**Лабораторная работа №5**

**Написание сценариев bash**

Возможность использования сценариев в **bash** предоставляет администратору **Linux** возможность легко автоматизировать рутинные задачи.

**Параметры и переменные**

Понятие *параметра* в оболочке **bash** подобно понятию переменной в обычных языках программирования. *Именем* (или *идентификатором*) параметра может быть слово, состоящее из алфавитных символов, цифр и знаков подчеркивания (только первый символ этого слова не может быть цифрой), а также число или один из следующих специальных символов: **\***, **@**, **#**,**?**, **-**, **$**, **!**, **0**, **\_**.

Параметры разделяются на три класса:

* + *позиционные параметры –* позволяют сценариям оболочки получать инфор­мацию, задаваемую в командной строке при их запуске (рассмотрены не будут)*,*
  + *специальные параметры –* их именами как раз и служат перечисленные только что специальные символы (ниже будут рассмотрены подробнее),
  + *переменные оболочки –* предоставляет возможность временного сохранения данных.

**Переменные оболочки**

Переменные оболочки размешаются в памяти автоматически при присвоении им значения.

Говорят, что переменная задана или установлена, если ей присвоено значение. Значением может быть и пустая строка. Чтобы вывести значение переменной, используют символ **$** перед ее именем. Так, команда

**echo**name

выдаст на экран слово name, а команда

**echo**$name

выдаст значение переменной name (если таковое, конечно, задано).

Значения переменным присваиваются с помощью знака «равно» (*без пробела*)

name**=**value

где **name** - имя переменной, a **value** - присваиваемое ей значение. Имя переменой может состоять только из цифр и букв и не может начинаться с цифры. Например,

>var**=**Hi!

>**echo** $var

Hi!

Если переменная должна содержать строку с пробелами, то строку следует экранировать с помощью одиночных или двойных кавычек, например,

>var='HiDude!'

Другой вариант: перед пробелами можно ставить экранирующий символ обратной косой черты «\», например,

>var=Hi\ \Dude

Для уничтожения переменной достаточно выполнить команду **unset**, указав в качестве ее аргумента имя переменной

>**unset**var

**Интерактивная установка значений переменных**

Для получения значения переменной непосредственно от пользователя предназначена команда **read**, которая читает вводимые данные из стандарт­ного потока ввода

>**echo-n** 'Введите значение переменной:'**;read**var

Введите значение переменной:Linux

>**echo** $var

Linux

В примере с клавиатуры должно быть введено значение переменной var. Опция **-n** команды **echo** использована здесь для отключения перевода строки после вывода строки приглашения.

|  |  |
| --- | --- |
| 256px-Information  256px-Information  256px-Information | Для разделения команд в одной строке используется «**;**». Если не поставить этот разделитель команд, то последующая команда может быть воспринята как аргумент предыдущей.  Таким образом, если написать в командной строке что-то вроде **command1 ;command2**, то оболочка вначале запустит на выполнение команду **command1**, дождется, пока ее выполнение завершится, после чего запустит **command2**, дождется ее завершения, после чего снова выведет приглашение командной строки, ожидая следующих действий пользователя. |

Можно одновременно ввести значения для нескольких переменных:

>**read**abc

Если в строке, поступающей из потока стандартного ввода для команды **read**, содержится больше значений, чем имеется аргументов у команды **read**, то все значения, для которых нет соответствующих переменных, в ви­де целой строки присваиваются последней переменной в списке.

Наоборот, если задано больше переменных, чем введено значений, то пере­менным, для которых не хватило значений, присваивается пустая строка.

**Переменные окружения**

Переменные оболочки доступны только в той оболочке, в которой они были описаны. Однако существует способ сделать переменную доступной для до­черних процессов этой оболочки. Для этого необходимо произвести опе­рацию преобразования переменной в *переменную окружения* с помощью команды **export**.

>var=Privet

>**export**var

>**bash**

>**echo** $var

Privet

>**exit**

Команда **export** записывает переменную в окружение оболочки, которое копируется в окружение дочерних процессов оболочки. Переменная var в примере получила значение privet, а затем преобразована в переменную окружения. Это сделало ее до­ступной в дочернем процессе - здесь в порожденной оболочке **bash**.

**Важнейшие переменные окружения**

***Окружение*** - это один из способов передачи информации процессов в сис­теме друг другу. Приведем наиболее важные переменные окружения и их содержимое:

**НОМЕ** - путь к домашнему каталогу пользователя;

**LOGNAME** и **USER** - имя пользователя;

**MAIL**- путь к почтовому ящику пользователя;

**PATH**- путь поиска исполняемых файлов, и значение этой переменной изменяется при каждом запуске программы **cd**.

**PS1** - вид приглашения оболочки;

**PWD**- имя текущего каталога;

**OLDPWD**- имя предыдущего каталога;

**HOSTNAME**- имя хоста;

**SHLVL**- номер оболочки и т.д.

Очень часто изменение значение какой-либо пе­ременной окружения приводит к изменению поведения запускаемой про­граммы. Так, например, если значение переменной **НОМЕ** установлено неверно, то команда **cd~** не будет возвращать в домашний каталог.

Значения всех переменных окружения могут быть получены с помощью команды **env**.

**Переменная окружения PS1**

**PS1** содержит вид приглашения, которое **bash** выводит, когда ожидает ввода очередной команды пользователем. При желании вид приглашения, а точнее значение переменной окружения **PS1** можно изменить. При этом можно использовать как любые символы, вводимые с клавиатуры, так и некоторое число специальных символов, которые при формировании строки приглашения декодируются в соответствии с таблицей (приведены избранные):

>**PS1**='\\'

Таблица 2. Формирование строки приглашения с помощью специальных символов

|  |  |
| --- | --- |
| **Символ** | **Его значение** |
| \d | дата в формате «день, месяц, число», например, СрдОкт17 |
| \u | имя пользователя |
| \t | текущее время в 24-часовом формате: hh:mm:ss |
| \\ | обратный слеш |
| \$ | символ **#**, если оболочка запущена суперпользователем, и символ **$**, если оболочка запущена обычным пользователем |
| \w | полное имя текущего рабочего каталога (начиная с корня) |
| \W | текущий рабочий каталог (без указания пути) |

**Вычисление арифметических выражений**

В командной строке можно вычислять выражения, заключенные либо в квадратные скобки, либо в двойные круглые скобки. Перед скобками должен стоять символ **$**, а результаты выражений можно переда­вать как аргумент какой-либо команде или назначать переменной. Например,

>**echo**$((10\*20))

200

Команда **echo** вы­водит в стандартный поток вывода строку, указанную в качестве аргумента. В следующем примере продемонстрировано назначение результата арифметиче­ского выражения переменной:

>a=5

>b=$(($a/2))

>**echo** $b

2

Здесь переменной a присвоено значение **5**. Далее производится присвоение пе­ременной b результата деления значения a на **2**. В силу того, что в **bash** возможно вычисление только целочисленных выражений, то в результате значением стало b число **2**.

**Сценарии оболочки**

*Сценарий оболочки* представляет собой текстовый файл, содержащий про­грамму, состоящую из системных и встроенных команд. Они предназначены для автоматизации выполнения задач, чаще всего связанных с администри­рованием.

Оболочка последовательно интерпретирует и выполняет команды, заданные в сценарии. Эти же команды могут быть выполнены простым последова­тельным вызовом их в командной строке оболочки.

Для файлов сценариев оболочки **bash** принято устанавливать расширение **.sh**.

Сценарии оболочки могут быть исполнены двумя различными путями:

* + *если файл сценария доступен для чтения*, то имя файла сценария указывается в качестве аргумента командной строки при явном запуске исполняемого файла оболочки, например,

>**bash** myscr.sh

* + *если файл сценария доступен для чтения и исполнения*, то сценарий можно запустить так же, как и обычные системные команды, например,

>**chmod** 555 myscrl.sh

> ~/myscrl.sh

В данном примере файлу сценария myscrl.sh установили права доступа на чтение и исполнение для всех групп пользователей, а затем запустили, указав путь к нему (при этом файл сценария находился в домашней папке пользователя).

**Создание сценария**

Сценарий создается с помощью команды **cat**. Например, создадим сценарий, содержащий всего одну строку и вычисляет произведение **10**⬝**20**

>**cat>** myscrl.sh

**echo**$((10\*20))

После окончания ввода кода сценария, надо нажать комбинацию клавиш **Ctrl+D**.

Та же команда, но без символа перенаправление вывода «**>**», позволяет просмотреть код сценария:

>**cat** myscrl.sh

echo$((10\*20))

**Специальные параметры**

В **bash** используются следующие специальные параметры:

**$\*** - строка, составленная из значений всех аргументов командной строки;

**$@** - содержит строку, составленную из значений всех аргументов ко­мандной строки, разделенных пробелами (аналогично, но не тождествен­но **$\***);

**$#** - количество аргументов командной строки;

**$?** - код возврата предыдущей команды;

**$$** - **PID** оболочки

**Проверка заданных условий**

Команда **test** позволяет проверить заданные условия. Виды проверок, вы­полняемых командой:

* + проверка файлов на предмет выполнения заданных условий;
  + сравнение файлов;
  + проверка установки опций оболочки;
  + сравнение строк;
  + сравнение целых чисел.

Если тест выполнен успешно, команда **test** передает *нулевой* код возврата. В противном случае - *ненулевой*. Так, используя опцию **-е** команды **test**, можно проверить существование файла

>**test -е** /etc/passwd

>**echo** $?

0

>**test -е**not\_existent\_file

>**echo** $?

1

В данном примере для того, чтобы узнать, какое значение вернула команда **test**, использовался специальный параметр**$?**. В первом случае команда **test** вернула нулевой код возврата, поскольку файл /etc/passwd существует. Во втором случае команда вернула код ошибки.

Команда **test** в сценариях обычно вызывается в другой форме, совершенно эквивалентной показанной ранее. В этой форме вместо строки **test** указы­вают квадратные скобки и условие в них: **[** *условие***]**.

Задача из предыду­щего примера, может быть решена иным способом

>**[-е** /etc/passwd**]**

>**echo** $?

0

Команда**[-е**/etc/passwd**]** эквивалентна команде **test-е**/etc/passwd.

Сравнение целых чисел производится с помощью следующих опций команды **test**:

**-eq** − равенство; **-nе** − неравенство;

**-lt** − меньше; **-1е** − меньше или равно;

**-gt** − больше; **-nе** − больше или равно

Рассмотрим пример сравнения чисел:

>**[** 1 **-lt** 2**]**

>**echo** $?

0

>**[** 1 **-eq** 2**]**

>**echo** $?

1

**Условное исполнение команд**

Оболочка **bash** поддерживает операторы выбора, а также операторы организации циклов, благодаря чему она превращается в мощный язык программирования.

**Операторы *if*и *test*(или*[]).***

Конструкция условного оператора в общем выглядит так:

**if [** *условие****1* ]**

**then***команды****1***

**elif [** *условие****2* ]**

**then***команды****2***

**...**

**else***команды****N***

**fi**

где *команды****1***, *команды****2*** и *команды****N*** — это последовательности команд, разделенные запятыми и оканчивающиеся точкой с запятой или символом новой строки.

Например, создадим простой сценарий

>**cat>**myif.sh

**read**a**; read** b

**if [** $a **–gt** $b **]**

**then**

**echo ‘**a больше b’

**else**

**echo ‘**a меньше b’

**fi**

Эту же задачу можно записать короче

**read**a**; read** b

**if [** $a **–gt** $b **]; then echo ‘**a больше b’; **else echo ‘**a меньше b’; **fi**

**Оператор *case.***

Формат оператора таков:

**case***слово***in**

*образец****1)***

*команды****1***

***;;***

*образец****2)***

*команды****2***

**;;**

**...**

*образец****N)***

*команды****N***

**;;**

**esac**

Например, создадим простой сценарий

>**cat>** mycase.sh

name1=’Ivan’; name2=’Egor’; **read** name

**case**name **in**

$name1**)**

**echo ‘**Hi, Ivan!’

;;

$name2**)**

**echo ‘**Hi, Egor!’

;;

**esac**

**Циклы**

В случаях, когда надо организовать последовательное выполнение одного и того же набора инструкций, удобно использовать программные циклы.

**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***имя***in***список*

**do**

*команды*

**done**

Переменной *имя*поочередно присваиваются полученные значения, и каждый раз выполняются *команды*.

Например, рассмотрим сценарий, который поочередно выводит права доступа к трем указанным каталогам

>**cat>** myfor.sh

**for**DIR **in** /etc /tmp /var

**do**

**echo** –n“Права доступа к $DIR”

**ls –ld**$DIR

**done**

**Операторы *while*и *until*** работает подобно **if**, только выполнение *команд* циклически продолжается до тех пор, пока верно условие, и прерывается, если условие не верно. Конструкция его выглядит так:

**while[** *условие* **]**

**do**

*команды*

**done**

Например,

>**cat>** mywhile.sh

i**=**1

**while[** $i**-le** 5 **]; do**

**echo-n** $i ‘ ‘ ; **sleep** 1

i**=**$((i+1))

**done**

Результат выполнения сценария:

1 2 3 4 5

Цикл **while** работает до тех пор, пока команда, указанная в качестве ее ар­гумента, возвращает успешный код завершения. Наоборот, **until** работает, пока команда-аргумент заканчивается неудачей.

**Практические задания**

1. Получите, используя переменные окружения, имена текущего каталога и домашнего каталога.
2. Создайте новую переменную **NEWVAR** со значением **1982** и проверьте, до­ступна ли она в порожденной оболочке. Экспортируйте переменную **NEWVAR** и снова проверьте, доступна ли она в порожденной оболочке.
3. Получите списки переменных окружения с их зна­чениями. Изменить приглашение **PS1** (любое отличное от «->»).
4. В сценарии **sl.sh** определите переменную **V** и выведите ее значение.
5. Перепишите сценарий **sl.sh** таким образом, чтобы из него вызывался сце­нарий **s2.sh**, который и печатал бы значение переменной **V**.
6. Напишите сценарий оболочки, считывающий значения трех переменных и вы­водящий их значения в стандартный поток вывода. Проверьте его работу, вво­дя два, три и четыре значения.
7. Создать сценарий, который при вводе пользователем:

1 – создает файл и копирует его в другую папку

2 – создает файл и перемещает его в другую папку

3 – просто создает каталог

1. Создать сценарий, который находит максимальное из трех, введенных пользователем, чисел и записывает их файл. В сценарии должны быть пояснения для ввода пользователя (приглашения).
2. Создать сценарий, который требует ввести какое-либо имя и, если оно равно имеющемуся программе имени, то выводится сообщение «Привет, Имя!». Если пользователь вводит новое имя, то выводится сообщение «Ты ни Имя1, ни Имя2, ни Имя3. Но все равно, привет, Имя!».
3. Создать сценарий, который вычисляет факториал числа.