Работа с файловой системой при помощи bash-скриптов

Язык BASH — *Bourne-Again SHell* (что может переводится как «перерожденный шел», или «Снова шел Борна(создатель *sh*)»), самый популярный командный интерпретатор в юниксоподобных системах, в особенности в *GNU/Linux*. По существу, *bash* представляет интерпретатор соответствующего языка программирования (ЯП). Каждый набор команд для *bash* представляет собой т.н. скрипт. Как правило скрипит используются в задачах автоматизации работы операционной системы. Например, может быть скрипт.

```
echo ls
echo who
```

Данный скрипт позволяет вывести содержимое текущего каталога и текущего пользователя одновременно.

Для выбора интерпретатора *bash* следует воспользоваться следующей записью, называемой *shebang*.

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

Для проверки интерпретатора по умолчанию можно воспользоваться следующей командой.

```
echo $SHELL
```

Данная команда выводит содержимое системной переменной *SHELL*.

Следует отметить, что знак # означает для *bash* комментарий. Поэтому, данная сточка не будет распознана самим интерпретатором.

Следует также отметить, что скрипты могут содержать и команды на иных языках программирования, например *#!/usr/bin/python* для ЯП *Python*. Если *shebang* указан, то bash выполняет скрипт в «подоболочке». Если указания *shebang* нет, то скрипт будет запущен в основной оболочке *Linux*. Для выхода из подоболочки следует использовать команду *«exit 0»*. Рассмотрим следующий пример команд.

```
#!/bin/bash
dir
dir > 1.txt
```

Команды *bash* могут быть организованы в виде отдельного скриптового файла, имеющего разрешение «*.sh*». Каждый такой файл предпочтительно начинать с указания компилятора.

Скрипт-файл – это обычный текстовый файл, содержащий последовательность команд *bash*, для которого установлены права на выполнение. Пример скрипта, выводящего содержимое текущего каталога на консоль и в файл:

```
#!/bin/bash
# look into the current directory
dir
dir > 1.txt # write the current content into the file
```

exit 0

Также следует отметить возможность работы скриптов с потоками ввода и вывода.

Для этого могут быть использованы следующие команды:

- > - перенаправить поток вывода в файл (файл будет создан или перезаписан)
- >> - дописать поток вывода в конец файла
- Перенаправление потока ввода (прием данных):
- < - файл в поток ввода (файл будет источником данных)
- <<< - чтение данных из строки вместо содержимого файла (для bash 3 и выше)
- Перенаправление вывода ошибок:
- 2> - перенаправить поток ошибок в файл
- 2>> - дописать ошибки в файл (файл будет создан или перезаписан)

Стандартные потоки:

- 0, stdin, ввод;
- 1, stdout, вывод;
- 2, stderr, поток ошибок.

Задание 1. Создание скрипта.

1. Создайте скриптовый файл *hello.sh (>hello.sh).*
2. Создайте в файле скрипт выводящий на экран время, дату и текущих пользователей
   #!/bin/bash
   # time and data
   date
   who
   exit 0
3. Сделайте файл исполняемых (*chmod +x ./hello.*sh)
4. Проверьте работоспособность скрипта.
5. Создайте файл с расширением «*.py*» (ЯП Питон) и сделайте скрипт вывода строки ‘*hello world’* в консоли.

При запуске скрипта можно использовать аргументы. Аргумент — это всё, что вы помещаете за командой сценария.

Задание 2. Создание скрипта с аргументами.

1. Создайте скриптовый файл *echo_args.sh*.
2. В файле задайте вывод на экран двух аргументов, например
   echo the first arg is $1.
3. Протестируйте скрипт.

На самом деле знак «*$*» перед цифрами обозначает вызов переменной. Могут быть неименованные переменные, как это проделано выше и именованные. Для именования переменных могут быть использованы несколько способов. Некоторые из них приведены ниже.

```
A=121
A="121"
let A=121
let "A=A+1"
```

Задание 3. Использование именованных переменных.

1. Перепишите задание 2 и использованием именованных переменных. То есть сначала присвойте значения введенных аргументов переменным, а затем уже выведите их обратно.

Команда *let,* описанная выше также позволяет производить арифметические операции над переменными.

Задание 4. Операции с переменными.

1. Создайте скрипт с двумя входными аргументами, значения которых присвойте переменным *a* и *b*.
2. Изучите особенности работы команды *let* на следующих примерах.

```
let "c = a + b"
echo "a+b= $c"
let "c = a / b"
echo "a/b= $c"
let "c <<= 2"
echo "c после сдвига на 2 разряда влево: $c"
let "c = a % b"
echo "$a / $b. остаток: $c "
```

Так как скрипты в *bash* могут быть сложными последовательностями команд, то в них предусмотрено использование функций. Например, так, как это показано ниже.

```
double_echo(){
   echo $1
   echo $2
}
test1 = $1
test2 = $2
double_echo() test1 test2
```

Задание 5. Операции с функциями.

1. Создайте скрипт с вычисления функции вида *(a-b)/(a+b)* для входных аргументов с использованием функции и ее вызова.

Для обработки больших массивов переменных в *bash* могут быть использованы циклы. Например, цикл «*for*», заданный как

for ...[counter] in ...[range] do ... done.

Следует также отметить, что условия в for могут быть заданы как описанным выше способом, так и следующим

for ((count=FIRST; count<LAST; count++)).

Помимо цикла «*for*» в ЯП *bash* доступны циклы *while* и *until*, однако особенности этих циклов требуют непосредственной проверки условий.

Задание 6. Использование циклов для работы с переменными и аргументами.

1. Добавьте в созданный в задание 2 скрипт счетчик аргументов.
   echo you have entered $# arguments
2. Добавьте вывод аргументов циклом.
   for i in $@
   do
   echo $i
   done
3. Протестируйте скрипт.

Примечание:

- *$#* — это счетчик, который показывает, сколько аргументов было использовано при запуске скрипта.
- *$@* — список всех аргументов, которые использовались при запуске скрипта.

Также доступны следующие операции:

- *$0* —имя выполняемой команды. Для скрипта – это путь, указанный при его вызове. Для функции – имя оболочки.
- *$** — строка аргументов, заключённая в двойные кавычки.
- *$?* — статус завершения последней выполненной команды.
- *$;* — номер процесса, соответствующего текущей оболочке.
- *$!* — номер процесса, соответствующий команде, запущенной в фоновом режиме.

По мимо того, чтобы обрабатывать только аргументы, введенные при вызове скрипта – можно обрабатывать аргументы, введенные по запросу во время исполнения скрипта. Например, такой запрос может быть выполнен по условию, для его считывания используется команда «*read*». При вызове команды необходимо записать ее результат в переменную. Затем можно присвоить первому вызванному аргументы значение данной переменной. Условие может быть проверено при помощи конструкции

if ... then ... elif (else if) ... then ... else ... fi.

Для проверки условия может быть использована команда «*test*». Например, вызов «*test*» с ключом *«-z»*, для проверки отсутствия.

Команда «*test*» – не является встроенной командой в *bash,* она является утилитой *linux*. Команда «*test*» может быть заменена встроенный для *bash* синоним *«[ ]*». Например:

if [-z $1]; if ["$a" -eq "$b"] или if ["$a" == "$b"]).

Также в *bach* доступны команда *«[[ ]]»* — расширенная версия *«[ ]»*, внутри которой могут быть использованы *«|| (*или*), & (*и*)»*. Например,

[[condition1 && condition2]] или if [["$a" < "$b"]]

– что эквивалентно записи:

if ["$a" \< "$b"].

Обратите внимание! Символ "<" необходимо экранировать внутри *«[ ]»*.

Также в *bach* доступны команда *«(( ))»* — представляющая собой математическое сравнение. Например,

(("$a" <= "$b")).

Также доступны следующие записи:

- *[[$a == z*]] # истина, если $a начинается с символа "z" (сравнение по шаблону)*
- *[[$a == "z*"]] # истина, если $a равна z**
- *[$a == z*] # имеют место подстановка имен файлов и разбиение на слова*
- *["$a" == "z*"] # истина, если $a равна z**

Условия сравнения:

- Файлы:
  - *-e* – проверить что файл существует *(-f, -d);*
  - *-f* – файл существует *(!-f* - не существует);
  - *-d* – каталог существует;
  - *-s* – файл существует и не пустой;
  - *-r* – файл существует и доступен на чтение;
  - *-w* – файл существует и доступен на запись;
  - *-x* - файл существует и доступен на выполнение;
  - *-h* - символическая ссылка.
- Строки:
  - *-z* – пустая строка;
  - *-n* – не пустая строка;
  - *==* – равно (*!=* - не равно);
- Числа:
  - *-eq* – равно;
  - *-ne* – не равно;
  - *-lt* – меньше;
  - *-le* – меньше или равно;
  - *-gt* – больше;
  - *-ge* - больше или равно.

Задание 7. Проверка условий

1. Изучите работу следующего скрипта.

```
#!/bin/bash
if test -z “$1”
then
   echo enter a text
   read TEXT
else
   TEXT=$1
fi
echo you have entered the text $TEXT
exit 0
```

2. Перепишите скрипт с использованием команды *«[ ]»*.
3. Добавьте в скрипт проверку переменной на равенство

Также следует отметить, что условия могут быть проверены не только в «*if*», но и в циклах «*while*» и «*untile*» а также в операторе «*case*». Например:

```
#!/bin/bash
count=0
while [ $count -lt 10 ]
do
   (( count++ ))
   echo $count
done
```

Задание 8. Проверка условий

1. Изучите следующий скрипт.

```
#!/bin/bash
echo "Выберите редатор для запуска:"
echo "1. Запуск программы nano"
echo "2. Запуск программы vi"
echo "3. Запуск программы emacs"
echo "4. Выход"
read doing # читаем в переменную $doing со стандартного ввода
case $doing in
   1) /usr/bin/nano # если $doing содержит 1, то запустить nano
   ;;
   2) /usr/bin/vi # если $doing содержит 2, то запустить vi
   ;;
   3) /usr/bin/emacs # если $doing содержит 3, то запустить emacs
   ;;
   4) exit 0
   ;;
   *)# по умполчению:
   echo "Введено неправильное действие"
esac #окончание оператора case.
```

2. Перепишите скрипт с использованием

```
if ... then ... else if ... then.
```

Задание 9. Циклы с условием для работы с файловой системой.

1. Изучите следующий код.

```
for f in $HOME/*; do
if[[-d $f]]; then
        echo $f; fi
done
```

2. Напишите скрипт, выводящий список только непустых файлов на экран.
3. Напишите скрипт, сканирующий домашнюю директорию и все вложенные папки.

Следует отметить, что анализ файлов может также быть осуществлен в рамках скрипта стандартными средствами *linux,* например так, как это показано ниже.

```
ls -la | grep ".sh*" | sort > sorted_list.txt
```

Вывод команды «*ls -la*» передается команде *grep*, которая отбирает все строки, в которых встретится слово «*.sh*», и передает их команде сортировке *sort*, которая пишет результат в файл *sorted_list.txt*.

Задание 10. Работа с файловой системой стандартными средствами Linux.

1. Напишите скрипт, выводящий в заданный файл список файлов домашней директории с использованием команды *grep*.

Задание 11. Циклы с условием для работы с файловой системой.

1. Изучите следующий код.

```
#!/bin/bash
dir="$HOME/.archive/" # directory for deleted files
if [ -d $dir ]; then # check the directory .archive/
file="$1"
null=""
else mkdir $dir | chmod 700 $dir # if there is no, create
fi
if [ $file == $null ]; then # error, if not specified file #
echo -e "/!\  No file.. Usage: $0 filename ;-) | archive directory - $dir  /!\ "
exit 1
fi
mv $file $dir$(date "+%H.%d.%m").$file # move file to .archive/
```

2. Напишите скрипт, копирующий все фалы текущего каталога в новый каталога архив.

Задание 11

1. Написать скрипт, выводящий в файл (имя файла задаётся пользователем в качестве первого аргумента командной строки) имена всех файлов с заданным расширением (третий аргумент командной строки) из заданного каталога (имя каталога задаётся пользователем в качестве второго аргумента командной строки).

Задание 12

1. Написать программу «*1.c*», выводящую на экран фразу "*HELLO World*". Компилировать полученную программу компилятором, например,
   gcc: gcc 1.c –o 1.exe.
2. Написать скрипт, компилирующий и запускающий программу «*1.exe*» (имя исходного файла и «*exe*»- файла результата задаётся пользователем в качестве аргументов командной строки). В случае ошибок при компиляции вывести на консоль сообщение об ошибках и не запускать программу на выполнение.