**What is Linux Shell ?**

In early days of computing, instruction were provided using binary language, which is difficult for us to read and write. So in OS there is a special program called **Shell**. Shell accepts your instruction or commands in English (mostly) and if it is a valid command, it is passed to the kernel.

Shell is a user program or it's an environment provided for user interaction. Shell is a ***command language interpreter*** that executes commands read from the standard input device (keyboard) or from a file.

**Several shell available with Linux including:**

* BASH ( Bourne-Again SHell )
* CSH (C SHell)
* KSH (Korn SHell)
* TCSH

**What is Shell Script ?**

Normally shells are interactive. It means shell accept command from keyboard and execute them. But if you use commands one by one (sequence of '*n*' number of commands) then, you can store these sequence of command in a text file and tell the shell to execute this text file instead of entering the whole sequence of commands. This is known as **shell script**.

**Why to Write Shell Script ?**

* Shell script can take input from user, file and output them on screen.
* Useful to create our own commands.
* Save lots of time.
* To automate some task of day today life.
* System Administration part can be also automated.

**Which Shell we are using to write shell script ?**

In this session we will be using the Linux **bash** **shell**.

**How to write shell script**

**Following steps are required to write shell scripts**:

1. Use any editor like **vi** to write shell script.

(2) After writing shell script, set ***execute* *permission*** for your script as follows:

*Syntax:*

**$ chmod permission your-script-name**

*Examples:*

$ chmod +x script-name

$ chmod 755 script-name

***Note:*** This will set **read write execute** (7) permission for owner. For group and others, permission is read and execute only (5).

(3) ***Execute*** your script as:

*Syntax:*

**sh script-name.sh**

*Example:*

$ sh bar.sh

**Example of a shell script:**

$ vi first

#

# My first shell script

#

clear

echo "Knowledge is Power"

***Exercise:***

Write the following shell script, save it, execute it and note down it's output.

# It is a test script

clear

echo "Hello $USER"

echo "Today is \c ";date

echo "Number of user login : \c" ; who | wc -l

echo "Calendar"

cal

exit 0

**Variables in Shell**

In Linux (Shell), there are two types of variable:

(1) **System variables** - Created and maintained by Linux itself. These types of variables are defined in UPPER CASE letters.

(2) **User defined variables (UDV)** - Created and maintained by user. These types of variables are defined in lower case letters.

**How to define User defined variables (UDV)**

To define UDV use following syntax:

*Syntax:*

**variable name=value**

'**value**' is assigned to given a '**variable name**' and value must be on right side of = sign.

*Example:*

$ no=10 # this is ok

$ 10=no # ***Error***, value must be on right side of = sign.

To define a variable called 'vech' having value Bus:

$ vech=Bus

To define a variable called n having value 10:

$ n=10

**Rules for giving variable names:**

(1) ***Don't put spaces on either side of the equal sign*** when assigning value to variable.

For e.g. in the following variable declaration there will be no error

$ no=10

But there will be problem for any of the following variable declaration:

$ no =10

$ no= 10

$ no = 10

(2) Variables are ***case-sensitive***, just like filename in Linux. For e.g.

$ no=10

$ No=11

$ NO=20

$ nO=2

All the variables are different, so to print value 20 we have to use $ echo $NO and not any of the following:

$ echo $no # will print 10 but not 20

$ echo $No # will print 11 but not 20

$ echo $nO # will print 2 but not 20

(3) You can define **NULL** variable as follows (NULL variable is variable which has no value at the time of definition) For e.g.

$ vech=

$ vech=""

Print its value by issuing following command:

$ echo $vech

Nothing will be shown because variable has no value i.e. NULL value.

(4) Do not use ***special characters*** like **? , \*** etc, to name your variable.

**How to print or access value of UDV**

To print or access UDV use following syntax:

*Syntax:*

**$ variablename**

Define variable vech and *n* as follows:

$ vech=Bus

$ n=10

To print contains of variable 'vech' type:

$ echo $vech # It will print 'Bus'

To print contains of variable '*n*' type:

$ echo $n

**echo Command**

Use echo command to display text or value of variable.

*Example:* echo "An apple a day keeps the doctor away \n"

**Shell Arithmetic**

Use to perform arithmetic operations.

*Syntax:*

**expr op1 operator op2**

*Examples:*

$ expr 1 + 3

$ expr 2 - 1

$ expr 10 / 2

$ expr 20 % 3

$ expr 10 \\* 3

$ echo `expr 6 + 3`

z = `expr $x + $y`

**The read statement**

Use to get input from keyboard (data from user) and store it in a variable.

*Syntax:*

**read variable1, variable2,...variableN**

*Example:*

$ vi say.sh

#

#Script to read your name from key-board

#

echo "Your first name please:"

**read fname**

echo "Hello **$fname**, Lets be friend!"

Run it as follows:

$ sh say.sh

*Output:*

Your first name please: **vivek**

Hello vivek, Lets be friend!

**Command Line arguments**

Telling the command / utility which option to use.

*Example:*

For shell script, **$ myshell foo bar**



In shell if we wish to refer this command line argument we refer above as follows:

 myshell it is **$0** (Shell Script name i.e. myshell)

 foo it is $1 (First command line argument passed to myshell i.e. foo)

 bar it is $2 (Second command line argument passed to myshell i.e. bar)

**$\* :** used to display the complete set of positional parameters as a single line without spaces.

*Example:*

myshellfoobar

**$@ :** used to display the complete set of positional parameters as a single line with spaces.

*Example*:

myshell foo bar

**$# :** used to show the number of arguments specified.

**$? :** exit status of last command.

*Example*:

exit 0 : used when everything goes fine.

exit 1 : used when something goes wrong.

In shell script every command returns an exit status to caller.

*Example*:

grep “director” emp.lst ; echo $?

**0**

Success ( it finds the pattern “director” in emp.lst)

grep “manager” emp.lst ; echo $?

**1**

Failure (does not finds the given pattern “manager” )

grep “manager” emp.lst ; echo $?

**2**

Failure (failure in opening the file)

**if condition**

if condition which is used for decision making in shell script, If given condition is true then command1 is executed.

*Syntax:*

**if condition**

**then**

command1 if condition is true

**...**

**...**

**fi**









**if...else...fi**

*Syntax:*

**if condition**

**then**

condition is zero (true - 0)

execute all commands up to else statement

**else**

if condition is not true then

execute all commands up to fi

**fi**

*Example:*

if [ $a -gt $b ]

then

echo "a is positive"

else

echo “b is negative"

fi

**Nested if-else-fi**

*Syntax:*

**if condition**

**then**

**if condition**

**then**

.....

..

do this

**else**

....

..

do this

**fi**

**else**

...

.....

do this

**fi**

**Multilevel if-then-else**

*Example:*

if [ $1 -gt 0 ]; then

echo "$1 is positive"

elif [ $1 -lt 0 ]

then

echo "$1 is negative"

elif [ $1 -eq 0 ]

then

echo "$1 is zero"

else

echo "Opps! $1 is not number, give number"

fi

**The case Statement**

*Syntax:*

**case** $variable-name **in**

pattern1) command

...

command;;

pattern2) command

...

…

command;;

patternN) command

...

…

command;;

**\*)** command

...

..

command;;

**esac**

*Example:*

case $rental in

"car") echo "For $rental Rs.20 per k/m";;

"van") echo "For $rental Rs.10 per k/m";;

"jeep") echo "For $rental Rs.5 per k/m";;

"bicycle") echo "For $rental 20 paisa per k/m";;

\*) echo "Sorry, I cannot get a $rental for you";;

esac

*Example:*

read choice

case “$choice” in

1. ls –l;;
2. date;;
3. who;;

\*) echo “invalid option”;;

esac