Shell Decision Making

2. Shell Decision Making

we will understand shell decision-making in Unix. While writing a shell script, there may be a situation when you need to adopt one path out of the given two paths. So you need to make use of conditional statements that allow your program to make correct decisions and perform the right actions.

The if…else statements

If else statements are useful decision-making statements which can be used to select an option from a given set of options.

Unix Shell supports following forms of if…else statement −

1. if…fi statement
2. if…else…fi statement
3. if…elif…else…fi statement

Ex. 1

Syntax for If.

If [expression]

then

Statements to be executed

Fi

Ex. 2

Syntax for if else.

if

[expression/condition]

then
Statements to be executed when condition true

else

Statements to be executed when condition not true

Fi

Ex. 3

```
if [ expression 1 ]
then
   Statement(s) to be executed if expression 1 is true
elif [ expression 2 ]
then
   Statement(s) to be executed if expression 2 is true
elif [ expression 3 ]
then
   Statement(s) to be executed if expression 3 is true
else
   Statement(s) to be executed if no expression is true
Fi
```

Shell Loop Types –

1. While loop
2. For loop
3. Until loop
4. Nested loop
5. Infinity loop
6. Loop control

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.

```
while command
```

do

Statement(s) to be executed if command is true

done

Ex.

```
#!/bin/bash
a=0
while [ $a -lt 10 ]
do
  echo $a
  a=`expr $a + 1`
done
```

==========

For Loop-

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

```
for ((initialize ; condition ; increment));
   do
#someting
  done
```

Ex.

```
for i in 1 2 3 4 5
do
echo “Looping … number $i”
done
```

Until Loop-

- Sometimes we need to execute the set of commands until a condition is true
- In simple words, here if the condition is false, then the given statement is executed

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

Eg

```
a=0
# -gt is greater than operator
#Iterate the loop until a is greater than 10
until [ $a -gt 10 ]
do
# Print the values
echo $a
# increment the value
a=`expr $a + 1`
done
```

Nested loop-

Nested for loops means loop within loop. They are useful for when you want to repeat something several times for several things.

```
#!/bin/bash
# A shell script to print each number five times.
for (( i = 1; i <= 5; i++ ))      ### Outer for loop ###
do
```


```
 for (( j = 1 ; j <= 5; j++ )) ### Inner for loop ###
    do
         echo -n "$i "
    done
  echo "" #### print the new line ###
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.

- **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.