Изучаем Linux, 101: Командная строка Linux

Освоение команд GNU и UNIX

Ян Шилдс (ishields@us.ibm.com)
Старший программист
IBM

13.09.2011

Несмотря на всю привлекательность графических интерфейсов, применить всю мощь Linux® может только командная строка. Из этой статьи вы узнаете о некоторых наиболее важных функциях интерпретатора bash. Особое внимание будет уделено функциям, о которых необходимо знать для успешной сдачи экзаменов LPI 101 программы сертификации на администратора Linux начального уровня. К концу этой статьи вы сможете свободно использовать основные команды Linux, такие как echo и exit, устанавливать значения переменных окружения и получать информацию о системе. [*Прим. ред.: два первых примечания после листинга 8 были исправлены с учетом реальных значений идентификаторов процессов (PID).*]

Больше статей из этой серии

Развить навыки по этой теме

Этот материал — часть knowledge path для развития ваших навыков. Смотри Основы системного администрирования Linux: работа с консолью

Краткий обзор

Эта статья кратко познакомит вас с некоторыми наиболее важными функциями интерпретатора bash. Будут рассмотрены следующие темы:

- Взаимодействие с командными интерпретаторами и командами при помощи командной строки.
- Правильное использование команд и последовательностей команд.
- Определение, изменение, использование и экспорт переменных окружения.
- История команд и средства редактирования.
- Вызов команд, как перечисленных, так и не перечисленных в переменной окружения PATH.
- Использование справочных man-страниц для поиска информации о командах.

Об этой серии

Эта серия статей поможет вам освоить задачи администрирования операционной системы Linux. Вы также можете использовать материал этих статей для подготовки к экзаменам первого уровня сертификации профессионального института Linux (LPIC-1).

Чтобы посмотреть описания статей этой серии и получить ссылки на них, обратитесь к нашему перечню материалов для подготовки к экзаменам LPIC-1. Этот перечень постоянно дополняется новыми статьями по мере их готовности и содержит самые последние (по состоянию на апрель 2009 года) цели экзаменов сертификации LPIC-1. Если какая-либо статья отсутствует в перечне, можно найти ее более раннюю версию, соответствующую предыдущим целям LPIC-1 (до апреля 2009 года), обратившись к нашим руководствам для подготовки к экзаменам института Linux Professional Institute.

Эта статья поможет вам подготовиться к сдаче экзамена LPI 101 на администратора начального уровня (LPIC-1) и содержит материалы цели 103.1 темы 103. Цель имеет вес 4. Материал этой статьи соответствует целям экзамена LPI 101 по состоянию на апрель 2009 года. Всегда обращайтесь к Web-сайту программы сертификации LPIC, чтобы уточнить последние цели.

Необходимые условия

Чтобы извлечь наибольшую пользу из наших статей, необходимо обладать базовыми знаниями о Linux и иметь работоспособный компьютер с Linux, на котором можно будет выполнять все встречающиеся команды. Иногда различные версии программ выводят результаты по-разному, поэтому содержимое листингов и рисунков может отличаться от того, что вы увидите на вашем компьютере.

Командный интерпретатор bash

Как связаться с Яном

Ян – один из наших наиболее популярных и плодовитых авторов. Ознакомьтесь со всеми статьями Яна (EN), опубликованными на сайте developerWorks. Вы можете найти контактные данные в профиле Яна и связаться с ним, а также с другими авторами и участниками ресурса My developerWorks.

Командный интерпретатор *bash* – это один из нескольких интерпретаторов, доступных в Linux. Другое свое название – *B*ourne-*a*gain *sh*ell – интерпретатор bash получил в честь Стивена Борна (Stephen Bourne), создателя его ранней версии (*/bin/sh*). По большей части bash совместим с sh, но содержит множество улучшений, затрагивающих как функциональность, так и возможности программирования. Bash сочетает возможности интерпретаторов Korn shell (ksh) и C shell (csh) и является POSIX-совместимым интерпретатором.

Прежде чем мы перейдем к дальнейшему рассмотрению bash, вспомните, что *командный интерпретатор* (или *командная оболочка*) – это программа, принимающая и выполняющая программы. Командный интерпретатор также поддерживает конструкции программирования, позволяя составлять сложные команды из более простых. Эти сложные команды, или *сценарии* можно сохранять в виде файлов, которые могут становиться новыми

самостоятельными командами. В действительности многие команды в обычной Linux-системе **являются** сценариями.

Интерпретаторы содержат ряд *встроенных* команд, таких как `cd`, `break` и `exec`. Другие команды являются *внешними*.

Кроме того, интерпретаторы используют три стандартных *потока* ввода/вывода.

- *stdin* – *стандартный поток ввода* (standard input stream), обеспечивающий ввод для команд.
- *stdout* – *стандартный поток вывода* (standard output stream), обеспечивающий отображение результатов выполнения команд.
- *stderr* – *стандартный поток ошибок* (standard error stream), обеспечивающий отображение ошибок, возникающих при выполнении команд.

При помощи потоков ввода обеспечивается ввод данных для команд (обычно с клавиатуры терминала). Потоки вывода отображают текстовые символы, которые обычно выводятся на терминал. Изначально терминал представлял собой ASCII печатающее устройство или дисплейный терминал, но сейчас, как правило, это просто окно на рабочем столе компьютера. Более подробно о перенаправлении потоков ввода/вывода можно узнать из другой статьи этой серии (см. перечень материалов для подготовки к экзаменам LPIC-1).

Мы считаем, что читатель этой статьи знает, как получить доступ к командной строке. Если вы не знаете этого, то прочитайте статью Основные задачи для начинающих Linux-разработчиков (EN), которая поможет вам решить эту и многие другие задачи.

Если при работе в Linux вы не используете графический интерфейс или открываете окно терминала на графическом рабочем столе, то в обоих случаях вы получите приглашение, которое может выглядеть следующим образом (листинг 1).

Листинг 1. Примеры приглашений обычного пользователя

```
[db2inst1@echidna db2inst1]$
ian@lyrebird:~>
$
```

Если вы войдете в систему как пользователь root (или суперпользователь), ваше приглашение может иметь следующий вид (листинг 2).

Листинг2. Примеры приглашений пользователя root, или суперпользователя

```
[root@echidna ~]#
lyrebird:~ #
#
```

Пользователь root имеет неограниченные права, поэтому следует осторожно использовать его учетную запись. Если вы обладаете привилегиями пользователя root, то, как правило, в конце приглашения будет отображаться знак решетки (#). Если вы работаете с привилегиями

обычного пользователя, в приглашении будет отображаться другой знак – как правило, это знак доллара ($). Приглашение на вашем компьютере может отличаться от приглашений, показанных в предыдущих примерах этой статьи. Оно может содержать имя пользователя, имя компьютера, текущую директорию, дату, время и так далее.

Примеры в статьях этой серии были выполнены на реальных компьютерах Linux с использованием приглашений, определенных по умолчанию для этих систем. Приглашения для суперпользователей заканчиваются символом #, поэтому вы легко сможете отличить их от приглашений для обычных пользователей, заканчивающихся символом $. Это соответствует соглашению, принятому в большинстве книг, посвященных Linux. Если вам покажется, что у вас что-то не работает или работает не так, как должно работать, сравните ваше приглашение с тем, которое вы видите в примере.

Команды и последовательности

Итак, вы работаете с командной строкой. Давайте посмотрим, что в ней можно делать. Главная задача интерпретатора заключается в преобразовании команд и обеспечении вашего взаимодействия с операционной системой Linux. Команды ОС Linux (а также UNIX®) состоят из *имени*, *опций* и *параметров*. Некоторые команды не имеют ни опций, ни параметров, некоторые имеют и то, и другое, а некоторые – только опции или только параметры..

Если строка содержит символ #, то все последующие символы в этой строке игнорируются. Таким образом, символ # может означать как приглашение для суперпользователя, так и комментарий. Как правило, его значение можно понять из контекста.

Команда echo

Команда `echo` выводит на экран свои аргументы, как показано в листинге 3.

Листинг 3. Примеры команды echo

```
[ian@echidna ~]$ echo Word
Word
[ian@echidna ~]$ echo A phrase
A phrase
[ian@echidna ~]$ echo Where     are   my   spaces?
Where are my spaces?
[ian@echidna ~]$ echo "Here     are   my   spaces." # plus comment
Here     are   my   spaces.
```

В третьем примере листинга 3 все последовательности пробелов были сокращены на выходе команды до одного пробела. Чтобы избежать этого, необходимо заключить строку в *кавычки* – либо в двойные ("), либо в одинарные ('). Bash использует *символы-разделители*, такие как пробелы, символы табуляции и символы новой строки для разделения входной строки на *маркеры*, которые передаются на вход вашей команде. Если же строка заключена в кавычки, то все дополнительные символы-разделители сохраняются, и вся строка воспринимается как один маркер. В вышеприведенном примере каждый маркер, размещенный в строке после имени команды, является параметром, следовательно, в каждом случае мы имеем 1, 2, 4 и 1 параметр соответственно.

Команда echo имеет несколько опций. Обычно команда echo добавляет в конце своего вывода символ новой строки. Чтобы запретить это, используйте опцию `-n`. Чтобы разрешить использование начинающихся с символа "\" escape-последовательностей, выполняющих специальные действия, используйте опцию `-e`. Некоторые из этих последовательностей перечислены в таблице 1.

Таблица 1. Escape-последовательности команды echo

Escape последовательность	Функция
\a	Сигнал тревоги (звонок)
\b	Забой
\c	Подавлять символ новой строки в конце (аналогично использованию опции –n)
\f	Перевод страницы (очищает экран дисплея)
\n	Новая строка
\r	Возврат каретки
\t	Горизонтальная табуляция

Escape-последовательности и продолжение строки

При использовании символа обратной косой черты ("\") в bash может возникнуть небольшая проблема. Если символ \ не заключен в кавычки, он сигнализирует bash о том, что следующий за ним символ должен восприниматься как буквенное значение. Это необходимо для специальных метасимволов интерпретатора bash, которые мы рассмотрим чуть позже. Существует одно исключение из этого правила: если вслед за символом обратной косой черты следует символ новой строки, то bash воспринимает эту последовательность как запрос на продолжение строки. Это удобно использовать для разбиения длинных строк, особенно в сценариях.

Чтобы рассмотренные выше последовательности правильно обрабатывались командой `echo`, а также многими другими командами, использующими те же самые управляющие escape-последовательности, вы должны заключить escape-последовательности в кавычки (либо они должны являться частью строки, заключенной в кавычки) или использовать два символа \ подряд, чтобы один из них был воспринят интерпретатором как символ escape-последовательности. В листинге 4 приведено несколько примеров использования символа \.

Листинг 4. Еще несколько примеров команды echo

```
[ian@echidna ~]$ echo -n No new line
No new line[ian@echidna ~]$ echo -e "No new line\c"
No new line[ian@echidna ~]$ echo "A line with a typed
> return"
A line with a typed
return
[ian@echidna ~]$ echo -e "A line with an escaped\nreturn"
A line with an escaped
return
[ian@echidna ~]$ echo "A line with an escaped\nreturn but no -e option"
A line with an escaped\nreturn but no -e option
[ian@echidna ~]$ echo -e Doubly escaped\\n\\tmetacharacters
Doubly escaped
        metacharacters
[ian@echidna ~]$ echo Backslash \
> followed by newline \
> serves as line continuation.
Backslash followed by newline serves as line continuation.
```

Обратите внимание на то, что когда вы набираете в строке текст без использования кавычек, bash отображает специальное приглашение (>). При этом ввод текста продолжается с новой строки, и к нему добавляется символ новой строки.

Метасимволы интерпретатора bash и операторы управления

В bash существует несколько *метасимволов*, которые, будучи не заключенными в кавычки, служат для разделения входной строки на отдельные слова. Помимо пробела существуют следующие метасимволы:

- |
- &
- ;
- (
-)
- <
- >

Некоторые из них будут подробно рассмотрены далее в этой статье, а сейчас запомните, что если вы хотите использовать метасимвол как часть текста, то его необходимо либо заключить в кавычки, либо поместить перед ним знак \, как было показано в листинге 4.

Символ новой строки и определенные метасимволы или пары метасимволов также исполняют роль *операторов управления*. Приведем примеры:

- ||
- &&
- &
- ;
- ;;
- |

- (
-)

Некоторые из этих операторов управления позволяют вам создавать *последовательности*, или *списки* команд.

Простейшая последовательность состоит из двух команд, разделенных точкой с запятой (;). Команды выполняются по очереди. В любой среде программирования команды возвращают код завершения (успех либо ошибка); команды Linux обычно возвращают нулевое значение в случае успешного завершения, и отличное от нуля значение – в случае возникновения ошибки. Можно задать выполнение команд по условию, используя операторы управления && и ||. Если разделить две команды оператором управления &&, то вторая команда будет выполнена только в том случае, если код завершения первой команды будет равен 0. Если разделить две команды оператором ||, то вторая команда будет выполнена только в том случае, если код завершения первой команды будет отличен от нуля. В листинге 5 показано несколько последовательностей с использованием команды echo. Эти примеры не очень интересны, поскольку команда echo возвращает 0, но в дальнейшем, когда вы познакомитесь с большим количеством команд, будут рассмотрены и другие примеры.

Листинг 5. Последовательности команд

```
[ian@echidna ~]$ echo line 1;echo line 2; echo line 3
line 1
line 2
line 3
[ian@echidna ~]$ echo line 1&&echo line 2&&echo line 3
line 1
line 2
line 3
[ian@echidna ~]$ echo line 1||echo line 2; echo line 3
line 1
line 3
```

Команда exit

Вы можете завершить работу интерпретатора, используя команду `exit`. Для этой команды можно задать параметр, который будет являться кодом ее завершения. Если интерпретатор запущен в окне терминала или в графическом окне рабочего стола, то это окно закроется. Аналогично, если вы подключились к удаленной системе, например, с помощью протокола ssh или telnet, то соединение завершится. Для завершения работы интерпретатора bash можно также удерживать клавишу **Ctrl** и нажать клавишу **d**.

Давайте рассмотрим еще один оператор управления. Если вы заключите команду или список команд в круглые скобки, то эта команда (или последовательность) будет выполнена в командной подоболочке, таким образом, команда exit завершит работу этой подоболочки, а не интерпретатора, в котором вы работаете. В листинге 6 приведен простой пример совместного использования операторов управления && и ||, а также двух различных кодов завершения.

Листинг 6. Подоболочки и последовательности

```
[ian@echidna ~]$ (echo In subshell; exit 0) && echo OK || echo Bad exit
In subshell
OK
[ian@echidna ~]$ (echo In subshell; exit 4) && echo OK || echo Bad exit
In subshell
Bad exit
```

О других последовательностях команд мы расскажем далее в этой статье.

Переменные окружения

Когда вы работаете в bash, ваше *окружение* состоит из многих параметров, таких как вид приглашения, домашняя директория, рабочая директория, имя интерпретатора, открытые вами файлы, определенные вами функции и так далее. Ваше окружение включает множество *переменных*, которые могут определяться как интерпретатором, так и вами. В bash также можно использовать *переменные командной оболочки*, которые можно экспортировать в ваше окружение для использования другими процессами, запущенными в командном интерпретаторе, или другими командными интерпретаторами, которые вы можете вызывать из текущего интерпретатора.

Как переменные окружения, так и переменные командной оболочки имеют *имена*. Для ссылки на значение переменной необходимо поставить перед ее именем знак доллара ('$'). В таблице 2 перечислены основные переменные окружения, которые могут вам встретиться.

Таблица 2. Основные переменные окружения bash

Имя	Функция
USER	Имя пользователя, вошедшего в систему
UID	Числовой идентификатор пользователя, вошедшего в систему
HOME	Домашняя директория пользователя
PWD	Текущая рабочая директория
SHELL	Имя командного интерпретатора
$	Идентификатор процесса (*PID*) запущенного интерпретатора bash (или другого процесса)
PPID	Идентификатор процесса, породившего данный процесс (т. е. идентификатор родительского процесса)
?	Код завершения последней команды

В листинге 7 показаны значения некоторых переменных bash.

Листинг 7. Переменные окружения и командной оболочки

```
[ian@echidna ~]$ echo $USER $UID
ian 500
[ian@echidna ~]$ echo $SHELL $HOME $PWD
/bin/bash /home/ian /home/ian
[ian@echidna ~]$ (exit 0);echo $?;(exit 4);echo $?
0
4
[ian@echidna ~]$ echo $$ $PPID
2559 2558
```

Не используете bash?

Интерпретатор bash используется по умолчанию во многих дистрибутивах Linux. Если же вы не работаете в командной оболочке bash, то можно попробовать работать с этим интерпретатором одним из следующих способов.

- Используйте команду
  `chsh -s /bin/bash`
  для изменения интерпретатора по умолчанию. Изменения вступят в силу при следующем входе в систему.
- Используйте команду
  `su - $USER -s /bin/bash`
  для создания дочернего процесса вашего текущего интерпретатора. Новый процесс будет являться командной оболочкой входа в систему, использующей bash.
- Создайте идентификатор пользователя с использованием интерпретатора bash по умолчанию для подготовки к сдаче экзамена LPI.

Вы можете создать или *задать значение* переменной командной оболочки, поставив знак равенства (=) сразу же после ее имени. Если переменная существует, то она получит новое значение. Имена переменных чувствительны к регистру, поэтому var1 и VAR1 – это две разные переменные. Существует соглашение, по которому имена переменных (в особенности, экспортируемых переменных) задаются в верхнем регистре, однако, это требование не является обязательным. С технической точки зрения $$ и $? – это не переменные, а *параметры* командной оболочки. На эти параметры можно ссылаться, но нельзя присвоить им новые значения.

Когда вы создаете новую переменную командной оболочки, часто может потребоваться *экспортировать* ее и, тем самым, сделать ее доступной для других процессов, которые могут быть запущены из этого интерпретатора. Экспортируемые переменные **не** доступны родительскому интерпретатору. Для экспорта переменной используется команда `export`. В bash вы можете задать значение переменной и сразу же экспортировать ее за один шаг.

Для демонстрации этого приема давайте запустим из текущего интерпретатора bash новый интерпретатор bash, а затем запустим из дочернего интерпретатора bash интерпретатор Korn shell (ksh). Для вывода информации о запущенных процессах мы воспользуемся командой `ps`. Более подробно об этой команде вы можете узнать из другой статьи этой серии (см. перечень материалов для подготовки к экзаменам LPIC-1 в разделе Ресурсы).

Листинг 8. Дополнительные переменные окружения и командной оболочки

```
[ian@echidna ~]$ ps -p $$ -o "pid ppid cmd"
```

```
  PID  PPID CMD
 2559  2558 -bash
[ian@echidna ~]$ bash
[ian@echidna ~]$ ps -p $$ -o "pid ppid cmd"
  PID  PPID CMD
 2811  2559 bash
[ian@echidna ~]$ VAR1=var1
[ian@echidna ~]$ VAR2=var2
[ian@echidna ~]$ export VAR2
[ian@echidna ~]$ export VAR3=var3
[ian@echidna ~]$ echo $VAR1 $VAR2 $VAR3
var1 var2 var3
[ian@echidna ~]$ echo $VAR1 $VAR2 $VAR3 $SHELL
var1 var2 var3 /bin/bash
[ian@echidna ~]$ ksh
$ ps -p $$ -o "pid ppid cmd"
  PID  PPID CMD
 2840  2811 ksh
$ export VAR4=var4
$ echo $VAR1 $VAR2 $VAR3 $VAR4 $SHELL
var2 var3 var4 /bin/bash
$ exit
[ian@echidna ~]$ echo $VAR1 $VAR2 $VAR3 $VAR4 $SHELL
var1 var2 var3 /bin/bash
[ian@echidna ~]$ ps -p $$ -o "pid ppid cmd"
  PID  PPID CMD
 2811  2559 bash
[ian@echidna ~]$ exit
exit
[ian@echidna ~]$ ps -p $$ -o "pid ppid cmd"
  PID  PPID CMD
 2559  2558 -bash
[ian@echidna ~]$ echo $VAR1 $VAR2 $VAR3 $VAR4 $SHELL
/bin/bash
```

Примечания:

1. В самом начале последовательности интерпретатор bash имеет идентификатор PID 2559.
2. Второй интерпретатор bash имеет идентификатор PID 2811, а его родительский процесс (т. е. исходный интерпретатор bash) имеет идентификатор PID 2559.
3. Во втором (дочернем) интерпретаторе bash мы создали три переменных – VAR1, VAR2 и VAR3, но экспортировали только переменные VAR2 и VAR3.
4. В интерпретаторе Korn мы создали переменную VAR4. Команда `echo` показала нам значения переменных VAR2, VAR3 и VAR4, тем самым подтвердив, что переменная VAR1 не была экспортирована. Вас не удивляет, что значение переменной SHELL не изменилось, несмотря на изменение приглашения? Вы не можете всегда полагаться на переменную SHELL, чтобы определить, в каком командном интерпретаторе вы работаете, однако команда `ps` показывает реальную картину. Заметьте, что команда `ps` ставит перед именем первого интерпретатора bash знак дефиса (-), который означает, что это *командный интерпретатор, назначаемый при входе в систему* (login shell).
5. Возвращаясь ко второму интерпретатору bash, мы видим переменные VAR1, VAR2 и VAR3.
6. Наконец, вернувшись в исходный интерпретатор, мы видим, что ни одной переменной не существует.

При обсуждении использования кавычек мы упоминали, что можно использовать как одинарные, таки двойные кавычки. Между ними существует важное отличие. Интерпретатор подставляет значения переменных командной оболочки, если они заключены в двойные кавычки ($quot;), но не делает этого при использовании одинарных кавычек ('). В предыдущем примере мы запустили дополнительный интерпретатор внутри исходного интерпретатора и получили новый идентификатор процесса. Используя опцию `-c`, можно передать команду другому интерпретатору, который выполнит ее и вернет управление. Если в качестве команды вы передаете строку, заключенную в одинарные кавычки, то внешний интерпретатор отбрасывает кавычки и передает строку в том виде, в каком она есть. Если же вы используете двойные кавычки, подстановка значения переменной происходит **до того**, как передается строка, поэтому результат может отличаться от ожидаемого. Интерпретатор и команда будут запущены в отдельном процессе и будут иметь другие идентификаторы PID. Эта концепция проиллюстрирована в листинге 9. Жирным шрифтом выделен идентификатор PID интерпретатора bash верхнего уровня.

Листинг 9. Использование кавычек и переменные командной оболочки

```
[ian@echidna ~]$ echo "$SHELL" '$SHELL' "$$" '$$'
/bin/bash $SHELL 2559 $$
[ian@echidna ~]$ bash -c "echo Expand in parent $$ $PPID"
Expand in parent 2559 2558
[ian@echidna ~]$ bash -c 'echo Expand in child $$ $PPID'
Expand in child 2845 2559
```

До сих пор все имена переменных оканчивались пробелами, поэтому было понятно, где они заканчиваются. В действительности имена переменных могут состоять только из букв, цифр или символа подчеркивания. Интерпретатор знает, что имя переменной заканчивается там, где встречается другой символ. Иногда необходимо использовать переменные в выражениях, значение которых не очевидно. В таких случаях имена переменных отделяются с помощью фигурных скобок, как показано в листинге 10.

Листинг 10. Использование фигурных скобок с именами переменных

```
[ian@echidna ~]$ echo "-$HOME/abc-"
-/home/ian/abc-
[ian@echidna ~]$ echo "-$HOME_abc-"
--
[ian@echidna ~]$ echo "-${HOME}_abc-"
-/home/ian_abc-
```

Команда env

Команда `env`, запущенная без каких-либо опций или параметров, выводит текущие значения переменных окружения. Также эту команду можно использовать для выполнения другой команды в пользовательском окружении. Опция `-i` (или просто `-`) очищает текущее окружение перед запуском команды, а опция `-u` сбрасывает значения переменных окружения, которые вы не хотите передавать.

В листинге 11 показана часть вывода команды `env`, запущенной без каких-либо параметров, а также приведено три примера вызова различных интерпретаторов без родительского окружения. Внимательно ознакомьтесь с примерами, прежде чем мы обсудим их.

Примечание. Если в вашей системе не установлен интерпретатор ksh (Korn) или tcsh, то необходимо установить его для выполнения этих примеров.

Листинг 11. Команда env

```
[ian@echidna ~]$ env
HOSTNAME=echidna
SELINUX_ROLE_REQUESTED=
TERM=xterm
SHELL=/bin/bash
HISTSIZE=1000
SSH_CLIENT=9.27.206.68 1316 22
SELINUX_USE_CURRENT_RANGE=
QTDIR=/usr/lib/qt-3.3
QTINC=/usr/lib/qt-3.3/include
SSH_TTY=/dev/pts/3
USER=ian
...
_=/bin/env
OLDPWD=/etc
[ian@echidna ~]$ env -i bash -c 'echo $SHELL; env'
/bin/bash
PWD=/home/ian
SHLVL=1
_=/bin/env
[ian@echidna ~]$ env -i ksh -c 'echo $SHELL; env'
/bin/sh
_=/bin/env
PWD=/home/ian
_AST_FEATURES=UNIVERSE - ucb
[ian@echidna ~]$ env -i tcsh -c 'echo $SHELL; env'
SHELL: Undefined variable.
```

Заметьте, что bash установил значение переменной SHELL, но не экспортировал ее в окружение, хотя создал в этом окружении три другие переменные. Во втором примере с интерпретатором ksh мы имеем две переменные окружения, но в результате попытки отобразить значение переменной SHELL мы получили пустую строку. Наконец, в последнем примере интерпретатор tcsh не создал ни одной переменной окружения, а при попытке получить значение переменной SHELL выдал ошибку.

Команды unset и set

Из листинга 11 видно, что интерпретаторы обрабатывают переменные и окружения по-разному. Хотя эта статья посвящена рассмотрению bash, следует учитывать, что не все интерпретаторы работают одинаково. Более того, работа интерпретаторов может изменяться в зависимости от того, являются ли они *интерпретаторами, назначаемым при входе в систему* (login shell), или нет. Пока я просто скажу, что назначаемый при входе в систему интерпретатор – это интерпретатор, который запускается при входе пользователя в систему; при желании вы можете запустить другие интерпретаторы, которые будут работать как интерпретаторы, назначаемые при входе. Три интерпретатора в вышеприведенных примерах, запущенные командой `env -i`, не являлись интерпретаторами, назначаемыми при входе. Попробуйте запустить интерпретатор с опцией `-l`, чтобы увидеть отличия между работой интерпретатора, назначаемого при входе, и интерпретатора, запущенного в обычном режиме.

Итак, давайте обсудим результаты нашей троекратной попытки отобразить значение переменной SHELL при работе с интерпретаторами, не являющимися назначаемыми при входе в систему.

1. При запуске интерпретатора bash было установлено значение переменной SHELL, но она не была автоматически экспортирована в окружение.
2. При запуске интерпретатора ksh значение переменной SHELL не было установлено. Однако ссылка на неопределенную переменную окружения эквивалентна ссылке на переменную с пустым значением.
3. При запуске интерпретатора tcsh значение переменной SHELL не было установлено. Кроме того, в этом случае поведение по умолчанию отличается от поведения ksh (а также bash), а именно, при попытке использования несуществующей переменной выдается сообщение об ошибке.

Вы можете использовать команду `unset` для сброса переменной и удаления ее из списка переменных командной оболочки. Если переменная была экспортирована в окружение, то она также будет удалена и из окружения. При помощи команды `set` вы можете управлять многими аспектами работы bash (или других интерпретаторов). Команда set является встроенной командой интерпретатора, поэтому для разных интерпретаторов ее опции будут разными. Если опция `-u` используется в bash, то при ссылке на неопределенную переменную интерпретатор сообщает об ошибке, а не расценивает ее как пустую. Различные опции команды `set` включаются с помощью символа `-` и выключаются с помощью символа `+`. Текущий список опций можно отобразить с помощью команды `echo $-`.

Листинг 12. Команды unset и set

```
[ian@echidna ~]$ echo $-
himBH
[ian@echidna ~]$ echo $VAR1

[ian@echidna ~]$ set -u;echo $-
himuBH
[ian@echidna ~]$ echo $VAR1
-bash: VAR1: unbound variable
[ian@echidna ~]$ VAR1=v1
[ian@echidna ~]$ VAR1=v1;echo $VAR1
v1
[ian@echidna ~]$ unset VAR1;echo $VAR1
-bash: VAR1: unbound variable
[ian@echidna ~]$ set +u;echo $VAR1;echo $-

himBH
```

Если вы используете команду `set` без каких-либо опций, то она выводит список всех переменных командной оболочки и их значения (если они существуют). Существует также команда `declare`, с помощью которой можно создавать, экспортировать и выводить значения переменных командной оболочки. Если вы хотите узнать о других опциях команды `set` или о команде `declare`, то обратитесь к соответствующим man-страницам. Справочные man-страницы будут рассмотрены далее в этой статье.

Команда exec

Последняя команда, которую нам осталось рассмотреть – это команда `exec`. Эта команда запускает другую команду, которая **замещает** текущий интерпретатор. Посмотрите на листинг 13, в котором сначала запускается дочерний интерпретатор bash, а затем выполняется команда `exec`, которая замещает его интерпретатором Korn. При выходе из Korn мы возвращаемся назад в исходный интерпретатор bash (в этом примере он имеет идентификатор PID 2852).

Листинг 13. Использование команды exec

```
[ian@echidna ~]$ echo $$
2852
[ian@echidna ~]$ bash
[ian@echidna ~]$ echo $$
5114
[ian@echidna ~]$ exec ksh
$ echo $$
5114
$ exit
[ian@echidna ~]$ echo $$
2852
```

Получение системной информации с помощью uname

Команда `uname` выводит информацию о вашей операционной системе и ее ядре. В листинге 14 показаны различные опции команды `uname` и результирующая информация. Описание опций содержится в таблице 3.

Листинг 14. Команда uname

```
[ian@echidna ~]$ uname
Linux
[ian@echidna ~]$ uname -s
Linux
[ian@echidna ~]$ uname -n
echidna.raleigh.ibm.com
[ian@echidna ~]$ uname -r
2.6.29.6-217.2.3.fc11.i686.PAE
[ian@echidna ~]$ uname -v
#1 SMP Wed Jul 29 16:05:22 EDT 2009
[ian@echidna ~]$ uname -m
i686
[ian@echidna ~]$ uname -o
GNU/Linux
[ian@echidna ~]$ uname -a
Linux echidna.raleigh.ibm.com 2.6.29.6-217.2.3.fc11.i686.PAE
#1 SMP Wed Jul 29 16:05:22 EDT 2009 i686 i686 i386 GNU/Linux
```

Таблица 3. Опции команды uname

Опция	Описание
-s	Выводит имя ядра. Если не указаны какие-либо опции, то это действие выполняется по умолчанию.
-n	Выводит имя узла (т. е. хоста).
-r	Выводит информацию о релизе ядра. Эта опция часто

	используется в командах, предназначенных для добавления или удаления модулей.
-v	Выводит информацию о версии ядра.
-m	Выводит название центрального процессора.
-o	Выводит имя операционной системы.
-a	Выводит всю вышеуказанную информацию.

Из примера в листинге 14 видно, что он был выполнен в ОС Fedora 11, работающей на компьютере на базе процессора Intel®. Команда `uname` доступна во многих UNIX-системах, а также в операционных системах Linux. Выводимая информация зависит от используемого Linux-дистрибутива и его версии, а также от архитектуры компьютера, на котором установлена ваша операционная система. В листинге 15 показан вывод команды uname, запущенной на компьютере AMD Athlon 64 под управлением Ubuntu 9.04.

Листинг 15. Пример выполнения команды uname в другой операционной системе

```
ian@attic4:~$ uname -a
Linux attic4 2.6.28-14-generic #47-Ubuntu SMP Sat Jul 25 01:19:55 UTC 2009 x86_64
GNU/Linux
```

История команд

Если по мере прочтения этой статьи вы набираете все встречающиеся в ней команды, то вы могли заметить, что часто используются одни и те же команды (возможно, лишь с незначительными изменениями). Хорошая новость заключается в том, что интерпретатор bash может вести *историю* ваших команд. Эта функциональность используется по умолчанию. Вы можете отключить историю команд с помощью команды `set +o history` и включить ее назад с помощью команды `set -o history`. Переменная окружения HISTSIZE сообщает интерпретатору bash о том, сколько строк с командами необходимо хранить в истории. Остальные параметры влияют на работу и управление историей.

Приведем примеры некоторых команд для работы с историей:

history
: Отображает всю историю

history *N*
: Отображает последние *N* строк истории

history -d *N*
: Удаляет строку *N* из истории; эту команду можно использовать, например, для удаления строки, содержащей пароль

!!
: Последняя введенная вами команда

!*N*
: Команда истории с номером *N*

!-*N*
: Команда, введенная вами *N* шагов назад (!-1 эквивалентно !!)

!#
: Текущая команда, которую вы набираете

!*string*
: Последняя введенная команда, начинающаяся на *string*

!?*string***?**
: Последняя введенная команда, содержащая строку *string*

Вы также можете использовать символ двоеточия (:), после которого указываются определенные значения, для получения доступа или изменения части команды из вашей истории. В листинге 16 продемонстрированы некоторые возможности работы с историей.

Листинг 16. Управление историей

```
[ian@echidna ~]$ echo $$
2852
[ian@echidna ~]$ env -i bash -c 'echo $$'
9649
[ian@echidna ~]$ !!
env -i bash -c 'echo $$'
10073
[ian@echidna ~]$ !ec
echo $$
2852
[ian@echidna ~]$ !en:s/$$/$PPID/
env -i bash -c 'echo $PPID'
2852
[ian@echidna ~]$ history 6
  595  echo $$
  596  env -i bash -c 'echo $$'
  597  env -i bash -c 'echo $$'
  598  echo $$
  599  env -i bash -c 'echo $PPID'
  600  history 6
[ian@echidna ~]$ history -d598
```

Команды из листинга 16 выполняют следующие действия:

1. Вывод идентификатора PID текущего интерпретатора.
2. Запуск команды echo в новом интерпретаторе и вывод его идентификатора PID.
3. Повторный запуск последней команды.
4. Повторный запуск последней команды, начинающейся с 'ec'; в данном случае повторно запускается первая команда нашего примера.
5. Повторный запуск последней команды, начинающейся с 'ec', но переменная '$PPID' заменяется на '$$; таким образом, выводится идентификатор PID родительского процесса.
6. Вывод последних 6 команд истории.
7. Удаление из истории записи с номером 598, т. е. последней команды echo.

Также можно редактировать историю команд в интерактивном режиме. Для управления редактированием команд и истории bash использует библиотеку readline. Для перемещения

по истории, а также для редактирования строк по умолчанию используются те же самые клавиши и комбинации клавиш, что и в редакторе GNU Emacs. Комбинации клавиш emacs обычно обозначаются в виде **C-x** или **M-x**, где **x** – это обычная клавиша, а **C** и **M** соответствуют клавишам *Control* и *Meta*. На обычном персональном компьютере клавише Control редактора Emacs соответствует клавиша **Ctrl**, а клавише Meta соответствует клавиша **Alt**. В таблице 4 перечислены некоторые команды для редактирования истории. Помимо комбинаций, перечисленных в таблице 4, можно использовать клавиши управления курсором (такие как стрелки вправо, влево, вверх и вниз) и клавиши Home и End, которые выполняют привычные действия. Если вы хотите узнать о дополнительных функциях, а также о том, как настраивать эти опции с помощью файла инициализации библиотеки readline, обратитесь к соответствующим man-страницам.

Таблица 4. Редактирование истории с помощью команд emacs

Команда	Комбинация клавиш на ПК	Описание
C-f	Стрелка вправо	Перемещение на один символ вправо.
C-b	Стрелка влево	Перемещение на один символ влево.
C-p	Стрелка вверх	Переход к предыдущей команде в истории.
C-n	Стрелка вниз	Переход к следующей команде в истории.
C-r		Обратный инкрементный поиск. Набирайте буквы для поиска текста в строке в обратном направлении. Нажмите комбинацию C-r еще раз для поиска следующего совпадения в этой же строке.
M-f	Alt-f	Перемещение на начало следующего слова; в графическом окружении эта комбинация клавиш используется для открытия меню **File** текущего окна.
M-b	Alt-b	Перемещение на начало предыдущего слова.
C-a	Home	Перемещение на начало строки.
C-e	End	Перемещение на конец строки.
Backspace	Backspace	Удаление символа перед курсором.
C-d	Del	Удаление символа, на котором стоит курсор (можно настроить клавиши Del и Backspace таким образом, что они будут работать наоборот).
C-k	Ctrl-k	Удалить весь текст до конца строки и сохранить его для последующего использования.
M-d	Alt-d	Удалить текст до конца слова и сохранить его для последующего использования.
C-y	Ctrl-y	Вставить текст, удаленный с помощью команд удаления.

Если вы предпочитаете управлять редактированием истории в режиме редактора vi, то вы можете переключиться в этот режим с помощью команды `set -o vi`. Чтобы переключиться обратно в режим emacs, используйте команду `set -o emacs`. Когда вы получаете команду в

режиме vi, то изначально вы находитесь в режиме вставки vi. Более подробно о редакторе vi вы можете узнать из другой статьи этой серии (см. перечень материалов для подготовки к экзаменам LPIC-1 в разделе Ресурсы).

Пути: где моя команда?

Некоторые команды интерпретатора bash являются встроенными, а другие – внешними. Давайте сейчас рассмотрим внешние команды, как их запускать и как определить, что команда является внутренней.

Где интерпретатор ищет команды?

Внешние команды – это обычные файлы на вашем компьютере. Основы управления файлами рассматриваются в другой статье этой серии (см. перечень материалов для подготовки к экзаменам LPIC-1 в разделе Ресурсы). В операционных системах UNIX и Linux все файлы структурированы в виде одного большого дерева, корнем которого является директория /. До сих пор в примерах этой статьи нашей текущей директорией являлась домашняя директория пользователя. Домашние директории обычных пользователей обычно расположены внутри директории /home (например, моя домашняя директория – это /home/ian). Домашней директорией пользователя root обычно является /root. Когда вы набираете имя команды, интерпретатор bash использует для ее поиска *путь*, задающий местоположение файла или директории и представляющий собой список разделенных двоеточиями директорий, перечисленных в переменной окружения PATH.

Если вы хотите узнать, какая команда будет выполнена, если вы напечатаете определенную строку, то используйте команду `which` или `type`. В листинге 17 показан путь по умолчанию для моей учетной записи, а также месторасположения нескольких команд.

Листинг 17. Поиск месторасположений команд

```
[ian@echidna ~]$ echo $PATH
/usr/lib/qt-3.3/bin:/usr/kerberos/bin:/usr/lib/ccache:/usr/local/bin:/bin:/usr/b
in:/home/ian/bin
[ian@echidna ~]$ which bash env zip xclock echo set ls
alias ls='ls --color=auto'
        /bin/ls
/bin/bash
/bin/env
/usr/bin/zip
/usr/bin/xclock
/bin/echo
/usr/bin/which: no set in (/usr/lib/qt-3.3/bin:/usr/kerberos/bin:/usr/lib/ccache
:/usr/local/bin:/bin:/usr/bin:/home/ian/bin)
[ian@echidna ~]$ type bash env zip xclock echo set ls
bash is hashed (/bin/bash)
env is hashed (/bin/env)
zip is /usr/bin/zip
xclock is /usr/bin/xclock
echo is a shell builtin
set is a shell builtin
ls is aliased to `ls --color=auto'
```

Заметьте, что почти все директории, перечисленные в пути, расположены в директории /bin. Это общепринятое соглашение о месторасположении команд, хотя оно и не является

обязательным требованием (пример – команда /usr/lib/ccache). Команда `which` сообщила нам о том, что команда `ls` является *псевдонимом*, а команда `set` не была найдена. В этом случае мы можем предположить два варианта: либо команда не существует, либо она является встроенной. Команда `type` тоже сообщила нам о том, что `ls` является *псевдонимом*, однако определила, что команда `set` является встроенной командой bash. Также она определила, что помимо команды `echo`, расположенной в директории /bin и найденной с помощью `which`, имеется и встроенная команда `echo`. Кроме того, команды `which` и `type` выводят результаты в различном порядке.

Мы видели, что команда `ls`, показывающая содержимое директорий, является *псевдонимом*. Псевдонимы являются удобным способом использования команд с различными наборами опций и назначения им дополнительных имен. В нашем примере опция `--color=tty` выводит список файлов в цвете; различные цвета соответствуют различным типам файлов и директорий. Чтобы посмотреть, как осуществляется управление цветами и какие цвета соответствуют тем или иным типам файлов, выполните команду `dircolors --print-database`.

Каждая из этих команд имеет дополнительные опции. Исходя из ваших задач, вы можете использовать обе команды. Я предпочитаю использовать команду `which`, когда я точно уверен, что искомая команда существует, и мне просто необходимо узнать ее полный путь. Однако команда `type` выдает более точную информацию, которая иногда требуется мне в сценариях.

Выполнение других команд

Из листинга 17 видно, что полные пути к исполняемым файлам начинаются с корневой директории /. Например, команда xclock на самом деле является файлом /usr/bin/xclock, т. е. файлом, расположенным в директории /usr/bin. В более старых системах эту команду можно было найти в директории /usr/X11R6/bin. Даже если команда не расположена в директории из списка переменной PATH, ее все равно можно запустить, указав в командной строке ее имя и путь к ней. Можно использовать пути двух типов.

- *Абсолютные* пути – это пути, начинающиеся с /, например те, что вы видели в листинге 17 (/bin/bash, /bin/env и т. д).
- *Относительные* пути – это пути относительно вашей *текущей рабочей директории*, которую можно определить по команде `pwd`. Эти команды не начинаются с /, но содержат, как минимум, одну комбинацию ./.

Абсолютные пути можно использовать независимо от текущей рабочей директории, а относительные пути наиболее удобно использовать тогда, когда команда расположена по соседству с рабочей директорией. Предположим, вы разрабатываете новую версию классической программы "Hello World!", которая находится в поддиректории mytestbin вашей домашней директории. Для запуска этой программы вы могли бы использовать относительный путь `mytestbin/hello`, однако вы можете использовать два специальных имени: точку (.), означающую текущую директорию, и пару точек (..), означающую родительскую директорию текущей директории. Поскольку обычно домашняя директория не перечислена в переменной PATH (как правило, она и не должна быть перечислена), вам потребуется явно указать путь к любому исполняемому файлу, который вы хотите

запустить, находясь в вашей домашней директории. Например, если в вашей домашней директории находилась бы копия программы "Hello World!", вы могли бы запустить ее с помощью команды `./hello`. При указании абсолютного пути вы можете использовать как одну, так и две точки (. или ..), хотя одинарная точка в подобном случае не очень полезна. Также вы можете использовать символ тильды (~), который означает вашу домашнюю директорию, и сочетание ~*username*, означающее домашнюю директорию пользователя с именем *username*. В листинге 18 приведены некоторые примеры.

Листинг 18. Абсолютные и относительные пути

```
[ian@echidna ~]$ /bin/echo Use echo command rather than builtin
Use echo command rather than builtin
[ian@echidna ~]$ /usr/../bin/echo Include parent dir in path
Include parent dir in path
[ian@echidna ~]$ /bin/./././echo Add a couple of useless path components
Add a couple of useless path components
[ian@echidna ~]$ pwd # See where we are
/home/ian
[ian@echidna ~]$ ../../bin/echo Use a relative path to echo
Use a relative path to echo
[ian@echidna ~]$ myprogs/hello # Use a relative path with no dots
-bash: myprogs/hello: No such file or directory
[ian@echidna ~]$ mytestbin/hello # Use a relative path with no dots
Hello world!
[ian@echidna ~]$ ./hello
Hello world!
[ian@echidna ~]$ ~/mytestbin/hello # run hello using ~
Hello world!
[ian@echidna ~]$ ../hello # Try running hello from parent
-bash: ../hello: No such file or directory
```

Смена вашей рабочей директории

Точно так же, как вы можете запускать программы из различных директорий, вы можете изменять вашу текущую рабочую директорию с помощью команды `cd`. Аргументом этой команды должен являться либо абсолютный, либо относительный путь к директории. Так же, как и для команд, при указании путей можно использовать символы ., .., ~ и ~*username* . Если вы запустите команду `cd` без параметров, вы окажетесь в вашей домашней директории. Если в качестве параметра указан дефис (-), будет выполнен переход в предыдущую рабочую директорию. Ваша домашняя директория хранится в переменной окружения HOME, а предыдущая директория – в переменной OLDPWD. Таким образом, запуск команды `cd` без параметров эквивалентен запуску команды `cd $HOME`, а команда `cd -` эквивалентна команде `cd $OLDPWD`. Вместо длинной фразы *сменить текущую рабочую директорию* обычно говорят просто *сменить директорию*.

Наряду с переменной PATH (которая используется для поиска команд) существует переменная окружения CDPATH, содержащая разделенный двоеточиями список директорий (в дополнение к текущей рабочей директории), которые необходимо просматривать при разрешении относительных путей. Если при разрешении пути используется путь из переменной CDPATH, то команда `cd` выводит полный путь найденной директории. Как правило, в случае удачной смены директории не выводится никакой информации, кроме нового (возможно, измененного) приглашения. В листинге 19 приведены некоторые примеры.

Листинг 19. Смена директорий

```
[ian@echidna ~]$ cd /;pwd
/
[ian@echidna /]$ cd /usr/local;pwd
/usr/local
[ian@echidna local]$ cd ;pwd
/home/ian
[ian@echidna ~]$ cd -;pwd
/usr/local
/usr/local
[ian@echidna local]$ cd ~ian/..;pwd
/home
[ian@echidna home]$ cd ~;pwd
/home/ian
[ian@echidna ~]$ export CDPATH=~
[ian@echidna ~]$ cd /;pwd
/
[ian@echidna /]$ cd mytestbin
/home/ian/mytestbin
```

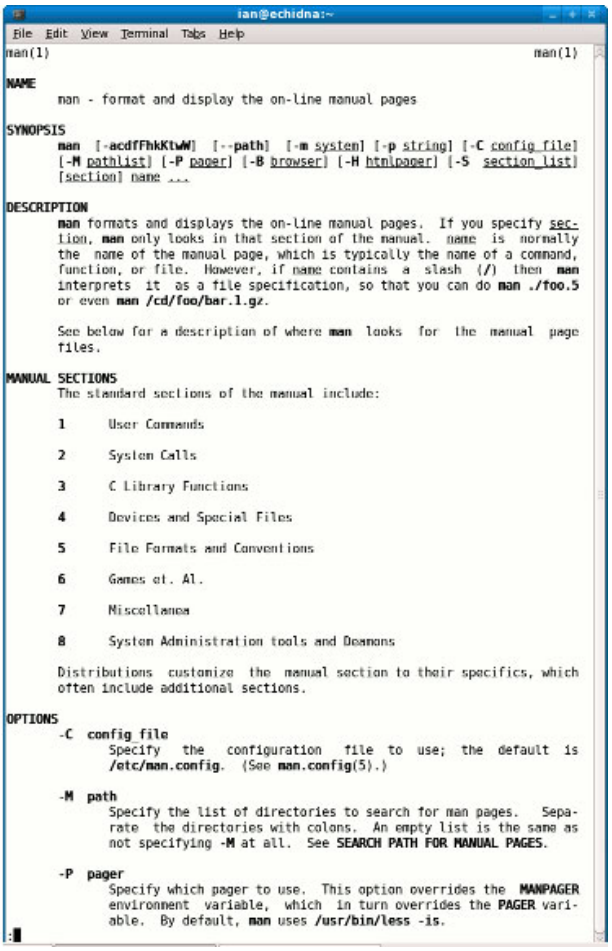
Справочные man-страницы

Мы подошли к последней теме нашей статьи, из которой вы узнаете, как получить справку по командам Linux с помощью man-страниц и других источников документации.

Справочные страницы и разделы

Главный (и традиционный) источник документации – это *man-страницы* (manual – справочник), к которым вы можете обращаться с помощью команды `man`. На рисунке 1 изображена man-страница самой команды `man`. Для для ее вызова выполните команду `man man`.

Рисунок 1. Man-страница команды man



На рисунке 1 показаны некоторые общие элементы man-страниц.

- Заголовок с именем команды, после которого в скобках указан номер раздела.
- Имена команды, а также связанных с ней команд, информация о которых приводится на этой man-странице.
- Сжатый список опций и параметров команды.
- Краткое описание команды.
- Подробное описание каждой опции.

Вы можете найти и другие разделы, содержащие сведения о том, как использовать команду или сообщить о найденной ошибке, информацию об авторе программы и список связанных команд. Например, man-страница команды `man` сообщает нам о следующих связанных командах (и их разделах справки):
apropos(1), whatis(1), less(1), groff(1) и man.conf(5).

Существуют восемь разделов, которые являются общими для всех man-страниц. Обычно man-страницы устанавливаются в процессе установки пакета, поэтому, если вы не установили какой-то пакет, то вряд ли вы найдете для него man-страницу. Аналогично, некоторые разделы man-страниц могут содержать совсем немного информации или вообще быть пустыми. Ниже перечислены общие разделы и приведены некоторые примеры.

1. Команды пользователя (env, ls, echo, mkdir, tty).
2. Системные вызовы или функции ядра (link, sethostname, mkdir).
3. Библиотечные процедуры (acosh, asctime, btree, locale, XML::Parser).
4. Информация об устройствах (isdn_audio, mouse, tty, zero).
5. Описания файловых форматов (keymaps, motd, wvdial.conf).
6. Игры (заметьте, что сегодня многие игры имеют графический интерфейс и графическую справочную систему, не связанную с man-страницами).
7. Разное (arp, boot, regex, unix utf8).
8. Системное администрирование (debugfs, fdisk, fsck, mount, renice, rpm).

Другие разделы могут иметь следующие обозначения: *9* (документация по ядру Linux), *n* (новая документация), *o* (старая документация) и *l* (локальная документация).

Некоторые сведения могут содержаться в нескольких разделах. Из вышеприведенного примера видно, что информация о команде mkdir содержится в разделах 1 и 2, а информация о tty – в разделах 1 и 4. При вызове справки вы можете указать определенный раздел (например, `man 4 tty` или `man 2 mkdir`) или использовать опцию `-a` для получения списка всех подходящих разделов справки.

На рисунке 1 вы могли заметить, что команда `man` имеет много опций, которые вы можете изучить самостоятельно. А сейчас давайте кратко рассмотрим некоторые команды, упомянутые в разделе "See also" на man-странице команды `man`.

Раздел "See also"

Две важные команды, связанные с командой `man`, это `whatis` и `apropos`. Команда `whatis` выводит информацию об указанном имени, найденную на соответствующих man-страницах. Команда `apropos` выводит список man-страниц, содержащих указанное ключевое слово. В листинге 20 приведены примеры этих команд.

Листинг 20. Примеры команд whatis и apropos

```
[ian@echidna ~]$ whatis man
man []               (1)  - format and display the on-line manual pages
man []               (1p)  - display system documentation
man []               (7)  - macros to format man pages
man []               (7)  - pages - conventions for writing Linux man pages
man.config []        (5)  - configuration data for man
man-pages           (rpm) - Man (manual) pages from the Linux Documentation Project
man                 (rpm) - A set of documentation tools: man, apropos and whatis
[ian@echidna ~]$ whatis mkdir
mkdir []             (1)  - make directories
mkdir []             (1p)  - make directories
mkdir []             (2)  - create a directory
mkdir []             (3p)  - make a directory
[ian@echidna ~]$ apropos mkdir
mkdir []             (1)  - make directories
mkdir []             (1p)  - make directories
mkdir []             (2)  - create a directory
mkdir []             (3p)  - make a directory
mkdirat []           (2)  - create a directory relative to a directory file descriptor
mkdirhier []         (1)  - makes a directory hierarchy
```

Между прочим, если вы не можете найти man-страницу для man.conf, то попытайтесь выполнить команду `man man.conf`**`ig`**.

Команда `man` разбивает выводимую на монитор информацию на страницы с помощью специальной программы. В большинстве Linux-систем это программа `less`. Вместо нее может использоваться другая, более старая программа – `more`. Если вы хотите распечатать страницу, используйте опцию `-t`, чтобы отформатировать страницу для последующей печати с помощью программы `groff` или `troff`.

У команды less имеется несколько команд, облегчающих поиск строк в полученном содержимом. Выполните команду `man less`, чтобы узнать о командах */* (прямой поиск), *?* (обратный поиск), *n* (повторить последний поиск), а также о многих других командах.

Другие источники документации

В дополнение к справочным man-страницам, доступным из командной строки, фондом Free Software Foundation был создан ряд *info*-файлов, с которыми работает программа *info*. Эти файлы обладают расширенными возможностями навигации, включая переход к другим разделам. Для получения дополнительной информации выполните команду `man info` или `info info`. Документация в формате info доступна не для всех команд, поэтому, так или иначе, вам придется использовать команду man, даже если вам больше нравится команда info.

Существует также и несколько графических интерфейсов для работы с man-страницами, таких как `xman` (входит в состав проекта XFree86) и `yelp` (справочный обозреватель Gnome 2.0).

Если вдруг вам не удалось найти справку по какой-либо команде, то попробуйте запустить эту команду с опцией `--help`. Возможно, вы получите необходимую справку или информацию о том, где можно ее найти.

Ресурсы

- Участвуйте в обсуждении данного материала на форуме.
- Оригинал статьи: Learn Linux, 101: The Linux command line (EN).
- Ознакомьтесь с целями экзаменов LPI 101 и LPI 102 по состоянию на апрель 2009 года. Всегда обращайтесь к Web-сайту программы сертификации LPIC, чтобы уточнить последние цели.
- На Web-сайте программы сертификации LPIC (EN) вы найдете подробные цели, списки задач и примерные вопросы всех трех уровней сертификации на администратора Linux-систем профессионального института Linux
- Прочитайте статью "Основные задачи для начинающих Linux-разработчиков" (EN) (developerWorks, март 2005 г.), где описано в частности, как открыть окно терминала или командной оболочки.
- Web-сайт Linux Documentation Project (EN) содержит большое количество полезной документации, в особенности, HOWTO-руководств.
- В разделе Linux сайта developerWorks можно найти дополнительные ресурсы для разработчиков Linux (включая разработчиков, только начинающих работу с Linux), а также самые популярные среди наших читателей статьи и руководства (EN).

Об авторе

Ян Шилдс

No bio.