Bash Shell Scripting Guide

Focus Training Services

VERSION 1.0

- **Unix commands that we need to know**

  Basic-
  uptime, uname, who, w, date, cal

  File related commands-
  ls, cat, cp, mv, rm, vi

  Directory related commands-
  ls, mkdir, rmdir, cd, rm -r

  Permission related commands-
  chmod, chgrp, chown

  Filters-
  cut, tr, sort, head, tail
  grep, uniq

  Important commands-
  find, sed, tar, awk

- UNIX Shell Scripting basics-

  Components:
  1. CPU
  2. RAM (Memory)
  3. Hard Disk

- What is a variable?

  Variable is a space in the memory(RAM) of the computer which stores some information.

- What are the characteristics of a variable?
  1. They have a name
  2. They have a value
  3. They have no size restriction

4. They have no data type restriction. All are strings

- How to assign a value to a variable in bash shell

  a=10 ← 10 is assigned to variable a

  Note: Please note that there is no space before and after the =

  a='10' ←10 is assigned to variable a

  a="10" ← 10 is assigned to variable a

  a=shekhar ← shekhar is assigned to variable a

  a='shekhar' ← single quote not necessary because there is no space in value

  a="shekhar" ← double quote is not necessary

  a=shekhar tulshibagwale ← This will not work because of the space in the value

  a='shekhar tulshibagwale' ← Since there is a space in the value of a variable, quotes are required (single or double)

Note: The difference between single quote and double quote will be clear in the coming sections

- How to find out what is the value inside a variable

Example No. 1

echo $a ← print the value of variable a

Example No. 2

echo ${a} ← print the value of variable a, good practice

Example No. 3

echo "${a}" ← print the value of variable a, better practice

- How to assign a value of one variable to another variable

Example No. 1

a=100 ← 100 is assigned to variable a

b=$a ← value of a will be assigned to variable b

echo $a ← Value of variable a is displayed

echo $b ← Value of variable b is displayed

Example No. 2

name=shekhar ← shekhar is assigned to variable named name

myname=${name} ← Value of variable name is assigned to new variable named myname

echo $name ← Value of variable name is displayed

echo $myname ← Value of variable myname is displayed

Example No. 3

my_name='Peter Parker' ← Peter Parker is assigned to variable named my_name

your_name="${my_name}" ← value of my_name is assigned to variable your_name

echo $my_name ← value of variable my_name is displayed

echo $your_name ←value of variable my_name is displayed

Example No. 4

a=100 ←100 is assigned to9 variable a

b=a ← a will be assigned to variable b and not the value of variable a (This is INCORRECT)

echo $b ← What will be printed?

- What is a shell script?
  It is a text file with any extension (.sh is not mandatory)
  It is not compiled before executing.
  Programs are compiled, scripts are not.
  It is run/executed with the help of an interpreter.

An interpreter is a shell program.
There are many shells (i.e. interpreters) available. They are as follows:

1. bash
2. sh
3. ksh
4. csh
5. tcsh

We will learn shell scripting ONLY in bash shell as it is the latest shell in the market.

- What is the difference between programming & scripting

C programming
Java programming
Java scripting
Shell Scripting
Perl scripting
Python scripting

Answer: programs are compiled and then executed. Scripts are not compiled. They are directly executed by the interpreter.

- How to execute a shell script

Method No. 1

Use the interpreter followed by the name of the shell script

1. bash 1.sh
   bash /home/shekhar/1.sh

2. ksh 1.sh
   ksh /home/shekhar/1.sh

3. csh a.sh
   csh /home/shekhar/1.sh

Method No. 2 (Most preferred method)

1. /home/shekhar/1.sh <== full name of script

2. ./1.sh <== relative name of script

Note: The script file must have an execute permission to the user who wants to execute it

Run the ls -l command to check the permissions-

```
ls -l 1.sh
```

Change the permissions as follows:

```
chmod u+x 1.sh
chmod g+x 1.sh
chmod o+x 1.sh
chmod a+x 1.sh
```

Assignment: Change the shell to csh for 1.sh script and execute the same. Does it run?

- What is a string literal

  -Anything between single quotes is called as a string literal. The following example will clarify the difference between single quote and double quote

Double quotes should be used if you want to evaluate the variable inside the quotes

Example No:1

```
a=shekhar

b="$a"    <== value of variable a i.e. shekhar will be
              assigned to variable b

b='$a'    <==  $a will be assigned to variable b
               That's why we say that anything in
               single quote is a literal.
```

Example No:2

a=100

echo "I have Rs. $a with me"

echo 'I have Rs. $a with me' ←This is a literal

```
#!/bin/bash
# ---------------------------------
# Script Name: 2.sh
# #! means She bang
# ---------------------------------
clear
first_name=Peter
last_name='Tulshibagwale'
full_name="${first_name} ${last_name}"
echo '--------------------------------------'
echo "Hello $full_name"
echo '--------------------------------------'
```

- How to accept input from a user

  read a ← Accepted input will be store in variable a
  read a b ← Two values will be accepted.
  read -p 'what is your name' a
  read -p 'Enter your full name ' fn ln
  read a < m.txt # only first line is read
  read -s -p 'Enter your password: ' p

```
#!/bin/bash
#--------------------------------
# Script Name: 3.sh
# This is my first shell script
# #! means She bang
# --------------------------------
clear
echo '---------------------------------------'
read -p 'What is your name : ' nm
echo "Hello $nm"
echo '---------------------------------------'
```

```
#!/bin/bash
#-----------------------------------------
#Script Name : 3a.sh
#-----------------------------------------
read -p 'Please enter your first name : ' fn
read -p 'Please enter your last  name : ' ln
read -p 'Please enter your city       : ' city
read -p 'Please enter your DOB        : ' dob
echo "$fn:$ln:$city:$dob" 1>> /tmp/data.txt
```

- All unix commands can use variable in them

Example No. 1

```
file1=a.txt          ← a.txt is assigned to variable file1
file2=b.txt          ←  b.txt is assigned to variable file2
cp $file1 $file2     ← This will execute "cp a.txt b.txt' command
```

Example No. 2

a='/tmp/a.tar' ← /tmp/a.tar is assigned to variable a single quotes is not mandatory in this case. Because no space in the value of the variable.

b="/home/shekhar" ← /home/shekhar is assigned to variable b. Double quotes not required here.

tar -cvf $a $b ← This will execute "tar -cvf /tmp/a.tar /home/shekhar"

tar -cvf ${a} ${b} ← better way to execute the same command

tar -cvf "${a}" "${b}" ← The best way to execute the same command

tar -cvf '${a}' '${b}' ← will not work

- Why use the best practice of double quotes and { } in varables

```
a=shekhar
b=tulshibagwale
c=$a_$b          ← What is the value of c
c=${a}_${b}      ← What is the value of c .

k='resume.pdf'
rm $k
rm ${k}
rm "${k}"  ← best practice
rm '${k}'  ← will not work

dir=/home/shekhar/backup
mkdir ${dir}
mkdir "${dir}"  ← best practice
```

- What is an escape character

  In UNIX \ is an escape character. It removes the special meaning of a character that follows it.

  a=100
  echo "I have $$a with me" ← What will be the output
  echo "I have \$$a with me"

  expr 10 * 20

- Special Characters in UNIX

  $ - to find out the value of a variable
  * - all file names in pwd (multiple wild card)
  ? - one wild card
  \ - acts as an escape char
  ~ - home dir
  - - previous dir (cd -)
  / - top most dir in tyhe filesystem
  . - pwd
  .. - parent dir
  ' - used for string literal
  " - used to evaluate variable
  ` - used to capture output of a unix command
  # - comment in shell script
  #! - she bang - first line in shell -

- I/O redirection

  command 1> file
  command 1>> file
  command 2> file
  command 2>> file
  command < file
  command <<EOF

command &> file
command &>> file
command | command
command | tee file | command

- Process Management

  & - Send process in background
  ctrl c - Terminate a process
  ctrl z - Suspend a process
  ctrl d -

- Regular expression

^ - Caret/Power symbol to match a starting at the beginning of line
$ - To match end of the line
[ ] - To match end of the line
( ) -
[!] -
. - To match any character
* - 0 or more occurrence of previous character

- How to assign an output of any UNIX command to a variable

  Use inverted quotes, back quotes, ticks-

  ```
  a=`whoami`
  b=`hostname`
  c=`users`

  a=$(whoami)
  b=$(hostname)
  c=$(users)
  ```

  Note: New line characters will be removed from the output of the

command before assigning it to the variable.

a=`cat b.txt` ← multiple lines will become 1 line

- How to assign an output of a unix command and put it into a file
  who > a.txt

- How to put contents of a variable into a file
  echo $a > a.txt

- How to put contents of a file into a variable
  a=`cat a.txt`

- What is a shell?

  It is a program that acts as a middle man between the user and kernel.

- What is a login shell?

  It is a program that starts automatically when a user logs in

  echo $SHELL ← show you what your login shell is

- What is your current shell

  It is a shell program that is currently interpreting your commands

  ehco $0 ← shows your current shell.

  -A Shell is started for each session

  Check using ps -f

-What shells are installed on your OS

cat /etc/shells

-How to change your login shell-

chsh

-How to change your current shell-

ksh ← will change your current shell to ksh
csh ← will change your current shell to csh

- What happens after you login?

- your login shell program(process) is
automatically started in the background
- Certain files are executed automatically
1. /etc/profile
2. /etc/bashrc
3. /home/$USER/.bash_profile
4. /home/$USER/.bashrc

-You are put in your home directory

- System Defined Variables

USER - username
HOSTNAME - name of the machine
HISTSIZE- no of unix commands stored in ram
for your session (used by the history commands)
~/.bash_history
SHELL - your login shell
(use chsh command to change your login shell)
0 - Current shell
PWD - rpesent working directory
PS1 - Login prompt
USERNAME - username

LOGNAME - username
MAIL - Directory where the mails are kept
HOME - your home directory
PATH - list of directories separated by :

-When any command is executed these directories are searched for the executable file of the command

- Which command shows you all system variables
  env

Assignment: Modify your .bash_profile to perform the following

1. Display the following welcome message

   Welcome Peter
   Today is 14/01/2025
   ---
2. Set your command prompt to display your login name only
   [Peter]
3. Store session history of last 2000 commands
4. Add $HOME/bin to your PATH variable. So your scripts in $HOME/bin dir can be executed like UNIX commands from any other directory

---

-All system variables are exported so their values can be accessed from the child shell.

Example No.1

```
a=100
export a
b=200
ksh
echo $a
echo $b
```

Example No.2

```
p=Peter; export p
q=Peter
bash
echo $p  ← Will display Peter
ehco $q  ← will display nothing
```

Example No.3

```
export x='Pune is my home'
y='I love Pune'
csh
echo $x   ← will display Pune is my home
ehco $y   ← will display nothing
```

Example No.4

```
Script :a.sh
#!/bin/bash
abc=100; export abc
pqr=200
echo "In a.sh script value of abc = $abc"
echo "In a.sh script value of pqr = $abc"
/home/shekhar/b.sh
```

Example No.5

Script :b.sh

```
#!/bin/bash
echo "In b.sh script value of abc = $abc"
echo "In b.sh script value of pqr = $abc"
```

```
#!/bin/bash
#----------------------------
#Script Name: 5.sh
# Script to print system info
#----------------------------
a=`whoami`
b=`date`
c=$(hostname)
d=$(who | wc -l)
e=`free | head -2 | tail -1 | tr -s ' ' | cut -d' ' -f2`
clear
echo '----------------------------'
echo '            System Info'
echo "  1. Username       : $a"
echo "  2. Today's date   : $b"
echo "  2. Server name    : $c"
echo "  3. Logged in Users: $d"
echo "  4. Total RAM (KB) : $e"
echo '----------------------------'
```

```
#!/bin/bash
#----------------------------
#Script Name: 6.sh
# Script to print system info
#----------------------------
a=`whoami`
b=`date`
c=$(hostname)
d=$(who | wc -l)
e=`free | head -2 | tail -1 | tr -s ' ' | cut -d' ' -f2`
clear
echo -en "-----------------------------\n
                System Info
\t\t1. Username       : $a\n
\t\t2. Today's date   : $b\n
\t\t2. Server name    : $c\n
\t\t3. Logged in Users: $d\n
\t\t4. Total RAM (KB) : $e\n
\t\t-----------------------------\n"
```

```
#!/bin/bash
#Script Name : 7.sh
echo '----------------------------------------'
#A script to print all system variables
echo "Value of variable HOME = $HOME"
echo "Value of variable PWD = $PWD"
echo "Value of variable SHELL = $SHELL"
echo "Value of variable MAIL = $MAIL"
echo "Value of variable PATH = $PATH"
echo "Value of variable 0 = $0"
echo "Value of variable $ = $$"
echo "Value of variable USER = $USER"
echo "Value of variable HISTSIZE = $HISTSIZE"
echo "Value of variable HOSTNAME = $HOSTNAME"
echo '----------------------------------------'
```

```
#!/bin/bash
#8.sh
#This script will take backup using cp command
clear
echo '---------------------------------'
echo '              Welcome'
echo '              -------'
echo -en '\n\n'
read -p 'Please enter dir name to be backed up : ' dir_name
mkdir /tmp/${USER}_04012015
cp -r ${dir_name} /tmp/${USER}_04012015
echo 'backup finished'
echo "backup is in /tmp/${USER}_04012015"
echo '---------------------------------'
```

```
#!/bin/bash
#9.sh
clear
echo '---------------------------------'
echo '              Welcome'
echo '              -------'
echo -en '\n\n'
read -p 'Please enter dir name to be backed up : ' dir_name1
read -p 'Please enter dir name where the tar file should be kept : '
dir_name2
tar -cvf ${dir_name2}/${USER}_mybackup.tar ${dir_name1} &>
/dev/null
echo 'backup finished'
echo '---------------------------------'
```

Assignments:

Write a script which will display as follows

The name of your server = _____
The IP address of your server = _____
Total no. of users logged on = _____
Total RAM of your server = _____

Print the follwing frm your shell script

My userID is : _____
My hostname is : _____
List of users currently logged in : _____
My operating system is : _____

- If-else-then

  -Where to use conditions

  Example: to check if you have more than 50 users logged in.

  Example: If your free RAM is less than 1GB

  Example: If free space in any partition is less than 500MB

- Syntax of if-else-then

```
#!/bin/bash
# 10.sh
# ------------------------------------------------
# This shell scripts will explain the syntax of
# the 'if' statement.It will also explain how
# to perform numeric (integer) comparisons in
# bash shell scripting
# ------------------------------------------------
a=10
if [ $a -gt 5 ]; then
   echo "hello"
    whoami
fi
```

- Numeric Comparisons

  > -gt

  < -lt

  = -eq

  != -ne

  <= -le

  >= -ge

- Another syntax to perform the same

```
a=10
if [ $a -gt 5 ]
then
        echo "hello"
        whoami
fi
```

- Important Points:

1. There should be a space after [
   and space before]

2. The body of if will be executed only
   if the condition is true.

3. some people call it an expression
   and some people call if a condition.

```
#!/bin/bash
# 11.sh
# ---------------------------------------------
# This script accepts 2 numbers from the user
# and checks if the first number is greater than
# the second
# ---------------------------------------------
read -p ' Please enter first number: ' a
read -p ' Please enter second number: ' b
if [ $a -gt $b ]; then
    echo "$a is greater that $b"
fi
```

```
#!/bin/bash
# 12.sh
# ---------------------------------------------
# This script accepts 2 numbers from the user
# and checks which one is bigger and which one
# is smaller
# We will learn how to write the "else" part
# ---------------------------------------------
read -p ' Please enter first number: ' a
read -p ' Please enter second number: ' b
if [ $a -gt $b ]; then
   echo "$a is greater that $b"
else
   echo "$a is smaller that $b"
fi
```

```
#!/bin/bash
#13.sh
# ---------------------------------------------
# This script will accept one number from a user
# and will tell you if this number is either positive
# or negative or zero
# We will write this script using
#       1. elif
#       2. nested if
# ---------------------------------------------
read -p ' Please enter any number: ' a
if [ $a -gt 0 ]; then
   echo "$a is is a positive number"
elif [ $a -lt 0 ]; then
   echo "$a is is a negative number"
else
   echo "$a is neither positive nor negative (its is zero)"
fi
```

```
#!/bin/bash
#13a.sh
# ------------------------------------------------------------------------
# This script will accept one number from a user
# and will tell you if this number is either positive
# or negative or zero
# We will write this script using nested if
#-------------------------------------------------------------------------
read -p ' Please enter any number: ' a

if [ $a -gt 0 ]; then
   echo "$a is is a positive number"
else
        if [ $a -lt 0 ]; then
                echo "$a is is a negative number"
        else
                echo "$a is neither positive nor negative (its is zero)"
        fi
fi
```

- Important Points
  1. there is always a condition written after
     if and elif
  2. There is never a condition written after else
  3. elif and else are optional in an if statement
  4. There is only 1 fi to complete the if statement

```
#!/bin/bash
# 14.sh
#-------------------------------------------------------------------------
# This script checks if a number entered by a user is
# between 0 and 100
#-------------------------------------------------------------------------
a=0
```

```
k=100
clear   # Let us clear the screen before asking user a question
read -p 'Ennter any number: ' l
echo "Please wait..."
echo "I am checking if your number is in between $a and $k"

if [ $l -gt $a  -a $l -lt $k ]; then
    echo "$l is between $a and $k"
else
    echo "$l is NOT between $a and $k"
fi
```

- Logical AND table

Q1	Q2	Combined Q
T	T	T
T	F	F
F	T	F
F	F	F

```
#!/bin/bash
#-----------------------------------------------------
#15.sh
#This script will compare number between the range 0 to 100
read -p 'Ennter any number: ' l
a=0
k=100
if [ $l -gt $a  -o $l -lt $k ]; then
    echo "$l is in between $a and $k"
else
    echo "$l is in NOT between $a and $k"
fi
```

```
# ----------------------------------------
# The above script will not have any syntax
# error. But will give wrong results as
# it has a logical error
# ----------------------------------------
```

- Logical OR table

Q1	Q2	Combined Q
T	T	T
T	F	T
F	T	T
F	F	F

```
#!/bin/bash
# 16.sh
# ------------------------------------------------------
# Now we will learn how to compare string
# String comparative operators are as follows
#  <
#  >
#  =
#  !=
#  <=
#  >=
# ------------------------------------------------------
read -p 'Please enter your user name :' nm
if [ $nm = 'Narendra Modi' ]; then
        echo 'Hello Prime Minister narendraji. How are you'
elif [ $nm = 'Sonia Gandhi' ]; then
        echo 'Soniaji, Where are you these days.'
else
        echo 'Kay yaar kya chal raha hai'
fi
```

Note : nm=`echo "${nm}" | tr 'a-z' 'A-Z'`
The above command will help the string comparison.

```
#!/bin/bash
#17.sh
# --------------------------------------------------------------------------
# We will learn the use of escape char i.e. \ again
# You must use the escape char for >  and < signs as
# they have special meaning in shell scripting
# --------------------------------------------------------------------------
read -p 'Please enter your user name :' nm
read -p 'Please enter your friend\'s name :' fnm
if [ $nm \> $fnm ]; then
        echo "Alphabetically your name is greater than your friend's
name"
else
        echo "Alphabetically your name is smaller than your friend's
name"
fi
```

```
#!/bin/bash
#18.sh
clear
read -p 'What city do you live in ' city
if [ $city = 'Bombay' ]; then
   read -p "Have you been to Bollywood? (Y/N) : " response
   if [ $response = 'Y' ]; then
      read -p "Have you met Amitabh ? (Y/N) : " response
      if [ $response = 'Y' ]; then
         echo "You are a lucky fellow"
      else
         echo "You must go a meet him once"
      fi
   else
```

```
        echo "Why not? Don't you like to watch bollywood movies"
    fi
elif [ $city = 'Pune' ]; then
    echo "Great. I also live in Pune"
else
    echo "I have never been to $city. May be I can visit you some
time"
fi
```

```
#!/bin/bash
# 19.sh
#---------------------------------------
# Ask user to enter his birth date
# Calculate his age based on his birth date
# Tell him whether he/she is an  teenager,
# adult or a senior citizen
#---------------------------------------

read -p "Enter your birth date(dd/mm/yyyy) " dob
by=$(echo $dbo | cut -d"/" -f3 )
cy=$(date +%Y)
age=`expr $cy - $by`

echo "You are $age years old"
if [ $age -ge 60 ]; then
   echo "You are a senior citizen"
elif [ $age -le 19 ]; then
   echo "You are a teenager"
elif [ $age -gt 19 -a $age -lt 60  ]; then
   echo "You are an  adult"
fi
```

Note: The above script will display "You are a teenager when a user enters 10. This is a bug (13,14,15,16,17,18 and 19 are the only teenagers). Please modify the script to fix this bug. Anyone less than 13 will consider a kid and you should print the message "You are a kid".

```
#!/bin/bash
#20.sh
#----------------------------------------
# Send an email to user shekhar@gmail.com
# if there are more that 5 users logged
# on to you system.
# The subject of the email should be
# 'High System Load' and the body of the
# email should be the sorted list of all
# users who are currently logged on to
# your system
#----------------------------------------
no_of_users_logged_in=`who | wc -l`

if [ ${no_of_users_logged_in} -gt 5 ]; then
   subject='High System Load'
   who | cut -d ' ' -f1 |sort > /tmp/list_of_users.txt
   mail -s ${subject} shekhar@gmail.com </tmp/list_of_users.txt
   rm -f /tmp/list_of_users.txt  # remove the file. it is not required
anymore.
fi
```

```
#!/bin/bash
#21.sh
# ----------------------------------------
# let us learn what an "exit status" is
# ----------------------------------------
ls -l
echo " After running ls -l, Value of exit status = $?"
ls -z
echo " After running ls -z, Value of exit status = $?"
useradd shekhar
echo " After running useradd, Value of exit status = $?"
```

```
#!/bin/bash
#22.sh
# ----------------------------------------
# let us check the exit status
# ----------------------------------------
read -p " Enter the name of the user to be created: " u
useradd $u &> /dev/null
if [ $? -eq 0 ]; then
        echo "User $u created successfully"
elif [ $? -eq 4 ]; then
        echo " UID for this user already exists"
elif [ $? -eq 9 ]; then
        echo " User with same username already exists"
else
        echo "User cannot be created. Some error has occurred"
fi
```

```
#!/bin/bash
#23.sh
# --------------------------------------------------------------------------
# This script prompts user to enter any file name
# It checks if this file exists or not
# --------------------------------------------------------------------------
clear
read -p "Enter Any File Name: " fn

if [ -e ${fn} ]; then
  echo "${fn} exists"
else
  echo "${fn}  does NOT exist"
fi
```

- Some other UNIX unary operators

[-d /tmp]	Does /tmp directory exist
[-r /tmp/a.txt]	Is /tmp/a.txt readable file
[-w /tmp/a.txt]	Is /tmp/a.txt writable file
[-x /tmp/a.txt]	Is /tmp/a.txt executable file

-a file
    True if file exists.
-b file
    True if file exists and is a block special file.
-c file
    True if file exists and is a character special file.
-d file
    True if file exists and is a directory.
-e file
    True if file exists.
-f file
    True if file exists and is a regular file.

-k file
    True if file exists and its sticky bit is set.
-p file
    True if file exists and is a named pipe (FIFO).
-r file
    True if file exists and is readable.
-s file
    True if file exists and has a size greater than zero.
-w file
    True if file exists and is writable.
-x file
    True if file exists and is executable.
-L file
    True if file exists and is a symbolic link.
-S file
    True if file exists and is a socket.
-N file
    True if file exists and has been modified since it was last read.
file1 -nt file2
    True if file1 is newer (according to modification date) than file2, or if
    file1 exists and file2 does not.
file1 -ot file2
    True if file1 is older than file2, or if file2 exists and file1 does not.
file1 -ef file2
    True if file1 and file2 refer to the same device and inode numbers.
-z string
    True if the length of string is zero.

```
#!/bin/bash
#24.sh
#-------------------------------------------------------------------------
# We will learn what Command Line Arguments
# are
# CLA are used instead of read command
# to accept the user input
#-------------------------------------------------------------------------

clear
echo "Hello $1"
```

```
#!/bin/bash
#25.sh
#-------------------------------------------------------------------------
# We will see some other important
# variables related to command line arguments
# $0
# $#
# $*
#-------------------------------------------------------------------------

echo "Hello $1 $2 $3 $4"

echo "Hello $3 $1 $2 $4"

echo "Name of your shell script = $0"

echo "Total Number of CLA passwd to this script = $#"

echo "All Command Line Arguments together = $*"
```

```
#!/bin/bash
#26.sh
# ------------------------------------------------------------------------
--------
# This script will accept one command line
# argument from the user. This command line
# argument will be the name of a directory
# that needs to be backed up
# ------------------------------------------------------------------------
--------

echo "Please wait..."
echo "Backing up directory $1"
tar -cvf /tmp/backup.tar $1
echo "Backup of $1 finished"
```

```
#!/bin/bash
#27.sh
# ---------------------------------------------
# This script will take a tar backup of
# a directory which is passed as the first
# command line argument
#
# The backup (tar) file will be stored in
# /tmp directory
#
# The script will check if the user has passwd
# the command line argument or not before
# taking the tar backup.
# ---------------------------------------------
```

```
if [ $# -ne 1 ]; then
    echo "----------------------------------------"
    echo "ERROR:You must pass Directory name to backup"
    echo "Example: $0 /etc"
    echo "The above command will backup /etc directory "
    echo "----------------------------------------"
else
    echo "Please wait..."
    echo "Backing up $dir"
    tar -cvf /tmp/backup.tar $1 &> /dev/null
    echo "Backup of $1 Finished"
fi
```

```
#!/bin/bash
#28.sh
# Check the free ram of the server

free_ram=`free | head -2 | tail -1 | tr -s ' ' | cut -d ' ' -f4`
threshold=`expr 1024 \* 1024`
echo "Free RAM = $free_ram"
if [ $free_ram -lt $threshold ]; then
        echo 'You are running out of ram'
else
        echo "You have enough ram on the server"
fi
```

```
#!/bin/bash
#29.sh
# ---------------------------------------------
# This script will take a tar backup of a directory which is passed
#as the first command line argument The backup (tar) file will be
#stored in /tmp directory The script will check if the user has
#passwd the command line argument or not before
# taking the tar backup.
# ---------------------------------------------

if [ $# -ne 1 ]; then
    echo "-----------------------------------------"
    echo "ERROR:You must pass Directory name to backup"
    echo "Example: $0 /etc"
    echo "The above command will backup /etc directory "
    echo "-----------------------------------------"
else
    # --------------------------------
    # Check if the CLA is a valid direcrory
    # or not. Take the backup ONLY if it
    # is a valid direcrory
    # --------------------------------
    if [ ! -d $1 ]; then
        echo "-----------------------------------------"
        echo "ERROR: $1 is NOT a valid directory"
        echo "ERROR: Cannot take the backup"
        echo "-----------------------------------------"
    else
        echo "Please wait..."
        echo "Backing up $1"
        tar -cvf /tmp/backup.tar $1 &> /dev/null
        echo "Backup of $1 Finished"
    fi
fi
```

```
#!/bin/bash
# 30.sh
# This script will backup a dir to a tar file

clear
if [ $USER != 'shekhar' ]; then
        echo 'Only shekhar can run this script'
        exit 9
fi
echo "This script's process id = $$"
echo "If you want to stop this backup script"
echo "run the following command in another window"
echo "kill $$"

if [ $# -ne 2 ]; then
        echo 'Error: must pass 2 CLA'
        echo "Example: $0 /home /tmp"
        echo 'The above command will backup /home directory'
        echo 'And the backup file will be in /tmp directory'
        exit 99
fi


#read -p 'Which dir do u want to backup? ' dir1
#read -p 'In which dir do you want to keep the tar file? ' dir2

dir1=$1
dir2=$2


echo "value of dir1 = $dir1"
echo "value of dir2 = $dir2"

if [ -d $dir1 -a -d $dir2 ]; then
        echo "$dir1 and $dir2 are a valid directories"
        echo 'Please wait....'
```

```
        echo 'Taking backup'
        if [ ! -w $dir2 ]; then
                echo "Directory $dir2 is not writable"
                echo 'Therefore exiting'
                exit
        fi
        tar -c -f $dir2/aaaaa.tar $dir1 &> /dev/null
        exit_status=$?
        if [ $exit_status -eq 0 ]; then
                echo 'backup finished'
                exit 0
        else
                echo 'backup failed'
                exit 99999
        fi
else
        echo "Directory $dir1 is not a valid directory"
        echo "       OR"
        echo "Directory $dir2 is not a valid directory"
        exit 999
fi
```

```
#!/bin/bash
#31.sh
#---------------------------------------------
# There are 3 types of loops in bash scripting
#       1. for
#       2. while
#       3. until
#---------------------------------------------
for i in 1 2 3 4   5
do
   echo "Value of i = $i"
done
```

```
#!/bin/bash
#32.sh
#-----------------------------------------------
# seq 1 1 5
# seq 1 2 50
# seq 50 -2 1
# seq 1 5
#-----------------------------------------------
a=$(seq 1 1 50)
for abcd in $a
do
   echo "Value of abcd = $abcd"
done
```

```
#!/bin/bash
#33.sh
for i in 1 2 3 shekhar 6 10
do
   echo "Value of i = $i"
done


k=`ls *.sh`
for i in $k
do
       cp $i /tmp/$i.bak
done
```

```
#!/bin/bash
#34.sh
#-----------------------------------------------------------
# This script will find out a list of all logged in user
# by executing the 'who' command and it will write a
# message to each of those users using the 'write' command
#-----------------------------------------------------------
a=`who | cut -d' ' -f1 | sort | uniq`
for i in $a
do
   echo "Value of i = $i"
   echo -en "Hi $i , how are you\n\n -- $USER \n" > /tmp/$$.txt
   write $i < /tmp/$$.txt
done
rm -f /tmp/$$.txt
```

```
#!/bin/bash
#35.sh
a=shekhar
b='peter parker'
c=jacob
for i in $a $b $c
do
       echo "value of i = $i"
done
```

```
#!/bin/bash
#36.sh
a=shekhar
b='peter parker'
c=jacob
for i in "${a}" "${b}" "${c}"
do
       echo "value of i = $i"
done
```

```
#!/bin/bash
#37.sh
#-------------------------------------------------------
# This script will take a backup of all .conf files
# in /etc directory usng the cp command.
# The backup file name will be different than the
# original file name. e.g
#httpd.conf  ==> httpd.conf.01.01.2015
# These backup file will be stored in /tmp/bk dir
#-------------------------------------------------------
cd /etc
a=`ls *.conf`
for filename in $a
do
        a=`date +%d`
        b=`date +%m`
        c=`date +%Y`
        cp $filename /tmp/bk/$filename.$a.$b.$c
done
```

```
#!/bin/bash
#38.sh
#-----------------------------------------
# This script shuts down all oracle databases
#-----------------------------------------
echo 'shutdown immediate;' > /tmp/$$.sql
echo 'exit; ' >> /tmp/$$.sql
a=`ps -ef | grep  ora_pmon | tr -s ' '| grep ^oracle | cut -d' ' -f8 | cut -
d'_' -f3`
for dbname in $a
do
        export ORACLE_SID=$dbname
        sqlplus '/ as sysdba' < $$.sql  &> /tmp/$$.err
        err_count=`grep ^ORA- /tmp/$$.err | wc -l `
```

```
        if [ $err_count -eq 0 ]; then
                mail -s 'Database Shutdown SUccessfully' $1 <
/tmp/$$.err
        else
                mail -s "$dbname: Database Shutdown Failed" $1 <
/tmp/$$.err

        fi

done
rm -f /tmp/$$.sql
```

```
#!/bin/bash
#39.sh
#-----------------------------------------
# Drop all tables listed in a file named t.txt
# Pass 3 CLA
#       1 - database name
#       2 - db username
#       3 - db password
#-----------------------------------------
export ORACLE_SID=$1
a=`cat t.txt`
> $$.sql
for tablename in $a
do
        echo "drop table $tablename;" >> $$.sql
done
echo "exit;" >> $$.sql
sqlplus $2/$3 < $$.sql  &> /tmp/$$.err
k=`grep ^ORA- /tmp/$$.err | wc -l`
if [ $k-eq 0 ]; then
        echo "All tables dropped successfully"
else
```

```
        echo "Some errors while dropping tables"
        cat /tmp/$$.err
fi

rm -f $$.sql
```

Assignments for Oracle DBAs:

Assignment No.1

A file named users.txt cotains a list of database users. Please write a script named lock_user.sh which will lock all of the users listed in the users.txt file.

Assignment No.2

/etc/oratab file contains list of all Oracle databases. Your job is to write a script to shutdown each database listed in that script. Please check the format of this file before writing the script.

```
#!/bin/bash
#40.sh
a=shekhar
b='peter parker'
c=jacob
for i in ${a} ${b} ${c}
do
        echo "value of i = $i"
done
```

```
#!/bin/bash
#41.sh
a=shekhar
b='peter parker'
c=jacob
for i in "${a}" "${b}" "${c}"
do
        echo "value of i = $i"
done
```

```
#!/bin/bash
#For UNIX Amins only
#42.sh
a=`chkconfig --list | grep '0:'| cut -d' ' -f1`
for i in $a
do
        echo '--------------------------------------------------'
        echo "Name of Service = $i"
        run3=`chkconfig --list $i | grep '3:on' | wc -l`
        run5=`chkconfig --list $i | grep '5:on' | wc -l`
        echo "Value of run3 = $run3"
        echo "Value of run5 = $run5"
        if [ $run3 -eq 1 -a $run5 -eq 1  ]; then
```

```
                echo "Changing run level for Service : $i"
                #chkconfig --level 5 $i off
        fi
done
```

```
#!/bin/bash
#Script for UNIX Admins
#43.sh
for i in `cut -d':' -f1 /etc/passwd`
do
        echo "username - $i"
        uid=`grep ^$i /etc/passwd | cut -d':' -f3`
        if [ $uid -gt 499 ]; then
                #chage -d 0 $i
                rand=`date +%d%m%Y%N`
                echo "${i}${rand}" > /tmp/$$.passwd
                #passwd --stdin $i < /tmp/$$.passwd
                echo "users = $i    Password=${i}${rand}"
        fi

done
```

```
#!/bin/bash
#44.sh
i=1
while [ $i -le 10 ]
do
   echo "Value of i = $i"
   i=`expr $i + 1`
done
```

```
#!/bin/bash
#44a.sh
i=10
while [ $i -ge 1 ]
do
   echo "value of i = $i"
   i=`expr $i - 1`
done
```

```
#!/bin/bash
#44b.sh
i=10
while [ 9399 -lt  999999 ]
do
   echo "value of i = $i"
   i=`expr $i - 1`
done
```

```
#!/bin/bash
#44c.sh
i=10
while true
do
   echo "value of i = $i"
   i=`expr $i - 1`
done
```

```
#!/bin/bash
#46.sh
trap "echo 'Trapped INT SIgnal....' " 2
trap "echo 'Trapped TERM SIgnal....' " 15
echo "starting infinite looop"
i=1
while [ 1 -eq 1 ]
do

    echo "$i. sleeping for 1 sec..."
    sleep 1
    i=$(( $i + 1 ))

done
```

Assigment: For Oracle DBAs

Write a script to take a backup of a database
using expdp command.
This script should remove the dump file if
user user types ctrl+c while the script is
being run.

Hint:
trap "rm backup.dmp; exit " 2

```
#!/bin/bash
#45.sh
i=1
#until [ $i -gt 10 ]
until [ 98 -gt 100 ]
do
   echo "Value of i = $i"
   i=`expr $i + 1`
done
```

```
#!/bin/bash
#47.sh
# ----------------------------------------------------------------------------
# This script shows how to use functions in your shell script.  A
function #is declared first  and then it is called
# Let us declare a function named
# printhello
# ----------------------------------------------------------------------------
function printhello
{
   echo "------------------------"
   echo "Hello There"
   echo "------------------------"
}
# -------------------------------------
# Now let us start writing the main  Shell script. We will call the
#above  declared function in this shell script
# -------------------------------------
clear
date
printhello
cal
printhello
```

```
#!/bin/bash
#48.sh

function printbyebye
{
        echo '---------------------------------'
        read -p 'Are you sure you want to exit (y/n) ' resp
        if [ $resp = 'Y' -o $resp = 'y' ]; then
                echo "Bye Bye"
                echo '---------------------------------'
                exit
        fi
        echo '---------------------------------'
}
trap "printbyebye" 2
trap "echo 'Trapped TERM SIgnal....' " 15
echo "starting infinite looop"
i=1
while [ 1 -eq 1 ]
do
   echo "$i. sleeping for 1 sec..."
   sleep 1
   i=$(( $i + 1 ))
done
```

```
#!/bin/bash
#49.sh
# -------------------------------------
# This script shows how to pass command line arguments to a
#function. Let us declare a function named printhello. We will
#pass one command line argument to this function.
#The first CLA is stored in a variable $1
# -------------------------------------
function printhello
```

```
{
   echo "-----------------------"
   echo "Hello $1"
   echo "-----------------------"
}
# -------------------------------------
# Now let us start writing the main  Shell script. We will call the
#above  declared function in this shell script
# -------------------------------------
clear
date
printhello Peter # Peter is passed as 1st CLA
cal
printhello Sadanand # Sadanand is passed is 1st CLA
```

```
#!/bin/bash
#50.sh

# -------------------------------------
# First let us source the file that
# contains the function/s that we want to
# call in this shell script
# -------------------------------------
. /var/ftp/pub/shellscript/functions/myfunc.sh

# -------------------------------------
# Now let us start writing the main
# Shell script. We will call the above
# declared function in this shell script
# -------------------------------------
clear
date
printhello Peter # Peter is passed as 1st CLA
cal
printhello Sadanand # Sadanand is passed is 2nd CLA
```

```
#!/bin/bash
#-------------------------------------------------------------------------------
#Add the following function in ~/.bashrc
#51.sh
#-------------------------------------------------------------------------------

function backuphome
{
        clear
        echo '----------------------------------------'
        echo 'Please wait.....'
        echo "Backing up $HOME directory"
        tar -cvf /tmp/${USER}_`date +%d_%m_%Y`.tar  $HOME &> /dev/null
        if [ $? -eq 0 ]; then
                echo 'backup successful'
                echo "backup file : /tmp/${USER}_`date +%d_%m_%Y`.tar"
        else
                echo 'backup failed'
        fi
        echo '----------------------------------------'
}
```

```
#!/bin/bash
#52.sh
# ------------------------------------------------
# This shell script will provide a menu driven
# program for mathemaical calculations
# In short it is calculator program
# Your task is:
#  Write the code in $HOME/bin/functions.sh
# ------------------------------------------------

# Source the file that has functions
. $HOME/bin/functions.sh

while [ 1 -eq 1 ]
do
clear
echo "--------------------------------"
echo "        Welcome - Calculator "
echo -e "\n\n\n"
echo "a> Add two numbers "
echo "s> Substract two numbers"
echo "m> Multiply two numbers"
echo "d> Divide two numbers"
echo "e> Exit the program"
echo -e "\n\n"
read -p "Please Enter your choice: " ch
echo "--------------------------------"
case $ch in
  a|A) add_numbers
       ;;
  s|S)
       clear
       read -p "Enter first number: " a
       read -p "Enter second number: " b
```

```
        substract_numbers a b
        read -p "Enter to continue" aa
        ;;
   m|M) multilpy_numbers
        ;;
   d|D) divide_numbers
        ;;
   e|E) echo "Bye Bye"
        break
        ;;
   *)   echo "Entered invalid choice"
esac
done
function add_numbers
{
    ans=$(($1+$2))
    echo "--------------------"
    echo -e "     Addition\n\n"
    echo "$1 + $2 = $ans"
    echo "--------------------"
}
```

Assignment:

Create a command called myip to display the IP address ← alias

Create a command called add to add two numbers ← function in .bashrc

Example : add 10 20

Create a command called createuser to create a UNIX user and set his password to focus123 ← using a shell script and PATH variable

Example : createuser shekhar