Занятие 8.

Тема: Командные оболочки. Занятие второе.

Вид занятия: лекция, практическое занятие.

Учебные вопросы:

1. Программирование для Bash.

Время: 90 минут

Литература:

1. Системная справочная служба Linux Man
2. Тейнсли Д. “Linux и UNIX: программирование в shell. Руководство разработчика”. Перевод с английского – К.: Издательская группа BHV, 2001. - 464 с.

Ход занятия.

Сценарии командных оболочек играют в Linux огромную роль. Кроме того, что каждый пользователь стремится по максимуму автоматизировать выполняемую им работу с помощью сценариев shell, так и сценарии инициализации системы и множества приложений есть ничто иное как просто сценарии shell. Как и на позапрошлом занятии, мы с Вами будем рассматривать программирование в shell на базе Bourne Shell.

1. Любой сценарий для bash начинается с указания в первой строке редактируемого файла комбинации:

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

Эта последовательность указывает на программу, которую следут использовать для обработки данного сценария – в нашем случае командную оболочку bash.

Язык bash довольно мощный. Важную часть представляют собой объявления переменных. Принять называть переменные буквами верхнего регистра, например:

```
#!/bin/bash
TERM=vt100
CONTER=0
```

Для извлечения значения переменной используются следующий синтаксис:

```
ALFA=$BETA+$GAMMA
```

Значок $ означает, что нужно извлечь значение переменной BETA, а не использовать строку “BETA”. Переменной можно присвоить вывод какой-либо команды. При этом переменная станет массивом, разделителем записей в котом будут служить символы пробела, табуляции и перевода строки. Для этого выполняемую команду необходимо заключить в обратные апострофы и присвоить переменной:

```
FILES=`/bin/ls -a /home/student`
```

Вывод на экран значений переменных, или просто фраз производится с помощью оператора echo:

```
echo $ALFA это строки BETA и GAMMA с плюсом посредине
echo Done.
```

При выводе выводимую информацию можно заключать в одинарные(') и двойные(“)кавычки. Одинарные полностью выводят текст, написанный внутри них, а в двойных указанные переменные заменяются их значениями:

```
[gserg@WebMedia serial]$ echo '$USER'
$USER
[gserg@WebMedia serial]$ echo “$USER”
student
[gserg@WebMedia serial]$
```

В качестве комментариев применяется знак #. Он может быть использован как в начале, так и в середине строки:

```
#Временный каталог
TMP='/tmp'     #используем кавычки
```

Как часть сценария может быть использована любая доступная команда операционной системы:

```
#очистим экран
```

```
clear
```

Bash поддерживает несколько операций сравнения. В случае, если условие верно, то оператор сравнения возвращает 0 (true). Проверить возвращаемое значение можно с помощью специального значения $?. Перечислю их:

```
#проверка кода возврата последней команды
[ -d abcd ]
echo $?
#проверка прав доступа к файлу:
[ -d abcd ] #является ли файл abcd каталогом?
[ -f abcd ] #является ли файл abcd обычным файлом?
[ -L abcd ] #является ли файл abcd символической ссылкой?
[ -r abcd ] #есть ли доступ на чтение к файлу abcd?
[ -w abcd ] #есть ли доступ на запись к файлу abcd?
[ -s abcd ] #файл abcd имеет ненулевой размер (он не пуст)?
[ -u abcd ] #имеет ли файл abcd установленный бит SUID?
[ -x abcd ] #является ли файл abcd исполняемым?
#проверка строк
[ -z $STRING ] #пуста ли строка STRING?
[ -n $STRING ] #строка STRING не пуста?
[ $STRING = $STRING1 ] #равны ли строки STRING и STRING1?
[ $STRING != $STRING1 ] #строки STRING и STRING1 не равны?
#проверка чисел
#при проверке чисел в условии их обязательно необходимо
#заключить в двойные кавычки
[ $DIGIT -eq $DIGIT1 ] #равны ли числа DIGIT и DIGIT1?
[ $DIGIT -ne $DIGIT1 ] #числа DIGIT и DIGIT1 не равны?
[ $DIGIT -gt $DIGIT1 ] #число DIGIT больше DIGIT1?
[ $DIGIT -lt $DIGIT1 ] #число DIGIT меньше DIGIT1?
[ $DIGIT -ge $DIGIT1 ] #число DIGIT больше или равно DIGIT1?
[ $DIGIT -le $DIGIT1 ] #число DIGIT меньше или равно DIGIT1?
```

Сценарию на bash могут быть переданы параметры командной строки. Значения их можно получить с помощью переменных $0, $1, $2 .. $n-1, $n. При этом $0 – имя файла сценария, $1 – первый параметр, $2 – второй, ... $n – последний.

Но операторы сравнения без операторов ветвления бесполезны. Простейшим оператором ветвления в языке сценариев bash является оператор if-then-else-fi. Полная структура для оператора выглядит следующим образом:

```
if <оператор сравнения1>; then
  <действия 1>
elif <оператор сравнения2>; then
   <действия 2>
else
 <действие 3>
fi
```

Оператор if позволяет упрощать конструкцию. Разделы elif и else не являются обязательными. Для правильной работы достаточно использования только конструкции if-fi.

Для примера приведу скрипт:

```
#!/bin/bash
if [ -f /etc/passwd ]; then
  echo “/etc/passwd – обычный файл”
else
 echo “/etc/passwd – не обычный файл”
fi
exit
```

В этом сценарии проверяется, обычный ли файл /etc/passwd и выводится

соответствующее сообщение на экран.

Естественно мы не сможем обойтись и без таких конструкций, как циклы.

Простейшим циклом является цикл перебора for.

```
for <ЭЛЕМЕНТ> in $<СПИСОК>
 do
  <действия>
 done
```

Циклу передается переменная со списком параметров, разделенных пробелом, знаком перевода строки или табуляции. Дальше цикл выполняет действия для каждого элемента из списка, например:

```
#!/bin/bash
echo “Поиск каталогов в $HOME”
echo
#перейдем в домашний каталог
pushd $HOME
#получим список файлов
LIST = `ls -a $HOME`
#организуем цикл для каждого из
#значений в переменной LIST
for ITEM in $LIST
 do
  #если вхождение – каталог, выведем его на экран
  if [ -d $ITEM ]; then
   echo “$ITEM”
  fi
done
#вернемся в прежнюю директорию
popd
#выход
exit
```

Однако цикл перебора подходит нам не во всех случаях. Во многих случаях нам необходим цикл с условием. Это цикл while.

```
while <условие>
 do
  <действия>
 done
```

Цикл while выполняется до тех пор, пока истинно <условие>. Попробую показать на примере организации цикла со счетчиком. Для вычислений я буду использовать команду expr, позволяющую выполнять математические операции с переменными bash:

```
#!/bin/bash
#определяем наличие первого параметра
#командной строки
if [ -z $1 ]; then
 echo “Использование: $0 количество_проверок”
 exit
fi
#очистим экран
clear
#определим переменную счетчика цикла
COUNTER=0
#организуем цикл на $1 итераций
while [ $COUNTER -lt $1 ]
 do
  echo “Загрузка системы (за $1 секунд)”
  echo   echo “секунда $COUNTER”
```

```
  echo “-- загрузка системы --”
  #команда определения средней загрузки
  /usr/bin/uptime
  echo “-- место на дисках --”
  #определение свободного места на винчестере
  df -h
  echo “----------------”
  echo
  #пауза 1 секунду
  /bin/sleep 1
  #очистим экран
  clear
  #приращение переменной цикла
  COUNTER=`expr $COUNTER + 1`
 done
#выход
exit
```

Организация вывода имени программы не очень удачно выполнено в данном случае, потому что будет выводится не только имя запущенной программы, а и путь, набранный нами в командной строке. Исправить это нам поможет директива basename, выделяющая из пути имя файла. Вот как будет выглядеть фрагмент приведенного выше сценария с ее применением:

```
#!/bin/bash
#определяем наличие первого параметра
#командной строки
if [ -z $1 ]; then
 echo Использование: `basename $0` количество_проверок
 exit
fi
...
```