичен MIT-MAGIC-COOKIE-1, за исключением того, что при передаче секретного кода используется шифрование при помощи симметричного алгоритма DES.

SUN-DES-1 – авторизация на основе безопасных вызовов удалённых процедур (secure RPC). При помощи механизма secure RPC обеспечивается аутентификация пользователя. Контроль доступа осуществляется по регистрационным именам пользователей. Этот механизм применим только в системах, поддерживающих Secure RPC.

MIT-KERBEROS-5 – авторизация по протоколу Kerberos 5.

Как правило, удалённые X-сервера используются только в пределах локальных вычислительных сетей, поэтому именно первые два механизма авторизации широко распространены, хоти они и менее безопасны, чем три остальных.

Если X-клиент и X-сервер работают на разных машинах, то, очевидно, наиболее полезна ситуация, когда удалённый X-клиент использует X-сервер на вашей машине (а не наоборот). Именно в этой ситуации обычно используют XDMCP (X Display Manager Control Protocol) и SSH X11 Forwarding.

Когда локальный X-сервер настраивается на соединение с удалённой машиной по протоколу XDMCP, удалённая машина выводит на вашем X-сервере окно регистрации. После успешной регистрации вы можете работать с удалённой машиной, причём все запущенные вами на удалённой машине X-клиенты будут соединяться с вашим X-сервером. В качестве XDMCP-сервера на удалённой машине обычно выступает программа **xdm** или аналогичная ей (**gdm**, **kdm**). Для запуска X-сервера в режиме XDMCP-клиента ему надо передать ключ *-query* с адресом XDMCP-сервера:

```
X -query huba.buba.com
```

Авторизацию обеспечивает механизм MIT-MAGIC-COOKIE-1. X-сервер будет использовать протокол XDMCP для согласования секретного кода. Сам код будет помещён XDMCP-сервером в вашу базу кодов на удалённой машине (~/.Xauthority).

Протокол XDMCP уязвим к перехвату сетевых пакетов, т. к. информация передаётся в открытом виде. Если требуется более безопасный механизм, следует использовать SSH X11 Forwarding.

На своём X-сервере вы запускаете SSH-клиент и регистрируетесь на удалённой машине. Если механизм X11 Forwarding поддерживается обеими сторонами, то на удалённой машине SSH создаёт виртуальный «прокси» X-сервер и устанавливает переменную окружения DISPLAY так, чтобы удалённые X-клиенты соединялись с этим виртуальным «прокси» X-сервером. На самом деле, данные, поступающие для этого виртуального сервера, передаются SSH по безопасному каналу вашему локальному X-серверу. При этом SSH генерирует для виртуального сервера свой *magic cookie*, а при перенаправлении данных, заменяет его на реальный секретный код для вашего сервера, так что *magic cookie* вашего X-сервера не хранится на удалённой машине и в открытом виде по сети не передаётся.

Контрольные вопросы и задания

1. При помощи команды **startx** запустите несколько X-серверов. Определите, какие виртуальные терминалы ими заняты. Сколько X-серверов вам удалось запустить?

2. Определите, какие адреса соответствуют X-серверам на разных консолях. Для этого с помощью средств вашего диспетчера окон запустите какой-нибудь эмулятор терминала для X (**xterm**, **rxvt**, **konsole** и т. п.). В окне эмулятора терминала выведите значение переменной окружения DISPLAY.
3. Из первого виртуального терминала (tty1) запустите на разных X-серверах программы **xcalc** и **xclock**.
4. Поэкспериментируйте с различными параметрами командной строки, описывающими вид окна, для программ **xcalc**, **xlogo**, **xclock**, **xfontsel**, **xload**, **xterm**, **rxvt**.
5. Измените базу данных ресурсов так, чтобы программа **xcalc** по умолчанию запускалась с голубым фоновым цветом, белым основным цветом и использовался шрифт misc-fixed с кодировкой iso8859-1.
6. Измените пользовательский файл описания ресурсов так, чтобы программа **xterm** запускалась с зелёным фоновым цветом, с красным основным цветом, шрифт – cronyx-courier с кодировкой koi8-r, размер окна по умолчанию – 100x43 симв. Перезапустите X-сервер и убедитесь, что новые описания ресурсов вступили в силу.
7. Попросите своего соседа, чтобы он для своего X-сервера при помощи команды **xhost** разрешил доступ с вашего компьютера. Запустите программу **xclock**, передав адрес X-сервера вашего соседа в переменной DISPLAY, и программу **xcalc**, передав адрес X-сервера вашего соседа в аргументе командной строки *-display*. Попросите соседа отключить доступ с вашего компьютера и сравните результат.
8. Выясните у своего соседа 128-битный код доступа (*magic cookie*) для его X-сервера. Добавьте этот код в свою базу cookies при помощи команды **xauth**. Запустите программы **xlogo** и **xload**, указав им адрес сервера вашего соседа.
9. Зайдите на указанную преподавателем машину при помощи **telnet** или **ssh** (X11 Forwarding запрещён). Запустите на удалённой машине несколько X-клиентов так, чтобы они выводили информацию на ваш X-сервер (настройте доступ для вашего X-сервера сначала при помощи Host Access Control, а затем при помощи MIT-MAGIC-COOKIE-1).
10. Запустите ваш X-сервер в режиме XDMCP-клиента. В этом режиме запустите на XDMCP-сервере несколько X-клиентов.
11. Ознакомьтесь с работой SSH X11 Forwarding.

ЛИТЕРАТУРА

1. Костромин В. Самоучитель Linux для пользователя. – СПб.: БХВ-Петербург, 2005. – 658 с.
2. Кузнецов С. Д. Операционная система UNIX. – М.: Центр информационных технологий, [б. г.] – URL: http://www.citforum.ru/ operating_systems/ unix/ contents.shtml
3. Олифер В. Г., Олифер Н. А. Сетевые операционные системы. – СПб.: Питер, 2008. – 672 с.
4. Рейчард К., Фолькердинг П. Linux: справочник / Пер. с англ. А. Выскубова. – СПб.: Питер, 2001. – 480 с.
5. Таненбаум Э., Вудхалл А. Операционные системы. Разработка и реализация. – СПб.: Питер, 2007. – 704 с.
6. Burgis H. The X Window User HOWTO. – 2002. – URL: http://tldp.org/HOWTO/XWindow-User-HOWTO/index.html
7. The GNU Bash Reference Manual. Ver. 3.2. – 2006. – URL: http://www.gnu.org/software/bash/manual/bashref.html
8. Grünbacher A. POSIX Access Control Lists on Linux. – Nuremberg: SuSE Labs, 2004. – URL: http://www.suse.de/~agruen/acl/linux-acls/online/
9. OpenAFS Documentation. – 2001. – URL: http://www.openafs.org/ doc/index.htm
10. Zweije V. Remote X Apps mini-HOWTO. – 2001. – URL: http://tldp.org/HOWTO/Remote-X-Apps.html

Учебное издание

Кипрушкин Сергей Альбертович
Соловьев Алексей Владимирович

ОСНОВЫ РАБОТЫ В LINUX

Учебное пособие

Редактор *Л. М. Колясева*
Компьютерная вёрстка *А. В. Соловьева*

В оформлении обложки использован логотип Linux – пингвин Такс, который разработан Ларри Юингом в 1996 г. и доступен для свободного применения и распространения при условии указания автора.

Подписано в печать 30.03.09. Формат 60×84 1/16
Бумага офсетная. Печать офсетная.
Уч.-изд. л. 4,6. Тираж 150 экз.
Изд. № 67.

Государственное образовательное учреждение
высшего профессионального образования
ПЕТРОЗАВОДСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

Отпечатано в типографии Издательства ПетрГУ
185910, г. Петрозаводск, пр. Ленина, 33

ISBN 978-5-8021-0997-7