LAB10. SHELL SCRIPTS (II)- PROGRAMMING (SELECTION CONSTRUCTS)

Instructor :

Murad Njoum

Objectives:

After completing this lab, the student should be able to:

- Include **programming selection constructs** in shell scripts.
- Use **the if/else** statement to manipulate **integer and string** values as well as file properties.
- Apply the **case statement** programming construct for efficient selections as well as **creating menus**

Unix commands return a value (**success = zero and failure or error = non-zero**) to the shell. This value is stored in the **variable (?)** as follows

CONT..

Run the command:

ls –al

Now run the command:

echo $?

What result did you get? 0 ***Why?***

0 : NO ERRORS, SUCCESS COMMAND EXECUTION.

Now run the command:

cp

followed by the command:

echo $?

What result did you get? 1 ***Why?***

(1) NONE ZERO :ERRORS FAILURE COMMAND EXECUTION.

EXAMPLE:

Write the following script (***checkcommand***):

```
#!/bin/bash

if $1

then

            echo command $1 succeed

        else

             echo command $1 failed

        fi
:wq
```

- Re-Write the following
- ```
  if $1 > out 2> err
  then
        echo Command $1 succeed
  else
        echo Command $1 failed
  fi
  :wq
  ```

- **checkcommand date**
  What result did you get? **SUCCESS** ***Why?*** **COMMAND DATE IS SUCCESS CORRECTLY AND VALUE OF ? IS ZERO (RETURN TO IF STATEMENT)**

- Now run the command:
  checkcommand mv

What result did you get? **FAILURE** ***Why?*** **COMMAND MV ISN'T SUCCESS AND VALUE OF ? IS NON-ZERO**


  ```

CONT..

- This is **one way** to use the if/else structure.
- Still, many scripts do not check commands, but rather check for **variable values, file properties, and number of arguments**.
- To do that we need to use one of two syntaxes:

- ***if test condition (e.g. if test $# -eq 2)***

  or

  if [condition] (e.g. if [$# -eq 2])

In Bash, we have the following conditional statements:

if..then..fi statement (Simple If)
if..then..else..fi statement (If-Else)
if..elif..else..fi statement (Else If ladder)
if..then..else..if..then..fi..fi..(Nested if)

```
if [ conditional expression ]
then
     statement1
     statement2
     …..
fi
```

```
if [ conditional
expression ]
then    statement1
        statement2
else
        statement3
        statement4
fi
```

if..elif..else..fi statement (Else If ladder)

```
if [ conditional expression1 ]
then
      statement1
      statement2
elif [ conditional expression2 ]
     then
      statement3
      statement4
else
      statement5

fi
```

if..then..else..if..then..fi..fi..(Nested if)

```
if [ conditional expression1 ]
then
    statement1
    statement2
else
if [ conditional expression2 ]
    then
statement3
    fi
fi
```

CONT..

To compare **integer values**, we use the following relational operators:

-lt (less than),

-gt (greater than)

-eq (equal)

-le (less than or equal)

-ge (greater than or equal),

-ne (not equal).

INTEGER VALUES:

- ***Write a script called sum, that accepts integer number and print the sum***

```
#! /bin/bash

echo "Enter two numbers"

read num1 num2

sum=$(expr $num1 + $num2)

#without spaces:print
concatof two numbers 10+10

echo "The sum is = $sum"
```

X=5

Y=10

expr $X + $Y

Or you can use

echo (($X + $Y))

```
#! /bin/bash

echo "Enter two numbers"

read num1 num2

sum=$(($num1+$num2))

echo "The sum is = $sum"
```

```
echo enter two numbers
read num1
read num2
sums=$(( num1+num2 ))
echo sum=$sums
```

```
echo sum=$(($1+$2))
```

Let us rewrite the delete script we wrote in the previous lab to check for the correct number of arguments as follows:

```
vi delete
if [ $# -eq 1 ]
then
     rm $1
    echo $1 has been deleted
exit 0    #This line return 0 from the script  (success)
else
     echo Usage: delete filename
     exit 1
  fi
:wq
```

CON..

- Now try the above script as follows:

- ***delete myfile*** (assuming myfile exists and is a regular file)
  Then run the command:
  echo $?
  Did it work? YES.
  What is the value of variable (?) ? 0

- Now try it as follows:
  delete
  Then run the command:
  echo $?
  What happened? DISPLAY ERROR MESSAGE
- ***Why?*** NO ARGUMENT.
  What is the value of variable (?) ? 1

To check file values we use the following operators:

-f filename (to check if file exists and is of type file)

-d filename (to check if directory exists and is of type directory)

-x,-r,-w (to check if a user has execute, read, or write permissions on a file)

Let us rewrite our delete script to include those:

```
#!/bin/bash

if [ $# -ne 1 ]
 then
        echo usage: delete filename
        exit  1
else
       if [ -f  $1 ]
       then
          rm –f  $1
          echo $1 has been deleted
          exit  0
 elif  [ -d  $1 ]
       then
        rm –rf  $1
        echo $1 directory has been deleted
       exit 0
 else
     echo  $1: No such file or directory
     exit  2
   fi
fi
```

Now create a file and a directory using the following commands:

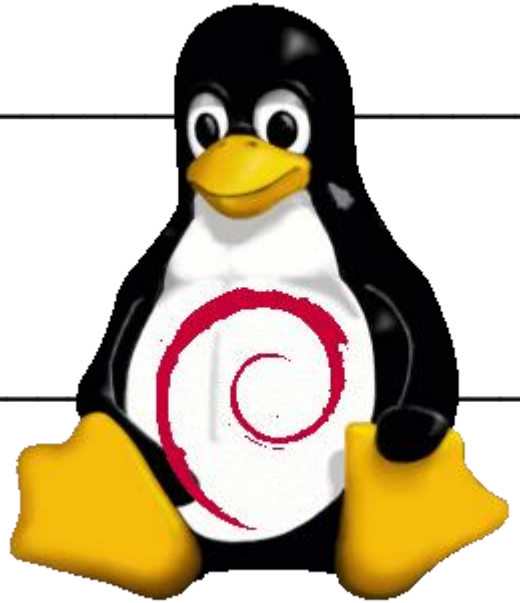
touch myfile; mkdir mydir

No try the updated delete script in the following ways:

delete

What happened? _____.

delete myfile (myfile exists and is a file)



What happened?_____.

delete mydir (mydir exists and is a directory)

What happened?_____.

delete wrong (wrong does not exist)

What happened? _____.

QUESTION:

Now rewrite the copy script to act as follows:

copy

Usage: copy src dest

copy myfile newfile

File myfile is copied to file newfile

copy mydir newdir

Directory mydir is copied to newdir

copy wrong good

wrong: No such file or directory

```
#! /bin/bash
if [ $# -ne 2 ]
   then
  echo Usage: copy file from source to destination
  exit 1
else
  if [ -f $1 ]
     then
     cp $1 $2
     echo $1 has bee copied to $2
     exit 0
 elif [ -d $1 ]
  then
   cp -r $1 $2
    echo $1 directory has been copied to $2 directory
   exit 0

 else
  echo No such file or directory has been copied
  exit 2
 fi
fi
```

Sometimes our scripts need to check string values. To do that we need to use the following operators:
= (equal), != (not equal) ,-n (none null string) -z (zero string (null))
Let us try some of those. let us write a script to check the value of the name entered by the user:
vi checkname

```
if [ $# -ne 1 ]
then echo Usage: checkname name
     exit 1
  else
  if [ "$1" = "ahmad" ]
     then echo $1:Hello
    exit 0
  else
        echo $1:Goodbye
      exit 0
  fi
fi
```

Try it as follows:
checkname ahmad
What happened?_____.
checkname suha
What happened?_____.
checkname
What happened?_____.

Try Update to following code:

```
#! /bin/bash
if [ -z "$1" ]
 then
 echo usage: cannot be empty, enter string
 exit 1
else
  if [ "$1" = "ahmad" ]
   then
   echo hello

else
   echo Goodbye
   exit 0
   fi
 fi
```

QUESTION

Write a script called checkusername which works as follows:

checkusername
No names were entered

checkusername u1112233
u1112233 = Ahmad Hamdan

checkusername u11
u11 = No such user name

checkusername bash
bash = No such user name

```
#! /bin/bash
if [ -z "$1" ]
  then
  echo No names were entered
  exit 1
 fi

var=$(grep ^$1 /etc/passwd |cut -d : -f1)

if [ "$var" = "$1" ]
     then
     name=$(grep ^$1 /etc/passwd |cut -d : -f5 |tr '_' ' ')
     echo $1=$name
     exit 0
 else
          echo $1=No Such user name
          exit 2

  fi
```

Case Statement

We can also use a case statement (similar to switch in c) to check for values. The syntax is as follows:

```
case value in

   pattern1) statements
                    ;;    # ;; is the break statement

   pattern2) statements

                    ;;

   *) statements     # * stand for default case

esac
```

The patterns may be strings or parts of strings. Those can include the * wild card, the (|) OR operator, as well as ranges (e.g [0-9] or [a-f]) as follows:

s* | S* | good)

means any pattern that starts with s or S or the word good.

[A-Z]*[0-5])

means any pattern with any size that starts with a capital letter and ends with a number between 0 and 5

[a-z][0-9][0-9][0-9] | [0-9][A-Z][A-Z][A-Z][a-f])

means the accepted pattern must consist of exactly four characters the first is a small letter and the next three are numbers or the pattern must be exactly five characters with the first being a number followed by three capital letters and then one small letter between a and f.

Case statements are usually used for handling menus and menu options. Let us try a simple example that uses a menu to call different scripts (modular programming):

Create three different scripts called ***script1***, ***script2***, and ***script3*** respectively. In each script put one line to display which script you're in (e.g in script1 put the line "echo this is script 1").

Now create a script called ***mainscript*** that displays the following menu:

Please select your choice (1-4):

1 - Run script1

2- Run script2

3- Run script3

4- Exit main script

```
#! /bin/bash
echo "Please Select your choice (1-4):
1-Run script1
2-Run Script2
3-Run Script3
4-Exit main script"
read choice
case $choice in
  1) ./script1
  ;;
  2) ./script2
  ;;
  3) ./script3
  ;;
  4) exit
esac
```

echo hi from script 1

echo hi from script 2

echo hi from script 3

THE END