THE UNIX PROGRAMMING ENVIRONMENT

Brian W. Kernighan, Rob Pike

PRENTICE HALL

HIGH TECH

UNIX

ПРОГРАММНОЕ ОКРУЖЕНИЕ

Брайан Керниган, Роб Пайк

Санкт-Петербург — Москва
2003

Брайан Керниган, Роб Пайк

UNIX. Программное окружение

Перевод П. Шера

Главный редактор	*А. Галунов*
Зав. редакцией	*Н. Макарова*
Научный редактор	*А. Козлихин*
Редактор	*В. Овчинников*
Художник	*В. Гренда*
Корректура	*С. Беляева*
Верстка	*Н. Гриценко*

Керниган Б., Пайк Р.
UNIX. Программное окружение. – Пер. с англ. – СПб: Символ-Плюс, 2003. – 416 с., ил.
ISBN 5-93286-029-4

Книга представляет собой введение в программное окружение UNIX и адресована тем, кто хочет научиться программировать с помощью всех тех инструментов, которые поставляются с операционной системой. Рассматривается вход в систему, работа с файлами (cat, mv, cp, rm) и каталогами (cd, mkdir, ...), основы окружения (переменные, маски), фильтры (grep, sed, awk), программирование оболочки (циклы, сигналы, аргументы, стандартный ввод-вывод), введение в системные вызовы (read, write, open, creat, ...), введение в программирование с использованием lex, yacc и make, работа с документацией с помощью troff, tbl и eqn. Приводимые примеры не придуманы специально для этой книги, – некоторые из них впоследствии стали частью комплекта программ, используемых каждый день. Программы написаны на Си. Предполагается, что читатель знает или хотя бы изучает этот язык.

Прочтение этой книги как новичками, так и опытными пользователями поможет понять, как сделать работу с системой эффективной и приносящей удовольствие.

ISBN 5-93286-029-4
ISBN 0-13-937681-X (англ)

Издательство «Символ-Плюс». 199034, Санкт-Петербург, 16 линия, 7,
тел. (812) 324-5353, edit@symbol.ru. Лицензия ЛП N 000054 от 25.12.98.
Налоговая льгота – общероссийский классификатор продукции
ОК 005-93, том 2; 953000 – книги и брошюры.
Подписано в печать 20.03.2003. Формат 70x100 1/16 . Печать офсетная.
Объем 26 печ. л. Тираж 3000 экз. Заказ N
Отпечатано с диапозитивов в Академической типографии «Наука» РАН
199034, Санкт-Петербург, 9 линия, 12.

Оглавление

Предисловие

«Количество инсталляций UNIX возросло до 10, а ожидается еще больший рост».

(Справочное руководство по системе UNIX, 2-е издание, июнь 1972 года.)

Операционная система UNIX[1] стартовала на неиспользовавшейся DEC PDP-7 в Bell Laboratories в 1969 году. Кен Томпсон (Ken Thompson) при помощи и поддержке Руда Канадея (Rudd Canaday), Дага Мак-Илроя (Doug McIlroy), Джо Оссанны (Joe Ossanna) и Денниса Ритчи (Dennis Ritchie) написал небольшую универсальную систему с разделением времени, достаточно удобную для того, чтобы привлечь пользователей, и в конечном счете создавшую достаточный кредит доверия для покупки более мощной машины – PDP-11/20. Одним из первых пользователей стал Ритчи, который помогал перенести систему на PDP-11 в 1970 году. Кроме того, Ритчи спроектировал и написал компилятор для языка программирования Си. В 1973 году Ритчи и Томпсон переписали ядро UNIX на Си, прервав таким образом традицию написания системного программного обеспечения на языке ассемблера. И после этой переделки система, по существу, стала тем, чем она и является сегодня.

Примерно в 1974 году система была разрешена для использования в университетах «в учебных целях», а через несколько лет стали доступны ее коммерческие версии. В это время системы UNIX процветали в Bell Laboratories – они проникли в лаборатории, проекты по разработке программного обеспечения, центры обработки текстов и системы поддержки операций в телефонных компаниях. С того времени UNIX распространилась по всему миру – десятки тысяч систем установлено на различное оборудование, от микрокомпьютеров до самых крупных мэйнфреймов.

Почему UNIX имела такой успех? Можно привести несколько причин. Во-первых, благодаря тому что она написана на языке Си, она перено-

---

[1] UNIX – это торговая марка Bell Laboratories. «UNIX» – это *не* акроним, это намек на MULTICS, операционную систему, над которой Томпсон и Ритчи работали до UNIX.

сима – UNIX-системы работают на всевозможных компьютерах, от микропроцессоров до мэйнфреймов; это важное коммерческое преимущество. Во-вторых, исходный код доступен и написан на языке высокого уровня, что делает простым адаптирование системы для специфических требований пользователей. Наконец, и это самое важное, UNIX – *хорошая* операционная система, особенно для программистов. Среда программирования UNIX поразительно богата и продуктивна.

Хотя UNIX и представляет ряд новаторских программ и технологий, действенность системы не определяется какой-то одной программой или концепцией. Эффективность UNIX определяется применением особого подхода к программированию, философией использования компьютера. Для того чтобы описать эту философию, одного предложения, конечно же, недостаточно, но вот ее основная идея – мощность системы в огромной степени определяется взаимодействием программ, а не их наличием. Многие UNIX-программы по отдельности выполняют довольно тривиальные задачи, но будучи объединенными с другими, образуют полный и полезный инструментарий.

Цель этой книги в том, чтобы представить философию программирования в UNIX. Основа этой философии – взаимодействие программ, поэтому, хотя основное место отводится обсуждению отдельных инструментов, на протяжении всей книги обсуждается и тема комбинирования программ, а также их использования для построения других программ. Чтобы правильно работать с системой UNIX и ее компонентами, надо не только понимать, как применять программы, но и знать, как они взаимодействуют с окружением.

По мере распространения UNIX становилось все меньше тех, кто квалифицированно использовал бы ее приложения. Нередко случалось так, что опытным пользователям, в том числе и авторам этой книги, удавалось найти только «топорное» решение задачи, или же они писали программы для выполнения заданий, которые без проблем обрабатывались уже существующими средствами. Конечно же, чтобы найти красивое решение, необходимы опыт и понимание системы. Хотелось бы надеяться, что прочтение этой книги поможет как новичкам, так и «бывалым» пользователям понять, как сделать работу с системой наиболее эффективной и приносящей удовольствие. Используйте систему UNIX правильно!

Мы адресуем книгу программистам-одиночкам в надежде, что, сделав их труд более продуктивным, мы тем самым сделаем более эффективной и групповую работу. Хотя книга предназначена в основном для специалистов, содержание первых четырех или даже пяти глав можно понять, не имея опыта программирования, так что они могут иметь ценность и для других пользователей.

Везде, где возможно, приводятся не специально придуманные, а реальные примеры. Случалось даже, что программы, написанные как примеры для этой книги, впоследствии становились частью комплек-

та программ, используемых каждый день. Все примеры были протестированы непосредственно из текста,[1] который представлен в машинно-считываемой форме.

Книга организована следующим образом. Глава 1 представляет собой введение в самые основы работы с системой. Она описывает процесс регистрации, почту, файловую систему, наиболее употребительные команды и содержит начальную информацию о командном процессоре. Опытные пользователи могут пропустить эту главу.

Глава 2 посвящена обсуждению файловой системы UNIX. Файловая система является центральным звеном в функционировании ОС и в ее использовании, поэтому вы должны хорошо понимать ее для правильной работы в UNIX. Обсуждаются файлы и каталоги, привилегии и права доступа к файлам, а также индексные дескрипторы. В конце главы дан обзор иерархии файловой системы и рассказано о файлах устройств.

Командный процессор, или *оболочка,* – это основной инструмент как для исполнения программ, так и для их написания. Глава 3 показывает, как использовать оболочку для создания новых команд, работы с аргументами команд, переменными оболочки, элементарной управляющей логикой и перенаправлением ввода-вывода.

Глава 4 рассказывает о фильтрах – программах, которые выполняют некоторые простые преобразования данных по мере «пропускания» последних через себя. Первый раздел знакомит читателей с командой поиска по шаблону `grep` и ее сородичами, в следующем разделе обсуждаются некоторые наиболее распространенные фильтры, такие как `sort`, а оставшаяся часть главы посвящена двум универсальным программам преобразования данных: `sed` и `awk`. Программа `sed` – это потоковый редактор, осуществляющий редактирование потока данных по мере его поступления. А `awk` – это язык программирования, обеспечивающий простой информационный поиск и генерирование отчетов. В ряде случаев применение этих программ (возможно, во взаимодействии с оболочкой) позволяет полностью избежать программирования в его традиционном виде.

В главе 5 рассматривается применение оболочки для создания программ, которые будут использоваться другими людьми. Обсуждается более сложная управляющая логика и переменные, обработка системных сигналов. Примеры в этой главе составлены с использованием как `sed` и `awk`, так и оболочки.

Со временем вы достигнете предела того, что может быть реализовано при помощи оболочки и других, уже существующих программ. Глава 6

[1] В процессе редактирования русского издания книги все предлагаемые авторами примеры также проверялись на работоспособность и возможность их компиляции, если она необходима, в современном окружении систем Solaris, Linux и FreeBSD. – *Примеч. науч. ред.*

посвящена написанию новых программ с использованием стандартной библиотеки ввода-вывода. Программы написаны на Си. Предполагается, что читатель знает или хотя бы изучает этот язык одновременно с прочтением книги. Приведены разумные методики разработки и организации новых программ, описано поэтапное проектирование, показано, как использовать уже существующий инструментарий.

Глава 7 знакомит с системными вызовами – фундаментом, на котором построены все остальные слои программного обеспечения. Обсуждаемые темы: ввод-вывод, создание файла, обработка ошибок, каталоги, индексы, процессы и сигналы.

Глава 8 рассказывает об инструментах разработки программ, таких как: `yacc` – генератор грамматического анализатора; `make`, контролирующий процесс компиляции больших программ, и `lex`, генерирующий лексические анализаторы. Описание основано на создании большой программы – Си-подобного программируемого калькулятора.

В главе 9 обсуждаются средства подготовки документов, проиллюстрированные описанием (на уровне пользователя) и страницей руководства для калькулятора из главы 8. Данная глава может быть прочитана независимо от остальных.

Приложение A – это краткое изложение материалов о стандартном редакторе `ed`. Вероятно, некоторые читатели предпочитают какой-либо другой редактор для каждодневной работы, но, тем не менее, `ed` – это общедоступный, действенный и производительный редактор. Его регулярные выражения являются основой других программ (например, `grep` и `sed`), и уже по этой причине он заслуживает изучения.

Приложение B представляет собой справочное руководство по языку калькулятора из главы 8.

В приложении C содержится листинг окончательной версии программы калькулятора, для удобства весь код собран в одном месте.

Приведем несколько фактов. Во-первых, система UNIX приобрела большую популярность и в настоящий момент широко используется несколько ее версий. Например, седьмая версия происходит от первоисточника создания системы UNIX – Computing Science Research Center, Bell Laboratories (Центра исследования вычислительных систем лабораторий Белла). System III и System V – это версии, официально поддерживаемые Bell Laboratories. Университет Беркли, Калифорния, распространяет системы, являющиеся производной седьмой версии, известные как UCB 4.xBSD. Кроме того, существует множество вариантов, также полученных на основе седьмой версии, в частности для микрокомпьютеров.

Чтобы справиться с этим разнообразием, авторы рассматривают только те аспекты системы, которые (насколько этого можно ожидать) одинаковы во всех вариантах. Значительная часть представленного материала не зависит от версии системы, а в тех случаях, когда детали

реализации отличаются, предпочтение отдается седьмой версии, так как именно она распространена наиболее широко. Примеры также выполнялись на System V в Bell Laboratories и на 4.1BSD Университета Беркли; лишь в небольшом количестве случаев потребовались незначительные изменения. Ваш компьютер будет работать с ними вне зависимости от версии операционной системы, а обнаруженные различия будут минимальными.

Во-вторых, несмотря на то что в книге представлено много материала, это не справочное руководство. При создании этой книги более важным представлялось описание подхода и способа применения, а не конкретные детали. Стандартным источником информации является справочное руководство по UNIX. В нем можно найти ответы на вопросы, которых вы не получили в данной книге, там же можно прочитать об отличиях конкретной системы от той, которая описана здесь.

В-третьих, по мнению авторов, лучший способ научиться чему бы то ни было состоит в том, чтобы сделать это. Эту книгу надо читать за компьютером, чтобы иметь возможность экспериментировать, проверять или опровергать написанное, исследовать пределы возможностей. Прочитайте немного, попробуйте сделать то, что написано, потом вернитесь и читайте дальше.

UNIX – это, конечно же, не совершенная, но удивительная и непостижимая вычислительная среда. Надеемся, что читатели придут к этому же выводу.

Хотелось бы выразить многим людям благодарность за конструктивные замечания и критику, а также за помощь в усовершенствовании кода. В особенности Джону Бентли (Jon Bentley), Джону Линдерману (John Linderman), Дагу Мак-Илрою (Doug McIlroy) и Питеру Вейнбергеру (Peter Weinberger), которые с огромным вниманием читали многочисленные черновики. Мы признательны Алу Ахо (Al Aho), Эду Бредфорду (Ed Bradford), Бобу Фландрене (Bob Flandrena), Дейву Хансону (Dave Hanson), Рону Хардину (Ron Hardin), Марион Харрис (Marion Harris), Джерарду Хользманну (Gerard Holzmann), Стиву Джонсону (Steve Johnson), Нико Ломуто (Nico Lomuto), Бобу Мартину (Bob Martin), Ларри Рослеру (Larry Rosler), Крису Ван Вику (Chris Van Wyk) и Джиму Вейтману (Jim Weytman) за их замечания по первому варианту этой книги. Мы также благодарим Мика Бьянчи (Mic Bianchi), Элизабет Биммлер (Elizabeth Bimmler), Джо Карфагно (Joe Carfagno), Дона Картера (Don Carter), Тома Де Марко (Tom De Marco), Тома Дафа (Tom Duff), Дэвида Гея (David Gay), Стива Махани (Steve Mahaney), Рона Пинтера (Ron Pinter), Денниса Ритчи (Dennis Ritchie), Эда Ситара (Ed Sitar), Кена Томпсона (Ken Thompson), Майка Тилсона (Mike Tilson), Поля Туки (Paul Tukey) и Ларри Вера (Larry Wehr) за их ценные предложения.

Брайан Керниган (Brian Kernighan)
Роб Пайк (Rob Pike)

1

UNIX для начинающих

Что такое «UNIX»? В самом узком смысле слова – это ядро операционной системы с разделением времени – программа, которая управляет ресурсами компьютера и распределяет их между пользователями. UNIX позволяет пользователям запускать их программы; он управляет периферийными устройствами (дисками, терминалами, принтерами и т. п.), соединенными с машиной; кроме того, UNIX предоставляет файловую систему, которая обеспечивает долгосрочное хранение информации: программ, данных и документов.

В более широком смысле под «UNIX» понимается не только ядро системы, но и основные программы, такие как компиляторы, редакторы, командные языки, программы для копирования и печати файлов и т. д.

В еще более широком смысле «UNIX» может включать в себя программы, созданные вами или другими пользователями для запуска на вашей системе, например средства подготовки документов, программы статистического анализа или графические пакеты.

Какой из этих смыслов слова «UNIX» правилен, зависит от того, о каком уровне системы идет речь. Какое именно значение «UNIX» подразумевается в том или ином разделе данной книги, будет следовать из контекста.

Система UNIX может показаться более сложной, чем есть на самом деле, – новичку трудно разобраться в том, как использовать доступные средства наилучшим образом. Но, к счастью, начало пути не такое уж трудное – достаточно изучить всего несколько программ, и дело пойдет. Цель этой главы – помочь как можно быстрее начать работу с системой. Это скорее обзор, чем учебник; в следующих главах информация будет представлена более подробно.

Здесь речь пойдет о следующих важных областях:

- об основах – входе и выходе из системы, простых командах, исправлении ошибок ввода с клавиатуры, почте, межтерминальной связи;
- о повседневной работе – файлах и файловой системе, печати файлов, каталогах, наиболее употребительных командах;
- о командном процессоре, или *оболочке* (*shell*) – шаблонах (масках) имен файлов, перенаправлении ввода-вывода, конвейерах, или программных каналах (pipes), установке символов удаления (erase) и аннулирования ввода (kill), задании пути поиска команд.

Для тех, кто уже работал с системой UNIX, большая часть информации, представленная в главе 1, окажется уже знакомой, и они могут сразу перейти к главе 2.

Уже во время прочтения этой главы вам потребуется экземпляр справочного руководства по UNIX (UNIX Programmer's Manual);[1] во многих случаях легче сослаться на информацию руководства, чем пересказывать его содержание. Эта книга предназначена не для того, чтобы заменить руководство, а для того, чтобы показать, как лучше всего использовать команды, описанные в нем. Более того, возможны отличия между тем, что написано в этой книге, и тем, что справедливо для вашей системы. В начале руководства есть пермутационный указатель команд, описанных в руководстве, незаменимый для того, чтобы найти программу, подходящую для решения задачи; учитесь им пользоваться.

И в заключение один совет: не бойтесь экспериментировать. Новичок, даже случайно, мало что может сделать такого, что нанесло бы вред ему самому или другим пользователям. Так что изучайте работу системы на практике. Это длинная глава, и лучше всего читать ее порциями по несколько страниц, сразу пробуя делать то, о чем вы читаете.

1.1. Давайте начнем

Начальные сведения о терминалах и вводе с клавиатуры

Чтобы не рассказывать с самого начала все о компьютерах, будем считать, что вы в общих чертах знаете, что такое терминал и как им пользоваться. Если все же какое-либо из утверждений, приведенных ниже, будет непонятным, проконсультируйтесь с местным специалистом.

Система UNIX является *полнодуплексной* – символы, набранные на клавиатуре, посылаются в систему, которая, в свою очередь, посылает

[1] Под «справочным руководством по UNIX» имеется в виду интерактивная справочная система UNIX (так называемые man pages). Для ее применения надо просто иметь доступ к UNIX-машине. – *Примеч. науч. ред.*

их на терминал для вывода на экран. Как правило, такой эхо-процесс копирует символы прямо на экран, так что пользователь может видеть то, что он ввел; но в некоторых случаях, например при вводе пароля, эхо отключается, и символы на экран не выводятся.

Большинство символов на клавиатуре – это обычные печатные символы, у которых нет какого-либо специального значения, но есть и клавиши, которые сообщают компьютеру, как интерпретировать ввод. Безусловно, самая важная из таких клавиш – это *Return*. Нажатие клавиши *Return* означает окончание строки ввода; система реагирует на это перемещением курсора на терминале в начало следующей строки экрана. Для того чтобы система приступила к интерпретации вводимых символов, необходимо нажать *Return*.

Return – это пример *управляющего символа,* невидимый символ, который управляет некоторыми аспектами ввода с терминала и вывода на него. На любом нормальном терминале есть специальная клавиша *Return*, в отличие от большинства остальных управляющих символов. Вместо этого они должны вводиться следующим образом: нажимается и удерживается клавиша *Control* (может также называться *Ctl*, *Cntl* или *Ctrl*), а затем нажимается другая клавиша, обычно с буквой. Например, чтобы ввести *Return*, можно нажать клавишу *Return*, а можно, удерживая клавишу *Control*, ввести букву «m», так что *ctl*-m представляет собой альтернативное имя для *Return*. Другие управляющие символы включают: *ctl*-d, который сообщает программе, что ввод закончен; *ctl*-g, который воспроизводится на терминале как звуковой сигнал; *ctl*-h, обычно называемый *Backspace* (возврат на одну позицию), с помощью которого исправляют ошибки; и *ctl*-i – символ табуляции, который перемещает курсор на следующую позицию табуляции почти так же, как на обычной пишущей машинке. Позиции табуляции в системах UNIX разделены восемью пробелами. Символы табуляции и возврата на одну позицию имеют собственные клавиши на многих терминалах.

Еще две клавиши со специальным значением: *Delete*, иногда называемая *Rubout*[1] или какой-нибудь аббревиатурой, и *Break*, иногда называемая *Interrupt*. В большинстве систем UNIX нажатие клавиши *Delete* немедленно останавливает программу, не ожидая ее завершения. В некоторых системах эту функцию выполняет *ctl*-c. А также на некоторых системах, в зависимости от того, как подключены терминалы, *Break* – это также синоним *Delete* или *ctl*-c.

Сессия UNIX

Давайте начнем с диалога между пользователем и его системой UNIX, который мы будем комментировать. Во всех примерах этой книги при-

[1] Rub out – стирать, вычищать. – *Примеч. ред.*

няты следующие обозначения: то, что печатает пользователь, записывается *наклонными буквами*, ответы компьютера – обычными символами, а комментарии – *курсивом*.

```
Установите соединение: наберите телефонный номер или включите
терминал. Система должна сказать
login: you                  Введите имя и нажмите Return
Password:                   Пароль не появится на экране, когда будет введен
You have mail.              У вас есть непрочитанные письма
$                           Теперь система готова к выполнению команд
$                           Нажмите Return несколько раз
$ date                      Который час и какое сегодня число?
Sun Sep 25 23:02:57 EDT 1983
$ who                       Кто сейчас пользуется системой?
jlb      tty0    Sep 25 13:59
you      tty2    Sep 25 23:01
mary     tty4    Sep 25 19:03
doug     tty5    Sep 25 19:22
egb      tty7    Sep 25 17:17
bob      tty8    Sep 25 20:48
$ mail                      Читать почту
From doug Sun Sep 25 20:53 EDT 1983
give me a call sometime Monday

?                           Return – перейти к следующему сообщению
From mary Sun Sep 25 19:07 EDT 1983     Следующее сообщение
Lunch at noon tomorrow?

? d                         Удалить это сообщение
$                           Больше нет сообщений
$ mail mary                 Отправить почту пользователю mary
lunch at 12 is fine
ctl-d                   Конец письма
$                       Повесьте трубку или выключите терминал. Это все
```

Иногда сеанс на этом заканчивается, хотя время от времени люди делают заодно какую-нибудь работу. Оставшаяся часть этого раздела будет посвящена обсуждению приведенного выше сеанса и некоторых других программ, которые могут оказаться полезными.

Вход в систему

У пользователя должны быть имя и пароль, которые можно получить у системного администратора. Система UNIX поддерживает работу с разными терминалами, но она строго ориентирована на устройства, имеющие *нижний регистр*. Регистр имеет большое значение! Так что если терминал работает только в верхнем регистре (как некоторые видео- и портативные терминалы), жизнь пользователя превратится в такую пытку, что ему придется поискать другой терминал.

Убедитесь в том, что все установки выполнены соответствующим образом: верхний и нижний регистр, полный дуплекс и другие параметры,

которые может посоветовать местный специалист, например быстродействие или *скорость соединения* (*baud rate*). Установите соединение, используя те заклинания, которые нужны вашему терминалу; это может означать набор телефонного номера или просто щелчок тумблером. В любом случае система должна написать

```
login:
```

Если она пишет что-то непонятное, вероятно, установлена неправильная скорость; проверьте это значение и другие установки. Если это не поможет, несколько раз, не торопясь, нажмите клавишу *Break* или *Interrupt*. Если сообщение о начале сеанса так и не появляется, остается только позвать на помощь.

Когда сообщение `login:` получено, введите свое имя пользователя в *нижнем регистре*, затем нажмите *Return*. Если нужен пароль, система попросит его ввести и отключит на это время вывод на экран.

Кульминация усилий по входу в систему – это *приглашение*, обычно одиночный символ, который указывает, что система готова принимать команды. Наиболее часто в строке приглашения выводится знак доллара `$` или процента `%`, но можно заменить его любым, наиболее понравившимся; дальше будет рассказано, как это сделать. Приглашение на самом деле печатается программой, называемой *командным процессором*, или *оболочкой* (*shell*)[1]; она является основным интерфейсом пользователя для системы.

Непосредственно перед приглашением может присутствовать уведомление о наличии почты или «сообщение дня». Также может быть задан вопрос о типе подключенного терминала; ответ поможет системе учитывать специфические свойства терминала.

Ввод команд

Как только получено приглашение, можно начинать вводить *команды*, которые являются просьбой к системе выполнить некое действие. Слово *программа* будет употребляться как синоним команды. Итак, когда получено приглашение (будем считать, что это `$`), введите `date` и нажмите клавишу *Return* (*Enter*). Система должна ответить, выдав дату и время, а затем вывести новое приглашение, таким образом, вся транзакция будет выглядеть на терминале следующим образом:

```
$ date                    Который час и какое сегодня число?
Sun Sep 25 23:02:57 EDT 1983
$
```

[1] В профессиональном разговоре вы вряд ли услышите слово «оболочка». По всей вероятности, ваш собеседник скажет просто «шелл». – *Примеч. науч. ред.*

Не забудьте нажать клавишу *Return* и не вводите символ `$`. Если кажется, что система не обращает на вас внимания, нажмите *Return*, – что-нибудь должно произойти. *Return* больше не будет упоминаться, но не забывайте нажимать эту клавишу в конце каждой строки.

Теперь опробуем команду `who`, которая сообщает о том, кто в настоящее время находится в системе:

```
$ who
rlm       tty0     Sep 26 11:17
pjw       tty4     Sep 26 11:30
gerard    tty7     Sep 26 10:27
mark      tty9     Sep 26 07:59
you       ttya     Sep 26 12:20
```

Первая колонка содержит имена пользователей. Во второй находятся системные имена для соединений (tty означает «teletype» (телетайп) – устаревший синоним «терминала»). Все остальное – это информация о том, когда пользователь вошел в систему. Можно также попробовать

```
$ who am i
you       ttya     Sep 26 12:20
```

Если при вводе названия команды была допущена ошибка или введена несуществующая команда, то в ответ система выдаст сообщение о том, что такое имя не найдено:

```
$ whom                Команда введена с ошибкой...
whom: not found       ...поэтому система не знает, как ее выполнить
$
```

Если же случайно введено имя некой другой существующей команды, она выполнится, и результат, вероятно, будет трудно постижимым.

Странное поведение терминала

Однажды терминал может начать вести себя странно, например каждая буква выводится дважды или же *Return* не перемещает курсор на начало следующей строки. Обычно можно исправить положение, выключив и заново включив терминал или же выйдя из системы и войдя снова. Можно также прочесть описание команды `stty` (set terminal options – задание установок терминала) в разделе 1 руководства (man1). Для правильной обработки знаков табуляции (если сам терминал их не обрабатывает) введите команду

```
$ stty -tabs
```

и система преобразует символ табуляции в соответствующее количество пробелов. Если терминал позволяет устанавливать шаг табуляции, то это можно сделать посредством команды `tabs`. (На самом деле может потребоваться ввести

```
$ tabs    тип-терминала
```

для того, чтобы все заработало (обратитесь к описанию команды `tabs` в руководстве.)

Ошибки ввода

Если при вводе сделана ошибка и она обнаружена до нажатия *Return*, то существуют два способа исправления: *стереть* символы один за другим или всю строку целиком, а потом набрать ее заново.[1]

Если вводится символ *стирания строки* (*line kill* character) (по умолчанию символ `@`), то вся строка будет проигнорирована, как будто она никогда и не была введена, и ввод будет продолжен с новой строки:

```
$ ddtae@                    Абсолютно неправильно, начинаем заново
date                        с новой строки
Mon Sep 26  12:23:39 EDT 1983
$
```

Символ `#` (знак диеза) стирает последний введенный символ; каждый новый `#` стирает еще один символ по направлению к началу строки (но не за ее пределами). Так что ошибочный ввод можно исправить следующим образом:

```
$ dd#atte##e                Исправляем прямо в процессе ввода
Mon Sep 26  12:24:02 EDT 1983
$
```

На какие именно символы возложены функции стирания символа (забоя) и удаления строки, в большой степени зависит от системы. Во многих системах (включая ту, с которой работают авторы) знак забоя заменен на символ возврата на одну позицию (backspace), что удобно для видеотерминалов. Можно без труда проверить, как обстоят дела в конкретной системе:

```
$ datee←                    Проверим ←
datee←: not found           Это не ←
$ datee#                    Попробуем #
Mon Sep 26  12:26:08 EDT 1983   Это #
$
```

(Символ возврата на одну позицию изображен как ← для большей наглядности). В качестве символа удаления строки также часто используется *ctl*-`u`.

В данном разделе символ забоя будет обозначаться как диез `#`, поскольку это отображаемый символ, но помните, что ваша система может отличаться. В разделе «Настройка окружения» будет показано, как

[1] Материал этого раздела вряд ли имеет отношение к тем UNIX-системам, с которыми будет иметь дело читатель. Приведенная информация касается «настоящих» UNIX-систем с «настоящими» терминалами. В PC-шных версиях UNIX все попроще и поудобнее. – *Примеч. науч. ред.*

можно раз и навсегда определить знаки забоя и удаления строки по усмотрению пользователя.

Что делать, если требуется ввести знак забоя или удаления строки как часть текста? Если предварить символы # или @ обратной косой чертой \, они потеряют свое особое значение. Поэтому, чтобы ввести # или @, наберите \# или \@. Система может переместить курсор на следующую строку после прочтения @, даже если перед ним стояла обратная косая черта. Не беспокойтесь, «коммерческое at» было записано.

Обратная косая черта, иногда называемая *escape-символом*, широко используется для указания того, что следующий за ней символ имеет специальное значение. Чтобы стереть обратную косую черту, надо ввести два символа исключения: \##. Понятно почему?

Вводимые символы, прежде чем они доберутся до места назначения, рассматриваются и интерпретируются целым рядом программ, а способ их интерпретации зависит не только от места назначения, но и от того, каким образом они туда попали.

Каждый введенный символ незамедлительно отображается на терминале, если только эхо не выключено, что бывает достаточно редко. Пока не нажата клавиша *Return*, символы временно хранятся ядром, поэтому опечатки можно поправить при помощи символов забоя и аннулирования строки. Если символ забоя или удаления строки предварен обратной косой чертой, то ядро отбрасывает черту и сохраняет последующий символ без интерпретации.

При нажатии клавиши *Return* сохраненные символы посылаются в программу, которая занимается чтением с терминала. Эта программа, в свою очередь, может интерпретировать символы специальным образом; например, оболочка (командный процессор) не воспринимает как имеющие особое значение символы, перед которыми стоит обратная косая черта. Об этом будет рассказано в главе 3. Пока же запомните, что ядро обрабатывает удаление строки, забой и обратную косую черту, только если черта стоит перед знаком удаления строки или забоя; оставшиеся же символы могут быть особым образом интерпретированы другими программами.

Упражнение 1.1. Объясните, что произойдет с

```
$ date\@
```

□

Упражнение 1.2. Большинство командных процессоров (в отличие от оболочки UNIX System 7) интерпретируют знак диеза # как начало комментария и игнорируют весь текст от # до конца строки. Учитывая это, поясните запись, представленную ниже, считая, что # – это знак исключения:

```
$ date
Mon Sep 26  12:39:56 EDT 1983
```

```
$ #date
Mon Sep 26  12:40:21 EDT 1983
$ \#date
$ \\#date
#date: not found
$
```

□

Опережающий ввод с клавиатуры

Ядро считывает ввод с клавиатуры по мере поступления, даже если оно одновременно занято чем-то еще, так что можно печатать сколь угодно быстро, в любой момент, даже если какая-то команда выполняет печать. Если ввод с клавиатуры выполняется в то время, пока система печатает, введенные символы появятся на экране вперемешку с выводимыми, но они сохранятся отдельно и будут интерпретированы корректно. Можно вводить команды одну за другой, не дожидаясь их завершения или даже старта.

Остановка программы

Большинство команд можно остановить, введя символ *Delete*.[1] Клавиша *Break*, которая есть на большинстве терминалов, тоже может остановить программу, но это зависит от конкретной системы. В некоторых случаях, например в текстовых редакторах, *Delete* останавливает любое действие, выполняемое программой, но оставляет вас внутри программы. Большинство программ будут остановлены при выключении терминала или разрыве телефонного соединения.

Если требуется лишь приостановить вывод, например, чтобы сохранить на экране выводимые данные, введите *ctl*-s. Вывод остановится практически сразу же; программа будет находиться в «подвешенном» состоянии до тех пор, пока ее не запустят вновь. Чтобы возобновить вывод, введите *ctl*-q.

Выход из системы

Правильный способ выхода из системы – это ввод *ctl*-d вместо команды; так оболочка получает сообщение о том, что ввод закончен. (В следующей главе будет подробно описано, как именно это происходит.) Обычно можно просто выключить терминал или повесить телефонную трубку, но осуществляется ли при этом на самом деле выход, зависит от системы.

---

[1] Для большинства распространенных UNIX-систем это код нажатия клавиш *ctl*-C. – *Примеч. науч. ред.*

Почта

В системе имеется почтовая служба, посредством которой пользователи могут общаться друг с другом, поэтому, войдя в систему, вы можете увидеть сообщение

```
You have mail.
```

перед первым приглашением на ввод команды. Чтобы прочитать почту, введите

```
$ mail
```

Сообщения будут показаны по одному, начиная с самого свежего. После каждого элемента `mail` ждет указания, как поступить с этим сообщением. Два основных ответа системе: `d` – удалить сообщение и *Return* – не удалять (то есть оно еще будет доступно, когда вы в следующий раз захотите почитать почту). Возможны варианты: `p` – распечатать сообщение, `s` *имя-файла* – сохранить сообщение в файле с указанным именем и `q` – выйти из `mail`. (Если вы не знаете, что такое файл, считайте, что это такое место, где можно сохранить информацию под выбранным именем и взять ее оттуда позже. Файлы – это тема раздела 1.2, да и большинства других разделов этой книги.)

Программа `mail` – это одна из тех программ, которая может существовать в различных вариантах. Она вполне может отличаться от приведенного здесь описания. Подробная информация представлена в руководстве.

Отправить почту другому пользователю очень просто. Пусть надо послать сообщение человеку с регистрационным именем `nico`. Самый простой способ сделать это выглядит так:

```
$ mail nico
Теперь введите текст письма, заполнив
столько строчек, сколько потребуется...
Завершив последнюю строку письма,
нажмите control-d
ctl-d
$
```

Символ *ctl*-`d` указывает на конец письма, сообщая команде `mail`, что больше данные вводиться не будут. Если на полпути вы передумаете отправлять письмо, нажмите *Delete* вместо *ctl*-`d`. Наполовину сформированное письмо будет не отправлено, а сохранено в файле с именем `dead.letter`.

Для тренировки отправьте письмо самому себе, затем введите `mail`, чтобы прочитать его. (На самом деле это не такое уж безумие, каким может показаться, – это простой и удобный способ напомнить себе о чем-нибудь важном.)

Есть и другие способы отправки сообщений – можно послать заранее подготовленное письмо, можно отправить одно сообщение сразу нескольким адресатам, можно также отправить сообщение пользователям, работающим на других компьютерах. Команда `mail` подробно описана в разделе 1 справочного руководства по UNIX. С этого момента будем использовать условное обозначение `mail(1)`, подразумевая страницу, описывающую команду `mail` в man1. Все команды, обсуждаемые в этой главе, также описаны в man1.

Кроме этого, в вашей системе может оказаться сервис напоминаний, который называется календарем (см. `calendar(1)`); в главе 4 будет описано, как его настроить, если это не было сделано ранее.

Общение с другими пользователями

Если в вашей UNIX-системе работают несколько пользователей, то однажды, как гром среди ясного неба, на вашем экране появится что-то вроде

```
Message from mary tty7...  Сообщение от mary
```

в сопровождении замечательного звукового сигнала. Мэри хочет написать вам что-то, но пока вы не выполните определенные действия, не сможете ответить ей тем же. Чтобы ответить, введите

```
$ write mary
```

Так устанавливается двусторонний канал связи. Теперь все, что Мэри введет на своем терминале, отобразится на вашем, и наоборот. Надо отметить, что происходит это очень медленно, как будто вы разговариваете с Луной.

Если вы работаете с какой-то программой, надо перейти в такое состояние, в котором будет возможен ввод команды. Обычно, какая бы программа ни была запущена, она должна остановиться или быть остановлена, но в некоторых программах, как, например, в каком-нибудь редакторе и в самой `write`, существует команда `!` для временного выхода в оболочку (табл. 2 в приложении 1).

Команда `write` не накладывает каких-либо ограничений, поэтому если вы не хотите, чтобы печатаемые вами символы перемешались с тем, что печатает Мэри, вам необходим некий протокол – соглашение об обмене сообщениями. Одно из соглашений заключается в том, чтобы соблюдать очередность, обозначая конец каждого фрагмента с помощью `(o)` – от английского «over» (окончено), а также сообщать о своем намерении закончить диалог при помощи `(oo)` – «over and out» (заканчиваю и выхожу).

Терминал Мэри:	*Ваш терминал:*
`$ write you`	
	`$ Message from mary tty7...`
	`write mary`

```
Message from you ttya...
did you forget lunch? (o)
                                    did you forget lunch? (o)
                                    five@
                                    ten minutes (o)
ten minutes (o)
ok (oo)
                                    ok (oo)
                                    ctl-d
EOF
ctl-d
                                    $ EOF
$
```

Еще один способ выйти из `write` – нажать *Delete*. Обратите внимание на то, что ошибки, сделанные при вводе, не отображаются на терминале Мэри.

Если предпринимается попытка пообщаться с кем-то, кто в данный момент не находится в системе или не хочет, чтобы его беспокоили, будет выдано соответствующее сообщение. Если адресат находится в системе, но не отвечает в течение достаточно большого промежутка времени, вероятно, он занят или отошел от терминала; тогда введите *ctl*-`d` или *Delete*. Если не хотите, чтобы вас беспокоили, выполните команду `mesg(1)`.

Новости

Многие UNIX-системы предоставляют сервис получения новостей, информируя пользователей о более или менее интересных событиях. Чтобы вызвать этот сервис, введите

```
$ news
```

Существует также обширная сеть UNIX-систем, которые поддерживают контакт друг с другом посредством телефонных звонков, расспросите специалиста о команде `netnews` и о USENET.[1]

Руководство по UNIX

Справочное руководство по UNIX (man) содержит большинство необходимой информации о системе. Раздел 1 знакомит читателя с командами, в том числе с представленными в данной главе. Раздел 2 описывает системные вызовы, обсуждаемые в главе 7, а раздел 6 – это информация об играх. Остальные разделы рассказывают о функциях, которые могут использовать программисты на Си, форматах файлов и

[1] Прошло уже два десятка лет с момента написания этой книги, и теперь можно так же осторожно расспросить специалиста об Интернете. – *Примеч. науч. ред.*

о сопровождении системы. (Нумерация разделов меняется от системы к системе.) Не забудьте и пермутационный указатель для начала; беглого просмотра достаточно для того, чтобы найти команды, которые могут быть полезными для выполнения конкретной задачи. Есть также введение, содержащее обзор работы системы.

Руководство часто представляет собой в системе оперативную справку, так что его можно читать прямо за терминалом. Если возникает какая-то проблема и рядом нет специалиста, способного помочь, можно вывести на терминал любую страницу руководства, набрав команду `man` *имя-команды*.

Так, чтобы прочитать справку о команде `who`, введите

```
$ man who
```

и

```
$ man man
```

чтобы прочитать о самой команде `man`.

Автоматизированное обучение

В систему может быть включена команда `learn`, которая предоставляет возможность получения информации о файловой системе и основных командах, о редакторе, подготовке документов и даже о программировании на Си. Попробуйте ввести

```
$ learn
```

Если команда `learn` присутствует в системе, она скажет, что делать дальше. Если с `learn` ничего не выходит, попробуйте команду `teach`.

Игры

Одно из лучших средств почувствовать себя свободно наедине с компьютером и терминалом (хотя это и не признается официально) – это компьютерные игры. В комплект поставки системы UNIX входит не много игр, но можно пополнить запасы на месте. Поспрашивайте товарищей или обратитесь к разделу 6 руководства (man 6).

1.2. Повседневная работа: файлы и основные команды

Информация в системе UNIX хранится в *файлах*, которые по сути своей очень похожи на обычные офисные папки для документов. У каждого файла есть имя, содержимое, место для хранения и некоторая административная информация о его владельце и размере. Файл

может содержать письмо, список имен или адресов, исходные тексты программы, данные для обработки программой или даже саму программу в исполняемой форме и другие нетекстовые материалы.

Файловая система UNIX организована таким образом, что вы можете работать со своими файлами, не вмешиваясь в работу других пользователей и не позволяя другим мешать себе. Несть числа командам, осуществляющим различные манипуляции с файлами; для начала рассмотрим только самые часто используемые. Глава 2 предлагает систематическое описание файловой системы и знакомит с многими другими командами, имеющими отношение к работе с файлами.

Создание файлов – редактор

Если надо напечатать письмо, документ или программу, каким образом извлечь информацию, хранящуюся в компьютере? Большинство задач такого рода выполняется с помощью *текстового редактора* – программы для хранения и обработки информации в компьютере. Практически в каждой UNIX-системе есть *экранный редактор* – редактор, который использует способность современных терминалов отражать редакционные изменения, как только они сделаны. Два наиболее популярных экранных редактора – это `vi` и `emacs`.

Мы не будем описывать здесь конкретные редакторы, частично из-за ограниченного объема книги, а частично потому, что ни один из них не является стандартным.[1] Однако есть более старый редактор `ed`, который наверняка доступен в каждой системе. Он не поддерживает специальные возможности, присущие некоторым терминалам, поэтому может работать на любом. Он также является основой других важнейших программ (в том числе нескольких экранных редакторов), так что стоит того, чтобы его изучили. Краткое описание этого редактора приводится в приложении 1.

Вне зависимости от того, какой редактор вы лично предпочитаете, его следует изучить достаточно хорошо для того, чтобы создавать файлы. В этой книге – для конкретизации и для гарантии работоспособности предложенных примеров на любой машине – обсуждается `ed`, но вы, безусловно, можете остановить свой выбор на другом редакторе – том, какой вам больше нравится.

Чтобы создать с помощью `ed` файл с именем `junk`, содержащий некоторый текст, выполните следующее:

```
$ ed                    Вызывает текстовый редактор
a                       команда ed для добавления текста
теперь вводите
любой текст ...
```

[1] Хотя сейчас, конечно, можно сказать, что редактор `vi` все-таки является стандартным. – *Примеч. науч. ред.*

```
.                    Введите отдельно «.» – завершение ввода текста
w junk               Записать текст в файл с именем junk
39                   ed выводит количество записанных символов
q                    Выйти из ed
$
```

Команда a (append – добавить) сообщает ed о начале ввода текста. Символ «.», обозначающий конец текста, должен быть введен в начале отдельной строки. Не забудьте его – до тех пор пока этот символ не будет введен, ни одна из последующих команд ed не будет распознана – все, что вводится, будет восприниматься как продолжение текста.

Команда редактора w (write – записать) сохраняет введенную информацию; w junk сохраняет ее в файле с именем junk. Любое слово может выступать в качестве имени файла; в данном примере было выбрано слово junk (мусор), чтобы показать, что файл не очень-то важный.

В ответ ed выдает количество символов, помещенных в файл. До ввода команды w ничего записано не будет, так что если отключиться и уйти, то информация не будет сохранена в файле. (Если отключиться во время редактирования, то данные, которые обрабатывались, сохраняются в файл ed.hup, с ним можно продолжать работать в следующей сессии.) Если во время редактирования происходит отказ системы (то есть из-за сбоя аппаратного или программного обеспечения система внезапно останавливается), то файл будет содержать только те данные, которые были записаны последней командой write. После же выполнения w информация записана навсегда; чтобы получить к ней доступ, введите

```
$ ed junk
```

Конечно же, введенный текст можно редактировать: исправлять орфографические ошибки, изменять стиль формулировок, перегруппировывать абзацы и т. д. Закончив, введите команду q (quit – выход), чтобы выйти из редактора.

Что там за файлы?

Создадим два файла, junk и temp, чтобы знать, чем мы располагаем:

```
$ ed
a
To be or not to be
.
w junk
19
q
$ ed
a
That is the question.
.
w temp
```

```
22
q
$
```

Счетчики количества символов в `ed` учитывают и символ конца каждой строки, называемый разделителем строк (newline), с помощью которого система представляет *Return*.

Команда `ls` выводит список имен (не содержимого) файлов:

```
$ ls
junk
temp
$
```

Это действительно два только что созданных файла. (В списке могли быть и другие файлы, которых вы не создавали.) Имена автоматически сортируются в алфавитном порядке.

Как и у многих других команд, у `ls` есть параметры, предназначенные для изменения поведения команды по умолчанию. Параметры вводятся в командной строке после имени команды, обычно они составлены из знака минус «-» и какой-то одной буквы, которая и определяет конкретное значение. Например, команда `ls -t` задает вывод файлов, упорядоченный по времени (time) их последнего изменения, начиная с самого свежего.

```
$ ls -t
temp
junk
$
```

Параметр `-l` обеспечивает расширенный (long) список, предоставляющий больше информации о каждом файле:

```
$ ls -l
total 2
-rw-r--r-- 1 you          19 Sep 26 16:25 junk
-rw-r--r-- 1 you          22 Sep 26 16:26 temp
$
```

`total 2` сообщает, сколько блоков дискового пространства занято файлами; блок – это обычно 512 или 1024 символа. Строка `-rw-r--r-` информирует о том, у кого есть права на чтение файла и запись; в данном случае владелец `you` может читать файл и писать в него, а остальные пользователи могут только читать. Следующий за строкой прав доступа символ `1` – это количество ссылок (links) на файл; пока не обращайте на него внимания, поговорим о нем в главе 2. Владелец файла (то есть пользователь, который его создал) – это `you`. `19` и `22` – это значения количества символов в соответствующих файлах; они совпадают

со значениями, выданными ed. Далее следуют дата и время последнего изменения файла.

Параметры могут быть сгруппированы: ls -lt выдает те же данные, что и ls -l, но отсортированные по времени, начиная с самого нового. Параметр -u предоставляет информацию о том, когда файлы использовались: ls -lut выдает расширенный (-l) список, отсортированный в порядке времени использования, начиная с последнего. Параметр -r изменяет порядок вывода на обратный, так что ls -rt выводит файлы в порядке, обратном времени их последнего изменения. Можно также после команды указать имена файлов, тогда ls выведет информацию только по указанным файлам:

```
$ ls -l junk
-rw-r--r-- 1 you          19 Sep 26 16:25 junk
$
```

Строки, следующие за именем команды в командной строке, как -l и junk в примере выше, называются *аргументами* команды. Обычно аргументы – это параметры или имена файлов, которые должны использоваться командой.

Запись параметра при помощи знака минус и одной (например, -t) или нескольких букв (например, -lt) – это общепринятое соглашение. Обычно если команда допускает использование таких необязательных параметров, то они должны предшествовать именам файлов, но возможен и другой порядок. Хотя надо отметить, что UNIX-программы достаточно капризны в том, что касается обработки нескольких параметров. Например, стандартная UNIX System 7 ls не воспримет

```
$ ls -l -t              Не работает в UNIX System 7
```

как синоним ls -lt, тогда как другие программы требуют разделения параметров.

По мере изучения UNIX станет понятно, что какая-то регулярность или система в отношении необязательных аргументов практически отсутствует. У каждой команды есть только ей присущие особенности, и каждая сама определяет, что будет обозначать какая буква (так что одна и та же функция в разных командах может быть обозначена разными буквами). Такая непредсказуемость приводит в замешательство, и именно ее часто называют главным недостатком системы. И хотя ситуация постепенно улучшается – в новых версиях уже больше единообразия, единственное, что можно порекомендовать – это стараться держать руководство поблизости, когда будете писать собственные программы.

Печать файлов: cat и pr

Итак, мы создали несколько файлов. Как просмотреть их содержимое? Есть много программ, способных выполнить эту задачу, может

быть, даже больше, чем нужно. Одна из возможностей состоит в том, чтобы запустить редактор:

```
$ ed junk
19                      ed сообщает о том, что в junk 19 символов
1,$p                    Напечатать строки с первой по последнюю
To be or not to be      В файле всего одна строка
q                       Все сделано
$
```

Сначала ed выводит количество символов в junk; команда 1,$p сообщает редактору, что надо напечатать все строки этого файла. Изучив ed, вы сможете печатать только необходимые фрагменты, а не целый файл.

Бывают случаи, когда редактор не подходит для печати. Например, существует предельный размер файла (несколько тысяч строк), который ed может обработать. Более того, он печатает только один файл за раз, а возможны ситуации, когда требуется печать нескольких файлов – одного за другим, без остановок. Итак, есть несколько других вариантов.

Первый вариант – это cat, самая простая из всех печатающих программ; она печатает содержимое всех файлов, имена которых указаны в ее аргументах:

```
$ cat junk
To be or not to be
$ cat temp
That is the question.
$ cat junk temp
To be or not to be
That is the question.
$
```

Названные в аргументах файлы или файл выводятся на терминал вместе (отсюда имя cat – catenate[1] – соединять), один за другим, без каких-либо разделителей.

С небольшими файлами проблем не возникает, а вот если файл большой и соединение с компьютером высокоскоростное, то надо иметь очень хорошую реакцию, чтобы успеть нажать *ctl*-s и остановить вывод cat, пока он еще не ускользнул с экрана. Нет «стандартной» команды, позволяющей выводить файл на видеотерминал порциями, помещающимися на одном экране, но во многих UNIX-системах такая возможность есть. В вашей системе такая команда может называться pg или more. Наша называется p; о ней будет рассказано в главе 6.

Как и cat, команда pr печатает содержимое всех файлов, перечисленных в списке, но в форме, пригодной для построчнопечатающих устройств: каждая страница содержит 66 строк (11 дюймов в высоту), к

[1] Catenate – это редко употребляемый синоним слова concatenate (связывать, соединять).

которым добавлены дата и время изменения файла, имя файла и номер страницы, а также несколько пустых строк на месте сгиба бумаги. Итак, чтобы аккуратно распечатать junk, потом перейти на начало новой страницы и так же распечатать temp, надо выполнить команду:

```
$ pr junk temp

Sep 26 16:25 1983  junk Page 1

To be or not to be
        (60 more blank lines)

Sep 26 16:26 1983  temp Page 1

That is the question.
        (60 more blank lines)
$
```

Программа pr также может осуществлять вывод в несколько колонок:

```
$ pr -3 имена-файлов
```

Каждый файл будет распечатан в 3 колонки. Вместо «3» можно написать любое разумное число, и pr постарается сделать все наилучшим образом. (На месте слов *имена-файлов* должен быть введен список файлов для печати.) Команда pr -m выведет каждый файл в отдельном столбце. См. описание pr(1).

Отметим, что pr *не является* программой форматирования текста в том, что касается перегруппировки строк или выравнивания краев. Настоящие средства форматирования – это nroff и troff, которые будут описаны в главе 9.

Существуют также команды, печатающие файлы на высокоскоростном принтере. Почитайте в man о командах lp и lpr или найдите статью «printer» в пермутационном указателе.[1] Выбор программы зависит от оборудования, подключенного к компьютеру. Команды pr и lpr часто используются вместе – после того как pr отформатирует информацию надлежащим образом, lpr запускает механизм вывода на построчно печатающий принтер. Вернемся к этому чуть позже.

Перемещение, копирование и удаление файлов – mv, cp, rm

Давайте посмотрим еще на какие-нибудь команды. Первое, что можно сделать, – это изменить имя файла. Переименование осуществляется посредством «перемещения» файла из одного имени в другое, например:

```
$ mv junk precious
```

[1] В большинстве современных UNIX для этой цели подойдут команды man -k printer или apropos printer. Попробуйте и увидите, что будет. – *Примеч. науч. ред.*

Файл, который назывался `junk`, теперь называется `precious`; содержимое при этом не изменилось. Если теперь запустить `ls`, список будет выглядеть по-другому: в нем больше нет `junk`, зато есть `precious`.

```
$ ls
precious
temp
$ cat junk
cat: can't open junk
$
```

Будьте осторожны: если переименовать файл в уже существующий, то файл назначения будет заменен.

Чтобы создать *копию* файла (то есть чтобы иметь две версии чего-то), используйте команду `cp`:

```
$ cp precious precious.save
```

Дубликат `precious` создается в `precious.save`.

И наконец, когда вам надоест создавать и перемещать файлы, команда `rm` удалит все указанные файлы:

```
$ rm temp junk
rm: junk nonexistent
$
```

Если один из файлов, который надо удалить, не существует, программа выдаст соответствующее сообщение, в остальных же случаях `rm`, как и большинство команд UNIX, работает «молча». Нет ни подсказок, ни пустой болтовни, сообщения об ошибках лаконичны и, бывает, бесполезны. Новичков такая краткость может привести в замешательство, а вот опытных пользователей раздражают «словоохотливые» программы.

Каким должно быть имя файла?

До сих пор мы использовали имена файлов, не обсуждая, какими они вообще могут быть, так что теперь пришло время привести несколько правил. Во-первых, длина имени файла ограничена 14 символами.[1] Во-вторых, хотя практически все символы разрешены в именах файлов, здравый смысл подсказывает, что разумнее придерживаться отображаемых символов и по возможности избегать символов, которые могут иметь другое значение. Например, в команде `ls`, рассмотренной ранее, `ls -t` означает вывод списка, упорядоченного по времени. Так что

[1] Современные версии операционных систем семейства UNIX позволяют присваивать файлам и каталогам имена, длина которых достигает 255 символов. – *Примеч. науч. ред.*

если дать файлу имя -t, тяжело будет добиться его вывода по имени. (Кстати, как это сделать?) Кроме знака минус в начале слова есть и другие символы со специальным значением. Чтобы избежать недоразумений, пока не разберетесь во всем досконально, лучше ограничиться буквами, цифрами, точками и знаками подчеркивания. (Точками и символами подчеркивания принято разделять имя файла на части, как в примере про precious.save.) И последнее – не забудьте о том, что регистр также имеет значение: junk, Junk и JUNK – это три разных имени.

Несколько полезных команд

Теперь, когда нам известны основы создания файлов, просмотра их имен и вывода их содержимого, изучим еще полдюжины команд, занимающихся обработкой файлов.

Чтобы конкретизировать обсуждение, будем использовать файл с именем poem, который содержит знаменитое стихотворение Огастеса Де Моргана (Augustus De Morgan). Создадим файл с помощью ed:

```
$ ed
a
Great fleas have little fleas
  upon their backs to bite 'em,
And little fleas have lesser fleas,
  and so ad infinitum.
And the great fleas themselves, in turn,
  have greater fleas to go on;
While these again have greater still,
  and greater still, and so on.
.
w poem
263
q
$
```

Первой рассмотрим команду wc, подсчитывающую количество строк, слов и символов в одном или нескольких файлах; она получила имя за свою деятельность – подсчет слов (word-counting):

```
$ wc poem
       8      46     263 poem
$
```

Это означает, что в poem 8 строк, 46 слов и 263 символа. Понятие «слово» определяется чрезвычайно просто: любая строка символов, не содержащая пробелов, знаков табуляции и разделителей строк.

Команда wc может обсчитать несколько файлов (и вывести общие результаты), может также исключить подсчет какой-либо категории. См. описание wc(1).

Следующая команда называется grep; она просматривает файл в поиске строк, соответствующих шаблону. (Имя команды происходит от названия команды g/*regular-expression*/p (g/*регулярное-выражение*/p) редактора ed, о которой рассказано в приложении 1.) Предположим, что требуется найти слово «fleas» в файле poem:

```
$ grep fleas poem
Great fleas have little fleas
And little fleas have lesser fleas,
And the great fleas themselves, in turn,
  have greater fleas to go on;
$
```

Если указать параметр -v, то grep будет искать строки, *не* соответствующие шаблону. (Параметр назван «v» по имени команды редактора; можно считать, что он меняет смысл соответствия на обратный.)

```
$ grep -v fleas poem
  upon their backs to bite 'em,
  and so ad infinitum.
While these again have greater still,
  and greater still, and so on.
$
```

Команда grep может быть использована для просмотра нескольких файлов; в этом случае перед началом каждой строки, удовлетворяющей шаблону, будет выведено имя файла, чтобы было понятно, где найдено соответствие. Существуют также параметры для подсчета, нумерации и т. д. Также grep может обрабатывать шаблоны гораздо более сложные, чем просто слово (как «fleas»), но оставим обсуждение этой возможности до главы 4.

Третья команда, sort, сортирует введенные данные по алфавиту, строка за строкой. Для стихотворения это выглядит достаточно странно, но все-таки попробуем ее применить, чтобы посмотреть, что получится:

```
$ sort poem
  and greater still, and so on.
  and so ad infinitum.
  have greater fleas to go on;
  upon their backs to bite 'em,
And little fleas have lesser fleas,
And the great fleas themselves, in turn,
Great fleas have little fleas
While these again have greater still,
$
```

Упорядочение выполнено построчно, при этом порядок сортировки по умолчанию выглядит следующим образом: сначала пробелы, потом буквы в верхнем регистре и затем в нижнем регистре, так что алфавитный порядок соблюдается не строго.

Команда `sort` имеет огромное количество параметров, определяющих порядок сортировки: обратный, по числовым значениям, в лексикографическом порядке, игнорируя начальные пробелы, по полям внутри строки и т. д. – обычно лучше просмотреть все эти параметры, чтобы быть уверенным в их значении. Перечислим наиболее часто используемые из них:

`sort -r`	*Порядок, обратный обычному*
`sort -n`	*Сортировать по значениям чисел*
`sort -nr`	*По значениям чисел в обратном порядке*
`sort -f`	*Игнорировать различие регистров*[1]
`sort +n`	*Сортировать начиная с n+1-го поля*

Более подробная информация о команде `sort` представлена в главе 4.

Еще одна команда для работы с файлом – это `tail`. Она выводит последние 10 строк файла. Это слишком для стихотворения из 8 строчек, но для больших файлов это хорошо. Кроме того, у `tail` есть параметр, задающий количество строк для просмотра, поэтому чтобы напечатать последнюю строку стихотворения:

```
$ tail -1 poem
  and greater still, and so on.
$
```

`tail` также может печатать файл, начиная с заданной строки:

```
$ tail +3 имя-файла
```

Вывод начнется со строки 3. (Обратите внимание на естественную инверсию соглашения о знаке минус в аргументе.)

И последняя пара команд, которая служит для сравнения файлов. Предположим, что вариант `poem` хранится в файле `new_poem`:

```
$ cat poem
Great fleas have little fleas
  upon their backs to bite 'em,
And little fleas have lesser fleas,
  and so ad infinitum.
And the great fleas themselves, in turn,
  have greater fleas to go on;
While these again have greater still,
  and greater still, and so on.
$ cat new_poem
Great fleas have little fleas
  upon their backs to bite them,
And little fleas have lesser fleas,
  and so on ad infinitum.
```

[1] При этом b сортируется как B: b раньше, чем D, но B всегда будет выведено раньше, чем b. – *Примеч. науч. ред.*

```
And the great fleas themselves, in turn,
  have greater fleas to go on;
While these again have greater still,
  and greater still, and so on.
$
```

Эти два файла мало чем отличаются друг от друга, и человеку тяжело обнаружить отличия. На помощь приходят команды сравнения файлов. Команда `cmp` находит первое несовпадение:

```
$ cmp poem new_poem
poem new_poem differ: char 58, line 2
$
```

Это означает, что первое отличие встречается во второй строке (что действительно так), но при этом нет никакой информации о том, в чем это отличие заключается и есть ли еще различия, кроме первого.

Вторая команда, выполняющая сравнение файлов, сообщает обо всех измененных, удаленных или добавленных строках и называется `diff`:

```
$ diff poem new_poem
2c2
<   upon their backs to bite 'em,
---
>   upon their backs to bite them,
4c4
<   and so ad infinitum.
---
>   and so on ad infinitum.
$
```

Это означает, что строка 2 первого файла (`poem`) должна быть заменена на строку 2 второго файла (`new_poem`), аналогично для строки 4.

Вообще говоря, `cmp` применяется, когда надо убедиться, что файлы действительно имеют одинаковое содержимое. Эта команда быстро выполняется и подходит для любых (не только текстовых) файлов. А команда `diff` применяется, когда есть подозрение в том, что у файлов есть какие-то отличия, и надо узнать, какие именно строки не совпадают. При этом `diff` обрабатывает только текстовые файлы.[1]

Команды обработки файлов: резюме

Краткий перечень рассмотренных команд обработки файлов представлен в табл. 1.1.

[1] Сравнив двоичные файлы, `diff` скажет: «Binary files *первый* and *второй* differ» или, если они не отличаются, не скажет ничего. – *Примеч. науч. ред.*

Таблица 1.1. Основные команды, относящиеся к файловой системе

Команда	Действие
`ls`	вывести список имен всех файлов в текущем каталоге
`ls` *имена-файлов*	вывести только указанные файлы
`ls -t`	вывести список, упорядоченный по времени, начиная с самого нового
`ls -l`	вывести список с более подробной информацией; также `ls -lt`
`ls -u`	вывести список, упорядоченный по времени последнего использования; также `ls -lu`, `ls -lut`
`ls -r`	вывести список в обратном порядке; также `ls -rt`, `ls -rlt` и т. д.
`ed` *имена-файлов*	редактировать указанные файлы
`cp` *file1 file2*	копировать *file1* в *file2*, перезаписывая старый *file2*, если он существовал
`mv` *file1 file2*	переместить *file1* в *file2*, перезаписывая старый *file2*, если он существовал
`rm` *имена-файлов*	безвозвратно удалить указанные файлы
`cat` *имена-файлов*	вывести содержимое указанных файлов
`pr` *имена-файлов*	вывести содержимое с заголовком, 66 строк на странице
`pr -n` *имена-файлов*	вывести в *n* колонок
`pr -m` *имена-файлов*	вывести указанные файлы рядом друг с другом (несколько колонок)
`wc` *имена-файлов*	сосчитать количество строк, слов и символов в каждом файле
`wc -l` *имена-файлов*	сосчитать количество строк в каждом файле
`grep` *шаблон имена-файлов*	вывести строки, соответствующие *шаблону*
`grep -v` *шаблон имена-файлов*	вывести строки, *не* соответствующие *шаблону*
`sort` *имена-файлов*	вывести строки файла в алфавитном порядке
`tail` *имена-файлов*	вывести 10 последних строк файла
`tail -n` *имена-файлов*	вывести *n* последних строк файла
`tail +n` *имена-файлов*	вывести, начиная с *n*-й строки
`cmp` *file1 file2*	вывести положение первого отличия
`diff` *file1 file2*	вывести все отличия между файлами

1.3. Снова о файлах: каталоги

Система отличает ваш файл `junk` от любого другого с таким же именем. Распознавание возможно благодаря тому, что файлы сгруппированы в *каталоги* (подобно тому, как книги в библиотеке расставлены по полкам), и файлы в разных каталогах могут иметь одинаковые имена, не создавая никаких конфликтов.

Обычно у каждого пользователя есть личный, или *домашний каталог*, который содержит информацию, принадлежащую только этому пользователю (иногда его еще называют каталогом регистрации[1] (login directory)). Войдя в систему, пользователь попадает в домашний каталог. Можно изменить каталог, в котором вы работаете, обычно он называется рабочим, или *текущим каталогом*, но домашний каталог всегда один и тот же. Если специально не указано иное, то при создании файла он помещается в текущий каталог. Поскольку изначально это домашний каталог, то файл не имеет никакого отношения к файлу с таким же именем, который может существовать в каталоге другого пользователя.

Каталог может включать в себя не только файлы, но и каталоги («Great directories have lesser directories...» – «У больших каталогов есть маленькие каталоги...», как в стихотворении Огастеса Де Моргана). Естественный способ представления структуры файловой системы – это дерево каталогов и файлов. Внутри этого дерева можно перемещаться в поиске файла от корня дерева по ветвям и, наоборот, можно начать с того места, где вы находитесь, и двигаться обратно к корню.

Начнем с реализации последней описанной возможности. Основным инструментом будет команда `pwd` (print working directory – показать рабочий каталог), которая выводит имя текущего каталога:

```
$ pwd
/usr/you
$
```

Это означает, что в настоящее время пользователь находится в каталоге `you`, который находится в каталоге `usr`, который, в свою очередь, находится в *корневом каталоге*, который условно обозначается как /. Символы / разделяют компоненты имени; длина каждого компонента такого имени ограничена 14 символами, как и имя файла.[2] Во многих системах `/usr` – это каталог, содержащий каталоги всех обычных пользователей системы. (Даже если домашний каталог и не `/usr/you`, команда `pwd` все равно выведет нечто подобное, так что приведенные ниже рассуждения подойдут и для этого случая.)

---

[1] Хотя на самом деле так его не называют никогда. – *Примеч. науч. ред.*

[2] Была ограничена. См. выше. – *Примеч. науч. ред.*

Если теперь ввести

```
$ ls /usr/you
```

должен быть выведен точно такой же список имен файлов, который выводился командой `ls` без параметров. Если параметры не указаны, то `ls` выводит содержимое текущего каталога, если же имя каталога указано, то команда выдает содержимое именно этого каталога.

Теперь попробуем

```
$ ls /usr
```

Будет выведен длинный список имен, среди которых будет и регистрационный каталог `you`.

Следующим шагом будет попытка вывести содержимое корневого каталога. Ответ должен быть примерно таким:

```
$ ls /
bin
boot
dev
etc
lib
tmp
unix
usr
$
```

(Не путайте два смысла символа /; это и обозначение корневого каталога, и разделитель компонентов имени файла.) Почти все перечисленные элементы представляют собой каталоги, но вот `unix` – это файл, содержащий ядро UNIX в исполняемом виде. Подробно поговорим об этом в главе 2.

Введем команду

```
$ cat /usr/you/junk
```

(если `junk` еще присутствует в домашнем каталоге). Имя

```
/usr/you/junk
```

называется *путем* к файлу. «Путь к файлу» – это название с интуитивно понятным смыслом, означающее полное имя пути от корня по дереву каталогов к конкретному файлу. В системе UNIX существует универсальное правило – везде, где может использоваться обычное имя файла, может использоваться и путевое имя.

Структура файловой системы напоминает генеалогическое дерево. Попробуем пояснить (рис. 1.1).

Файл `junk` пользователя `you` не имеет никакого отношения к файлу с таким же именем, принадлежащему Полу (`paul`) или Мэри (`mary`).

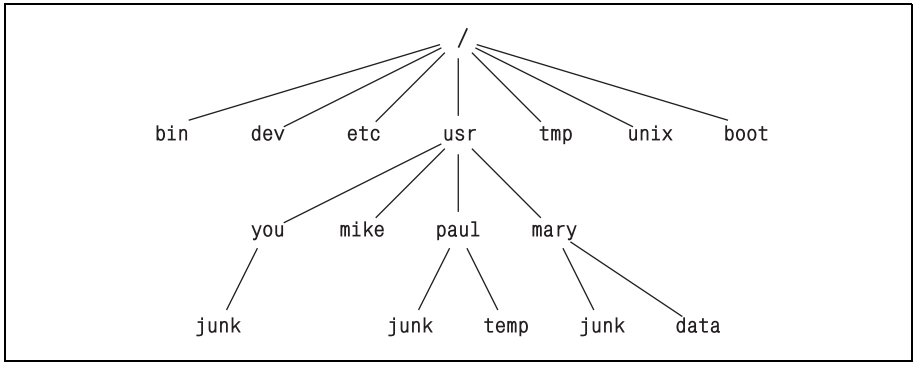

Рис. 1.1. *Дерево каталогов*

Если все интересующие пользователя файлы находятся в его личном каталоге, то путевые имена не имеют особого значения, а вот если он работает вместе с кем-то или занимается несколькими проектами одновременно, тут путевые имена становятся действительно полезными. Например, друзья могут распечатать ваш файл `junk`, просто введя команду

```
$ cat /usr/you/junk
```

Аналогично и вы можете, например, просмотреть, какие файлы есть у Мэри:

```
$ ls /usr/mary
data
junk
$
```

или сделать себе копию одного из ее файлов:

```
$ cp /usr/mary/data data
```

или даже отредактировать ее файл:

```
$ ed /usr/mary/data
```

Если пользователь не хочет, чтобы другие копались в его файлах, он может ограничить доступ к ним. Для каждого каталога и файла задаются права на чтение-запись-выполнение для владельца, группы и всех остальных, с помощью которых можно контролировать доступ. (Вспомните `ls -l`.) Пользователи наших локальных систем видят больше преимуществ в открытости, а не в секретности, но для других систем политика может быть иной. Вернемся к обсуждению этого вопроса в главе 2.

Итак, последняя серия экспериментов с путевыми именами:

```
$ ls /bin /usr/bin
```

Похожи ли полученные названия на что-то уже знакомое? Когда вы вызываете команду, печатая ее имя в командной строке (после соответствующего приглашения со стороны системы), система ищет файл с таким именем. Обычно она сначала просматривает текущий каталог (где, вероятно, файл не будет найден), затем каталог `/bin` и, наконец, `/usr/bin`. В командах, подобных `cat` или `ls`, нет ничего особенного, кроме того, что они собраны в нескольких каталогах для простоты поиска и администрирования. Чтобы проверить это, запустите какие-нибудь из подобных программ, указывая их полные путевые имена:

```
$ /bin/date
Mon Sep 26 23:29:32 EDT 1983
$ /bin/who
srm      tty1    Sep 26 22:20
cvw      tty4    Sep 26 22:40
you      tty5    Sep 26 23:04
$
```

Упражнение 1.3. Введите

```
$ ls /usr/games
```

и сделайте, что первым придет в голову. В нерабочее время некоторые вещи выглядят забавнее. □

Переход в другой каталог – cd

Если регулярно возникает необходимость работы с информацией из каталога другого пользователя, можно сказать, например: «Я хочу работать не со своими файлами, а с файлами Мэри». Такое пожелание реализуется с помощью изменения текущего каталога командой `cd`:

```
$ cd /usr/mary
```

Если теперь в качестве аргументов команд `cat` или `pr` использовать имена файлов без указания полного пути, то подразумеваться будут файлы из каталога, принадлежащего Мэри. Переход к другому каталогу никак не влияет на права доступа к файлу – если файл был недоступен из домашнего каталога, то переход в другой каталог этого факта не изменит.

Обычно бывает удобно организовать свои файлы таким образом, чтобы все, относящееся к одному проекту, находилось в одном каталоге, отдельно от других проектов. Например, кто-то, кто пишет книгу, может поместить все тексты в каталог под названием `book` (книга). Команда `mkdir` создаст новый каталог.

```
$ mkdir book        Создать каталог
$ cd book           Перейти в него
$ pwd               Убедиться, что переход осуществлен правильно
/usr/you/book
```