Shell Loop Types

we will examine the following types of loops available to shell programmers −

* [The while loop](https://www.tutorialspoint.com/unix/while-loop.htm)
* [The for loop](https://www.tutorialspoint.com/unix/for-loop.htm)
* [The until loop](https://www.tutorialspoint.com/unix/until-loop.htm)
* [The select loop](https://www.tutorialspoint.com/unix/select-loop.htm)

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.

Once you have good programming practice you will gain the expertise and thereby, start using appropriate loop based on the situation. Here, **while** and **for** loops are available in most of the other programming languages like **C**, **C++** and **PERL**, etc.

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

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

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=10

until [ $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

The break command can also be used to exit from a nested loop using this format −

break n

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

Example

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

#!/bin/sh

a=0

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 −

0

1

2

3

4

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

Like with the break statement, an integer argument can be given to the continue command to skip commands from nested loops.

continue n

Here **n** specifies the **nth** enclosing loop to continue from.

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 "Number is an even number!!"

continue

fi

echo "Found 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