

SHELL Scripting

Part 3

Loops

Operator	Description	Example
-eq	Checks if the value of two operands are equal or not; if yes, then the condition becomes true.	[\$a -eq \$b] is not true.
-ne	Checks if the value of two operands are equal or not; if values are not equal, then the condition becomes true.	[\$a -ne \$b] is true.
-gt	Checks if the value of left operand is greater than the value of right operand; if yes, then the condition becomes true.	[\$a -gt \$b] is not true.
-lt	Checks if the value of left operand is less than the value of right operand; if yes, then the condition becomes true.	[\$a -lt \$b] is true.
-ge	Checks if the value of left operand is greater than or equal to the value of right operand; if yes, then the condition becomes true.	[\$a -ge \$b] is not true.
-le	Checks if the value of left operand is less than or equal to the value of right operand; if yes, then the condition becomes true.	[\$a -le \$b] is true.

Conditional Statements

- There are total 5 conditional statements which can be used in bash programming
- if statement
- if-else statement
- if..elif..else..fi statement (Else If ladder)
- if..then..else..if..then..fi..fi..(Nested if)

if statement

- This block will process if specified condition is true.

Syntax:

```
if [ expression ]  
then  
    statement  
fi
```

if-else statement

- If specified condition is not true in if part then else part will be execute.

Syntax:

```
if [ expression ]  
then  
    statement1  
else  
    statement2  
fi
```

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

- To use multiple conditions in one if-else block, then elif keyword is used in shell. If expression1 is true then it executes statement 1 and 2, and this process continues. If none of the condition is true then it processes else part.

Syntax:

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

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

Nested if-else block can be used when, one condition is satisfies then it again checks another condition. In the syntax, if expression1 is false then it processes else part, and again expression2 will be check.

Syntax:

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

Looping Statements

- There are total 3 looping statements which can be used in bash programming
 - while statement
 - for statement
 - until statement
- To alter the flow of loop statements, two commands are used they are,
 - break
 - continue

while statement

Here command is evaluated and based on the result loop will executed, if command raise to false then loop will be terminated

Syntax:

```
while [condition]
do
    statement 1
done
```

```
#!/bin/bash
a=0
# -lt is less than operator

#Iterate the loop until a less than 10
while [ $a -lt 10 ]
do
    # Print the values
    echo $a

    # increment the value
    a=`expr $a + 1`
done
```

```
4 a=1
5 while (($a < 10))
6 do
7     # Print the values
8     echo $a
9
10    # increment the value
11    #a=`expr $a + 1`
12    ((a++))
13 done
14
```

1
2
3
4
5
6
7
8
9



saima@ubuntu: ~

saima@ubuntu:~\$ chmod +x while.sh

saima@ubuntu:~\$./while.sh

0

1

2

3

4

5

6

7

8

9

saima@ubuntu:~\$



for statement

- The for loop operate on lists of items. It repeats a set of commands for every item in a list.

Here var is the name of a variable and word1 to wordN are sequences of characters separated by spaces (words). Each time the for loop executes, the value of the variable var is set to the next word in the list of words, word1 to wordN.

Syntax

- For [condition]
- do
- statements
- done



file.sh x while.sh x for.sh x

```
#!/bin/bash
#Start of for loop
for a in 1 2 3 4 5 6 7 8 9 10
do
    # if a is equal to 5 break the loop
    if [ $a == 5 ]
    then
        break
    fi
    # Print the value
    echo "Iteration no $a"
done
```

saima@ubuntu: ~

saima@ubuntu:~\$ chmod +x for.sh

saima@ubuntu:~\$./for.sh

Iteration no 1

Iteration no 2


Iteration no 3

Iteration no 4

saima@ubuntu:~\$

For loop similar to C syntax

```
5  
6  echo "Hello World";  
7  for ((i=1;i<5;i+=1));  
8  do  
9  echo "0.${i}" ;  
10 done  
11
```



```
Hello World  
0.1  
0.2  
0.3  
0.4
```

until statement

- The until loop is executed as many as times the condition/command evaluates to false. The loop terminates when the condition/command becomes true.

Syntax :

- Until [condition]
- do
- statements
- done

Example Until loop

```
#!/bin/bash
echo "until loop"
i=10
until [ $i == 1 ]
do
    echo "$i is not equal to 1";
    i=$((i-1))
done
echo "i value is $i"
echo "loop terminated"
```

```
until loop
10 is not equal to 1
9 is not equal to 1
8 is not equal to 1
7 is not equal to 1
6 is not equal to 1
5 is not equal to 1
4 is not equal to 1
3 is not equal to 1
2 is not equal to 1
i value is 1
loop terminated
```

Arrays

Syntax

Result

```
arr=()
```

Create an empty array

```
arr=(1 2 3)
```

Initialize array

```
${arr[2]}
```

Retrieve third element

```
${arr[@]}
```

Retrieve all elements

```
${!arr[@]}
```

Retrieve array indices

```
${#arr[@]}
```

Calculate array size

```
arr[0]=3
```

Overwrite 1st element

```
arr+=(4)
```

Append value(s)

```
str=$(ls)
```

Save `ls` output as a string

```
arr=( $(ls) )
```

Save `ls` output as an array of files

```
${arr[@]:s:n}
```

Retrieve n elements starting at index s

Lab assignment

1. Find the first 50 prime numbers using while, for and until loop.
2. Find first 10 multiples of your roll no.