COMP311
Linux OS Laboratory
Lab11:Bash Programming (Looping Constructs)

By

Alaa' Omar

Objectives

1

Include programming looping constructs in shell scripts.

2

Understand and use the while, until, and for loops constructs.

3

Learn how to make for loops more efficient by using command outputs as lists.

Case statements usage examples

Example 1: Simple menu selection

```bash
#!/bin/bash
echo "Please select an option:"
echo "1. Option 1"
echo "2. Option 2"
echo "3. Option 3"
echo "4. Quit"

read choice

case $choice in
    1) echo "You selected Option 1"
        # Add your code here for Option 1
        ;;
    2) echo "You selected Option 2"
        # Add your code here for Option 2
        ;;
    3) echo "You selected Option 3"
        # Add your code here for Option 3
        ;;
    4) echo "Goodbye!"
        exit 0
        ;;
    *) echo "Invalid option"
        ;;
esac
```

Case statements usage examples

Example 3: Handel file extension

```
#!/bin/bash

filename="myfile.txt"

case "$filename" in
    *.txt)
        echo "Text file"
        ;;
    *.jpg|*.png)
        echo "Image file"
        ;;
    *.sh)
        echo "Shell script"
        ;;
    *)
        echo "Unknown file type"
        ;;
esac
```

Case statements usage examples

Example 2: Handel user input

```
#!/bin/bash

echo "Enter your age:"
read age

case $age in
    [0-9])
        echo "You are a child"
        ;;
    1[0-9]|2[0-9])
        echo "You are a teenager"
        ;;
    [3-9][0-9])
        echo "You are an adult"
        ;;
    *)
        echo "Invalid age"
        ;;
esac
```

Case statements usage examples

Example 4 match pattern

```
#!/bin/bash

input="a123"

case $input in
    [a-z][0-9][0-9][0-9]|[0-9][A-Z][A-Z][A-Z][a-f])
        echo "Pattern matched: $input"
        ;;
    *)
        echo "No pattern matched"
        ;;
esac
```

```
#!/bin/bash

input="A123"

case $input in
    [A-Z]*[0-5])
        echo "Pattern matched: $input"
        ;;
    *)
        echo "No pattern matched"
        ;;
esac
```

Flow control constructs

- In Lab 10 we have studied the following **selection constructs**:
  - **If/else**
  - **Case**
- In this lab we will study the following **loop constructs**:
  - While
  - for
  - Until

bash supports the following flow control constructs:

if/else

Execute a list of statements if a certain condition is/is not true

for

Execute a list of statements a fixed number of times

while

Execute a list of statements repeatedly *while* a certain condition holds true

until

Execute a list of statements repeatedly *until* a certain condition holds true

case

Execute one of several lists of statements depending on the value of a variable

Source: Learning bash shell book

while and until

These two flow control constructs bash provides are while and until. These are similar; they both allow a section of code to be run repetitively while (or until) a certain condition becomes true.

While syntax

```
while condition
  do
    statement(s)
  done
```

until syntax

```
until false
  do
    statements
  done
```

listarguments (while example)

```
vi listarguments

while [ $# -ne 0 ]
do
  echo $1
  shift
done

:wq
```

while and until

These two flow control constructs bash provides are while and until. These are similar; they both allow a section of code to be run repetitively while (or until) a certain condition becomes true.

While syntax

```
while condition
  do
    statement(s)
  done
```

until syntax

```
until false
  do
    statements
  done
```

listarguments (while example)

```
#!/bin/bash
while test $# -ne 0
do
        echo '------------------'
        echo $*
        echo '------------------'
        echo $1
        shift
done
```

Practice

Rewrite the delete script we wrote in the last lab such that it works as follows:

```
delete file1 wrong dir1 file2
File file1 is deleted
wrong: No such file or directory
Directory dir1 is deleted
File file2 is deleted
```

delete_file

```
#!/bin/bash
if test $# -eq 0; then
echo Usage: delete_file filename
exit 1
else
while test $# -ne 0; do
if      test -f $1; then
        rm $1
        echo File $1 has been removed
elif test -d $1; then
        rm -rf $1
        echo Directory $1 has been removed
else
        echo $1: No such file or directory
fi
shift
done
fi
```

Practice

Rewrite the delete script we wrote in the last lab such that it works as follows:

```
delete file1 wrong dir1 file2
File file1 is deleted
wrong: No such file or directory
Directory dir1 is deleted
File file2 is deleted
```

delete

```
#!/bin/bash
while test $# -ne 0
do
        ./delete_file $1
        shift
done
```

delete_file

```
#!/bin/bash
if test $# -ne 1; then
echo Usage: delete_file filename
exit 1
elif test -f $1; then
        rm $1
        echo File $1 has been removed
        exit 0
elif test -d $1; then
        rm -rf $1
        echo Directory $1 has been removed
        exit 0
else
        echo $1: No such file or directory
        exit 2
fi
```

Practice (delete multiple files)

Rewrite the delete script we wrote in the last lab such that it works as follows:

```
delete file1 wrong dir1 file2
File file1 is deleted
wrong: No such file or directory
Directory dir1 is deleted
File file2 is deleted
```

delete

```
#!/bin/bash
while test $# -ne 0
do
        ./delete_file $1
        shift
done
```

delete_file

```
#!/bin/bash
if test $# -ne 1; then
echo Usage: delete_file filename
exit 1
elif test -f $1; then
        rm $1
        echo File $1 has been removed
        exit 0
elif test -d $1; then
        rm -rf $1
        echo Directory $1 has been removed
        exit 0
else
        echo $1: No such file or directory
        exit 2
fi
```

Example 2 (findahmad script)

findahmad

```
#!/bin/bash
echo Enter name
read name
while [ "$name" != "ahmad" ]
do
  echo $name: wrong name. Try again.
  echo Enter name
  read name
done
```

Now modify the ***checkusername*** script from the previous lab such that it is called ***checkusernames*** instead and works as follows:

Example 2 (findahmad script)

Now modify the ***checkusername*** script from the previous lab such that it is called ***checkusernames*** instead and works as follows:

```
checkusernames
        Enter user name to check or  word "enough" to stop
        u1112345
        Enter user name to check or  word "enough" to stop
        u11
        Enter user name to check or  word "enough" to stop
        u1123456
        Enter user name to check or  word "enough" to stop
        enough

        u1112345  =  Salem  Hamdi
        u11  =  No such user name
        u1123456  =  Sabah  Khaled
```

Example 2 (findahmad script)

```
checkusernames
        Enter user name to check or  word "enough" to stop
        u1112345
        Enter user name to check or  word "enough" to stop
        u11
        Enter user name to check or  word "enough" to stop
        u1123456
        Enter user name to check or  word "enough" to stop
        enough

        u1112345  =  Salem  Hamdi
        u11  =  No such user name
        u1123456  =  Sabah  Khaled
```

```bash
#!/bin/bash

echo "Enter username to check or word enough to stop"
read uname
while test  "$uname" != "enough"; do
if test -n "$uname"; then
full_name=$(grep -w $uname pass | cut  -d: -f5| tr '_' ' ')
if test -z "$full_name"
then
        echo $uname=No such username
else
        echo $uname=$full_name
fi
fi
echo "Enter username to check or word 'enough' to stop"
read uname
done
```

Until loop

until loop

The until loop is similar to the while loop but stops when the condition becomes true.

```
#!/bin/bash
until false
do
statements
done
```

Break and Continue Statements

The programmer can use break and continue statements inside shell script loops which mean the same as they do in the C language:

- break - exit the loop immediately.
- continue – stop running the current cycle but go back and check the condition.

In addition, they can use break and continue followed by a number to specify how many loop levels they want them to work for. For example:
break 2
Will exit out of two nested loops if they exist.

For loop

```
#!/bin/bash
for item in list of items
do
statement(s)
done
```

For loop

```
#!/bin/bash
for item in list of items
do
statement(s)
done
```

```
#!/bin/bash

for name in $*; do

        echo $name
done
```

For loop

Rewrite the delete script we wrote at the beginning of this lab such that it uses a for loop instead of a while loop. Did it work? _____.

```
#!/bin/bash

for file in $*;do

./delete_file $file

done
```

For loop

Using a for loop, write a script called comp311 that lists the full names of all the users that are registered in the comp311 course.

```
#!/bin/bash
for line in $(cat /etc/passwd); do
        full_name=$(echo $line | cut -d: -f5| tr '_' ' ' ); echo $full_name
done
```

For loop

Now rewrite the script comp311 such that it will display only the names of the users that are currently logged in to the system. (hint: use the output of the who command)
Answer:

```
#!/bin/bash

for uname in $(who | tr -s ' ' | cut -d' ' -f1); do
    echo $uname
    full_name=$(grep $uname /etc/passwd | cut -d: -f5 | tr '_' ' ')
    echo $full_name
done
```

For loop

```
#!/bin/bash
for file in *; do
        echo $file
done
```

For loop

Write a script called filetypes that uses a for loop to type the name and type (file, dir, or unknown) for each file in a given directory as follows:
Assume that I use the script in the following way:
filetypes /etc
then the script should display the names of all the files under directory /etc and the type of each of those files:

```
#! /bin/bash
for file in $1/*
do
if [ -f $file ]
then
echo $file : is File type
elif [ -d $file ]
then
echo $file :is Directory type
else
echo $file : is Unknown type
fi
done
```

For loop

Write a script called ***mywhich*** that simulates the which command. You are not allowed to use the which command in your script. (***hint: use the for loop and the sed command***).

The End