Loops in Shell Scripting

In this chapter, we will discuss shell loops in Unix. 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
- The select loop

Nesting Loops

All the loops support nesting concept which means you can put one loop inside another similar one or different loops. This nesting can go up to unlimited number of times based on your requirement.

Here is an example of nesting **while** loop. The other loops can be nested based on the programming requirement in a similar way −

Nesting while Loops

It is possible to use a while loop as part of the body of another while loop.

Syntax

```
while command1 ; # this is loop1, the outer loop
do
   Statement(s) to be executed if command1 is true

   while command2 ; # this is loop2, the inner loop
   do
      Statement(s) to be executed if command2 is true
   done

   Statement(s) to be executed if command1 is true
done
```

Example

Here is a simple example of loop nesting. Let's add another countdown loop inside the loop that you used to count to nine −

```
#!/bin/sh

a=0
while [ "$a" -lt 10 ]       # this is loop1
do
    b="$a"
    while [ "$b" -ge 0 ]   # this is loop2
    do
        echo -n "$b "
        b=`expr $b - 1`
    done
    echo
    a=`expr $a + 1`
done
```

for loop syntax

Numeric ranges for syntax is as follows:

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

Example:

```
        #!/bin/bash
        for i in 1 2 3 4 5
            do
                echo "Welcome $i times"
            done
```

```
#!/bin/bash
  for (( n=1; n<=10; n++ ))
      Do
          echo "$n"
      done
```

Another way to use this loop is like this:

```
#!/bin/bash
for user in Kate Jake Patt
   do
      echo "$user"
   done
```

Here we execute the loop for every string instance, which in this case is "Kate", "Jake", and "Patt".

```
#!/bin/bash
  users=(John Harry Jake Scott Philis)
  for u in "${users[@]}"
     Do
       echo "$u is a registered user"
     done
```

Reading inputs from terminal:

```
#!/bin/bash
myArray=("$@")
for arg in "${myArray[@]}"; do
    echo "$arg"
Done
```

Break and Continue statements:

```
#!/bin/sh

a=0
while [ $a -lt 10 ]do
    echo $a
    if [ $a -eq 5 ]
```

```
then
    break
fi
a=`expr $a + 1`done
```

break n
Here n specifies the nth enclosing loop to the exit from.

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

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.

```
#!/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 "Number is an even number!!"
        continue
    fi
    echo "Found odd number"
done
```