Информационные технологии

Лекция №4

Скрипты оболочки bash

Скрипты оболочки

Скрипт оболочки – это файл, содержащий команды оболочки. Скрипты можно выполнять как обычные команды. Если при запуске такого файла заданы аргументы, на время выполнения скрипта они становятся позиционными параметрами.

В каждом файле, задающем скрипт первая строка имеет вид:

```
#!/bin/bash
```

Это означает, что когда мы запускаем скрипт на выполнение как обычную команду, /bin/bash будет выполнять ее для нас.

Скрипт - это просто текстовый файл

Обычно имеет имя типа <script_name>.sh

```
install_my_program.sh
print_diplom.sh
```

Скрипты оболочки

Помимо использования позиционных параметров, возможно использование и других переменных, определяемых внутри скрипта.
Например:

```
fruit = apple (определение);
echo $fruit (доступ);
```

Возможна конкатенация (соединение) строк:

```
$ fruit = apple
$ fruit = pine$fruit
$ echo $fruit
pineapple
$ fruite = apple
$ wine = ${fruite}jack
$ echo $wine
applejack
```

Скрипты оболочки

Ввод с клавиатуры.
Переменные можно считывать со стандартного ввода. Для этого используется команда read

```
echo -n Enter number of elements:
read x
```

Управляющие структуры

Оболочка bash поддерживает:

- операторы выбора `if … then … else` и `case`
- операторы организации циклов `for, while, until,`

благодаря чему bash превращается в мощный язык программирования.

Операторы if и test (или [ ])

Конструкция условного оператора в слегка упрощенном виде выглядит так:

```
if list1 then list2 else list3 fi
```

где `list1, list2 и list3` — это последовательности команд, разделенные запятыми и оканчивающиеся точкой с запятой или символом новой строки.
Кроме того, эти последовательности могут быть заключены в фигурные скобки: {list}.

Оператор if проверяет значение, возвращаемое командами из list1.
Если в этом списке несколько команд, то проверяется значение, возвращаемое последней командой списка.
Если это значение равно 0, то будут выполняться команды из list2; если это значение не нулевое, будут выполнены команды из list3.
Значение, возвращаемой таким составным оператором if, совпадает со значением, выдаваемым последней командой выполняемой последовательности.
Полный формат команды if имеет вид:

```
if list then list [ elif list then list ] ... [ else list ] fi
```

(квадратные скобки означают необязательность присутствия в операторе того, что в них содержится).

В качестве выражения, которое стоит сразу после if или elif, часто используется команда test, которая может обозначаться также квадратными скобками [ ].
Команда test выполняет вычисление некоторого выражения и возвращает значение 0, если выражение
истинно, и 1 в противном случае.
Выражение передается программе test как аргумент.
Вместо того, чтобы писать

```
test expression,
```

можно заключить выражение в квадратные скобки:

```
[ expression ].
```

Заметьте, что test и [— это два имени одной и той же программы, а не какое-то магическое преобразование, выполняемое оболочкой bash (только синтаксис [требует, чтобы была поставлена закрывающая скобка). Заметьте также, что вместо test в конструкции if может быть использована любая программа.

```
if [ -e textmode2.htm ] ; then
  ls textmode*
else
  pwd
fi
```

Проверка файловых атрибутов

`-a file`
Верно, если файл с именем file существует.

`-b file`
Верно, если file существует и является специальным файлом блочного устройства.

`-c file`
Верно, если file существует и является специальным файлом символьного устройства.

`-d file`
Верно, если file существует и является каталогом.

`-e file`
Верно, если файл с именем file существует.

`-f file`
Верно, если файл с именем file существует и является обычным файлом.

`-r file`
Верно, если файл с именем file существует и для него установлено право на чтение

`-s file`
Верно, если файл с именем file существует и его размер больше нуля.

`-w file`
Верно, если файл с именем file существует и для него установлено право на запись.

`-x file`
Верно, если файл с именем file существует и является исполняемым.

`file1 -nt file2`
Верно, если файл file1 имеет более позднее время модификации, чем file2.

`file1 -ot file2`
Верно, если файл file1 старше, чем file2

Оценка строк

`-z string`

Верно, если длина строки равна нулю.

`-n string`

Верно, если длина строки не равна нулю.

`string1 == string2`

Верно, если строки совпадают.

`string1 !== string2`

Верно, если строки не совпадают.

`string1 < string2`

Верно, если строка string1 лексикографически предшествует строке string2

`string1 > string2`

Верно, если строка string1 лексикографически стоит после строки string2

Арифметические сравнения

`arg1 OP arg2`

Здесь OP — это одна из операций арифметического сравнения: -eq (равно), -ne (не равно), -lt (меньше чем), -le (меньше или равно), -gt (больше), -ge (больше или равно). В качестве аргументов могут использоваться положительные или отрицательные целые.

`!(expression)`

Булевский оператор отрицания.

`expression1 -a expression2`

Булевский оператор AND (И). Верен, если верны оба выражения.

`expression1 -o expression2`

Булевский оператор OR (ИЛИ). Верен, если верно любое из двух выражений.

Оператор case

Формат оператора case:

```
case word in [ [(] pattern [ | pattern ] ... ) list ;; ] ...
esac
```

Команда case вначале производит раскрытие слова word, и пытается сопоставить результат с каждым из образцов pattern поочередно.
После нахождения первого совпадения дальнейшие проверки не производятся, выполняется список команд, стоящий после того образца, с которым обнаружено совпадение.
Значение, возвращаемое оператором, равно 0, если совпадений с образцами не обнаружено.
В противном случае возвращается значение, выдаваемое последней командой из соответствующего списка.

Каждая строка с условием должна завершаться правой (закрывающей) круглой скобкой).

Каждый блок команд, отрабатывающих по заданному условию, должен завершаться двумя символами точка-с-запятой ;;.

```
#!/bin/bash
echo -n " Какую оценку ты получил на экзамене?:
"
 read z
 case $z in
   5) echo Отлично!!!!
     ;;
   4) echo Хорошо !
     ;;
   3) echo удовлетворительно !
     ;;
   2) echo Неудовлетворительно!
     ;;
   *) echo !
     ;;
 esac
```

Оператор select
Оператор select позволяет организовать интерактивное взаимодействие с пользователем.

Он имеет следующий формат:

```
select name [ in word; ] do list ; done
```

Вначале из шаблона word формируется список слов, соответствующих шаблону.
Этот набор слов выводится в стандартный поток ошибок, причем каждое слово сопровождается порядковым номером.
Если шаблон word пропущен, таким же образом выводятся позиционные параметры.
После этого выдается стандартное приглашение bash, и оболочка ожидает ввода строки на стандартном вводе.
Если введенная строка содержит число, соответствующее одному из отображенных слов, то переменной name присваивается значение, равное этому слову.
Если введена пустая строка, то номера и соответствующие слова выводятся заново.
Если введено любое другое значение, переменной name присваивается нулевое значение.
Введенная пользователем строка запоминается в переменой REPLY.
Список команд list выполняется с выбранным значением переменной name.

```
#!/bin/bash
echo "Какую ОС Вы предпочитаете?"
select var in "Linux" "Windows" "Free BSD" "Other"; do
break
done
echo "Вы бы выбрали $var"
```

Если сохранить этот текст в файле, сделать файл исполняемым и запустить, на экран будет выдан следующий запрос:

```
Какую ОС Вы предпочитаете?
1) Linux
2) Windows
3) Free BSD
4) Other
#?
```

Нажмите любую из 4 предложенных цифр (1,2,3,4). Если вы, например, введете 1, то увидите собщение:

```
“Вы бы выбрали Linux”
```

Оператор for

Оператор for работает немного не так, как в обычных языках программирования.
Вместо того, чтобы организовывать увеличение или уменьшение на единицу значения некоторой переменной при каждом проходе цикла, он при каждом проходе цикла присваивает переменной очередное значение из заданного списка слов.
В целом конструкция выглядит примерно так:

```
for name in words do list done.
```

Правила построения списков команд (list) такие же, как и в операторе if.

Пример. Следующий скрипт создает файлы foo_1, foo_2 и foo_3:

```
for a in 1 2 3 ; do
  touch foo_$a
done
```

В общем случае оператор for имеет формат:

```
for name [ in word; ] do list ; done
```

Вначале производится раскрытие слова word в соответствии с правилами раскрытия выражений.
Затем переменной name поочередно присваиваются полученные значения, и каждый раз выполняется список команд list.
Если "in word" пропущено, то список команд list выполняется один раз для каждого позиционного параметра, который задан.
Программа seq воспринимает в качестве аргументов два числа и выдает последовательность всех чисел, расположенных между заданными.
С помощью этой команды можно заставить for в bash работать точно так же, как аналогичный оператор
работает в обычных языках программирования. Для этого достаточно записать цикл for следующим образом:

```
for a in $( seq 1 10 ) ; do
cat file_$a
done
```

Эта команда выводит на экран содержимое 10-ти файлов: "file_1", ..., "file_10".

Операторы while и until

Оператор while работает подобно if, только выполнение операторов из списка list2 циклически продолжается до тех пор, пока верно условие, и прерывается, если условие не верно.

Конструкция выглядит следующим образом:

```
while list1 do list2 done.
```

Оператор until аналогичен оператору while:

```
until list1 do list2 done.
```

Отличие заключается в том, что результат, возвращаемый при выполнении списка операторов list1, берется с отрицанием: list2 выполняется в том случае, если последняя команда в списке list1 возвращает ненулевой статус выхода.

Пример:

```
while [ -d mydirectory ] ; do
  ls -l mydirectory >> logfile
  echo -- SEPARATOR -- >> logfile
  sleep 60
done
```

Такая программа будет протоколировать содержание каталога "mydirectory" ежеминутно до тех пор, пока директория существует.

Функции

Оболочка bash позволяет пользователю создавать собственные функции.
Функции ведут себя и используются точно так же, как обычные команды оболочки, т. е. мы можем сами
создавать новые команды.
Функции конструируются следующим образом:

```
function name () { list }
```

Причем слово function не обязательно, name определяет имя функции, по которому к ней можно обращаться, а тело функции состоит из списка команд list, находящегося между { и }.

Этот список команд выполняется каждый раз, когда имя name задано как имя вызываемой команды.

Отметим, что функции могут задаваться рекурсивно, так что разрешено вызывать функцию, которую мы задаем, внутри нее самой.

Функции выполняются в контексте текущей оболочки: для интерпретации функции новый процесс не запускается (в отличие от выполнения скриптов оболочки).

Аргументы
Когда функция вызывается на выполнение, аргументы функции становятся позиционными параметрами (positional parameters) на время выполнения функции.
Они именуются как $n, где n — номер аргумента, к которому мы хотим получить доступ.
Нумерация аргументов начинается с 1, так что $1 — это первый аргумент.
Мы можем также получить все аргументы сразу с помощью $*, и число аргументов с помощью $#.
Позиционный параметр 0 не изменяется.

Если в теле функции встречается встроенная команда return, выполнение функции прерывается и управление передается команде, стоящей после вызова функции.
Когда выполнение функции завершается, позиционным параметрам и специальному параметру # возвращаются те значения, которые они имели до начала выполнения функции.

Локальные переменные (local)

Если мы хотим создать локальный параметр, можно использовать ключевое слово local. Синтаксис ее задания точно такой же, как и для обычных параметров, только определению предшествует ключевое слово local: local name=value.

Вот пример задания функции, реализующей упоминавшуюся выше команду seq:

```
seq()
{local I=$1;
while [ $2 != $I ]; do
{
  echo -n "$I ";
  I=$(( $I + 1 ))
};
done;
echo $2
}
```

Функция вычисления факториала fact

```
fact()
{
if [ $1 = 0 ]; then
  echo 1;
else
{
  echo $(( $1 * $( fact $(( $1 – 1 )) ) ))
};
fi
}
```

Это функция факториала, пример рекурсивной функции. Обратите внимание на арифметическое расширение и подстановку команд.