```
###############
Shell Scripting
###############

++++++++++++++
What is shell ?
++++++++++++++

-> Shell is responsible for reading commands given by user

-> Shell will verify command and will give instructions to kernel to process that command

-> If command is invalid shell will give error

-> Kernel will execute our command with System Hardware Components

-> Shell acts as mediator between User and Kernel

++++++++++++++++++
What is Scripting ?
++++++++++++++++++

-> Scripting means set of commands mentioned in a file for execution

-> Scripting is used to automate our routine work

-> For example i want to execute below commands every day as a linux user

$ date
$ cal
$ whoami
$ pwd
$ ls -l

-> Instead of executing all these commands manually we can keep them in a file and we can execute
that file.

-> The file which contains set of commands for execution is called as 'Script file'

-> Script file will have .sh extension

        Ex:    task.sh

######################
What is Shell Scripting ?
######################

-> Shell Scripting is used to execute set of commands using a script file

-> When we execute script file then shell will read those commands and will verify commands syntax

-> Shell will give instructions to 'Kernel'

-> Kernel will give instructions to hardware components to perform actual operations



##############
Types of Shells
###############

-> There are several shells available in linux OS

1) Bourne Shell
```

```
2) Bash Shell
3) Korn Shell
4) CShell
5) TShell
6) ZShell


# Display all the shells of our linux machines
$ cat /etc/shells

# Display the default shell of our linux machine
$ echo $SHELL


Note: The most commonly used shell in linux is 'BASH SHELL'.

Note: Shell Script files will have .sh extension

###################################
Working with First Shell Script Program
###################################

# Create a script file
$ vi task.sh

whoami
pwd
date

-> Save the file and close it (ESC +  :wq)

# Run the shell script (Option-1)
$ sh task.sh

Note: If we get permission denied then we should provide 'execute' permission using 'chmod' command

# Run the shell script (Option-2)
$ ./task.sh



##############################
What is sha-bang in shell script ?
##############################

-> Sha-bang is used to specify which shell should be used to process our script

Syntax :

#! /bin/bash



**********Shell Script - 2 (print output to console) **********
#! /bin/bash

echo "Welcome to Scripting"
echo "Scripting is used to automate regular work"
echo "Scripting requires lot of practise"

**********Shell Script - 3 (take input from user) **********

#! /bin/bash

echo "Enter your name:"
```

```
read name
echo "Good Morning $name"

******************** Shell Script - 4 *****************************

#! /bin/bash

a=10
b=20

c=$(($a + $b))

echo "Sum of $a and $b is =  $c"


************************** Shell Script - 5 *****************************

#! /bin/bash

echo "Enter First Number"
read a
echo "Enter Second Number"
read b

c=$(($a + $b))

echo "Sum of $a and $b is =  $c"


*********************************
Variables
Control Statements
Case Statements
Loops
Functions
*********************************


-> Variables are place-holders to store the value

-> Varibles are key-value pairs

-> In Shell Scripting there is no concept of Data Type.

-> Every value will be treated as text/string


Ex:

name=ashok
age=30
email=ashokitschool@gmail.com
phno=1234

-> Variables are divided into 2 types


1) Environment Variables or System variables
2) User Defined Variables


-> The variables which are already defined and using by our system are called as Environment/System
variables

Ex:
```

```
$ echo $USER
$ echo $SHELL


-> Based on our requirement we can define our own variables those are called as user defined
variables

Ex:

name=ashok
age=30

$echo $name $ age


################
Variable Rules
#################

-> We should not use special symbols like -, @, # etc....
-> Variable name should not start with digit

Note: It is recommended to use uppercase characters for variable name



-> we can use 'readonly' for variable so that variable value modification will not be allowed

########################
Command Line Arguments
########################

-> The arguments which will pass to script file at the time of execution

-> Cmd args are used to supply the values dynamically to the script file

Ex:

$ sh demo.sh ashokit 30

-> We can access cmd args in script file like below

$# - no.of args
$0 - script file name
$1 - First Cmd Arg
$2 - Second Cmd Arg
$3 - Third Cmd Arg
$* -  All Cmd Args



-> To comment any single line we can use '#'

echo 'hi'
#echo 'hello'


-> We can comment multiple lines also in script file like below

<<COMMENT
        ......................

COMMENT
```

-> We can hold script execution for some time using 'sleep' command

```
#! /bin/bash

echo $#
echo $0
echo $1
sleep 30s
echo $2
#echo $*
```

```
#######################
Conditional Statements
#######################
```

-> Conditional statements are used to execute commands based on condition

Syntax:

```
if [ conition ]
then
        stmts
else
        stmts
fi
```

-> If given condition satisfied then if statments will be executed otherwise else statements will be executed

```
if [ condition ]
then
        stmts
elif [ condition ]
then
        stmts
else
        stmts
fi
```

Ex:

```
#!/bin/bash

echo "Enter Your Favorite Color"

read COLOR

if [ $COLOR == 'red' ]
then
        echo "Your are cheerful"
elif [ $COLOR == 'blue' ]
then
        echo "You are joyful"
else
         echo "You are lucky"
fi
```

```
###################
Working with loops
###################
```

-> Loops are used to execute stmts multiple times

-> We can use 2 types of loops

1) Range based loop (ex: for loop)
2) Conditional based loop (ex: while loop)

################
For Loop Example
################

```
#! /bin/bash

for ((i=1; i<=10; i++))
do
echo "$i"
done
```

###################
While loop Example
###################

```
#! /bin/bash
i=10
while [ $i -ge 0 ]
do
echo "$i"
let i--;
done
```

#################
Infinite Loop
#################

```
#! /bin/bash
while true
do
echo "This is my loop stmt"
done
```

Note: To stop infinite loop we will use 'ctrl + c'

##############
Functions
##############

-> The big task can be divided into smaller tasks using functions

-> Function is used to perform an action / task

-> Using functions we can divide our tasks logically

-> Functions are re-usable

Syntax:

```
function functionaName( ) {
```

```
      // commands to execute
}


########################
Writing welcome function
########################

#! /bin/bash

function welcome(){
  echo "Welcome to functions...";
  echo "I am learning Shell Scripting";
  echo "Shell Scripting is used to automate our regular work";
}

# call the function
welcome

########################
Function with Parameters
########################

#! /bin/bash

function welcome ( ) {
      echo "$1";
}

# call function
welcome Linux
welcome AWS
welcome DevOps
```