Лабораторная работа №

Скрипты. Их составление, редактирование и выполнение

Целью работы является изучение правил создания и редактирования командных файлов на языке shell-интерпретатора — так называемых скриптов.

В предыдущих лабораторных работах мы познакомились с основными командами, которые может интерпретировать и либо самостоятельно выполнить, либо передать на исполнение соответствующей утилите (программе) интерпретатор **bash**. Вместо интерпретатора **bash** может использоваться какой-нибудь иной. Для обобщения их принято называть shell-интерпретатором. Однако эти shell-интерпретаторы одновременно являются и языком программирования, который используется для написания командных файлов (shell-файлов). Часто командные файлы также называются скриптами и сценариями. Синтаксис языка, на котором составляются командные файлы, достаточно прост и прямолинеен. Короткие скрипты практически не нуждаются в отладке, и даже отладка больших скриптов отнимает весьма незначительное время.

Скрипт на языке bash должен начинаться со следующей строки

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

Эта строка однозначно указывает на интерпретатор **bash**, который должен будет исполнять скрипт.

Для обозначения переменных в shell-скриптах используются последовательность из букв, цифр и символа подчёркивания, причём переменные не могут начинаться с цифры. Присваивание значений переменным осуществляется с использованием знака **=** , например, **Var3 = '<'**. Для обращения к значению переменной перед именем ставится знак **$** .

Переменные можно разделить на следующие группы:

- простые переменные, значения которых может задавать пользователь, или они могут устанавливаться интерпретатором;

- позиционные переменные вида **$n**, где n — целое число;

- специальные переменные **#** , **?** , **-** , **!** , **$** устанавливаются интерпретатором и позволяют получить информацию о числе позиционных переменных, коде завершения последней команды, идентификационном номере текущего и фонового процессов, о текущих флагах интерпретатора shell.

Простые переменные. Shell присваивает значения переменным:

```
z=500
x=$z
echo $x
```

500

В приведённом примере переменной **x** было присвоено значение, которое получила переменная **z**.

Позиционные переменные. Переменные вида **$n** используются для идентификации позиций элементов в командной строке с помощью номеров, начиная с нуля. Например, в командной строке **cat text_1 text_2 … text_9** аргументы идентифицируются параметрами **$1** , **$2** , … , **$9**. Для имени собственно команды всегда используется **$0**. В данном случае **$0** — это **cat** , **$1** — **text_1** , **$2** — **text_2** и т. д. Для присваивания значений позиционным переменным используется команда **set** , например:

set arg_1 arg_2 … arg_9

здесь переменной **$1** присваивается значение аргумента **arg_1** , **$2** — **arg_2** и т. д.

Для получения информации обо всех аргументах (включая последний) используют метасимвол ***** . Например:

echo $*

arg_2 arg_3 … arg_10 arg_11 arg_12

С помощью позиционных переменных shell можно сохранить имя команды и её аргументы. При выполнении команды интерпретатор shell должен передать ей аргументы, порядок которых может регулироваться также с помощью позиционных переменных.

Специальные переменные. Переменные **- ? # $!** устанавливаются только shell. Они позволяют с помощью команды **echo** получить следующую информацию:

- - текущие флаги интерпретатора (установка флагов может быть изменена командой **set**);

- число аргументов, которое было сохранено интерпретатором при выполнении какой-либо команды;

? - код возврата последней выполняемой команды;

$ - числовой идентификатор текущего процесса PID;

! - PID последнего фонового процесса.

Арифметические операции

Команда **expr** (express ~ выражать) вычисляет выражение expression и записывает результат в стандартный вывод. Элементы выражения разделяются пробелами; символы, имеющие специальный смысл в командном языке, нужно экранировать. Строки, содержащие специальные символы, заключают в апострофы. Используя команду **expr**, можно выполнять сложение, вычитание, умножение, деление, взятие остатка, сопоставление символов и т. д. Пример. Сложение, вычитание:

b=190

a=`expr 200 - $b`

где ` - обратная кавычка.

Умножение *****, деление **/**, взятие остатка **%**:

```
d= `expr $a + 125 "*" 10`
c= `expr $d % 13`
```

Здесь знак умножения заключается в двойные кавычки, чтобы интерпретатор не воспринимал его как метасимвол. Во второй строке переменной **c** присваивается значение остатка от деления переменной **d** на **13**.

Сопоставление символов с указанием числа совпадающих символов:

```
concur= `expr "abcdefgh" : "abcde"`
echo $concur
```

ответ 5 .

Операция сопоставления обозначается двоеточием (**:**). Результат - переменная **concur**.

Подсчет числа символов в цепочках символов. Операция выполняется с использованием функции **length** в команде **expr:**

```
chain = "The program is written in Assembler"
str = `expr length "$chain"`
Echo $str
```

ответ 35 . Здесь результат подсчета обозначен переменной **str**.

Оператор if

Как и во многих языках программирования, в `bash` есть условный оператор, который имеет несколько конструкций. Они имеют формат, представленный ниже. Будьте внимательны: слова "`if`" и "`then`" должны находится на разных строках. Старайтесь выравнивать горизонтально всю конструкцию, включая заключительный "`fi`" и все "`else`". Это делает код намного удобнее для чтения и отладки. В дополнении к простой форме "`if ... else`" есть еще несколько других форм условных конструкций:

```
if      [ условие ]
then
        действие
fi
```

В приведенной записи 'действие' выполняется только если 'условие' верно, в противном случае скрипт продолжает выполнение инструкций со строки идущей за "fi".

```
if [ условие ]
then
        действие
elif [ условие_2 ]
then
        действие_2
elif [ условие_3 ]
then
.
.
.
else
        действие_x
fi
```

А эта конструкция последовательно проверяет условия и если они верны, то исполняет соответствующее действие. Если ни одно из условий не верно, то выполняется 'действие_*x*' стоящее после '`else`' (если оно есть). Потом продолжается исполнение команд идущих за этой конструкцией "`if then else`", если таковые есть.

Группирование команд

Группы команд или сложные команды могут формироваться с помощью специальных символов (метасимволов):

& - процесс выполняется в фоновом режиме, не дожидаясь окончания предыдущих процессов;

? - шаблон, распространяется только на один символ;

***** - шаблон, распространяется на все оставшиеся символы;

| - программный канал - стандартный вывод одного процесса является стандартным вводом другого;

> - переадресация вывода в файл;

< - переадресация ввода из файла;

; - если в списке команд команды отделяются друг от друга точкой с запятой, то они выполняются друг за другом;

&& - эта конструкция между командами означает, что последующая команда выполняется только при нормальном завершении предыдущей команды (код возврата 0);

| - последующая команда выполняется только, если не выполнилась предыдущая команда (код возврата 1);

( ) - группирование команд в скобки;

{ } - группирование команд с объединенным выводом;

[ ] - указание диапазона или явное перечисление (без запятых);

>> - добавление содержимого файла в конец другого файла.

Для того, чтобы скрипт **file_script** мог быть выполнен, ему необходимо присвоить соответствующие права. Например, чтобы его смог запустить любой пользователь, необходимо выполнить например такую команду:

```
chmod +rx file_script
```

Порядок выполнения работы

1. Напишите сценарий, который выводит на экран дату, время, список зарегистрировавшихся пользователей, имя компьютера и **uptime** системы, и сохраняет эту информацию в файле, имя которого соответствует Вашей фамилии.
2. Измените этот сценарий так, чтобы имя файла, в который будет записываться информация, нужно было задать во время выполнения программы.

3. Измените сценарий еще раз так, чтобы в тот же файл дописывалась информация о процессоре, на базе которого работает Ваш компьютер, и о версии операционной системы, на которой Вы выполняет данное задание. Эту информацию можно взять из файлов, расположенных в Вашей системе; вспомните — где они расположены?