SHELL LOOPS

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

```
a=0
while [ $a -lt 10 ]
do
  echo $a
  a=`expr $a + 1`
done
```

FOR LOOP

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

Syntax

```
for var in word1 word2 ... wordN
do
   Statement(s) to be executed for every word.
done
```

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.

Example

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

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 c*ommand* 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

```
a=0
until [ ! $a -lt 10 ]
do
  echo $a
  a=`expr $a + 1`
done
```

LOOP CONTROL

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 −

```
a=0
while [ $a -lt 10 ]
do
  echo $a
  if [ $a -eq 5 ]
  then
    break
  fi
  a=`expr $a + 1`
done
```

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

```
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
```