Linux - Shell Loop Types

A loop is a powerful programming tool that enables you to execute a set of commands repeatedly. In this chapter, we will examine the following types of loops available to shell programmers −

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

You will use different loops based on the situation. For example, the **while** loop executes the given commands until the given condition remains true; the **until** loop executes until a given condition becomes true.

The while Loop

The **while** loop enables you to execute a set of commands repeatedly until some condition occurs. It is usually used when you need to manipulate the value of a variable repeatedly.

Syntax

```
while command
do
   Statement(s) to be executed if command is true
done
```

Here the Shell *command* is evaluated. If the resulting value is *true*, given *statement(s)* are executed. If *command* is *false* then no statement will be executed and the program will jump to the next line after the done statement.

Example

Here is a simple example that uses the **while** loop to display the numbers zero to nine.

```
#!/bin/sh

a=0

while [ $a -lt 10 ]
do
   echo $a
   a=`expr $a + 1`
done
```

Upon execution, you will receive the following result −

```
0
1
2
3
4
5
6
7
8
9
```

Each time this loop executes, the variable **a** is checked to see whether it has a value that is less than 10. If the value of **a** is less than 10, this test condition has an exit status of 0. In this case, the current value of **a** is displayed and later **a** is incremented by 1.

The for Loop

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

Syntax

```
for VARIABLE in 1 2 3 4 5 .. N
do
        command1
        command2
        commandN
done
```

or

```
for VARIABLE in file1 file2 file3
do
        command1 on $VARIABLE
        command2
        commandN
done
```

Each time the for loop executes, the value of the variable var is set to the next value in the list.

Example

Here is a simple example that uses the **for** loop to span through the given list of numbers

```
#!/bin/sh

for var in 0 1 2 3 4 5 6 7 8 9
do
   echo $var
done
```

Upon execution, you will receive the following result −

```
0
1
2
3
4
5
6
7
8
9
```

Sometimes you may need to set a step value (allowing one to count by two's or to count backwards for instance). Latest **bash version 3.0+** has inbuilt support for setting up ranges:

```
#!/bin/bash
for i in {1..5}
do
   echo "Welcome $i times"
done
```

Bash v4.0+ has inbuilt support for setting up a step value using {START..END..INCREMENT} syntax:

```
#!/bin/bash
echo "Bash version ${BASH_VERSION}"
for i in {0..10..2}
  do
     echo "Welcome $i times"
 done
```

Sample outputs:

```
Bash version 4.0.33(0)-release
Welcome 0 times
Welcome 2 times
Welcome 4 times
Welcome 6 times
Welcome 8 times
Welcome 10 times
```

Following is the example to display all the files starting with **.bash** and available in your home. We will execute this script from my root −

```
#!/bin/sh

for FILE in $HOME/.bash*
do
   echo $FILE
done
```

The above script will produce the following result −

```
/root/.bash_history
/root/.bash_logout
/root/.bash_profile
/root/.bashrc
```

The until Loop

The while loop is perfect for a situation where you need to execute a set of commands while some condition is true. Sometimes you need to execute a set of commands until a condition is true.

Syntax

```
until command
do
   Statement(s) to be executed until command is true
done
```

Here the Shell *command* is evaluated. If the resulting value is *false*, given *statement(s)* are executed. If the *command* is *true* then no statement will be executed and the program jumps to the next line after the done statement.

Example

Here is a simple example that uses the until loop to display the numbers zero to nine.

```
#!/bin/sh

a=0

until [ ! $a -lt 10 ]
do
   echo $a
   a=`expr $a + 1`
done
```

Upon execution, you will receive the following result −

```
0
1
2
3
4
```

```
5
6
7
8
9
```

Loop Control

In this chapter, we will learn following two statements that are used to control shell loops

- The **break** statement
- The **continue** statement

The infinite Loop

All the loops have a limited life and they come out once the condition is false or true depending on the loop.

A loop may continue forever if the required condition is not met. A loop that executes forever without terminating executes for an infinite number of times. For this reason, such loops are called infinite loops.

Example

Here is a simple example that uses the **while** loop to display the numbers zero to nine.

```
#!/bin/sh

a=9

while [ $a -lt 10 ]
do
   echo $a
   a=`expr $a - 1`
done
```

This loop continues forever because **a** is always **greater than** or **equal to 10** and it is never less than 10.

The break Statement

The **break** statement is used to terminate the execution of the entire loop, after completing the execution of all of the lines of code up to the break statement. It then steps down to the code following the end of the loop.

Syntax

The following **break** statement is used to come out of a loop −

```
break
```

Example

Here is a simple example which shows that loop terminates as soon as **a** becomes 5.

```
#!/bin/sh

a=9

while [ $a -lt 10 ]
do
   echo $a
   if [ $a -eq 5 ]
   then
      break
   fi
   a=`expr $a - 1`
done
```

Upon execution, you will receive the following result.

```
9
8
7
6
5
```

Here is a simple example of nested for loop. This script breaks out of both loops if **var1 equals 2** and **var2 equals 0**

```
#!/bin/sh

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
```

Upon execution, you will receive the following result. In the inner loop, you have a break command with the argument 2. This indicates that if a condition is met you should break out of outer loop and ultimately from the inner loop as well.

```
1 0
1 5
```

The continue statement

The **continue** statement is similar to the **break** command, except that it causes the current iteration of the loop to exit, rather than the entire loop.

This statement is useful when an error has occurred but you want to try to execute the next iteration of the loop.

Syntax

```
continue
```

Example

The following loop makes use of the **continue** statement which returns from the continue statement and starts processing the next statement −

```
#!/bin/sh

NUMS="1 2 3 4 5 6 7"

for NUM in $NUMS
do
   Q=`expr $NUM % 2`
   if [ $Q -eq 0 ]
   then
      echo "$NUM is an even number!!"
      continue
   fi
   echo "$NUM is an odd number"
done
```

Upon execution, you will receive the following 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
```