Assignment No. 01(B)

Date:

TITLE: Shell programming

Write a program to implement an address book with options given below:
a) Create address book. b) View address book. c) Insert a record. d) Delete a record.
e) Modify a record. f) Exit.

OBJECTIVE:

- Study the concepts of Shell Scripting in Linux.
- Study various command in Linux.

SOFTWARE REQUIREMENTS:

1. Ubuntu 16.04
2. GNU C Compiler

THEORY:

What is Linux Shell ?

Computer understand the language of 0's and 1's called binary language.

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

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

Shell is not part of system kernel, but uses the system kernel to execute programs, create files etc.

Several shell available with Linux including:

Shell Name	Developed by	Where	Remark
BASH (Bourne-Again SHell)	Brian Fox and Chet Ramey	Free Software Foundation	Most common shell in Linux. It's Freeware shell.
CSH (C SHell)	Bill Joy	University of California (For BSD)	The C shell's syntax and usage are very similar to

			the C programming language.
KSH (Korn SHell)	David Korn	AT & T Bell Labs	--
TCSH	See the man page. Type $ man tcsh	--	TCSH is an enhanced but completely compatible version of the Berkeley UNIX C shell (CSH).

To find all available shells in your system type following command:
$ cat /etc/shells

To find your current shell type following command
$ echo $SHELL

What is Shell Script ?

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

Shell script defined as:

"*Shell Script is **series of command** written **in plain text file**. Shell script is just like batch file is MS-DOS but have more power than the MS-DOS batch file.*"

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.

How to write shell script

Following steps are required to write shell script:

(1) Use any editor like vi or mcedit 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 your-script-name
$ chmod 755 your-script-name
```

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

(3) Execute your script as
syntax:
bash your-script-name
sh your-script-name
./your-script-name

Examples:

```
$ bash bar
$ sh bar
$ ./bar
```

NOTE In the last syntax ./ means current directory, But only . (dot) means execute given command file in current shell without starting the new copy of shell, The syntax for . (dot) command is as follows
Syntax:
. command-name

Example:

```
$ . foo
```

Now you are ready to write first shell script that will print "Knowledge is Power" on screen. See the common vi command list , if you are new to vi.

```
$ vi first
#
# My first shell script
#
clear
echo "Knowledge is Power"
```

After saving the above script, you can run the script as follows:

```
$ ./first
```

This will not run script since we have not set execute permission for our script *first*; to do this type command

```
$ chmod 755 first
$ ./first
```

First screen will be clear, then Knowledge is Power is printed on screen.

Script Command(s)	Meaning
$ vi first	Start vi editor
# # My first shell script #	# followed by any text is considered as comment. Comment gives more information about script, logical explanation about shell script. *Syntax:* # comment-text
clear	clear the screen
echo "Knowledge is Power"	To print message or value of variables on screen, we use echo command, general form of echo command is as follows *syntax:* echo "Message"

Variables in Shell

To process our data/information, data must be kept in computers RAM memory. RAM memory is divided into small locations, and each location had unique number called memory location/address, which is used to hold our data. Programmer can give a unique name to this memory location/address called memory variable or variable (Its a named storage location that may take different values, but only one at a time).

In Linux (Shell), there are two types of variable:
(1) **System variables** - Created and maintained by Linux itself. This type of variable defined in CAPITAL LETTERS.
(2) **User defined variables (UDV)** - Created and maintained by user. This type of variable defined in lower letters.

You can see system variables by giving command like **$ set**, some of the important System variables are:

System Variable	Meaning
BASH=/bin/bash	Our shell name
BASH_VERSION=1.14.7(1)	Our shell version name
COLUMNS=80	No. of columns for our screen
HOME=/home/vivek	Our home directory
LINES=25	No. of columns for our screen
LOGNAME=students	students Our logging name
OSTYPE=Linux	Our Os type
PATH=/usr/bin:/sbin:/bin:/usr/sbin	Our path settings
PS1=[\u@\h \W]\$	Our prompt settings
PWD=/home/students/Common	Our current working directory
SHELL=/bin/bash	Our shell name
USERNAME=vivek	User name who is currently login to this PC

How to define User defined variables (UDV)

To define UDV use following syntax

Syntax:

variable name=value

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

Example:

```
$ no=10# this is ok
$ 10=no# Error, NOT Ok, Value must be on right side of = sign.
```

To define variable called 'vech' having value Bus

```
$ vech=Bus
```

To define variable called n having value 10

```
$ n=10
```

Rules for Naming variable name (Both UDV and System Variable)

(1) Variable name must begin with Alphanumeric character or underscore character (_), followed by one or more Alphanumeric character. For e.g. Valid shell variable are as follows

HOME

SYSTEM_VERSION

vech

no

(2) Don't put spaces on either side of the equal sign when assigning value to variable. For e.g. In 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
```

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

```
$ no=10
$ No=11
$ NO=20
$ nO=2
```

Above all are different variable name, 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
```

(4) 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=""
Try to print it's value by issuing following command
$ echo $vech
Nothing will be shown because variable has no value i.e. NULL variable.

(5) Do not use **?,*** etc, to name your variable names.

How to print or access value of UDV (User defined variables)

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 command as follows
$ echo $n

Caution: Do not try **$ echo vech**, as it will print vech instead its value 'Bus' and **$ echo n**, as it will print n instead its value '10', You must *use $ followed by variable name*.

echo Command

Use echo command to display text or value of variable.

echo [options] [string, variables...]
Displays text or variables value on screen.
Options
-n Do not output the trailing new line.
-e Enable interpretation of the following backslash escaped characters in the strings:
\a alert (bell)
\b backspace
\c suppress trailing new line
\n new line
\r carriage return
\t horizontal tab
\\ backslash

For e.g. **$ echo -e "An apple a day keeps away \a\t\tdoctor\n"**

Shell Arithmetic

Use to perform arithmetic operations.

Syntax:
expr op1 math-operator op2

Examples:

```
$ expr 1 + 3
$ expr 2 - 1
$ expr 10 / 2
$ expr 20 % 3
$ expr 10 ¥* 3
$ echo `expr 6 + 3`
```

Note:
expr 20 %3 - Remainder read as 20 mod 3 and remainder is 2.
expr 10 * 3 - Multiplication use * and not * since its wild card.

For the last statement not the following points

(1) First, before expr keyword we used ` (back quote) sign not the (single quote i.e. ') sign. Back quote is generally found on the key under tilde (~) on PC keyboard OR to the above of TAB key.

(2) Second, expr is also end with ` i.e. back quote.

(3) Here expr 6 + 3 is evaluated to 9, then echo command prints 9 as sum

(4) Here if you use double quote or single quote, it will NOT work
For e.g.
$ echo "expr 6 + 3" # It will print expr 6 + 3
$ echo 'expr 6 + 3' # It will print expr 6 + 3

About Quotes

There are three types of quotes

Quotes	Name	Meaning
"	Double Quotes	"Double Quotes" - Anything enclose in double quotes removed meaning of that characters (except \ and $).
'	Single quotes	'Single quotes' - Enclosed in single quotes remains unchanged.
`	Back quote	`Back quote` - To execute command

Example:
$ echo "Today is date"
Can't print message with today's date.
$ echo "Today is `date`".
It will print today's date as, Today is Tue Jan ....,Can you see that the `date` statement uses back quote?

Exit Status

By default in Linux if particular command/shell script is executed, it return two type of values which is used to see whether command or shell script executed is successful or not.

(1) If return *value is zero* (0), command is successful.
(2) If return *value is nonzero*, command is not successful or some sort of error executing command/shell script.

This value is know as ***Exit Status***.

But how to find out exit status of command or shell script?
Simple, to determine this exit Status you can use **$?** special variable of shell.

For e.g. (This example assumes that **unknow1file** doest not exist on your hard drive)
$ rm unknow1file
It will show error as follows
rm: cannot remove `unkowm1file': No such file or directory
and after that if you give command
$ echo $?
it will print nonzero value to indicate error. Now give command
$ ls
$ echo $?
It will print 0 to indicate command is successful.

The read Statement

Use to get input (data from user) from keyboard and store (data) to variable.
Syntax:
read variable1, variable2,...variableN

Following script first ask user, name and then waits to enter name from the user via keyboard. Then user enters name from keyboard (after giving name you have to press ENTER key) and entered name through keyboard is stored (assigned) to variable fname.

```
$ vi sayH
#
#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:

```
$ chmod 755 sayH
$ ./sayH
```

Your first name please: ***vivek***
Hello vivek, Lets be friend!

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 or if exit status
        of condition is 0 (zero)
        ...
        ...
fi
```

Condition is defined as:
"*Condition is nothing but comparison between two values.*"

For compression you can use test or [expr] statements or even exist status can be also used.

Expreession is defined as:
"*An expression is nothing but combination of values, relational operator (such as >,<, <> etc) and mathematical operators (such as +, -, / etc).*"

Following are all examples of expression:
5 > 2
3 + 6
3 * 65
a < b
c > 5
c > 5 + 30 -1

Type following commands (assumes you have file called **foo**)
$ cat foo
$ echo $?
The cat command return zero(0) i.e. exit status, on successful, this can be used, in if condition as follows, Write shell script as

```
$ cat > showfile
#!/bin/sh
#
#Script to print file
#
if cat $1
then
echo -e "¥n¥nFile $1, found and successfully echoed"
fi
```

Run above script as:
$ chmod 755 showfile
$./showfile foo
Shell script name is showfile ($0) and foo is argument (which is $1).Then shell compare it as follows:
if cat $1 which is expanded to if cat foo.

Detailed explanation
if cat command finds foo file and if its successfully shown on screen, it means our cat command is

successful and its exist status is 0 (indicates success), So our if condition is also true and hence statement echo -e "\n\nFile $1, found and successfully echoed" is proceed by shell. Now if cat command is not successful then it returns non-zero value (indicates some sort of failure) and this statement echo -e "\n\nFile $1, found and successfully echoed" is skipped by our shell.

test command or [expr]

test command or [expr] is used to see if an expression is true, and if it is true it return zero(0), otherwise returns nonzero for false.

Syntax:

test expression OR [expression]

Example:

Following script determine whether given argument number is positive.

```
$ cat > ispostive
#!/bin/sh
#
# Script to see whether argument is positive
#
if test $1 -gt 0
then
echo "$1 number is positive"
fi
```

if...else...fi

If given condition is true then command1 is executed otherwise command2 is executed.

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
```

Loops in Shell Scripts

Loop defined as:

"*Computer can repeat particular instruction again and again, until particular condition satisfies. A group of instruction that is executed repeatedly is called a loop.*"

Bash supports:

- ❿ for loop
- ❿ while loop

Note that in each and every loop,

(a) First, the variable used in loop condition must be initialized, then execution of the loop begins.

(b) A test (condition) is made at the beginning of each iteration.

(c) The body of loop ends with a statement that modifies the value of the test (condition) variable.

for Loop

Syntax:

```
for { variable name } in { list }
do
        execute one for each item in the list until the list is
        not finished (And repeat all statement between do and done)
done
```

Before try to understand above syntax try the following script:

```
$ cat > testfor
for i in 1 2 3 4 5
do
echo "Welcome $i times"
done
```

As you see the if statement can nested, similarly loop statement can be nested. You can nest the for loop. To understand the nesting of for loop see the following shell script.

```
$ vi nestedfor.sh
for (( i = 1; i <= 5; i++ ))      ### Outer for loop ###
do

    for (( j = 1 ; j <= 5; j++ )) ### Inner for loop ###
    do
          echo -n "$i "
    done

  echo "" #### print the new line ###

done
```

while loop

Syntax:

```
while [ condition ]
do
      command1
      command2
      command3
      ..
      ....
 done
```

Loop is executed as long as given condition is true.

The case Statement

The case statement is good alternative to Multilevel if-then-else-fi statement. It enable you to match several values against one variable. Its easier to read and write.

Syntax:

```
case  $variable-name  in
     pattern1)   command
                    ...
                    ..
                    command;;
     pattern2)   command
                    ...
                    ..
                    command;;
     patternN)   command
                    ...
                    ..
                    command;;
     *)             command
                    ...
                    ..
                    command;;
esac
```

The *$variable-name* is compared against the patterns until a match is found. The shell then executes all the statements up to the two semicolons that are next to each other. The default is *) and its executed if no match is found. For e.g. write script as follows:

```
$ cat > car
#
# if no vehicle name is given
# i.e. -z $1 is defined and it is NULL
#
# if no command line arg

if [ -z $1 ]
then
  rental="*** Unknown vehicle ***"
elif [ -n $1 ]
then
# otherwise make first arg as rental
  rental=$1
fi

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 can not gat a $rental for you";;
esac
```

Conclusion: Thus we have implemented an address book with given options