ASSIGNMENT NO.: 01

AIM:

To study Basic Linux Commands and Shell programming.

A. Study of Basic Linux Commands: echo, ls, read, cat, touch, test, loops, arithmetic comparison, conditional loops, grep, sed etc.

B. 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

PREREQUISITE:

1. C Programming
2. Fundamentals of Data Structure

OBJECTIVE:

To study

1. Basic Linux commands
2. Shell script

THEORY:

Linux Commands

The Linux command is a utility of the Linux operating system. All basic and advanced tasks can be done by executing commands. The commands are executed on the Linux terminal. The terminal is a command-line interface to interact with the system, which is similar to the command prompt in the Windows OS. Commands in Linux are case-sensitive

Basic Linux Commands

❖ mkdir Command

The mkdir command is used to create a new directory under any directory

Syntax:

mkdir <directory name>

❖ rmdir Command

The rmdir command is used to delete a directory.

Syntax:

```
rmdir <directory name>
```

- ## cd Command

  The cd command is used to change the current directory.

  Syntax:

  ```
  cd <directory name>
  ```

- ## touch Command

  The touch command is used to create empty files. We can create multiple empty files by executing it once.

  Syntax:

  ```
  touch <file name>
  touch <file1>  <file2>
  ```

- ## cat Command

  The cat command is a multi-purpose utility in the Linux system. It can be used to create a file, display content of the file, copy the content of one file to another file, and more.

  Syntax:

  ```
  cat [OPTION]... [FILE]..
  ```

  To create a file, execute it as follows:

  ```
  cat > <file name>
  // Enter file content
  ```

  Press "CTRL+ D" keys to save the file. To display the content of the file, execute it as follows:

  ```
  cat <file name>
  ```

- ## rm Command

  The rm command is used to remove a file.

  Syntax:

```
rm <file name>
```

- ❖ ls Command

  The ls command is used to display a list of content of a directory.

  Syntax:

  ```
  ls
  ```

- ❖ cat Command

  The cat command is also used as a filter. To filter a file, it is used inside pipes.

  Syntax:

  ```
  cat <fileName> | cat or tac | cat or tac |
  ```

- ❖ grep Command

  The grep is the most powerful and used filter in a Linux system. The 'grep' stands for "global regular expression print." It is useful for searching the content from a file. Generally, it is used with the pipe.

  Syntax:

  ```
  command | grep <searchWord>
  ```

- ❖ sed command

  The sed command is also known as stream editor. It is used to edit files using a regular expression. It does not permanently edit files; instead, the edited content remains only on display. It does not affect the actual file.

  Syntax:

  ```
  command | sed 's/<oldWord>/<newWord>/'
  ```

- ❖ sort Command

  The sort command is used to sort files in alphabetical order.

  Syntax:

  ```
  sort <file name>
  ```

❖ cal Command

The cal command is used to display the current month's calendar with the current date highlighted.

Syntax:

cal<

❖ read command

The read command is used to read from a file descriptor. This command read up the total number of bytes from the specified file descriptor into the buffer. If the number or count is zero then this command may detect the errors. But on success, it returns the number of bytes read. Zero indicates the end of the file. If some errors found then it returns -1

Syntax:

read

❖ echo command

echo command in linux is used to display line of text/string that are passed as an argument . This is a built in command that is mostly used in shell scripts and batch files to output status text to the screen or a file.

Syntax:

echo [option] [string]

❖ test command

Test is used as part of the conditional execution of shell commands. test exits with the status determined by EXPRESSION. Placing the EXPRESSION between square brackets ([and]) is the same as testing the EXPRESSION with test. To see the exit status at the command prompt, echo the value "$?" A value of 0 means the expression evaluated as true, and a value of 1 means the expression evaluated as false

Syntax:

test EXPRESSION

[EXPRESSION]

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 known as shell script. Shell Script is series of command written in plain text file. This manual is meant as a brief introduction to features found in Bash.

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 known 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 does 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.

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:
To define variable called n having value 10
$ n=10

To print or access UDV use following syntax
Syntax:
$variablename

$ n=10

To print contains of variable 'n' type command as follows
$ echo $n

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`".

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
(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.

(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.

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.

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!"
```

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
HOME=/home/vivek	Our home directory
LOGNAME=students	students Our logging name
OSTYPE=Linux	Our Os type
PATH=/usr/bin:/sbin:/bin:/usr/sbin	Our path settings
PWD=/home/students/Common	Our current working directory
SHELL=/bin/bash	Our shell name

You can print any of the above variables contains as follows:

$ echo $HOME

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.

```
if test $1 -gt 0
then
echo "$1 number is positive"
fi
```

test or [expr] works with

1.Integer (Number without decimal point)

2.File types

3.Character strings

For Mathematics, use following operator in Shell Script

Operator in Shell Script	Meaning	Normal Arithmetical	But in Shell	
-eq	is equal to	5 == 6	if test 5 -eq 6	if [5 -eq 6]
-ne	is not equal to	5 != 6	if test 5 -ne 6	if [5 -ne 6]
-lt	is less than	5 < 6	if test 5 -lt 6	if [5 -lt 6]
-le	is less than or equal to	5 <= 6	if test 5 -le 6	if [5 -le 6]
-gt	is greater than	5 > 6	if test 5 -gt 6	if [5 -gt 6]
-ge	is greater than or equal to	5 >= 6	if test 5 -ge 6	if [5 -ge 6]

NOTE: == is equal, != is not equal.

For string Comparisons use

Operator	Meaning
string1 = string2	string1 is equal to string2
string1 != string2	string1 is NOT equal to string2
string1	string1 is NOT NULL or not defined
-n string1	string1 is NOT NULL and does exist
-z string1	string1 is NULL and does exist

Shell also test for file and directory types

	Meaning
-s file	Non empty file
-f file	Is File exist or normal file and not a directory
-d dir	Is Directory exist and not a file
-w file	Is writeable file
-r file	Is read-only file
-x file	Is file is executable

Logical Operators:

Logical operators are used to combine two or more condition at a time

Operator	Meaning
! expression	Logical NOT
expression1 -a expression2	Logical AND
expression1 -o expression2	Logical OR

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.

Loops in Shell Scripts

Bash supports:

1) for loop

2) while loop

while :

The syntax of the while is:

```
          while test-commands
          do
commands
done
```

Execute *commands* as long as *test-commands* has an exit status of zero.

for :

The syntax of the for is:

```
          for variable in list
          do
commands
         done
```

Each white space-separated word in list is assinged to variable in turn and commands executed until list is exhausted.

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.

CONCLUSION:

Thus in shell script we can write series of commands and execute as a single program.

Answer the following questions.

1. What are different types of shell & differentiate them.
2. Explain Exit status of a command.
3. Explain a) User define variables.
4. System variables.
5. What is man command?
6. Explain test command.
7. Explain how shell program get executed.
8. Explain syntax of if-else, for, while, case.
9. Explain use of functions in shell and show it practically.
10. Write a menu driven shell script to execute different commands with options.

OUTPUT:

(Attach Screenshots of your output in sequence)