For loop Script, do-while Scripts with Programs

Most languages have the concept of loops: If we want to repeat a task twenty times, we don't want to have to type in the code twenty times, with maybe a slight change each time. As a result, we have **loops in the Bourne shell for and while**. This is somewhat fewer features than other languages, but nobody claimed that shell programming has the power of C.

You may use different loops based on the situation.

They are:

#1) Linux For loop statement

Example: **This program will add 1+2+3+4+5 and the result will be 15**

```
for i in 1 2 3 4 5
 do
 sum=`expr $sum + $i`
 done
 echo $sum
```

for loops iterate through a set of values until the list is exhausted:

for.sh

```
#!/bin/sh
for i in 1 2 3 4 5
do
  echo "Looping ... number $i"
done
```

For loops are entry-controlled looping structures that check the condition before entering the loop.

Syntax:

```
for iteration_variable in {set of space separated values}
do
    statement 1
    statement 2
    statement 3
    ...
done
```

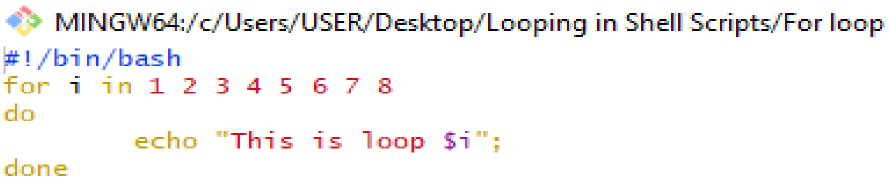
The iteration variable accesses each element of the set of values one by one and performs the task specified within the loop.

Example:

```
#!/bin/bash
for i in 1 2 3 4 5 6 7 8
do
        echo "This is loop $i";
done
```

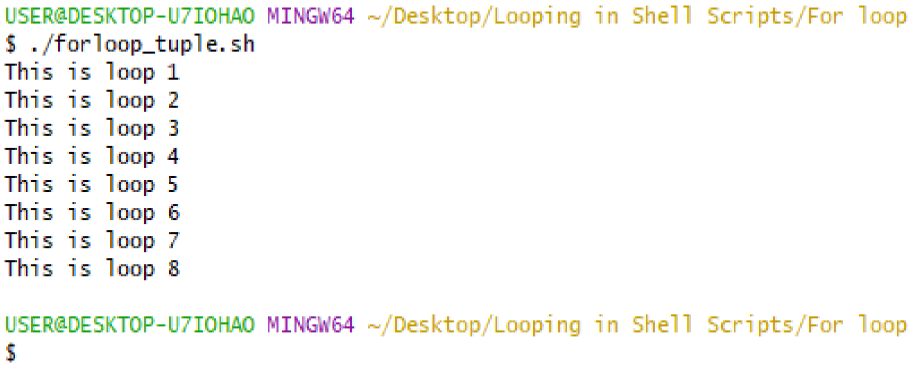
Here, for loop access each numeric value and prints in different lines.

Shell code on vi editor



```
MINGW64:/c/Users/USER/Desktop/Looping in Shell Scripts/For loop
#!/bin/bash
for i in 1 2 3 4 5 6 7 8
do
        echo "This is loop $i";
done
```

Output:



```
USER@DESKTOP-U7IOHAO MINGW64 ~/Desktop/Looping in Shell Scripts/For loop
$ ./forloop_tuple.sh
This is loop 1
This is loop 2
This is loop 3
This is loop 4
This is loop 5
This is loop 6
This is loop 7
This is loop 8

USER@DESKTOP-U7IOHAO MINGW64 ~/Desktop/Looping in Shell Scripts/For loop
$
```

do 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 the command is true

done
```

Here the Shell *command* is evaluated. If the resulting value is *true*, given *statement(s)* are executed. If the *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.

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
```

This will produce the following result. It is important to note how **echo -n** works here. Here **-n** option lets echo avoid printing a new line character.

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

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 argument 2. This indicates that if a condition is met you should break out of the outer loop and ultimately from the inner loop as well.

```
1 0
1 5
```