LECTURE 4: REPETITIVE TASKS, LOOP CONTROL AND PASSING PARAMETERS TO SHELL SCRIPTS

4.0 Introduction

A loop is a powerful programming tool that enables a person to execute a set of commands repeatedly.

The following loops available to shell programmers are:-

- The while loop
- The for loop
- The until loop
- The select loop

4.1 The while loop

Example 1

Create a bash file with the name, 'while_use.sh', to know the use of while loop. In the example, while loop will iterate for 10 times. The value of count variable will increment by 1 in each step. When the value of count variable will 10 then the while loop will terminate.

```
#!/bin/bash
valid=true
count=1
while [ $valid ]
do
echo $count
if [ $count -eq 10 ];
then
break
fi
((count++))
done
```

RESULT

```
1
2
3
4
5
```

6
7
8
9
10

4.2 The for loop

The basic for loop declaration is shown in the following example. Create a file named 'for_use.sh' and add the following script using for loop. Here, for loop will iterate for 10 times and print all values of the variable, counter in single line.

Example 2

```
#!/bin/bash
for (( counter=10; counter>0; counter-- ))
do
echo -n "$counter "
done
printf "\n"
```

RESULT

10 9 8 7 6 5 4 3 2 1 0

4.3 Shell Loop Control in UNIX/LINUX

We have the basic knowledge of how loops are created and how various tasks can be carried out with these loops. There are statements used to control shell loops. They are :-

- The break statement
- The continue statement

4.4 Infinite Loop

As a long as a condition set to be met to end a loop is not met, such loop will continue running and this continues to infinity. Such loops are referred to as infinite loops.

Example 3 is shown below.

Example 3

```
#!/bin/bash
z = 20
until [$z –lt 15 ]
do
        echo $z
        a = 'expr $z + 1'
```

```
done
```

It is obvious that the value of z will never be less than 15 and this will make a continuous loop to infinity.

4.5 The Break Statement

The break statement is used to terminate the execution of the entire loop, after the execution of lines of codes before the break statement.

Syntax

To exit from a loop, the command below is used:-

```
break
```

To exit from a nested loop, the command below is used:-

```
break n
```

In this case, the n specifies the nth closing loop to exit from.

For example, we have a script in example 4 below that terminates as soon as z becomes 7.

Example 4

```
#!/bin/bash
z = 0
while [ $z –lt 10 ]
do
        echo $z
        if [$z –eq 7]
        then
                break
        fi
        z=’expr $z +1’
done
```

RESULT

```
0
1
2
3
4
5
6
7
```

Example 5

This is an example of a 'nested for loop'. The script breaks out of both loops if var1 equals 2 and var2 equals 0.

```
#!/bin/bash
for var1 in 1 2 3
do
        for var2 in 0 5
        do
                if  [$var1 –eq 2 –a $var2 –eq 0]
                then
                  break 2
                else
                        echo " $var1 $var2"
                fi
                done
done
```

RESULT

```
1 0
1 5
```

4.6 The Continue Statement

The continue statement is similar to the break command, but in this case it only causes the current iteration of the loop to exit rather than exit the entire loop. The use of this statement becomes important when an error has occurred but there is need to continue the execution of the next loop's iteration.

Syntax

continue

As with the break statement, an integer argument can be given to the continue command to skip commands from nested loops. Where the integer indicates the nth enclosing loop to continue from.

continue n

Example 6

```
#!/bin/bash
NUMS= "1 2 3 4 5 6 7"
for NUM in $NUMS
do
```

```
        Q = ‘expr $NUM / 2’
        if  [ $Q –eq 0 ]
        then
                echo “Number is an even number!!”
                continue
        fi
        echo “Found odd number”
done
```

RESULT
Found odd number
Number is an even number!!
Found odd number
Number is an even number!!
Found odd number
Number is an even number!!
Found odd number

4.7 Pass Return Value from a Function

Bash function can pass both numeric and string values. How you can pass a string value from the function is shown in example 7. Create a file named, ‘function_return.sh’ and add the following code. The function, greeting() returns a string value into the variable, val which prints later by combining with other string.

Example 7

```
#!/bin/bash
function greeting() {
str="Hello, $name"
echo $str
}
echo "Enter your name"
read name
val=$(greeting)
echo "Return value of the function is $val"
```

RESULT

Enter your name
Daniel
Return value of the function is Hello, Daniel

4.8 Passing Parameters to Shell Scripts

Bash can't declare function parameter or arguments at the time of function declaration. But you can use parameters in function by using other variable. If two values are passed at the time of function calling then $1 and $2 variable are used for reading the values. Create a file named 'area_a_parameter.sh' and add the following code. Here, the function, 'Rectangle_Area' will calculate the area of a rectangle based on the parameter values.

Example 8

```
#!/bin/bash
Rectangle_Area() {
area=$(($1 * $2))
echo "Area is : $area"
}
Rectangle_Area 20 15
```

RESULT

```
Area is : 300
```