COMP311
Linux OS Laboratory
Lab10:Programming (Selection Constructs)

By

Alaa' Omar

Objectives

1

Include programming selection constructs in shell scripts.

2

Use the if/else statement to manipulate integer and string values as well as file properties.

3

Apply the case statement programming construct for efficient selections as well as creating menus.

Script Selection Constructs

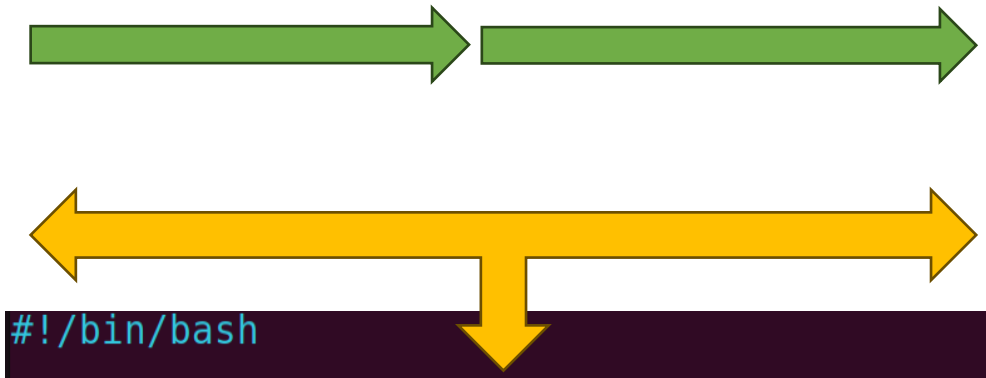
Unix commands return a value (success = **zero** and failure or error = **non-zero**) to the shell. This value is stored in the variable **(?)** as follows:

Run the command:
$ls –al
Now run the command:
echo $?
What result did you get? _____ Why? _____.
Now run the command:
$cp
followed by the command:
echo $?
What result did you get? _____ Why? _____.

Script Selection Constructs

bash

```
#!/bin/bash
if $1
then
    echo "Command $1 succeed"
else
    echo "Command $1 failed"
fi
```



```
#!/bin/bash


$1>out 2>error
exit_code=$?

if [ $exit_code -eq 0 ]
    then
        echo command $1 executed successfully

else
        echo command $1 faild
fi
```

bash

```
vi checkcommand
if $1 > out 2> err
then
    echo "Command $1 succeeded"
else
    echo "Command $1 failed"
fi
:wq
```

If statement syntax

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])

If statement syntax

The general syntax for the if/else statement is as follows:

if-else

```
#!/bin/bash

if condition
    then
        statments
else
        statments
fi
```

If condition; then
statement
else
statement
fi

If statement syntax

```
if condition
then
  statements
elif condition
then
  statements
else
statements
fi
```

Conditions

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).

Conditions

summ_v1

```
#!/bin/bash
if [ $1 -gt $2 ]
then
        echo $1 is greater than $2
else
        echo $1 is smaller than $2
fi
sum=$(($1 + $2))
echo The summation of the $1 and $2 is $sum
```

summ_v2

```
#!/bin/bash
if test $1 -gt $2
then
        echo $1 is greater than $2

else
        echo $1 is smaller than $2
fi
sum=$(expr $1 + $2)
echo The summation of the $1 and $2 is $sum
```

- You can use either test keyword or [ ] in the if condition
- You can use **expr** or **()** for integer numbers

Delete as an example

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

delete - lab 9

```
vi delete
rm $1
echo $1 has been deleted

:wq
```

delete

```
#!/bin/bash
vi delete
if [ $# -eq 1 ]
  then
    rm $1
    echo $1 is deleted
    exit 0 # This line returns 0 from the script (success)
else
  echo Usage: delete filename
  exit 1
fi
:wq
```

Writing scripting (practice)

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?_____.
What is the value of variable (?) ?_____
Now try it as follows:
delete
Then run the command:
echo $?
What happened? _____ Why?_____.
What is the value of variable (?) ?_____

Check for file values

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 executed, read, or write permissions on a file)

Modify the delete script

If statement syntax

```
if [number of arguments is not equal to 1]
then
    display "Usage: delete filename"
    exit with status code 1
else
    if [$1 exists and is a file]
    then
        remove $1
        display "File $1 is deleted"
        exit with status code 0
    else if [$1 exists and is a directory]
    then
        remove directory $1 recursively
        display "Directory $1 is deleted"
        exit with status code 0
    else
        display "$1: No such file or directory"
        exit with status code 2
    end if
end if
```

```
vi delete

if [ $# -ne 1 ]
then
    echo "Usage: delete filename"
    exit 1
else
    if [ -f $1 ] # $1 exists and is a file name
    then
        rm $1
        echo "File $1 is deleted"
        exit 0
    elif [ -d $1 ]
    then
        rm -r $1 # $1 exists and is a directory
        echo "Directory $1 is deleted"
        exit 0
    else
        echo "$1: No such file or directory"
        exit 2
    fi
fi

:wq
```

Try the delete script

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?_____.

Copy script, example two

```
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
```

```bash
#!/bin/bash
vi copy
if [ $# -ne 2 ]
then
    echo "Usage: copy src dest"
    exit 1
elif [ -f $1 ]
then
    cp $1 $2
    echo "File $1 is copied to file $2"
    exit 0
elif [ -d $1 ]
then
    cp -r $1 $2
    echo "Directory $1 is copied to $2"
    exit 0
else
    echo "$1: No such file or directory"
    exit 2
fi
```

Try the new copy script and make sure it works as above?

Did it work correctly?

_____.

Equal Operators:

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))

Operators example checkname script

```bash
#!/bin/bash

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?_____.

Example two, Checkusername script

Write a script called **checkusername** which works as follows:

- **checkusername**
  No names were entered
- **checkusername** u1112233
  u1112233 = Ahmad Hamdan
- **checkusername** u11
  u11 = No such username
- **checkusername** bash
  bash = No such username

Solution1: use grep with –w option to ensure the exact match of username.

```
#!/bin/bash
if test -z $1
   then
        echo Usage: checkusername name
        exit 1
fi
full_name=$(grep -w $1 pass | cut  -d: -f5| tr '_' ' ')
if test -z "$full_name"
then
         echo $1=No such username
         exit 2
else
        echo $1=$full_name
        exit 0
fi
```

Solution2: find username first, then search for full name

checkusername

```
#!/bin/bash
if test -z $1
   then
        echo No names were entered
        exit 1
fi
uname=$(grep $1 /etc/passwd | cut -d: -f1)
if test $uname = $1
then
        full_name=$(grep $1 /etc/passwd | cut  -d: -f5| tr '_' ' ')
        echo $1=$full_name
        exit 0
else
        echo $1=No such user name
fi
```

Case statment

The bash case statement is commonly utilized to streamline intricate conditionals in situations where you need to handle various choices. By opting for the case statement over nested if statements, you can enhance the readability and maintainability of your bash scripts.

The case statement in Bash operates on a similar principle to the switch statement in JavaScript or C. However, it differs in that, unlike the C switch statement, the Bash case statement halts the search for pattern matches once it finds a match and executes the associated statements. This ensures that only the actions corresponding to the first matching pattern are performed, making the code more efficient and predictable.

CASE statment syntax

```
case EXPRESSION in

  PATTERN_1)
    STATEMENTS
    ;;

  PATTERN_2)
    STATEMENTS
    ;;

  PATTERN_N)
    STATEMENTS
    ;;

  *)
    STATEMENTS
    ;;
esac
```

Case Statement

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 Statement

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

checkusername

```
#!/bin/bash
echo "Please select your choice (1-4):"
echo "1. Run script1"
echo "2. Run script2"
echo "3. run script3"
echo "4. Exit main script"

read choice
case $choice in
        1) ./script1 ;;
        2) ./script2 ;;
        3) ./script3 ;;
        4) exit 0    ;;
        *) echo invalid choice;  exit 2 ;;
esac
```

The End