Оболочки и вызов сценариев

Пользовательская оболочка bash может работать в двух режимах — интерактивном и, соответственно, не интерактивном. Открыть оболочку в Ubuntu можно комбинацией клавиш `Ctrl + Alt + F1`, привычный графический интерфейс исчезнет, а перед вами откроется один из семи виртуальных терминалов, доступных в дистрибутиве Ubuntu.

Если оболочка выдает приглашение (что-то вроде того, которое можно увидеть ниже), то вы работаете в интерактивном режиме:

```
user@host:~$
```

Здесь можно вводить самые разнообразные unix-команды (как например: ls, grep, cd, mkdir, rm) и видеть результат их выполнения. Интерактивной эта оболочка называется потому, что она взаимодействует с пользователем напрямую. Окружение рабочего стола (графический интерфейс), в семействе систем Debian (к которым относится и Ubuntu), принято размещать в седьмом виртуальном терминале, для того чтобы вернуться к привычному окружение рабочего стола наберите комбинацию `Ctrl + Alt + F7`.

Конечно работать в виртуальных терминалах не слишком удобно, особенно, если нужно редактировать документ и одновременно выполнять какие-либо команды, поэтому в дальнейшем мы будем пользоваться встроенным в графический интерфейс эмулятором виртуального терминала, встроенным в Ubuntu. Открыть его можно комбинацией клавиш `Ctrl + Alt + T`, или Unity Dash, найдя его в списке программ.

В не интерактивном режиме оболочка читает команды из некоторого файла и последовательно выполняет их. Когда интерпретатор дойдет до конца файла, работа с оболочкой автоматически завершится. Запустить оболочку в не интерактивном режиме можно с помощью команд:

```
sh скрипт
```

```
bash скрипт
```

Где `скрипт` — это путь к файлу, содержащему команды для выполнения. Такой файл является обычным текстовым документом, который можно создать с помощью любого текстового документа. Впрочем, можно упростить вызов скрипта всего лишь сделав его исполняемым. Для этого необходимо предоставить соответствующие права доступа этому файлу с помощью команды chmod:

```
chmod +x скрипт
```

Кроме этого, в первой строке скрипта необходимо указать какая именно оболочка должна выполнять этот сценарий. Это можно сделать, разместив в начале соответствующее указание `#! / Bin / sh` (для оболочки sh) или `#! / Bin / bash`(соответственно для bash). После этого файл можно будет вызвать на выполнение обратившись к нему в терминале:

```
./скрипт
```

Комментарии

Сценарии могут содержать комментарии. Комментарии — это операторы, их можно размещать в сценарии оболочки, но который игнорируется при исполнении.

Комментарии должны начинаются с символа `#` и продолжаются до символа новой строки.

Например:

```
#!/bin/bash

# Сценарий выведет имя пользователя

whoami
```

Переменные

Оболочка позволяет создавать и удалять переменные, а также выполнять операции над ними. Переменные в bash могут находиться в 3-х областях видимости:

Локальные переменные — это обычные переменные внутри одного сценария. Они не доступны другим программам и сценариям, которые запускаются с этой оболочки. Объявляются переменные с помощью символа `=` (обратите внимание на то, что перед и после `=` нет пробелов), а к их значениям обращаются с помощью символа `$`:

```
name="Петро Петрович"

echo $name # вывод значения

unset name # вывод переменной
```

Также можно создать локальную переменную внутри функции, которая будет доступна только в теле этой функции:

```
local локальная_переменная=значение
```

Переменные окружения — это переменные, которые доступны любым программам, запущенные с данной оболочки. Объявляются они так же как и локальные переменные, но с командой `export`:

```
export глобальная_переменная=значение
```

В bash есть много переменных окружения, которые достаточно часто встречаются в сценариях, например:

- HOME — путь к домашнему каталогу пользователя;
- PATH — список каталогов, в которых оболочка ищет исполняемые файлы;
- PWD — путь к рабочему каталогу;
- RANDOM — формирует целое случайное число;
- HOSTNAME — имя компьютера, на котором выполняется оболочка;

Переменные оболочки — это переменные, которые устанавливаются оболочкой и необходимы ей для корректной работы. Эти переменные имеют имена порядкового номера (`$ 1`, `$ 2`, `$ 3`, …) и содержат аргументы, которые передавались сценарию при запуске, как:

```
./some_script.sh VAL1 VAL2 # внутри сценария $1='VAL1', $2='VAL2'
```

Переменным можно присваивать значения по умолчанию следующим образом:

```
: ${VAR:='значение по умолчанию'} # Если переменная VAR пустая, присвоить “значение по умолчанию”
```

Массивы и списки

В bash также есть возможность работы с массивами. При работе с ними часто пользуются переменной окружения IFS — разделителя полей для входных строк (IFS — Input Field Separator). По умолчанию IFS равен символу пробела, но может быть изменен для разбиения строки на элементы массива, например, запятыми. Обратите внимание, что для формирования переменных оболочки, которые доступны через `$ 1`, `$ 2`и т.д., используется именно переменная IFS, то есть введенная после имени скрипта строка аргументов будет разделена именно первым символом, который хранится в этой переменной.

Объявить массив можно следующим образом:

```
files[0]=Яблоко

files[1]=Груша

echo ${files[*]} # напечатает элементы массива без учета

IFS echo ${files[@]} # напечатает элементы массива с IFS в качестве разделителя
```

Получить доступ к элементу массива можно с помощью срезов: `$ {arr: 0: 1}`.
Удалить первый элемент массива можно с помощью сдвига: `shift arr`.
Добавить в элементы в массив: `arr = ("$ {arr [@]}" "Item 1" "Item 2")`.

Проверка вхождения элемента в массив реализуется с помощью несколько более сложной конструкции:

```
if [[ ${arr[(r)some]} == some ]]; then # команды,если элемент входит

else # команды, если не входит

fi
```

В этом примере `arr` — некоторый массив, а `some` — это элемент, который мы проверяем на вхождение.

Результаты операций

Присвоить переменной результат работы команды или арифметических операций можно с помощью апострофов, или конструкции `$ (выражение)`:

```
now=data +%T

# или

now=$(data +%T)

echo now # 19:08:26
```

Арифметические операции необходимо помещать в двойные скобки:

```
foo=$(( ((10 + 5*3) - 7) / 2 ))
```

```
echo $foo #> 9
```

С помощью фигурных скобок можно генерировать строки произвольного вида, или учитывать различные варианты написания слова:

```
echo beg{i,a,u}n #> begin began begun
```

Стоит вспомнить и о строгости кавычек в bash: одинарные кавычки — строгие, двойные — нестрогие. Это означает, что при подстановке переменных в строку с двойными кавычками, интерпретатор подставит соответствующее значение переменной. Одинарные кавычки выведут строку так, как вы её написали. Пример:

```
echo "Домашняя директория: $HOME" #> Домашняя директория: /home/user
```

```
echo 'Домашняя директория: $HOME' #> Домашняя директория: $HOME
```

Потоки

Файл с которого происходит чтение, называют стандартным потоком ввода, а тот в который происходит запись, соответственно — стандартным потоком вывода. В bash есть три стандартных потока:

```
0 stdin ввод
```

```
1 stdout вивод
```

```
2 stderr поток ошибок
```

Для перенаправления потоков используют основные операторы:

- > — перенаправление потока вывода в файл (файл будет создан, или перезаписан)
- >> — дописать поток вывода в конец файла;
- < — перенаправляет данные из файла в поток ввода;
- <<< — чтение данных из строки, вместо всего содержимого файла (работает для bash 3+)
- 2> — перенаправляет поток ошибок в файл (файл будет создан, или перезаписан)
- 2>> — дописать ошибки в конец файла;

Каналы

Стандартные потоки можно перенаправить не только в файлы, но и на вход других сценариев. Соединение потока вывода одной программы с потоком ввода другой называют каналом или пайпом (pipe) . Ниже приведен простой конвейер из трех команд: `команда1` перенаправляет свой вывод на вход `команды2`, которая, в свою очередь, перенаправляет собственный вывод на вход `команды3`:

```
cmd1 | cmd2 | cmd3
```