Лабораторная работа № 3-6ч
Знакомство с командной строкой Linux и bash-скриптами.

Цель лабораторной работы: ознакомиться с основные команды ОС Linux, научиться писать скрипты и запускать с командной строки, а также изучить утилиту make и запускать компиляцию кода на C++.

Теоретическая часть

Операционная система Linux создана на основе ОС Unix и во многом имеет схожую структуру и систему команд. Пользователь может работать в текстовом режиме с помощью командной строки (оболочки). Оболочка является одной из важнейших частей системы Unix. *Оболочка* – это программа, запускающая команды (например, те, которые вводит пользователь). Оболочка выступает также в роли небольшой среды программирования. Программисты, работающие в Unix, часто разбивают обычные задачи на несколько небольших компонентов, а затем используют оболочку, чтобы управлять задачами и собирать части воедино.

Многие важные части системы в действительности являются *сценариями оболочки* – текстовыми файлами, которые содержат последовательность команд оболочки. Если вам до этого приходилось работать в системе Windows, то вы можете представлять сценарии оболочки как очень мощные файлы .BAT.

Одно из свойств оболочки позволяет, если вы совершили ошибку, сразу же увидеть, что именно набрано неверно, и попробовать заново.

Существуют различные варианты оболочки Unix, но все они заимствуют некоторые функции от оболочки Bourne shell (/bin/sh) – стандартной оболочки, разработанной в компании Bell Labs для ранних версий системы Unix.

Каждой системе Unix для правильной работы необходимо наличие оболочки Bourne shell. Система Linux использует улучшенную версию оболочки Bourne shell – bash, или «заново рожденную» оболочку. Оболочка bash является оболочкой по умолчанию в большинстве дистрибутивов Linux, а путь /bin/sh, как правило, – ссылка на эту оболочку. При запуске примеров следует применять оболочку bash.

После входа в систему откройте окно оболочки, которое часто называют *терминалом*. Проще всего это выполнить с помощью запуска терминального приложения из графического интерфейса пользователя Gnome или Unity в Ubuntu или сочетанием клавишей ***Ctrl+T***. При этом оболочка открывается в новом окне. После запуска оболочки в верхней части окна должно отобразиться приглашение, которое обычно заканчивается символом доллара ($). В Ubuntu это приглашение будет выглядеть примерно так: *name@host:path$*. Если вы хорошо знакомы с Windows, окно оболочки

будет выглядеть подобно окну командной строки DOS; приложение Terminal в OS X, по сути, такое же, как окно оболочки Linux.

Чтобы обозначить приглашение, все они начинаются с символа $. Многие команды начинаются символом #. Такие команды следует запускать с правами пользователя superuser (root). Как правило, с этими командами следует обращаться осторожно.

Рассмотрим основные команды системы и их параметры.

1.1. Команда cat

Команда cat системы Unix является одной из простейших для понимания. Она просто выводит содержимое одного или нескольких файлов. Общий синтаксис команды cat выглядит следующим образом:

$ **cat** *file1 file2 ...*

После запуска команда cat выполняет вывод содержимого файлов *file1*, *file2* и каких-либо еще (они обозначены символом ...), а затем завершает работу. Эта команда названа так, поскольку она выполняет конкатенацию (сцепление) содержимого файлов, если выводится более одного файла.

1.2. Команда ls

Команда ls выводит перечень содержимого какого-либо каталога. По умолчанию это текущий каталог. Используйте вариант ls -l, чтобы получить детализированный (длинный) список, или ls –F, чтобы отобразить информацию о типах файлов.

Ниже приведен пример длинного перечня, он содержит информацию о владельце файла (столбец 3), группе (столбец 4), размере файла (столбец 5), а также о дате и времени его изменения (между столбцом 5 и названием файла):

$ **ls -l**

```
total 3616
-rw-r--r-- 1 juser users 3804 Apr 30 2011 abusive.c
-rw-r--r-- 1 juser users 4165 May 26 2010 battery.zip
-rw-r--r-- 1 juser users 131219 Oct 26 2012 beav_1.40-13.tar.gz
-rw-r--r-- 1 juser users 6255 May 30 2010 country.c
drwxr-xr-x 2 juser users 4096 Jul 17 20:00 cs335
-rwxr-xr-x 1 juser users 7108 Feb 2 2011 dhry
-rw-r--r-- 1 juser users 11309 Oct 20 2010 dhry.c
-rw-r--r-- 1 juser users 56 Oct 6 2012 doit
drwxr-xr-x 6 juser users 4096 Feb 20 13:51 dw
drwxr-xr-x 3 juser users 4096 May 2 2011 hough-stuff
```

1.3. Команда cp

В своей простейшей форме команда cp копирует файлы. Например, чтобы скопировать *file1* в файл *file2*, введите следующее:

$ **cp** *file1 file2*

Чтобы скопировать несколько файлов в какой-либо каталог (папку) с названием *dir*, попробуйте такой вариант:

$ **cp** *file1* **...** *fileN dir*

1.4. Команда mv

Команда mv (от англ. move – «переместить») подобна команде cp. В своей простейшей форме она переименовывает файл. Например, чтобы переименовать файл *file1* в *file2*, введите следующее:

$ **mv** *file1 file2*

Можно также использовать команду mv, чтобы переместить несколько файлов в другой каталог:

$ **mv** *file1* **...** *fileN dir*

1.5. Команда touch

Команда touch создает файл. Если такой файл уже существует, команда touch не изменяет его, но обновляет информацию о времени изменения файла, выводимую с помощью команды ls -l. Например, чтобы создать пустой файл, введите следующее:

$ **touch** *file*

Теперь примените к этому файлу команду ls -l. Вы должны увидеть результат, подобный приведенному ниже. Символом ❶ отмечены дата и время запуска команды touch:

$ **ls -l** *file*

-rw-r--r-- 1 juser users 0 May 21 18:32❶ *file*

1.6. Команда rm

Чтобы удалить файл, воспользуйтесь командой rm (от англ. remove – «удалить»). После удаления файла он исчезает из системы и, как правило, не может быть восстановлен.

$ **rm** *file*

1.7. Команда cd

Текущий рабочий каталог – это каталог, в котором в данный момент находится процесс (например, оболочка). Команда cd изменяет текущий рабочий каталог оболочки:

$ **cd** *dir*

Если вы опустите параметр *dir*, оболочка вернет вас в *домашний каталог*, с которого вы начали работу при входе в систему.

1.8. Команда mkdir

Команда mkdir создает новый каталог с именем *dir*:
$ **mkdir** *dir*

1.9. Команда rmdir

Команда rmdir удаляет каталог с именем *dir*:
$ **rmdir** *dir*

Если каталог *dir* не пуст, эта команда не сработает. Чтобы удалить каталог со всем его содержимым, используйте команду rm -rf *dir*. Однако будьте осторожны! Это одна из немногих команд, которая может причинить существенный вред, особенно если вы работаете как superuser. Параметр –r задает *рекурсивное удаление*, которое последовательно удаляет все, что находится внутри каталога *dir*, а параметр -f делает эту операцию принудительной. Не используйте флаги -rf с такими джокерными символами, как звездочка (*). Перепроверяйте команды перед их запуском.

1.10. Джокерные символы

Оболочка способна сопоставлять простые шаблоны с именами файлов и каталогов. Этот процесс называется *универсализацией файловых имен*. Он напоминает применение джокерных символов в других системах. Простейшим из таких символов является звездочка (*), которая указывает оболочке на то, что вместо него можно подставить любое количество произвольных символов. Например, следующая команда выдаст список файлов в текущем каталоге:

$ **echo ***

Оболочка сопоставляет аргументы, которые содержат джокерные символы, с именами файлов, заменяет аргументы подходящими именами, а затем запускает исправленную командную строку. Такая подстановка называется *развертыванием*, поскольку оболочка подставляет все подходящие имена файлов. Вот несколько примеров того, как можно использовать символ * для развертывания имен файлов:

- at* – развертывается во все имена файлов, которые начинаются с символов at;
- *at – развертывается во все имена файлов, которые заканчиваются символами at;
- *at* – развертывается во все имена файлов, которые содержат символы at.

Если ни один из файлов не удовлетворяет шаблону, оболочка не выполняет развертывание и вместо него буквально использует указанные

символы, в том числе и *. Попробуйте, например, набрать такую команду, как echo *dfkdsafh.

ПРИМЕЧАНИЕ

Если вы привыкли к работе в системе MS-DOS, то вы могли бы инстинктивно набрать символы *.*, которые подходят для всех файлов. Расстаньтесь теперь с этой привычкой. В системе Linux и других версиях системы Unix вы должны использовать символ *. В оболочке Unix комбинация *.* соответствует только тем файлам и каталогам, названия которых содержат точку. Для названий файлов в системе Unix расширения не являются обязательными, и файлы часто обходятся без них.

Еще один джокерный символ, вопросительный знак (?), указывает оболочке на то, что необходимо подставить только один произвольный символ. Например, комбинации b?at будут соответствовать имена boat и brat.

Если вы желаете, чтобы оболочка не развертывала джокерный символ в команде, заключите его в одиночные кавычки (' '). Так, например, команда '*' выдаст символ звездочки. Вы увидите, что это удобно применять в некоторых командах, например grep и find (они рассматриваются в следующем разделе, подробности об использовании кавычек вы узнаете из раздела 11.2).

ПРИМЕЧАНИЕ

Важно помнить о том, что оболочка выполняет развертывание перед запуском команд и только в это время. Следовательно, если символ * приводит к отсутствию развертывания, оболочка ничего не будет с ним делать; и уже от команды зависит решение о дальнейших действиях.

В современных версиях оболочки существуют дополнительные возможности настройки шаблонов, однако сейчас вам необходимо знать лишь джокерные символы * и ?.

1.11. Команда grep

Команда grep выдает строки из файла или входного потока, которые соответствуют какому-либо выражению. Например, чтобы напечатать строки из файла /etc/passwd, которые содержат текст root, введите следующее:

$ **grep root /etc/passwd**

Команда grep чрезвычайно удобна, когда приходится работать одновременно с множеством файлов, поскольку она выдает название файла в дополнение к найденной строке. Например, если вы желаете отыскать все файлы в каталоге /etc, которые содержат слово root, можно применить данную команду так:

$ **grep root /etc/***

Двумя наиболее важными параметрами команды grep являются –i (для соответствий, нечувствительных к регистру символов) и –v (который инвертирует условие поиска, то есть выдает все строки, не отвечающие условию). Существует также более мощный вариант команды под названием egrep (это всего лишь синоним команды grep -E).

Команда grep понимает шаблоны, которые известны как *регулярные выражения*. Они укоренились в теории вычислительной техники и являются весьма обычными для утилит системы Unix. Регулярные выражения более мощные, чем шаблоны с джокерными символами, и имеют другой синтаксис. Следует помнить два важных момента, относящихся к регулярным выражениям:

- сочетание .* соответствует любому количеству символов (подобно джокерному символу *);
- символ . соответствует одному произвольному символу.

ПРИМЕЧАНИЕ

Страница руководства grep(1) содержит подробное описание регулярных выражений, но оно может оказаться трудным для восприятия. Чтобы узнать больше, можете прочитать книгу Mastering Regular Expressions («Осваиваем регулярные выражения»), 3-е издание (O'Reilly, 2006) или главу о регулярных выражениях в книге Programming Perl («Программирование на языке Perl»), 4-е издание (O'Reilly, 2012). Если вы любите математику и вам интересно, откуда возникли регулярные выражения, загляните в книгу Automata Theory, Languages and Computation («Теория автоматов, языки и вычисления»), 3-е издание (Prentice Hall, 2006).

1.12. Команда less

Команда less становится удобной тогда, когда файл довольно большой или когда выводимый результат длинен и простирается за пределы экрана.

Чтобы постранично просмотреть такой большой файл, как /usr/share/dict/words, воспользуйтесь командой less /usr/share/dict/words. После запуска команды less вы увидите содержимое файла, разбитое на части и умещающееся в пределах одного экрана. Нажмите клавишу Пробел, чтобы переместиться далее по содержимому, или клавишу B, чтобы вернуться назад на один экран. Чтобы выйти, нажмите клавишу Q.

ПРИМЕЧАНИЕ

Команда less является улучшенной версией устаревшей команды more. Большинство рабочих станций и серверов на основе системы Linux содержат команду less, однако она не является стандартной для многих встроенных и других систем на основе Unix. Если вы когда-либо столкнетесь с невозможностью использовать команду less, попробуйте применить команду more.

Можно также воспользоваться поиском текста внутри команды less. Например, чтобы отыскать слово *word*, наберите /word. Для поиска в обратном направлении применяйте вариант ?word. Когда результат будет найден, нажмите клавишу N для продолжения поиска.

Как вы узнаете из раздела 2.14, можно отправить стандартный вывод практически из любой команды непосредственно на стандартный вход другой команды. Это исключительно полезно, если команда выводит большое количество результатов, которые вам пришлось бы просеивать с

использованием какой-либо команды вроде less. Вот пример отправки результатов команды grep в команду less:

$ **grep ie /usr/share/dict/words | less**

Попробуйте самостоятельно поработать с этой командой. Возможно, вы станете часто применять команду less подобным образом.

1.13. Команда find и locate

Бывает досадно, если вы знаете, что какой-либо файл точно расположен где-то в данном дереве каталогов, но вы не помните точно, где именно. Запустите команду find, чтобы отыскать файл *file* в каталоге *dir*:

$ **find** *dir* **-name** *file* **-print**

Подобно большинству команд в этом разделе, команда find обладает некоторыми интересными особенностями. Однако не пробуйте применять параметры, описанные здесь как -exec, пока не выучите наизусть их форму и станете понимать, зачем нужны параметры -name и -print. Команда find допускает применение специальных шаблонных символов вроде *, но вы должны заключать такие символы в одиночные кавычки ('*'), чтобы оградить их от функции универсализации, которая действует в оболочке. Вспомните из подраздела 2.4.4 о том, что оболочка выполняет развертывание джокерных символов *перед* выполнением команд.

В большинстве систем есть также команда locate для поиска файлов. Вместо отыскивания файла в реальном времени эта команда осуществляет поиск в индексе файлов, который система периодически создает. Поиск с помощью команды locate происходит гораздо быстрее, чем с помощью find, но если искомый файл появился после создания индекса, команда locate не сможет его найти.

1.14. Команда head и tail

Чтобы быстро просмотреть фрагмент файла или потока данных, используйте команды head и tail. Например, команда head /etc/passwd отобразит первые десять строк файла с паролем, а команда tail /etc/passwd покажет заключительные десять строк.

Чтобы изменить количество отображаемых строк, применяйте параметр -*n*, в котором число *n* равно количеству строк, которые необходимо увидеть (например, head -5 /etc/passwd). Чтобы вывести строки, начиная со строки под номером *n*, используйте команду tail +*n*.

1.15. Команда chmod

Каждый файл системы Unix обладает набором *прав доступа*, которые определяют, можете ли вы читать, записывать или запускать данный файл. Команда ls -l отображает эти права доступа. Вот пример такой информации:

-rw-r--r--❶ 1 juser somegroup 7041 Mar 26 19:34 endnotes.html

Режим файла ❶ представляет права доступа и некоторые дополнительные сведения. Режим состоит из четырех частей, как показано на рис. 2.1. Первый символ в режиме обозначает тип файла. Дефис (-) на этом месте, как в примере, указывает на то, что файл является *обычным* и не содержит никаких особенностей. Безусловно, это самый распространенный тип файлов. Каталоги также весьма обычны и обозначаются символом d в позиции, указывающей тип файла (в разделе 3.1).

Оставшаяся часть режима файла содержит сведения о правах доступа, которые разделены на три группы: *права пользователя*, *права группы* и *другие права* – в указанном порядке. Символы rw- в приведенном примере относятся к правам доступа пользователя, за ними следуют символы r--, определяющие права доступа для группы, и, наконец, символы r – определяют другие права доступа.

Каждый из наборов прав доступа может содержать четыре основных обозначения (табл. 2.5).

Рис. 2.1. Составляющие режима файла

Таблица 2.5. Обозначение прав доступа

Обозначение	**Описание**
r	Файл доступен для чтения
w	Файл доступен для записи
x	Файл является исполнимым (можно запустить его как программу)
–	Ничего не обозначает

Права доступа пользователя (первый набор символов) относятся к пользователю, который является владельцем файла. В приведенном выше примере это juser. Второй набор символов относится к группе (в данном случае somegroup). Любой пользователь из этой группы обладает указанными правами доступа. Воспользуйтесь командой groups, чтобы узнать, в какую группу вы входите. Дополнительную информацию можно получить в подразделе 7.3.5.

В третьем наборе (другие права доступа) указано, кто еще будет обладать в системе правами доступа. Их часто называют правами доступа *для всех*.

ПРИМЕЧАНИЕ

Каждая позиция, содержащая обозначение права доступа на чтение, записи или исполнение, иногда называется битом прав доступа. По этой причине вы можете слышать, как пользователи упоминают про «биты доступа для чтения».

У некоторые исполняемых файлов в правах доступа пользователя вместо символа x указан символ s. Это говорит о том, что исполняемый файл является файлом setuid – при его выполнении он запускается так, словно владельцем файла является пользователь с указанным идентификатором, а не

вы. Многие команды используют бит setuid, чтобы получить корневые права доступа, необходимые для изменения системных файлов. В качестве примера можно привести команду passwd, которой необходимо изменять файл /etc/passwd.

2.17.1. Изменение прав доступа

Чтобы изменить права доступа, используйте команду chmod. Сначала укажите набор прав, который вы желаете изменить, а затем укажите бит, подлежащий изменению. Например, чтобы добавить права доступа на чтение файла для группы (g) и всех пользователей (o, по первой букве слова other – «остальные»), нужно запустить следующие две команды:

$ **chmod g+r** *file*
$ **chmod o+r** *file*

Можно также объединить их таким образом:

$ **chmod go+r** *file*

Чтобы удалить эти права доступа, используйте go-r вместо go+r.

ПРИМЕЧАНИЕ

Довольно очевидно, что не следует предоставлять всем пользователям права доступа на запись файла, поскольку каждый получит возможность изменить системные файлы. Но сможет ли при этом кто-либо, подключившись через Интернет, изменить ваши файлы? Вероятно, не сможет, если только в вашей системе нет уязвимости в сетевой защите. Если же она уязвима, то права доступа к файлам ничем вам не помогут.

Иногда пользователи меняют права доступа, указывая числа, например, так:

$ **chmod 644** *file*

Такой способ называется *абсолютным* изменением, поскольку при нем сразу же устанавливаются все биты прав доступа. Чтобы разобраться, как это устроено, вам необходимо понять, как представлять биты прав доступа в восьмеричной форме (каждое значение является числом в системе счисления по основанию 8 и соответствует набору прав доступа). Дополнительную информацию об этом можно прочитать в руководстве на странице chmod(1).

Вам не обязательно знать, как составить абсолютные значения режимов, запомните те из них, которыми вы будете пользоваться чаще всего. В табл. 2.6 перечислены наиболее распространенные варианты.

Таблица 2.6. Абсолютные режимы прав доступа

Режим	**Значение**	**Применение**
644	пользователь: чтение/запись; группа, другие: чтение	Файлы
600	пользователь: чтение/запись;	Файлы

	группа, другие: нет	
755	пользователь: чтение/запись/исполнение; группа, другие: чтение/исполнение	Каталоги, команды
700	пользователь: чтение/запись/исполнение; группа, другие: нет	Каталоги, команды
711	пользователь: чтение/запись/исполнение; группа, другие: исполнение	Каталоги

Каталогам также можно назначить права доступа. Вы можете вывести список содержимого каталога, если он доступен для чтения, но доступ к файлу в этом каталоге можно получить лишь в том случае, если каталогу назначено право доступа на исполнение. Часто при указании прав доступа к каталогам пользователи совершают ошибку, ненароком удаляя разрешение на исполнение при применении абсолютных значений режимов.

Наконец, вы можете указать набор прав доступа по умолчанию с помощью команды оболочки umask, которая применяет заранее определенный набор прав доступа к любому создаваемому новому файлу. В общем, применяйте команду umask 022, если вы желаете, чтобы каждый мог видеть все создаваемые вами файлы и каталоги, или команду umask 077 в противном случае. Вам необходимо поместить команду umask с указанием желаемого режима в один из файлов запуска системы, чтобы новые права доступа по умолчанию применялись в следующих сеансах работы (см. главу 13).

1.16. Команда su

2. Введение в сценарий оболочки
2.1. Основные сценарии оболочки
2.2. Кавычки и литералы
2.3. Специальные переменные
2.4. Коды выхода
2.5. Условные операторы
2.6. Циклы
3. Задания
4. Контрольные вопросы
5. Список используемых источников

2.5. Вспомогательные команды

В приведенных ниже разделах рассмотрены наиболее важные вспомогательные команды системы Unix.

2.5.1. Команда grep

2.5.2. Команда less

2.5.3. Команда pwd

Команда pwd (отобразить рабочий каталог) выводит название текущего рабочего каталога. Возникает вопрос: зачем нужна эта команда, если в большинстве версий системы Linux учетные записи настроены так, чтобы в строке приглашения отображался рабочий каталог? На это есть две причины.

Во-первых, не все приглашения включают текущий рабочий каталог, и вам может даже потребоваться избавиться от его отображения, поскольку путь занимает слишком много места. Если это так, вам понадобится команда pwd.

Во-вторых, символические ссылки, о которых вы узнаете из подраздела 2.17.2, иногда могут делать неясным подлинный полный путь к текущему рабочему каталогу. Чтобы избежать путаницы, вам потребуется применить команду pwd -P.

2.5.4. Команда diff

Чтобы увидеть различия между двумя текстовыми файлами, воспользуйтесь командой diff:

$ **diff** *file1 file2*

Формат вывода можно контролировать с помощью нескольких параметров, однако для человеческого восприятия наиболее понятен формат, принятый по умолчанию. Тем не менее большинство программистов предпочитает использовать формат diff -u, когда приходится отправлять выходные данные еще куда-либо, поскольку автоматизированные устройства могут лучше справиться с таким форматом.

2.5.5. Команда file

Если вы видите файл и не уверены в том, какой у него формат, попробуйте использовать команду file, чтобы система попыталась выяснить это за вас:

$ **file** *file*

2.5.6. Команды find и locate

2.5.7. Команды head и tail

2.5.8. Команда sort

Команда sort быстро выстраивает строки текста в алфавитно-числовом порядке. Если строки файла начинаются с чисел и вам необходимо выстроить их в порядке следования чисел, применяйте параметр -n. Параметр -r изменяет порядок следования на обратный.

2.6. Изменение вашего пароля и оболочки

Для изменения своего пароля воспользуйтесь командой passwd. Система попросит вас указать старый пароль, а затем дважды пригласит ввести новый. Выбирайте пароль, который не содержит реальных слов какого-либо языка, а также не старайтесь комбинировать слова.

Один из простейших способов придумать хороший пароль состоит в следующем. Выберите какую-либо фразу, составьте из нее акроним (оставив только первые буквы входящих в нее слов), а затем измените этот акроним с помощью цифр и знаков пунктуации. После этого вам понадобится только запомнить исходную фразу.

Вы можете сменить оболочку (на альтернативную, например ksh или tcsh) с помощью команды chsh, но помните о том, что в данной книге предполагается, что вы работаете в оболочке bash.

2.7. Файлы с точкой

Перейдите в домашний каталог, посмотрите его содержимое с помощью команды ls, а затем запустите команду ls -a. Видите различия в результатах вывода? После запуска команды ls без параметра -a вы не увидите конфигурационные файлы, которые называются *файлами с точкой*. Имена таких файлов и каталогов начинаются с точки (.). Обычными файлами с точкой являются файлы .bashrc и .login. Существуют также и каталоги с точкой, например .ssh.

У файлов или каталогов с точкой нет ничего особенного. Некоторые команды по умолчанию не показывают их, чтобы при отображении содержимого домашнего каталога вы не увидели полнейшую неразбериху. Так, например, команда ls не показывает файлы с точкой, если не указан параметр -a. Кроме того, паттерны оболочки не рассматривают файлы с точкой, если вы намеренно не укажете это с помощью шаблона .*.

ПРИМЕЧАНИЕ

У вас могут возникнуть трудности с шаблонами, поскольку комбинации .* соответствуют варианты . и .. (то есть текущий и родительский каталоги). Следует использовать шаблоны вроде .[^.]* или .??*, чтобы получить список всех файлов с точкой, кроме текущего и родительского каталогов.

2.8. Переменные окружения и оболочки

Оболочка может хранить временные переменные, которые называются *переменными оболочки* и содержат значения текстовых строк. Переменные оболочки весьма полезны для отслеживания значений в сценариях. Некоторые переменные оболочки контролируют режим работы оболочки (например, оболочка bash считывает значение переменной PS1 перед отображением приглашения).

Чтобы присвоить значение переменной оболочки, используйте знак равенства (=). Вот простой пример:

$ **STUFF=blah**

В этом примере переменной с именем STUFF присваивается значение blah. Чтобы обратиться к этой переменной, применяйте синтаксис $STUFF (попробуйте, например, запустить команду echo $STUFF). Множество вариантов использования переменных оболочки приведено в главе 11.

Переменная окружения подобна переменной оболочки, но она не является привязанной к оболочке. Все процессы в системах Unix пользуются хранилищем переменных окружения. Основное отличие переменных окружения от переменных оболочки заключается в том, что операционная система передает все переменные окружения вашей оболочки командам, запускаемым оболочкой, в то время как переменные оболочки не могут быть доступны командам, которые вы запускаете.

Назначение переменной окружения производится с помощью команды export. Если, например, вы желаете сделать переменную оболочки $STUFF переменной окружения, используйте следующий синтаксис:

$ **STUFF=blah**

$ **export STUFF**

Переменные окружения практичны, поскольку многие команды считывают из них значения своих настроек и параметров. Например, вы можете поместить ваши излюбленные параметры для команды less в переменную окружения LESS, и тогда команда less будет использовать их при запуске. Многие страницы руководства содержат раздел с названием Environment («Окружение»), в котором описаны такие переменные.

2.9. Командный путь

PATH является специальной переменной окружения, которая содержит *командный путь*, или просто *путь*. Командный путь – это перечень системных каталогов, которые просматривает оболочка, пытаясь найти какую-либо команду. Например, когда вы запускаете команду ls, оболочка просматривает каталоги, перечисленные в переменной PATH, в поисках команды ls. Если команда встречается сразу в нескольких каталогах, оболочка запустит первый найденный экземпляр.

Если вы запустите команду echo $PATH, вы увидите, что компоненты пути отделены с помощью двоеточия (:). Например, так:

$ **echo $PATH**

/usr/local/bin:/usr/bin:/bin

Чтобы указать оболочке дополнительные места для поиска команд, измените переменную окружения PATH. Например, с помощью приведенной ниже команды можно добавить в начало пути каталог *dir*, чтобы оболочка начала просмотр каталогов с него, до других каталогов из переменной PATH.

$ **PATH=***dir***:$**PATH

Можно также добавить название каталога в конец переменной PATH, тогда оболочка обратится к нему в последнюю очередь:

$ **PATH=$PATH:***dir*

ПРИМЕЧАНИЕ

Будьте осторожны при изменении пути, поскольку вы можете случайно полностью стереть его, если неправильно наберете $PATH. Если это случилось, не следует паниковать. Ущерб не является необратимым, вы можете просто запустить новую оболочку. Чтобы сильнее нарушить работу, необходимо сделать ошибку при редактировании некоторого файла конфигурации, но даже и в этом случае ее нетрудно исправить. Один из простейших способов вернуться в нормальное состояние – закрыть текущее окно терминала и открыть другое.

2.10. Специальные символы

Вам следует знать названия некоторых специальных символов системы Linux. Если вам нравятся вещи подобного рода, загляните на страницу *Jargon File* (http://www.catb.org/jargon/html/) или посмотрите ее печатную версию в

книге *The New Hacker's Dictionary* («Новый словарь хакера») (MIT Press, 1996).

В табл. 2.1 описан набор специальных символов, многие из них вы уже встречали в этой главе. Некоторые утилиты, например язык программирования Perl, используют почти все эти символы! Помните о том, что здесь приведены американские варианты названий символов.

Таблица 2.1. Специальные символы

Символ	Название	Применение
*	Звездочка	Регулярные выражения, джокерные символы
.	Точка	Текущий каталог, разделитель имени файла и хоста
!	Восклицательный знак	Отрицание, история команд
|	Вертикальная черта	Командный конвейер
/	Прямой слеш	Разделитель каталогов, команда поиска
\	Обратный слеш	Литералы, макросы (но не каталоги)
$	Доллар	Обозначение переменной, конец строки
'	Одиночная кавычка	Буквенные строки
`	«Обратная галочка»	Замена команды
"	Двойная кавычка	Частично буквенные строки
^	Знак вставки	Отрицание, начало строки
~	Тильда	Отрицание, ярлык каталога
#	Диез, знак фунта	Комментарии, препроцессор, подстановки
[ ]	Квадратные скобки	Массивы
{ }	Фигурные скобки	Блоки инструкций, массивы
_	Подчеркивание	Просто замена символа пробела

2.11. Редактирование командной строки

Во время работы с оболочкой обратите внимание, что вы можете редактировать командную строку, используя клавиши ← и →, а также перемещаться к предыдущим командам с помощью клавиш ↑ и ↓. Такой способ работы является стандартным в большинстве систем Linux.

Кроме того, вместо них можно применять управляющие сочетания клавиш. Если вы запомните сочетания, перечисленные в табл. 2.2, то обнаружите, что у вас появились дополнительные возможности ввода текста во многих командах Unix по сравнению с использованием стандартных клавиш.

Таблица 2.2. Сочетания клавиш для командной строки

Сочетание клавиш	Действие
Ctrl+B	Перемещение курсора влево
Ctrl+F	Перемещение курсора вправо
Ctrl+P	Просмотр предыдущей команды (или перемещение курсора вверх)
Ctrl+N	Просмотр следующей команды (или

перемещение курсора вниз)

2.13. Получение интерактивной справки

Операционные системы Linux снабжены документацией. *Страницы руководства* расскажут вам обо всем необходимом для основных команд. Например, чтобы увидеть страницу руководства по команде ls, введите следующий запрос:

$ **man ls**

Большинство страниц руководства содержат главным образом справочную информацию, возможно, с некоторыми примерами и перекрестными ссылками. Не ожидайте увидеть учебное пособие и не рассчитывайте на воодушевляющий литературный стиль.

Когда у команды много параметров, они перечисляются на странице руководства в каком-либо систематизированном виде (например, в алфавитном порядке). Степень их важности при этом определить невозможно. Если вы достаточно терпеливы, то в большинстве случаев сможете найти все необходимые сведения в руководстве.

Чтобы выполнить поиск на странице руководства по ключевому слову, используйте параметр -k:

$ **man -k** *keyword*

Это полезно, если вы не вполне уверены в названии необходимой команды. Например, если вы ищете команду для сортировки чего-либо, введите следующее:

$ **man -k sort**

--snip--

comm (1) - сравнивает два отсортированных файла строка за строкой

qsort (3) - сортирует массив

sort (1) - сортирует строки текстового файла

sortm (1) - сортирует сообщения

tsort (1) - выполняет топологическую сортировку

--snip--

Результат включает название страницы руководства, раздел руководства (см. ниже), а также краткое описание того, что содержит данная страница руководства.

ПРИМЕЧАНИЕ

Если у вас есть какие-либо вопросы насчет команд, рассмотренных в предыдущих разделах, можете попробовать найти ответы с помощью команды man.

На страницы руководства ссылаются с помощью пронумерованных разделов. Когда кто-либо приводит ссылку на страницу руководства, то в скобках после названия указывается номер раздела, например, так: ping(8). В табл. 2.3 перечислены все разделы с их нумерацией.

Таблица 2.3. Разделы интерактивного руководства

Раздел	**Описание**
1	Команды пользователя
2	Системные вызовы

3	Документация к высокоуровневой программной библиотеке Unix
4	Интерфейс устройств и информация о драйверах
5	Описание файлов (файлы конфигурации системы)
6	Игры
7	Форматы файлов, условные обозначения и кодировки (ASCII, суффиксы и т. д.)
8	Системные команды и серверы

Разделы 1, 5, 7 и 8 служат хорошим дополнением к данной книге. Раздел 4 может пригодиться в редких случаях, а раздел 6 был бы замечателен, если бы его сделали немного больше. Вероятно, вы не сможете воспользоваться разделом 3, если вы не программист, однако вам удастся понять раздел 2, когда вы узнаете из этой книги немного больше о системных вызовах.

Вы можете выбрать страницу руководства по разделу. Это важно, поскольку система отображает первую из страниц, на которой ей удается обнаружить нужное понятие. Например, чтобы прочитать описание файла /etc/passwd (а не команды passwd), можно добавить номер раздела перед названием страницы:

$ **man 5 passwd**

Страницы руководства содержат основные сведения, но есть и другие способы получить интерактивную справку. Если вам необходимо узнать что-либо о параметре команды, введите название команды, а затем добавьте параметр --help или -h (варианты различны для разных команд). В результате вы можете получить множество ответов (как, например, в случае с командой ls --help) или, возможно, сразу то, что нужно.

Когда-то участники проекта GNU Project решили, что им не очень нравятся страницы руководства, поэтому появился другой формат справки – info (или texinfo). Зачастую объем этой документации больше, чем у обычных страниц руководства, и иногда она более сложная. Чтобы получить доступ к этой информации, введите команду info и укажите название команды:

$ **info** *command*

Некоторые версии системы сваливают всю доступную документацию в каталог /usr/share/doc, не заботясь о предоставлении какой-либо справочной системы вроде man или info. Загляните в этот каталог, если вам необходима документация, и, конечно же, поищите в Интернете.

2.14. Ввод и вывод с помощью оболочки

Теперь, когда вы знакомы с основными командами системы Unix, файлами и каталогами, вы готовы к изучению перенаправления стандартных ввода и вывода. Начнем со стандартного вывода.

Чтобы направить результат команды *command* в файл, а не в терминал, используйте символ перенаправления >:

$ *command* **>** *file*

Оболочка создаст файл *file*, если его еще нет. Если такой файл существует, то оболочка сначала *сотрет* его *(затрет данные)*. Некоторые оболочки снабжены параметрами, предотвращающими затирание данных. В оболочке bash, например, наберите для этого set –C.

Вы можете добавить выводимые данные к файлу вместо его перезаписи с помощью такого синтаксиса перенаправления >>:

$ *command* **>>** *file*

Это удобный способ собрать все выходные данные в одном месте, когда выполняется последовательность связанных команд.

Чтобы отправить стандартный вывод какой-либо команды на стандартный вход другой команды, используйте символ вертикальной черты (|). Чтобы понять, как он работает, попробуйте набрать следующие команды:

$ **head /proc/cpuinfo**
$ **head /proc/cpuinfo | tr a-z A-Z**

Можно пропустить выходные данные через любое количество команд. Для этого добавляйте вертикальную черту перед каждой дополнительной командой.

2.14.1. Стандартная ошибка

Иногда при перенаправлении стандартного вывода вы можете обнаружить, что команда выводит в терминал что-то еще. Это называется *стандартной ошибкой* (stderr). Она является дополнительным выходным потоком для диагностики и отладки.

Например, такая команда вызовет ошибку:

$ **ls /fffffffff > f**

После ее выполнения файл f должен быть пустым, однако вы увидите в терминале следующее сообщение стандартной ошибки:

ls: cannot access /fffffffff: No such file or directory

Если желаете, можете перенаправить стандартную ошибку. Чтобы, например, отправить стандартный вывод в файл f, а стандартную ошибку в файле, используйте синтаксис 2> следующим образом:

$ **ls /fffffffff > f 2> e**

Число 2 определяет *идентификатор потока*, который изменяет оболочка. Значение 1 соответствует стандартному выводу (по умолчанию), а значение 2 – стандартной ошибке.

Вы можете также направить стандартную ошибку туда же, куда осуществляется стандартный вывод, с помощью нотации >&.

Например, чтобы отправить стандартный вывод и стандартную ошибку в файл f, попробуйте такую команду:

$ **ls /ffffffff > f 2>&151**

2.14.2. Перенаправление стандартного ввода

Чтобы отправить файл на стандартный ввод команды, используйте оператор <:

$ **head < /proc/cpuinfo**

Вам может встретиться команда, которой потребуется указанный выше тип перенаправления, но поскольку большинство команд системы Unix принимает имена файлов в качестве аргументов, такое происходит нечасто. Эту команду можно было бы переписать в виде head /proc/cpuinfo.

2.15. Объяснение сообщений об ошибках

Когда у вас возникают проблемы при работе в системе, похожей на Unix (например, в Linux), вы *должны* читать сообщения об ошибках. В отличие от сообщений в других операционных системах, в системе Unix ошибки, как правило, точно указывают вам на то, что именно вышло из строя.

2.15.1. Структура сообщений об ошибке в Unix

Большинство команд системы Unix выдает одинаковые основные сообщения об ошибках, однако окончательный вид может немного различаться для разных команд. Вот пример сообщения, которое в том или ином виде обязательно вам встретится:

$ **ls /dsafsda**
ls: cannot access /dsafsda: No such file or directory

Это сообщение состоит из трех частей.

- Название команды: ls. Некоторые команды опускают такую идентифицирующую информацию, и это может раздражать при написании сценариев оболочки, хотя, по сути, это не так уж и важно.
- Имя файла, /dsafsda, которое является более конкретной информацией. Указанный путь содержит ошибку.
- Сообщение об ошибке No such file or directory указывает на ошибку в имени файла.

Если собрать эти части воедино, получится нечто вроде «команда ls пыталась открыть файл /dsafsda, но не смогла, поскольку такого файла нет».

Это может казаться очевидным, однако подобные сообщения сбивают с толку, если вы запустите сценарий оболочки, который содержит ошибку в другой команде.

При устранении ошибок всегда начинайте разбираться с первой ошибки. Некоторые команды сообщают о том, что они ничего не смогут сделать до выяснения причин других ошибок. Представьте, например, что вы запускаете выдуманную команду под названием scumd, которая выдает такое сообщение об ошибке:

scumd: cannot access /etc/scumd/config: No such file or directory

За ним следует огромный перечень других сообщений об ошибках, который выглядит катастрофически. Не отвлекайтесь на остальные ошибки. Скорее всего, вам всего лишь надо создать файл /etc/scumd/config.

ПРИМЕЧАНИЕ

Не смешивайте сообщения об ошибках с предупреждениями. Предупреждения часто выглядят как ошибки, но они содержат слово warning. Они говорят о наличии какой-либо неисправности, однако команда будет пытаться продолжить работу. Чтобы устранить проблему, указанную в предупреждении, вам потребуется отыскать ошибочный процесс и завершить его, прежде чем делать что-либо еще (о списке процессов и об их завершении вы узнаете из раздела 2.16).

2.15.2. Общие ошибки

Многие ошибки, которые вы встретите при выполнении команд системы Unix, являются результатом неправильных действий с файлами или процессами. Приведем «хит-парад» сообщений об ошибках.

No such file or directory

Вероятно, вы пытались получить доступ к несуществующему файлу. Поскольку ввод/вывод в системе Unix не делает различий между файлами и каталогами, это сообщение об ошибке появляется везде. Вы получите его, если попытаетесь выполнить чтение несуществующего файла, или решите перейти в отсутствующий каталог, или попробуете записать файл в несуществующий каталог и т. д.

File exists

В данном случае вы, возможно, пытались создать файл, который уже существует. Это часто бывает, когда вы создаете каталог, имя которого уже занято каким-либо файлом.

Not a directory, Is a directory

Эти сообщения возникают, когда вы пытаетесь использовать файл в качестве каталога или каталог в качестве файла. Например, так:

$ **touch a**
$ **touch a/b**

touch: a/b: Not a directory

Обратите внимание на то, что сообщение об ошибке относится только к части a пути a/b. Когда вы столкнетесь с такой проблемой, вам потребуется время, чтобы отыскать компонент пути, с которым обращаются как с каталогом.

No space left on device

На вашем жестком диске закончилось свободное пространство.

Permission denied

Эта ошибка возникает, когда вы пытаетесь выполнить чтение или запись, указав файл или каталог, к которым вам не разрешен доступ (вы обладаете недостаточными правами). Эта ошибка говорит также о том, что вы пытаетесь запустить файл, для которого не установлен бит выполнения (даже если вы можете читать этот файл). Из раздела 2.17 вы больше узнаете о правах доступа.

Operation not permitted

Обычно такая ошибка возникает, когда вы пытаетесь завершить процесс, владельцем которого не являетесь.

Segmentation fault, Bus error

Суть *ошибки сегментации* состоит в том, что разработчик программы, которую вы только что запустили, где-то ошибся. Программа пыталась получить доступ к области памяти, к которой ей не разрешено обращаться, в результате операционная система завершила работу программы. Подобно ей, *ошибка шины* означает, что программа пыталась получить доступ к памяти недолжным образом. Если вы получаете одну из этих ошибок, то, вероятно, вы передали на ввод программы какие-либо неожиданные для нее данные.

для которого не установлен бит выполнения (даже если вы можете читать этот файл). Из раздела 2.17 вы больше узнаете о правах доступа.

Operation not permitted

Обычно такая ошибка возникает, когда вы пытаетесь завершить процесс, владельцем которого не являетесь.

Segmentation fault, Bus error

Суть *ошибки сегментации* состоит в том, что разработчик программы, которую вы только что запустили, где-то ошибся. Программа пыталась получить доступ к области памяти, к которой ей не разрешено обращаться, в результате операционная система завершила работу программы. Подобно ей, *ошибка шины* означает, что программа пыталась получить доступ к памяти недолжным образом. Если вы получаете одну из этих ошибок, то, вероятно, вы передали на ввод программы какие-либо неожиданные для нее данные.

2.16. Получение списка процессов и управление ими

Процесс – это работающая программа. Каждому процессу в системе присвоен числовой *идентификатор процесса* (PID). Чтобы быстро получить перечень работающих процессов, запустите команду ps. Вы получите результат вроде этого:

```
$ ps
PID TTY STAT TIME COMMAND
520 p0 S 0:00 -bash
545 ? S 3:59 /usr/X11R6/bin/ctwm -W
548 ? S 0:10 xclock -geometry -0-0
2159 pd SW 0:00 /usr/bin/vi lib/addresses
31956 p3 R 0:00 ps
```

Эти поля означают следующее.

- PID – идентификатор процесса.
- TTY – оконечное устройство, в котором запущен процесс (об этом подробнее чуть позже).
- STAT – статус процесса, а именно: что выполняет данный процесс и где расположена отведенная для него память. Например, символ S обозначает ждущий процесс, а символ R – работающий (описание всех символов можно найти на странице ps(1) в руководстве).
- TIME – количество времени центрального процессора в минутах и секундах, которое использовал данный процесс к настоящему моменту. Другими словами, это общее количество времени, потраченное процессом на выполнение инструкций в процессоре.
- COMMAND – поле может показаться очевидным, однако имейте в виду, что процесс может изменить исходное значение этого поля.

2.16.1. Параметры команды ps

Команда ps обладает множеством параметров. Чтобы запутать дело еще больше, можно указывать параметры в трех разных стилях: Unix, BSD и GNU. Многие пользователи считают, что стиль BSD наиболее удобен (вероятно, в силу большей краткости набора), поэтому в данной книге мы будем применять именно его. В табл. 2.4 приведено несколько наиболее полезных сочетаний параметров.

Таблица 2.4. Параметры команды ps

Команда с параметром	Описание
ps x	Показать все процессы, запущенные вами
ps ax	Показать все процессы системы, а не только те, владельцем которых являетесь вы
ps u	Включить детализированную информацию о процессах
ps w	Показать полные названия команд, а не только те,

что помещаются в одной строке

Как и в других командах, эти параметры можно комбинировать, например, так: ps aux или ps auxw. Чтобы проверить какой-либо конкретный процесс, добавьте значение PID к списку аргументов команды ps. Например, чтобы просмотреть информацию о текущем процессе оболочки, можно использовать команду ps u $$, поскольку параметр $$ является переменной оболочки, которая содержит значение PID текущего процесса. В главе 8 вы найдете информацию о командах администрирования top и lsof. Они могут пригодиться при локализации процессов, причем не обязательно тогда, когда необходимо заниматься обслуживанием системы.

2.16.2. Завершение процессов

Чтобы завершить процесс, отправьте ему *сигнал* с помощью команды kill. Сигнал – это сообщение процессу от ядра. Когда вы запускаете команду kill, вы просите ядро отправить сигнал другому процессу. В большинстве случаев для этого необходимо набрать следующее:

$ **kill** *pid*

Существует много типов сигналов. По умолчанию используется сигнал TERM («прервать»). Вы можете отправлять другие сигналы, добавляя параметр в команду kill. Например, чтобы приостановить процесс, не завершая его, применяйте сигнал STOP:

$ **kill -STOP** *pid*

Остановленный процесс остается в памяти и ожидает повторного вызова. Используйте сигнал CONT, чтобы продолжить выполнение этого процесса:

$ **kill -CONT** *pid*

ПРИМЕЧАНИЕ

Применение сочетания клавиш Ctrl+C для прерывания процесса, работающего в терминале, равносильно вызову команды kill для завершения процесса по сигналу INT («прервать»).

Самый грубый способ завершения процесса – с помощью сигнала KILL. Другие сигналы дают процессу шанс прибрать за собой, а сигнал KILL – нет. Операционная система прерывает процесс и принудительно выгружает его из памяти. Используйте это в качестве последнего средства. Не следует завершать процессы без разбора, особенно если вы не знаете, что они делают.

Вы можете увидеть, что некоторые пользователи вводят числа вместо названий параметров команды kill. Например, kill -9 вместо kill -KILL. Это объясняется тем, что ядро использует числа для обозначения различных сигналов. Можете применять команду kill подобным образом, если вам известно число, которое соответствует необходимому сигналу.

2.16.3. Управление заданиями

Оболочка поддерживает также *управление заданиями* – один из способов отправки командам сигналов TSTP (подобен сигналу STOP) и CONT с помощью различных сочетаний клавиш и команд. Например, вы можете отправить сигнал TSTP с помощью сочетания клавиш Ctrl+Z, а затем возобновить процесс командой fg (вывести из фона) или bg (перевести в фон; см. следующий раздел). Несмотря на практичность управления заданиями и его привычное использование многими опытными пользователями, оно не является необходимым и может запутать начинающих. Очень часто пользователи нажимают сочетание Ctrl+Z вместо Ctrl+C, забывают о запущенных процессах и в итоге получают множество «подвешенных» процессов.

СОВЕТ

Чтобы узнать, не висят ли в текущем терминале приостановленные процессы, выполните команду jobs.

Если вы намерены использовать несколько оболочек, запустите каждую из них в отдельном окне терминала, переведите в фон неинтерактивные процессы (см. следующий раздел) или научитесь использовать команду screen.

2.16.4. Фоновые процессы

Обычно при запуске из оболочки команды в системе Unix вы не увидите строки приглашения, пока команда не завершит работу. Тем не менее можно отделить процесс от оболочки и поместить его в «фон» с помощью символа амперсанда (&); после этого строка приглашения вернется. Если вам необходимо распаковать большой архив с помощью команды gunzip (о ней вы узнаете из раздела 2.18), а тем временем вы намерены заняться чем-либо другим, запустите команду такого вида:

$ **gunzip** *file***.gz &**

Оболочка должна в ответ выдать номер PID нового фонового процесса, а строка приглашения появится немедленно, чтобы вы смогли работать далее. Процесс продолжит свое выполнение и после того, как вы выйдете из системы. Это чрезвычайно удобно, если приходится запускать программу, которая производит довольно много вычислений. В зависимости от настроек системы оболочка может уведомить вас о завершении процесса.

Обратной стороной фоновых процессов является то, что они могут ожидать начала работы со стандартным вводом (и, хуже того, выполнять чтение прямо из терминала). Если фоновая программа пытается считывать что-либо из стандартного ввода, она может зависнуть (попробуйте команду fg, чтобы вызвать ее из фона) или прекратить работу. К тому же, если программа выполняет запись в стандартный вывод или стандартную ошибку, все это может отобразиться в терминале вне всякой связи с какими-либо

выполняемыми командами: вы можете увидеть неожиданный результат, пока работаете над чем-то другим.

Лучший способ добиться того, чтобы фоновый процесс не беспокоил вас, – перенаправить его вывод (и, возможно, ввод), как описано в разделе 2.14.

Узнайте о том, как обновить окно терминала. Оболочка bash и большинство полноэкранных интерактивных программ поддерживают сочетание клавиш Ctrl+L для повторной отрисовки всего экрана. Если программа производит считывание из стандартного ввода, сочетание клавиш Ctrl+R обычно обновляет текущую строку, однако нажатие неверного сочетания в неподходящий момент может привести вас к более печальной ситуации, чем была раньше. Нажав, например, Ctrl+R в строке приглашения оболочки bash, вы окажетесь в режиме реверсивного поиска isearch mode (для выхода из него нажмите клавишу Esc).

2.17. Режимы файлов и права доступа

2.17.2. Символические ссылки

Символическая ссылка – это файл, который указывает на другой файл или каталог, создавая, по сути, псевдоним (подобно ярлыку в Windows). Символические ссылки обеспечивают быстрый доступ к малопонятным путям каталогов.

В длинном списке каталогов символические ссылки будут выглядеть примерно так (обратите внимание на символ l, указанный в режиме файла в качестве типа файла):

```
lrwxrwxrwx 1 ruser users 11 Feb 27 13:52 somedir -> /home/origdir
```

Если вы попытаетесь получить доступ к каталогу somedir в данном каталоге, система выдаст вам вместо этого каталог /home/origdir. Символические ссылки являются лишь именами, указывающими на другие имена. Их названия, а также пути, на которые они указывают, не обязаны что-либо значить. Так, например, каталог /home/origdir может и вовсе отсутствовать.

В действительности, если каталог /home/origdir не существует, любая команда, которая обратится к каталогу somedir, сообщит о том, что его нет (кроме команды ls somedir, которая проинформирует вас лишь о том, что каталог somedir является каталогом somedir). Это может сбивать с толку, поскольку вы прямо перед собой видите нечто с именем somedir.

Это не единственный случай, когда символические ссылки могут вводить в заблуждение. Еще одна проблема состоит в том, что вы не можете установить характеристики объекта, на который указывает ссылка, просто посмотрев на название ссылки; вы должны перейти по ней, чтобы понять, ведет ли она к файлу или в каталог. В системе могут быть также ссылки, указывающие на другие ссылки – *цепные символические ссылки*.

2.17.3. Создание символических ссылок

Чтобы создать символическую ссылку от цели *target* к имени ссылки *linkname*, воспользуйтесь командой ln -s:

$ **ln -s** *target linkname*

Аргумент *linkname* является именем символической ссылки, аргумент *target* задает путь к файлу или каталогу, к которому ведет ссылка, а флаг -s определяет символическую ссылку (см. следующее за этим предупреждение).

При создании символической ссылки перепроверьте команду перед ее запуском. Например, если вы поменяете аргументы местами (ln –s *linkname target*), то возникнет забавная ситуация, если каталог *linkname* уже существует. Когда такое происходит (а это случается часто), команда ln создает ссылку с именем *target* внутри каталога *linkname*, и эта ссылка указывает на себя, если только *linkname* не является полным путем. Когда вы создаете символическую ссылку на каталог, проверьте его на наличие ошибочных символических ссылок и удалите их.

Символические ссылки могут также создать множество неприятностей, когда вы не знаете об их существовании. Например, вы легко можете отредактировать, как вам кажется, копию файла, хотя в действительности это символическая ссылка на оригинал.

ВНИМАНИЕ

Не забывайте о параметре -s при создании символической ссылки. Без него команда ln создает жесткую ссылку, присваивая дополнительное реальное имя тому же файлу. У нового имени файла тот же статус, что и у старого: оно непосредственно связывает с данными, а не указывает на другое имя файла, как это делает символическая ссылка. Жесткие ссылки могут запутать сильнее, чем символические. Пока вы не усвоите материал раздела 4.5, избегайте применять их.

Если символические ссылки снабжены таким количеством предупреждений, зачем их используют? Во-первых, они являются удобным способом упорядочения файлов и предоставления совместного доступа к ним, а во-вторых, позволяют справиться с некоторыми мелкими проблемами.

2.18. Архивирование и сжатие файлов

После того как вы узнали о файлах, правах доступа и возможных ошибках, вам необходимо освоить команды gzip и tar.

2.18.1. Команда gzip

Команда gzip (GNU Zip) – одна из стандартных команд сжатия файлов в системе Unix. Файл, имя которого оканчивается на .gz, является архивом в формате GNU Zip. Используйте команду gunzip *file.gz* для декомпрессии

файла <*file*>.gz и удаления суффикса из названия; для сжатия применяйте команду gzip file.

2.18.2. Команда tar

В отличие от программ сжатия в других операционных системах, команда gzip не создает архивы файлов, то есть она не упаковывает несколько файлов и каталогов в один файл. Для создания архива используйте команду tar:

$ **tar cvf archive.tar** *file1 file2* **...**

Архивы, созданные с помощью команды tar, обычно снабжены суффиксом .tar (по договоренности; он не является необходимым). В приведенном выше примере команды параметры *file1*, *file2* и т. д. являются именами файлов и каталогов, которые вы желаете упаковать в архив <*archive*>.tar. Флаг c активизирует режим создания. У флагов v и f более специфичные роли.

Флаг v активизирует подробный диагностический вывод, при котором команда tar отображает имена файлов и каталогов в архиве по мере работы с ними. Если добавить еще один флаг v, команда tar отобразит такие подробности, как размер файла и права доступа к нему. Если вам не нужны сообщения команды о том, что она сейчас делает, опустите флаг v.

Флаг f обозначает файл-параметр. Следующий после этого флага аргумент в командной строке должен быть именем файла-архива, который создаст команда tar (в приведенном выше примере это файл <*archive*>.tar). Вы всегда *должны* использовать этот параметр, указывая за ним имя файла (исключая вариант работы с накопителями на магнитной ленте). Для применения стандартных ввода и вывода введите дефис (-) вместо имени файла.

Распаковка файлов tar

Чтобы распаковать файл .tar с помощью команды tar, используйте флаг x:

$ **tar xvf** *archive***.tar**

Флаг x переводит команду tar *в режим извлечения (распаковки)*. Вы можете извлечь отдельные части архива, указав их имена в конце командной строки, но при этом вы должны знать их в точности. Чтобы быть уверенными, посмотрите краткое описание режима содержания.

ПРИМЕЧАНИЕ

При использовании режима извлечения помните о том, что команда tar не удаляет архивный файл .tar по окончании извлечения его содержимого.

Режим содержания

Перед началом распаковки следует просмотреть файл .tar в *режиме содержания*, использовав в команде флаг t вместо флага x. Этот режим

проверяет общую целостность архива и выводит имена всех находящихся в нем файлов. Если вы не просмотрите архив перед его распаковкой, то в результате можете получить в текущем каталоге хаос из файлов, который будет довольно трудно разобрать.

При проверке архива в режиме t убедитесь в том, что все расположено внутри разумной структуры каталогов, то есть все пути файлов в архиве должны начинаться с одного названия каталога. Если вы не уверены в этом, создайте временный каталог, перейдите в него, а затем выполните распаковку. Вы всегда сможете использовать команду mv * .., если архив не распаковался в виде хаоса.

При распаковке подумайте о возможности применить параметр p, чтобы сохранить права доступа. Используйте его в режиме распаковки, чтобы переопределить параметры команды umask и получить в точности те же права доступа, которые указаны в архиве. Параметр p применяется по умолчанию при работе в учетной записи superuser. Если у вас в качестве superuser-пользователя возникают сложности с правами доступа и владения при распаковке архива, дождитесь завершения работы команды и появления приглашения оболочки. Хотя вам может понадобиться извлечь лишь малую часть архива, команда tar должна обработать его полностью. Вам не следует прерывать этот процесс, поскольку команда выполняет назначение прав доступа только после проверки всего архива.

Выучите *все* параметры и режимы команды tar, которые упомянуты в этом разделе. Если для вас это сложно, выпишите их на карточки-подсказки – при работе с данной командой очень важно не допускать ошибок по небрежности.

2.18.3. Сжатые архивы (.tar.gz)

Начинающих пользователей часто смущает факт, что архивы обычно являются сжатыми, а их имена заканчиваются на .tar.gz. Чтобы распаковать сжатый архив, действуйте справа налево: сначала избавьтесь от суффикса .gz, а затем займитесь суффиксом .tar. Например, приведенные ниже команды производят декомпрессию и распаковку архива <*file*>.tar.gz:

$ **gunzip** *file*.**tar.gz**
$ **tar xvf** *file*.**tar**

Поначалу вы можете выполнять эту процедуру пошагово, запуская сначала команду gunzip для декомпрессии, а затем – команду tar для проверки и распаковки архива. Чтобы создать сжатый архив, выполните действия в обратном порядке: сначала запустите команду tar, а затем – gzip. Делайте это достаточно часто, и вскоре вы запомните, как работают процедуры архивации и сжатия.

2.18.4. Команда zcat

Описанный выше метод не является самым быстрым или наиболее эффективным при вызове команды tar для работы со сжатым архивом. Он расходует дисковое пространство и время ядра, затрачиваемое на ввод/вывод.

Лучший способ заключается в комбинации функций архивирования и компрессии в виде конвейера. Например, данная команда-конвейер распаковывает файл <*file*>.tar.gz:

$ **zcat** *file***.tar.gz | tar xvf —**

Команда zcat равносильна команде gunzip -dc. Параметр -d отвечает за декомпрессию, а параметр -c отправляет результат на стандартный вывод (в данном случае команде tar).

Поскольку использование команды zcat стало общепринятым, у версии команды tar, входящей в систему Linux, есть ярлык-сокращение. Можно применять в качестве параметра z, чтобы автоматически задействовать команду gzip. Это срабатывает как при извлечении архива (как в режимах x или t у команды tar), так и при создании (с параметром c). Используйте, например, следующую команду, чтобы проверить сжатый архив:

$ **tar ztvf** *file***.tar.gz**

Вам следует хорошо освоить длинный вариант команды, прежде чем применять сокращение.

ПРИМЕЧАНИЕ

Файл .tgz – это то же самое, что и файл .tar.gz. Суффикс приспособлен для использования в файловых системах FAT (на основе MS-DOS).

2.18.5. Другие утилиты сжатия

Еще одной командой для компрессии в системе Unix является bzip2, которая создает файлы, оканчивающиеся на .bz2. Работая чуть медленнее по сравнению с gzip, команда bzip2 зачастую сжимает текстовые файлы немного лучше и поэтому чрезвычайно популярна при распространении исходного программного кода. Для декомпрессии следует использовать команду bunzip2, а параметры для обоих компонентов довольно похожи на параметры команды gzip, так что вам не придется учить ничего нового. Для команды tar параметром компрессии/декомпрессии в режиме bzip2 является символ j.

Новая команда для сжатия под названием xz также приобретает популярность. Соответствующая ей команда декомпрессии называется unxz, а ее аргументы сходны с параметрами команды gzip.

В большинстве дистрибутивов системы Linux присутствуют команды zip и unzip, которые совместимы с ZIP-архивами Windows. Они работают как с обычными файлами .zip, так и с самораспаковывающимися архивами, файлы которых оканчиваются на .exe. Однако если вам встретится файл, который оканчивается на .Z, то вы обнаружили реликт, который был создан командой, бывшей когда-то стандартом для системы Unix. Команда gunzip способна распаковать такие файлы, однако создать их с помощью gzip невозможно.

2.20. Запуск команд с правами пользователя superuser

Прежде чем продвигаться дальше, вам следует узнать о том, как запускать команды в качестве пользователя superuser. Вероятно, вы уже знаете о том, что можно запустить команду su и указать пароль для входа в корневую оболочку. Такой вариант сработает, но у него есть некоторые неудобства:

- вы не отслеживаете команды, которые изменяют систему;
- вы не отслеживаете пользователей, которые выполнили команды, изменяющие систему;
- у вас нет доступа к обычной среде оболочки;
- вы должны вводить пароль.

2.20.1. Команда sudo

В большинстве дистрибутивов применяется пакет sudo, который позволяет администраторам запускать команды в корневом режиме, зайдя в систему со своей учетной записью. Например, из главы 7 вы узнаете об использовании команды vipw для редактирования файла /etc/passwd. Это можно выполнить так:

$ **sudo vipw**

При таком запуске команда sudo отмечает данное действие с помощью сервиса syslog в устройстве local2. Подробности о системных журналах изложены в главе 7.

2.20.2. Файл /etc/sudoers

Система не позволит *любому* пользователю запускать команды в качестве пользователя superuser. Вы должны настроить права таких привилегированных пользователей в файле /etc/sudoers. Пакет sudo обладает множеством параметров (которые, вероятно, вы никогда не станете применять), вследствие чего синтаксис файла /etc/sudoers довольно сложен. Так, приведенный ниже фрагмент позволяет пользователям *user1* и *user2* запускать любую команду в качестве корневого пользователя, не вводя при этом пароль:

```
User_Alias ADMINS = user1, user2
ADMINS ALL = NOPASSWD: ALL
root ALL=(ALL) ALL
```

В первой строке для двух пользователей определен псевдоним ADMINS, а во второй строке им предоставляются права доступа. Фрагмент ALL = NOPASSWD: ALL означает, что пользователи с псевдонимом ADMINS могут использовать пакет sudo для выполнения команд в корневом режиме. Второе слово ALL значит «любая команда». Первое слово ALL обозначает «любой хост».

ПРИМЕЧАНИЕ

Если у вас несколько компьютеров, можно указать различные типы доступа для каждого из них или для группы машин, но мы не будем рассматривать такой вариант.

Фрагмент root ALL=(ALL) ALL означает, что пользователь superuser может также использовать пакет sudo для запуска любой команды на любом хосте. Дополнительный параметр (ALL) означает, что пользователь superuser может также запускать команды как и любой другой пользователь. Можно распространить это право на пользователей ADMINS, добавив параметр (ALL) в строку файла /etc/sudoers, как отмечено ниже символом ❶:

```
ADMINS ALL = (ALL)❶ NOPASSWD: ALL
```

ПРИМЕЧАНИЕ

Воспользуйтесь командой visudo для редактирования файла /etc/sudoers. Эта команда проверяет отсутствие синтаксических ошибок после сохранения файла.

Если вам необходимо использовать более продвинутые функции команды sudo, обратитесь к страницам sudoers(5) и sudo(8). Подробности механизма переключения между пользователями рассмотрены в главе 7.

Задания

Для участия в розыгрыше призов каждый называет свою фамилию.

Каждая буква имени имеет значение, которое является ее рангом в английском алфавите. A и a имеют ранг 1, B и b - ранг 2 и так далее.

Длина фамилии складывается с суммой этих рангов, таким образом, получается число сом.

Массив случайных весов связан с фамилиями, и каждый сом умножается на соответствующий вес для получения так называемого выигрышного числа.

Пример:

имена: "COLIN, AMANDBA, AMANDAB, CAROL, PauL, JOSEPH".

веса: [1, 4, 4, 5, 2, 1]

PauL -> som = длина имени + 16 + 1 + 21 + 12 = 4 + 50 -> 54

Вес, связанный с PauL, равен 2, поэтому *выигрышное число* PauL равно 54 * 2 = 108.

Теперь можно отсортировать фамилии в порядке убывания выигрышных номеров. Если два человека имеют одинаковые выигрышные номера, отсортируйте их в алфавитном порядке по фамилиям.

Задача:

параметры: st - строка фамилий, we - массив весов, n - ранг.

return: фамилия участника, чей ранг равен n (ранги нумеруются от 1).

Пример:

имена: "COLIN,AMANDBA,AMANDAB,CAROL,PauL,JOSEPH".

веса: [1, 4, 4, 5, 2, 1]

n: 4

Функция должна вернуть: "PauL"

Примечания:

Массив весов по крайней мере такой же длины, как и количество имен, он может быть длиннее.

Если st пуст, то возвращается "Нет участников".

Если n больше числа участников, то возвращается "Недостаточно участников".

Дополнительные примеры см. в разделе Примеры тестовых примеров.

Переведено с помощью www.DeepL.com/Translator (бесплатная версия)

2 (7). Возьмите целое число n (n >= 0) и цифру d (0 <= d <= 9).

Возведите в квадрат все числа k (0 <= k <= n) между 0 и n.

Подсчитайте количество цифр d, использованных при записи всех k**2.

Вызовите функцию nb_dig (или nbDig или ...), принимающую n и d в качестве параметров и возвращающую этот подсчет.