SHELL SCRIPTING

Shell: It is a command line interface.

-shell is an interface between user and the kernel.

Kernel: It acts an intermediate between hardware and software.

-kernel is also a program in Linux that keep on running and it will process the commands given by shell.

Types of Shell

1.Gnome shell

2.KDE

3.sh

4.Bash shell

5.csh and tcsh

6.ksh

1.Gnome shell: Genome is like a graphical environment that is Linux starting with version 3. which was released on April 6, 2011.

-GNOME [GNU-Network object Model Environment] it provides basic functions like launching applications, switching between windows.

2.KDE: K Desktop Environment, it is a desktop environment for Linux based operating system.

Command Line Shell

3.Sh-Bourne shell: Unix computers by Stephen Bohn

-The Bourne shell is used for scripting. It provides command based programming to interpret and execute user commands.

-As a user types a command, the shell interprets it, so the operating system can take actions, such as automating a task.

-The Bourne shell was the default shell for Unix version 7.

- It offers features such as input and output redirections, shell scripting with string integer variables and condition testing and looping.

4.Bash: Bourne again shell

-It is free and enhanced version of the Bourne shell distributed with Linux and GNU operating system. Bash is similar to the original, but has added features such as command line editing.

5.csh and tcsh: csh is a Unix shell that provides command line user interface to interact with an operating system.

-It was created by Bill Joy at the university of California at Berkley in the late 1970s, c shell is one of the oldest Unix shell used today.

-Tcsh is an enhanced version of the csh, it behaves exactly like csh but includes some additional utilities such as command line editing and filename/commands.

6.ksh: Korn shell

-korn shell is an operating system command shell that was developed for unix by David korn at Bell labs.

Find your shell:

echo $0

Available shells: cat /etc/shells

Shell Scripting:

A shell script is an executable file containing multiple shell commands that are executed sequentially.

The file can contains

- Shell(#!/bin/bash)
- Comments(#comments)
- Commands(echo,cp,mv,grep etc)
- Statements(if,while,for etc)

-Shell script should have executable permissions

Ex: -rwx r-x r-x

-Shell script has to be called from absolute path

Ex: /home/userdir/script.sh

-If it is called from current location then ./script.sh

Variables:

-variables is a container which will store our data.

-data holders.

Declaring the variables:

Syntax: variable-name = variable_value

To print the value of the variable:

Echo "$variable_name" or echo $variable_name

To take the input from the user

Syntax: read variable_name

To make a variable readonly

Syntax: readonly variable name

Operators:

The operators are the special symbol and characters that will perform a specific task.

Types of Operators:

1.Arithmetic operators

2.Logical operators

3.Relational operators

4.String operators

1.Arithmetic operators:

-The arithmetic operators are the special symbols which will Perform mathematical operations(+,-,*,/,%)

= = --->Equality operator

= --->Assignment operator

!= --->Not equality operator

ex: a=10 b=20

[Sa == $b] [&a != $b]

2.Logical operators:

-Logical operators will check the condition and compare the values of the given variables then it will return either true or false.

-Logical AND: it will check the condition of 2 operands and if both the conditions are true then it will return true or else it will return false.

-a ---> ex: a=10 b=20

[$a < $b] -a [$a == 1]

-Logical OR: It will check the condition of 2 operands and if one of the conditions is true then it will return true.

-o ---> ex: a=10 b=20

[$a < $b] -o [$a == 1]

-Logical NOT: It will check the condition and if the condition is true then it will return false, if the condition is false then it will return true (vice versa).

! ---> ex: ![$a >= $b] condition is false

Output-->true

3.Relational operators:

-It will check the relation between the 2 operands and if the condition is true then it will return true or else false.

-eq : It will check the values of variables are equal or not, if equal then it will return true.

ex: a=10 b=20 [$a -eq $b]

-ne[not equal]:It will check the relation between the 2 operands and if the values are not equal then it will return true.

ex: a=10 b=20 [$a -ne $b]

-gt[greater than]: It will compare the values of 2 operands, if the left operand is greater than the right operand then it will return true.

ex: a=10 b=20 [$b -gt $a]

-lt[lesser than]: It will compare the values of 2 operands, if the left operand is lesser than the right operand then it will return true.
ex: a=10 b=20 [$a -lt $b]

-ge[greater than or equal]: It will compare the values of 2 operands, if the left operand is either greater than or equal to right operand then, it will return true.
ex: a=10 b=20 [$b -ge $a]

-le[lesser than or equal]: It will compare the values of 2 operands, if the left operand is either lesser than or equal to right operand then, it will return true.
ex: a=10 b=20 [$a -le $b]

4.String operator:

-It will compare the values of 2 strings, if the condition is true it will return true and if the condition is false it will return false.

= ---> It will check, if the values of 2 operands are equal or not, if equal then condition will become true.

!= ---> It will check, if the values of 2 operands are equal or not, if not equal then condition will become true.

-z ---> It will check, if the given string operand size is zero or not, if the size is zero then it will return true.

-n ---> It will check, if the given string operand size is zero or not, if the size is not zero then it will return true.

Conditional Statements:

1.if: It will check the condition, if condition is true then it will execute, if the condition is false it will stop execution.

Syntax:

```
if [condition]
then
 <statement>
fi
```

2.if-else: It will check the condition, if condition is true it will return true, if the condition is false it will return false.

Syntax:

```
if [condition]
then
  <statement>
else
  <statement>
fi
```

3.if-elseif: It will check multiple conditions, if any one of the condition is true, it will not check for the other conditions, it will return true.

Syntax:

```
if [condition]
then
  <statement>
elif [condition]
then
   <statement>
elif [condition]
then
  <statement>
else
   <statement>
fi
```

4.Nested if: if you want to write condition inside another condition we will go with nested if.
So, if condition is true then again it will check for another condition inside the block.
Syntax:

```
if [condition]
then
      if [condition]
      then
            <statement>
      else
            <statement>
       fi
else
     <statement>
fi
```

Case statement:

When decision making is based on multiple choices, According to the user input a particular choice will get executed and the other choices will be skipped.
Syntax:

```
case condition variable in
case1) statement
   ;;
case2) statement
   ;;
case3) statement
   ;;
case4) statement
   ;;
case n) default
   ;;
esac
```

Looping Statements:

1.for
2.while
3.until

1.for loop: for loop moves through a specified list of values until the, list is exhausted.
Syntax:

```
for var in item1 item2 item3
do
   <statement>
done
```

2.while loop: while loop runs the program until the expression becomes false.
Syntax:

```
while  [condition]
do
    <statement>
done
```

3.until loop: It is opposite to while loop, it will execute until the condition becomes true, if the execution becomes true it will stop the execution.
Syntax:

```
until [condition]
do
    <statement>
done
```