What is Shell Scripting?

In Linux, shells like bash and korn support programming construct which are saved as scripts. These scripts become shell commands and hence many Linux commands are script.

You can get the name of your shell prompt, with following command :

Syntax:

```
echo $SHELL
```

The $ sign stands for a shell variable, echo will return the text whatever you typed in.

The sign #! is called she-bang and is written at top of the script. It passes instruction to program **/bin/sh.**

To run your script in a certain shell (shell should be supported by your system), start your script with #! Followed by the shell name.

Example:

```
1. #!/bin/bash
2. echo Hello World
3. #!/bin/ksh
4. echo Hello World
```

Any line starting with a hash (#) becomes comment. Comment means, that line will not take part in script execution.

Scripting Variables

Scripts can contain variables inside the script.

```
#!/bin/bash

var1=Hello
var2=Jai
#
echo "$var1 $var2"
```

Two variables are assigned to the script **$var1** and **$var2.**

Steps to write and execute a script

- Open the terminal. Go to the directory where you want to create your script.
- Create a file with **.sh** or **.bash** extension.
- Write the script in the file using an editor.
- Make the script executable with command **chmod +x <fileName>**.
- Run the script using ./<**fileName**>.**sh** Or **bash filename.bash**

Note: In the last step you have to mention the path of the script if your script is in other directory.

Hello World script

Here we'll write a simple programme for Hello World.
First of all, create a simple script in any editor or with echo. Then we'll make it executable with **chmod +x** command. To find the script you have to type the script path for the shell.

```
sssit@JavaTpoint:~$ echo echo Hello World > hello_world
sssit@JavaTpoint:~$ chmod +x hello_world
sssit@JavaTpoint:~$ ./hello_world
Hello World
sssit@JavaTpoint:~$
```

Look at the above snapshot, script **echo Hello World** is created with echo command as **hello_world.** Now command **chmod +x hello_world** is passed to make it executable. We have given the command **./hello_world.sh or ./hello_world.bash** to mention the hello_world path. And output is displayed.

OPERATORS

Assume variable **a** holds 10 and variable **b** holds 20 then

Operator	Description	Example
-eq	Checks if the value of two operands are equal or not; if yes, then the condition becomes true.	[$a -eq $b] is not true.
-ne	Checks if the value of two operands are equal or not; if values are not equal, then the condition becomes true.	[$a -ne $b] is true.
-gt	Checks if the value of left operand is greater than the value of right operand; if yes, then the condition becomes true.	[$a -gt $b] is not true.

-lt	Checks if the value of left operand is less than the value of right operand; if yes, then the condition becomes true.	[$a -lt $b] is true.
-ge	Checks if the value of left operand is greater than or equal to the value of right operand; if yes, then the condition becomes true.	[$a -ge $b] is not true.
-le	Checks if the value of left operand is less than or equal to the value of right operand; if yes, then the condition becomes true.	[$a -le $b] is true.

It is very important to understand that all the conditional expressions should be placed inside square braces with spaces around them. For example, **[$a <= $b]** is correct whereas, **[$a <= $b]** is incorrect.

Control and Loop Structure in Shell Script

Condition is a comparison between two values.for compression you can use test or [expr] statements or even exist status can be also used.expression is defined as - an expression is nothing but combination of values,relational operator (>,<,<>) and mathematical operator (+,-,/).there are the following kinds of statements available in shell programming as decision making statements.

□ if

simple if is used for decision making in shell script.if the given condition is true then it will execute the set of code that you have allocated to that block.

Syntax

if [condition]

then

Execute the statements

fi

Example

```
#Check Number is 1
echo "Enter Number:-"
read no
if [ $no -eq 1]
then
    echo "Number 1 "
```

```
fi
```

if..else

if..else is used for decision making in shell script where the given condition is true then it will execute the set of code that you have allocated to that block otherwise you can execute the rest of code for the false condition.

Syntax

```
if [ condition ]
then
   Execute Statement if Condition is True
elif
   Execute Statement if Condition is False
fi
```

Example

```
#Check Number is Positive or Not
echo "Enter Number:"
read no
if [ $no -gt 0 ]
then
   echo "Number is Positive"
elif
   echo "Number is Negative"
fi
```

if..elif..else

it is possible to create compound conditional statements by using one or more else if(elif) clause.if the 1st condition is false,then subsequent elif statements are checked.when an elif condition is found to be true,the statements following that associated parts are executed.

Syntax

```
if [ condition ]
then
   Execute Statement if Condition 1
elif [ condition ]
   Execute Statement if Condition 2
elif [ condition ]
   Execute Statement if Condition 3
elif
   Else Condition
fi
```

Example

```
#Find Student Class

echo "Enter Student Mark:-"
read mark

if [ $mark -gt 70]
then
   echo "Distinction"
elif [ $mark -gt 60]
then
   echo "First Class"
elif [ $mark -gt 50]
then
   echo "Second Class"
elif [ $mark -gt 40]
then
   echo "Pass Class"
elif
   echo "Fail"
fi
```

Nested if

if statement and else statement can be nested in bash shell programming.the keyword "fi" indicates the end of the inner if statement and all if statement should end with "fi".

Syntax

```
if [ condition ]
then
   if [ condition ]
        then
           Execute Statement
        elif
           Execute Statement
        fi
elif
   Execute Statement
fi
```

Example

```
#Nested if Example

echo "Enter Your Country:"
read cn

if [$cn -eq 'India']
then
   echo "Enter Your State:"
        read st
   if [$st -gt 'Gujarat']
        then
           echo "Welcome to Gujarat"
        elif
           echo "You are Not Gujarati"
        fi
elif
   echo "Other Country"
fi
```

Case Statement

The case statement is good alternative to multilevel if then else fi statement.it enables you to match several values against one variable.its easier to read and write multiple conditions.

Syntax

```
case $[ variable_name ] in
value1)
   Statement 1
        ;;
value2)
   Statement 2
        ;;
value3)
   Statement 3
        ;;
value4)
   Statement 4
        ;;
valueN)
   Statement N
        ;;
*)
   Default Statement
        ;;
esac
```

Example

```
#Case Statement Example
echo "Enter Country Code:"
read co

case $co in
'IN') echo "India"
   ;;
'PK') echo "Pakistan"
   ;;
*) echo "Enter Vail id Country Code"
```

```
   ;;
esac
```

Shell Loops and Different Loop Types like:

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 for and while loops in the Bourne shell. This is somewhat fewer features than other languages, but nobody claimed that shell programming has the power of C.

- For Loop
- While Loop
- Until Loop

For Loop

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

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.

Syntax

for var in word1 word2 ...word n

do

  Statement to be executed

done

E.g :

for.sh

```
#!/bin/sh
for i in 1 2 3 4 5
do
```

```
 echo "Looping ... number $i"
done
```

b) E.g :

```
#!/bin/bash
for i in {1..5}
do
  echo "Welcome $i times"
done
```

c) E.g

```
#!/bin/bash
for i in {0..10..2}
 do
   echo "Welcome $i times"
 done
```

d) E.g

```
for (( c=1; c<=5; c++ ))
do
  echo "Welcome $c times"
done
```

e) E.g

```
for (( ; ; ))
do
  echo "infinite loops [ hit CTRL+C to stop]"
done
```

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 to be executed

Done

E.g

```
a=1
 while [ $a -le 5 ]
 do
 echo “value of a=” $a
 a=`expr $a + 1`
 done
```

b) E.g

```
while :
do
  echo "Please type something in (^C to quit)"
  read INPUT_STRING
  echo "You typed: $INPUT_STRING"
done
```

c) E.g (Factorial)

```
counter=$1
factorial=1
while [ $counter -gt 0 ]
do
   factorial=$(( $factorial * $counter ))
   counter=$(( $counter - 1 ))
done
echo $factorial
```

d) E.g

```
#!/bin/sh
while read f
do
  case $f in
```

```
        hello)          echo English        ;;
        howdy)          echo American       ;;
        gday)           echo Australian     ;;
        bonjour)        echo French ;;
        "guten tag")    echo German         ;;
        *)              echo Unknown Language: $f
                ;;
  esac
done < myfile
```

e) E.g

```
while :
do
        read -p "Enter two numbers ( - 1 to quit ) : " a b
        if [ $a -eq -1 ]
        then
                break
        fi
        ans=$(( a + b ))
        echo $ans
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 command 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

Here is a simple example that uses the until loop to display the numbers zero to nine −

```
#!/bin/sh
a=0
until [ ! $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

Example

Create a shell script called until.sh:

```
#!/bin/bash
i=1
until [ $i -gt 6 ]
do
        echo "Welcome $i times."
        i=$(( i+1 ))
done
```

Sample outputs:

Welcome 1 times.

Welcome 2 times.

Welcome 3 times.

Welcome 4 times.

Welcome 5 times.

Welcome 6 times.