

# **DEV BHOOMI INSTITUTE OF TECHNOLOGY**

Department of Computer Science and Engineering

**Year: 2nd**

**Semester: 4th**



UNIX AND SHELL PROGRAMMING LAB-PCS-402

## **LAB MANUAL**

Prepared By:

HOD(CSE)

# DEV BHOOMI INSTITUTE OF TECHNOLOGY


Department of Computer Science and Engineering

## INDEX

<b>S. No</b>	<b>Practical's Name</b>	<b>S/w used</b>	<b>Remark</b>
1	<b>Execution of various file/directory handling commands.</b>	UNIX OS /ANY Flavor of Linux	
2	<b>Simple shell script for basic arithmetic and logical calculations.</b>	UNIX OS /ANY Flavor of Linux	
3	<b>Shell scripts to check various attributes of files and directories</b>	UNIX OS /ANY Flavor of Linux	
4	<b>Shell scripts to perform various operations on given strings.</b>	UNIX OS /ANY Flavor of Linux	
5	<b>Shell scripts to explore system variables such as PATH, HOME etc.</b>	UNIX OS /ANY Flavor of Linux	
6	<b>Shell scripts to check and list attributes of processes.</b>	UNIX OS /ANY Flavor of Linux	
7	<b>Execution of various system administrative commands.</b>	UNIX OS /ANY Flavor of Linux	
8	<b>Write awkscript that uses all of its features.</b>	UNIX OS /ANY Flavor of Linux	
9	<b>Use sedinstruction to process /etc/password file.</b>	UNIX OS /ANY Flavor of Linux	
10	<b>Write a shell script to display list of users currently logged in.</b>	UNIX OS /ANY Flavor of Linux	

# DEV BHOOMI INSTITUTE OF TECHNOLOGY

## LAB MANUAL

	<b>Course Name:</b> System Administration Lab	<b>Experiment No. 1</b>	
	<b>Course Code :</b> PCS-402 <b>Faculty :</b> Ms. RituRawat	<b>Branch:</b> CSE	<b>Semester:</b> IV

**Objective:** Execution of various file/directory handling commands.

**Tools:** UNIX operating system/any flavor of Linux.

### **Procedure:**

Analyzing the Problem:

Start the Linux and enter the user name and password.

Now write startx and after that open the terminal.

At the terminal try the different commands and see the output.

### **Designing the Solution:**

At the terminal first perform the command without and with the different Options available for it. The exercises in this lab cover the usage of some of the most basic system utilities that users and administrators alike need to be familiar with. Most of the commands are used in navigating and manipulating the file system. The file system is made up of files and directories.

### **Theory:**

**1) pwdCommand:** pwd Print Working Directory. pwd command prints the full filename of the current working directory.

**SYNTAX:**

The Syntax is pwd [options]

**2) cdCommand:**

cd command is used to change the directory.

SYNTAX: The Syntax is cd [directory | ~ | ./ | ../ | ]

### **3) lsCommand:**

ls command lists the files and directories under current working directory.

SYNTAX:

The Syntax is

ls [OPTIONS]... [FILE]

OPTIONS:

Lists all the files, directories and their mode, Number of links,

l

owner of the file, file size, Modified date and time and filename. t Lists in order of last modification time. a Lists all entries including hidden files. d Lists directory files instead of contents. p Puts slash at the end of each directories. u List in order of last access time. i Display inode information.

### **4) rmCommand:**

rmlinux command is used to remove/delete the file from the directory.

SYNTAX:

The Syntax is rm [options..] [file | directory]

OPTIONS:

f Remove all files in a directory without prompting the user.

i Interactive. With this option, rm prompts for confirmation before removing any files.

### **5) mvCommand:**

mv command which is short for move. It is used to move/rename file from one directory to another. mv command is different from cp command as it completely removes the file from the source and moves to the directory specified, where cp command just copies the content from one file to another.

SYNTAX:

The Syntax is mv [f] [i] oldnamenewname

OPTIONS:

f This will not prompt before overwriting (equivalent to reply=yes). mvf will move the file(s) without prompting even if it is writing over an existing target.

iPrompts before overwriting another file.

#### **6) catCommand:**

catlinux command concatenates files and print it on the standard output.

SYNTAX: The Syntax is cat [OPTIONS] [FILE]...

OPTIONS:

A Show all. b Omits line numbers for blank space in the output.

E Displays a \$ (dollar sign) at the end of each line. n Line numbers for all the output lines.

#### **7) cmpCommand:**

cmplinux command compares two files and tells you which line numbers are different.

SYNTAX:

The Syntax is cmp [options..] file1 file2

OPTIONS:

c Output differing bytes as characters.

l Print the byte number (decimal) and the differing byte values (octal) for each difference. s

Prints nothing for differing files, return exit status only.

#### **8) cpCommand:**

cp command copy files from one location to another. If the destination is an existing file, then the file is overwritten; if the destination is an existing directory, the file is copied into the directory (the directory is not overwritten).

SYNTAX:

The Syntax is

cp [OPTIONS]... SOURCE DEST

#### **10) echoCommand:**

echo command prints the given input string to standard output.

SYNTAX: The Syntax is echo [options..] [string]

#### **11)mkdirCommand:**

mkdir command is used to create one or more directories.

SYNTAX:

The Syntax is mkdir [options] directories

OPTIONS:

## **12) pasteCommand:**

paste command is used to paste the content from one file to another file. It is also used to set column format for each line.

### **SYNTAX:**

The Syntax is paste [options]

- m Set the access mode for the new directories.
- p Create intervening parent directories if they don't exist.
- v Print help message for each directory created.

### **OPTIONS:**

- s Paste one file at a time instead of in parallel.
- d Reuse characters from LIST instead of TABs .

## **13) rmdirCommand:**

rmdir command is used to delete/remove a directory and its subdirectories.

### **SYNTAX:**

The Syntax is rmdir [options..] Directory

### **OPTIONS:**


Allow users to remove the directory dir name and its parent p directories which become empty.

## **Outcome:**

In this way we can run different file and directory handling commands and see the output on standard output window.

# DEV BHOOMI INSTITUTE OF TECHNOLOGY

## LAB MANUAL

	<b>Course Name:</b> System Administration Lab	<b>Experiment No. 2</b>	
	<b>Course Code :</b> PCS-402 <b>Faculty :</b> Ms. RituRawat	<b>Branch:</b> CSE	<b>Semester:</b> IV

**Objective:** Simple shell script for basic arithmetic and logical calculations.

**Tools:**UNIX operating system/any flavor of Linux.

### **Algorithm:**

Step 1: start UNIX OS in your computer and login in it and enter Username and Password.

Step 2: Create a folder with your Id Number or Name Followed by Roll No.

Step 3: now go to your folder from the terminal and after that open the VI editor with the desired program name with extension .sh.

Step 4: now write your program and quit back to the terminal.

Step 5: After writing program make it executable by using \$chmod +x program name.

Step 6: Now run the program using any one of the method...

sh program name

./program name

### **Theory:**

There are various operators supported by each shell. Our tutorial is based on default shell (Bourne) so we are going to cover all the important Bourne Shell operators in the tutorial.

There are following operators which we are going to discuss:

Arithmetic Operators.

Logical Operators.

String operators.

File operators

1) Relational operators. But in this section we are only concentrating on first two i.e. arithmetic & logical operators.

2) Arithmetic Operators:

There are following arithmetic operators supported by Bourne Shell. Assume variable a holds 10 and variable b holds 20 then:

<b>Operator</b>	<b>Description</b>	<b>Example</b>
!	This is logical negation. This inverts a true condition into false and vice versa.	[ ! false ] is true.
o	This is logical OR. If one of the operands is true then condition would be true.	[ \$a lt 20 o \$b gt 100 ] is true.
a	This is logical AND. If both the operands are true then condition would be true otherwise it would be false.	[ \$a lt 20 a \$b gt 100 ] is false.
/	Division Divides left hand operand by right hand operand	`expr \$b / \$a` will give 2
%	Modulus Divides left hand operand by right hand operand and returns remainder	`expr \$b % \$a` will give 0
=	Assignment Assign right operand in left operand	a=\$b would assign value of b into a
==	Equality Compares two numbers, if both are same then returns true.	[ \$a == \$b ] would return false.
!=	Not Equality Compares two numbers, if both are different then returns true.	[ \$a != \$b ] would return true.

3) Logical Operators:

There are following Boolean operators supported by Bourne Shell. Assume variable a holds 10 and variable b holds 20 then:



# DEV BHOOMI INSTITUTE OF TECHNOLOGY


## LAB MANUAL

### **Conclusions:**

With the help of given procedure and information about the operators we can write shell program to perform various operation.

### **Outcome:**

To understand the Simple shell script for basic arithmetic and logical calculations.

	<b>Course Name:</b> System Administration Lab	<b>Experiment No. 3</b>	
	<b>Course Code :</b> PCS-402  <b>Faculty :</b> Ms. RituRawat	<b>Branch:</b> CSE	<b>Semester:</b> IV

**Objective :Shell scripts to check various attributes of files and directories.**

**Tools:** UNIX operating system/any flavor of Linux.

**Algorithm:**

Step 1: start UNIX OS in your computer and login in it and enter Username and Password.

Step 2: Create a folder with your Id Number or Name Followed by Roll No.

Step 3: now go to your folder from the terminal and after that open the VI editor with the desired program name with extension .sh.

Step 4: now write your program and quit back to the terminal.

Step 5: After writing program make it executable by using \$chmod +x program name.

Step 6: Now run the program using any one of the method...

sh program name

./program name

**Theory:**

Basic File Attributes Read, Write and Execute

There are three basic attributes for plain file permissions: read, write, and execute.

**Read Permission of a file**

If you have read permission of a file, you can see the contents. That means you can use more, cat, etc.

**Write Permission of a file**

If you have write permission of a file, you can change the file. This means you can add to a file, or overwrite a file. You can empty a file called "yourfile" by copying the empty (/dev/null) file on top of it

cat /dev/null yourfile

**Execute Permission of a file**

If the file has execute permission, then you can ask the operating system to run the file as if it were a program. If it's a binary file/program, you can execute it like any other program.

If the file is a shell script, then the execute attribute says you can treat it as if it were a program.

To put it another way, you can create a file using your favorite editor, add the execute attribute to it, and it "becomes" a program. However, since a shell has to read the file, a shell script has to be readable and executable. A compiled program does not need to be readable.

The basic permission characters, "r", "w", and "x" r means read w means write, and x means execute. Using chmod to change permissions

The chmod command is used to change permission. The simplest way to use the chmod command is to add or subtract the permission to a file. A simple plus or minus is used to add or subtract the permission.

You may want to prevent yourself from changing an important file. Remove the write permission of the file "myfile" with the command `chmodw myfile` If you want to make file "myscript" executable, type `chmod +x myscript` You can add or remove more than one of these attributes at a time `chmodrwx file`

```
chmod +wx file
```

You can also use the "=" to set the permission to an exact combination This command removes the write and execute permission, while adding the read permission:

```
chmod =r myfile
```

Note that we can change permissions of files you own. That is, you can remove all permissions of a file, and then add them back again. You can make a file "read only" to protect it. However, making a file read only does not prevent you from deleting the file. That's because the file is in a directory, and directories also have read, write and execute permission. And the rules are different. Read on.

### **Basic Directory Attributes Read, Write and Search**

Directories use these same permissions, but they have a different meaning. Yes, very different meanings.

#### **Read permission on a directory**

If a directory has read permission, you can see what files are in the directory. That is, you can do an "ls" command and see the files inside the directory. However, read permission of a directory does not mean you can read the contents of files in the directory.

### **Write permission on a directory**

Write permission means you can add a new file to the directory. It also means you can rename or move files in the directory.

### **Execute permission on a directory**

Execute allows you to use the directory name when accessing files inside that directory. The "x" permission means the directory is "searchable" when searching for executables. If it's a program, you can execute the program.

### **File Test Operators:**

There are following operators to test various properties associated with a Unix file.

Assume a variable file holds an existing file name "test" whose size is 100 bytes and has read, write and execute permission on:

<b>Operator</b>	<b>Description</b>	<b>Example</b>
b file	Checks if file is a block special file if yes then condition becomes true.	[ b \$file ] is false.
c file	Checks if file is a character special file if yes then condition becomes true.	[ b \$file ] is false.
d file	Check if file is a directory if yes then condition becomes true.	[ d \$file ] is not true.
f file	Check if file is an ordinary file as opposed to a directory or special file if yes then condition becomes true.	[ f \$file ] is true.
g file	Checks if file has its set group ID (SGID) bit set if yes then condition becomes true.	[ g \$file ] is false.
k file	Checks if file has its sticky bit set if yes then condition becomes true.	[ k \$file ] is false.
p file	Checks if file is a named pipe if yes then condition becomes true.	[ p \$file ] is false.
t file	Checks if file descriptor is open and associated with a terminal if yes then condition becomes true.	[ t \$file ] is false.
u file	Checks if file has its set user id (SUID) bit set if yes then condition becomes true.	[ u \$file ] is false.
r file	Checks if file is readable if yes then condition becomes true.	[ r \$file ] is true.
w file	Check if file is writable if yes then condition becomes true.	[ w \$file ] is true.

<b>Operator</b>	<b>Description</b>	<b>Example</b>
x file	Check if file is execute if yes then condition becomes true.	[ x \$file ] is true.
s file	Check if file has size greater than 0 if yes then condition becomes true.	[ s \$file ] is true.
e file	Check if file exists. Is true even if file is a directory but exists.	[ e \$file ] is true.

**Example:**


```
#!/bin/sh
file="/home /jnec /test.sh"
if [ r $file ] then echo "File has read access" else echo "File does not have read access" fi
if [ w $file ] then echo "File has write permission" else echo "File does not have write
permission" fi
if [ x $file ] then echo "File has execute permission" else echo "File does not have execute
permission" fi
if [ f $file ] then
echo "File is an ordinary file" else echo "This is special file" fi
if [ d $file ] then echo "File is a directory" else echo "This is not a directory" fi
if [ s $file ] then echo "File size is zero" else echo "File size is not zero" fi
if [ e $file ] then echo "File exists" else echo "File does not exist" fi
```

**Outcome:**

With the help of given procedure and information about the File operators we can write shell program to check various attributes of files and directories.

# DEV BHOOMI INSTITUTE OF TECHNOLOGY

## LAB MANUAL

	<b>Course Name:</b> System Administration Lab	<b>Experiment No. 4</b>	
	<b>Course Code :</b> PCS-402 <b>Faculty :</b> Ms. RituRawat	<b>Branch:</b> CSE	<b>Semester:IV</b>

**Objective:** Shell scripts to perform various operations on given strings.

**Tools:** UNIX operating system/any flavor of Linux.

### **Algorithm:**

Step 1: start UNIX OS in your computer and login in it and enter Username and Password.

Step 2: Create a folder with your Id Number or Name Followed by Roll No.

Step 3: now go to your folder from the terminal and after that open the VI editor with the desired program name with extension .sh.

Step 4: now write your program and quit back to the terminal.

Step 5: After writing program make it executable by using \$chmod +x program name.

• Step 6: Now run the program using any one of the method...

sh program name

./program name

### **Theory:**

There are various functions by using which you can perform different operation on given string.

String Manipulation Functions:

1) String Length

`${#string}` expr length \$string These are the equivalent of strlen() in C. expr "\$string" : '.\*'

2) Length of Matching Substring at Beginning of String

expr match "\$string" '\$substring' \$substring is a regular expression. expr "\$string" : '\$substring'

e.g. stringZ=abcABC123ABCabc # || # 12345678

```
echo `expr match "$stringZ" 'abc[AZ]*.2` # 8 echo `expr "$stringZ" : 'abc[AZ]*.2` # 8
```

### 3) Index

expr index \$string \$substring Numerical position in \$string of first character in \$substring that matches. This is the near equivalent of strchr() in C.

e.g.

```
stringZ=abcABC123ABCabc # 123456 ... echo `expr index "$stringZ" C12` # 6
```

# C position.

```
echo `expr index "$stringZ" 1c` # 3
```

### 4) Substring Extraction

`${string: position}` Extracts substring from \$string at \$position. If the \$string parameter is "\*" or "@", then this extracts the positional parameters, starting at \$position. `${string: position: length}` Extracts \$length characters of substring from \$string at \$position.

e.g. stringZ=abcABC123ABCabc # 0123456789..... # 0based indexing.

```
echo ${stringZ:0} # abcABC123ABCabc echo ${stringZ:1} # bcABC123ABCabc echo  
${stringZ:7} # 23ABCabc
```

```
echo ${stringZ:7:3} # 23A # Three characters of substring.
```

### 5) Substring Removal

`${string#substring}` Deletes shortest match of \$substring from front of \$string.

`${string##substring}` Deletes longest match of \$substring from front of \$string.

### 6) Substring Replacement

`${string/substring/replacement}` Replace first match of \$substring with \$replacement.

`${string//substring/replacement}` Replace all matches of \$substring with \$replacement.


## Outcome:

With the help of given procedure and information about the String manipulating Functions we can write shell program to perform various operation on given string.



# DEV BHOOMI INSTITUTE OF TECHNOLOGY

## LAB MANUAL

	<b>Course Name:</b> System Administration Lab	<b>Experiment No. 5</b>	
	<b>Course Code :</b> PCS-402 <b>Faculty :</b> Ms. RituRawat	<b>Branch:</b> CSE	<b>Semester:</b> IV

**Objective:** Shell scripts to explore system variables such as PATH, HOME etc.

**Tools:** UNIX operating system/any flavor of Linux.

### **Algorithm:**

Step 1: start UNIX OS in your computer and login in it and enter Username and Password.

Step 2: Create a folder with your Id Number or Name Followed by Roll No.

Step 3: now go to your folder from the terminal and after that open the VI editor with the desired program name with extension .sh.

Step 4: now write your program and quit back to the terminal.

Step 5: After writing program make it executable by using \$chmod +x program name.

• Step 6: Now run the program using any one of the method...

sh program name

./program name

### **Theory:**

UNIX and all UNIXlike operating systems such as OpenBSD, Linux, Redhat, CentOS, Debian allows you to set environment variables. When you log in on UNIX, your current shell (login shell) sets a unique working environment for you which is maintained until you log out. Following are most command examples of environment variables used under UNIX operating systems:

PATH Display lists directories the shell searches, for the commands.

HOME User's home directory to store files.

TERM Set terminal emulator being used by UNIX.

PS1 Display shell prompt in the Bourne shell and variants.

MAIL Path to user's mailbox.

TEMP Path to where processes can store temporary files.

PWD Path to the current directory.

HISTFILE The name of the file in which command history is saved

HISTFILESIZE The maximum number of lines contained in the history file

HOSTNAME The system's host name

PATH It is a colonseparated set of directories where libraries should be searched for.

USER Current logged in user's name.

DISPLAY Network name of the X11 display to connect to, if available.

SHELL The current shell.

EDITOR The user's preferred text editor.

PAGER The user's preferred text pager.

MANPATH Colon separated list of directories to search for manual pages.

### **Display Environment Variable**

Open the terminal and type the following commands to display all environment variables and their values under UNIXlike operating systems:

`$set` Or `$printenv` Or `$env`


To explore any of the variables mentioned above you have to simply write `$echo $ Variable name`

### **Outcome:**

With the help of given procedure and information about Environment variables we can write shell program to Explore all the Environment variables.

# DEV BHOOMI INSTITUTE OF TECHNOLOGY

## LAB MANUAL

	<b>Course Name:</b> System Administration Lab	<b>Experiment No. 6</b>	
	<b>Course Code :</b> PCS-402 <b>Faculty :</b> Ms. RituRawat	<b>Branch:</b> CSE	<b>Semester:</b> IV

**Objective:** Shell scripts to check and list attributes of processes.

**Tools:**UNIX operating system/any flavor of Linux.

### **Algorithm:**

Step 1: start UNIX OS in your computer and login in it and enter Username and Password.

Step 2: Create a folder with your Id Number or Name Followed by Roll No.

Step 3: now go to your folder from the terminal and after that open the VI editor with the desired program name with extension .sh.

Step 4: now write your program and quit back to the terminal.

Step 5: After writing program make it executable by using \$chmod +x program name.

• Step 6: Now run the program using any one of the method...

sh program name

./program name

### **Theory:**

A process can be simply defined as an instance of a running program. It should be understood that a program is an entity that resides on a nonvolatile media (such as disk), and a process is an entity that is being executed (with at least some portion, i.e. segment/page) in RAM.

A process has a series of characteristics:

The process ID or PID: a unique identification number used to refer to the process.

The parent process ID or PPID: the number of the process (PID) that started this process.

Nice number: the degree of friendliness of this process toward other processes (not to be confused with process priority, which is calculated based on this nice number and recent CPU usage of the process).

Terminal or TTY: terminal to which the process is connected.

The pscommand displays active processes. The syntax for the pscommand is: ps [options]

Options

a Displays all processes on a terminal, with the exception of group leaders.

c Displays scheduler data.

d Displays all processes with the exception of session leaders.

e Displays all processes.

f Displays a full listing.

glist Displays data for the list of group leader IDs.

j Displays the process group ID and session ID.

l Displays a long listing

plist Displays data for the list of process IDs.

slist Displays data for the list of session leader IDs.

tlist Displays data for the list of terminals.


ulist Displays data for the list of usernames.

### **Outcome:**

With the help of given procedure and information about the ps with option we can write shell program to check and list attributes of process.

# DEV BHOOMI INSTITUTE OF TECHNOLOGY

## LAB MANUAL

	<b>Course Name:</b> System Administration Lab	<b>Experiment No. 7</b>	
	<b>Course Code :</b> PCS-402 <b>Faculty :</b> Ms. RituRawat	<b>Branch:</b> CSE	<b>Semester:</b> IV

**Objective:** Execution of various system administrative commands.

**Tools:**UNIX operating system/any flavor of Linux.

### **Algorithm:**

Analyzing the Problem:

Start the Linux and enter the user name and password.

Now write startx and after that open the terminal.

At the terminal try the different commands and see the output.

Designing the Solution:

- At the terminal first perform the command without and with the different Options available for it.

The exercises in this lab cover the usage of some of the most basic system utilities that users and administrators alike need to be familiar with. Most of the commands are used in navigating and manipulating the file system. The file system is made up of files and directories.

### **Theory:**

Users and Groups users

Show all logged on users. This is the approximate equivalent of who q. groups

Lists the current user and the groups she belongs to. This corresponds to the \$GROUPS internal variable, but gives the group names, rather than the numbers.

```
bash$ groups bozitacdromcdwriter audio xgrp
```

```
bash$ echo $GROUPS 501
```

chown, chgrp

The chown command changes the ownership of a file or files. This command is a useful method that root can use to shift file ownership from one user to another. An ordinary user may not change the ownership of files, not even her own files. [1]

```
root# chown bozo *.txt
```

The chgrp command changes the group ownership of a file or files. You must be owner of the file(s) as well as a member of the destination group (or root) to use this operation.

```
chgrp recursive dunderheads *.data # The "dunderheads" group will now own all the "*.data" files #+ all the way down the $PWD directory tree (that's what "recursive" means).
```

useradd, userdel

The useradd administrative command adds a user account to the system and creates a home directory for that particular user, if so specified. The corresponding userdel command removes a user account from the system

## **[2] and deletes associated files.**

The adduser command is a synonym for useradd and is usually a symbolic link to it.

usermod

Modify a user account. Changes may be made to the password, group membership, expiration date, and other attributes of a given user's account. With this command, a user's password may be locked, which has the effect of disabling the account.

groupmod


Modify a given group. The group name and/or ID number may be changed using this command.

## **Outcome:**

With the help of given procedure and information about the commands we can execute system administrative task.

# DEV BHOOMI INSTITUTE OF TECHNOLOGY

## LAB MANUAL

	<b>Course Name:</b> System Administration Lab	<b>Experiment No. 8</b>	
	<b>Course Code :</b> PCS-402 <b>Faculty :</b> Ms. RituRawat	<b>Branch:</b> CSE	<b>Semester:</b> IV

**Objective:** Write awkscript that uses all of its features.

**Tools:**UNIX operating system/any flavor of Linux.

### **Algorithm:**

Analyzing the Problem:

Start the Linux and enter the user name and password.

Now write startx and after that open the terminal.

At the terminal try the different commands and see the output.

Designing the Solution:

- At the terminal first perform the command without and with the different Options available for it.

The exercises in this lab cover the usage of some of the most basic system utilities that users and administrators alike need to be familiar with. Most of the commands are used in navigating and manipulating the file system. The file system is made up of files and directories.

### **Theory:**

The AWK utility is an interpreted programming language typically used as a data extraction and reporting tool. It is a standard feature of most Unixlike operating Systems. Awk is an excellent tool for building UNIX/Linux shell scripts. AWK is a Programming language that is designed for processing textbased data, either in files or data streams or using shells pipes. In other words you can combine awk with shell scripts or directly use at a shell prompt.

The essential organization of an AWK program follows the form:

pattern { action } The pattern specifies when the action is performed. Like most UNIX utilities, AWK is line oriented. That is, the pattern specifies a test that is performed with each line read as input. If the condition is true, then the action is taken. The default pattern is something that matches every line. This is the blank or null pattern. Two other important patterns are specified by the keywords "BEGIN" and "END." As you might expect, these two words specify actions to be taken before any lines are read, and after the last line is read. The AWK program below:

```
BEGIN { print "START" }
```

```
{ print } END { print "STOP" }
```

adds one line before and one line after the input file. This isn't very useful, but with a simple change, we can make this into a typical AWK program:

```
BEGIN { print "File\tOwner", " } { print $8, "\t", $3 } END { print " DONE " }
```

Functions in awk :


### **Outcome:**

To understand the awk scripts .With the help of given procedure and information about the use of awk commands we can successfully execute all awk features.



# DEV BHOOMI INSTITUTE OF TECHNOLOGY

## LAB MANUAL

	<b>Course Name:</b> System Administration Lab	<b>Experiment No. 9</b>	
	<b>Course Code :</b> PCS-402 <b>Faculty :</b> Ms. RituRawat	<b>Branch:</b> CSE	<b>Semester:</b> IV

**Objective:** Use sed instruction to process /etc/password file.

**Tools:** UNIX operating system/any flavor of Linux.

### **Algorithm:**

Analyzing the Problem:

Start the Linux and enter the user name and password.

Now write start x and after that open the terminal.

At the terminal try the different commands and see the output.

### **Designing the Solution:**

- At the terminal first perform the command without and with the different Options available for it.

The exercises in this lab cover the usage of some of the most basic system utilities that users and administrators alike need to be familiar with. Most of the commands are used in navigating and manipulating the file system. The file system is made up of files and directories.

### **Theory:**

sed stream editor for filtering and transforming text

#### SYNTAX

```
sed [OPTION]... {scriptonlyifnootherscript} [inputfile]...
```

#### DESCRIPTION

Sed is a stream editor. A stream editor is used to perform basic text transformations on an input stream (a file or input from a pipeline). While in some ways similar to an editor which permits scripted edits (such as ed), sed works by making only one pass over the input(s), and is

consequently more efficient. But it is sed's ability to filter text in a pipeline which particularly distinguishes it from other types of editors.

n, quiet, silent suppress automatic printing of pattern space e script, expression=script add the script to the commands to be executed f scriptfile, file=scriptfile add the contents of scriptfile to the commands to be executed i[SUFFIX], inplace[=SUFFIX] edit files in place (makes backup if extension supplied) c, copy use copy instead of rename when shuffling files in i mode (avoids change of input file ownership) l N, linelength=N specify the desired linewidth length for the 'l' command posix disable all GNU extensions. r, regextended use extended regular expressions in the script. s, separate consider files as separate rather than as a single continuous long stream.

```
#sed 'ADDRESSs/REGEXP/REPLACEMENT/FLAGS' filename #sed  
'PATTERNs/REGEXP/REPLACEMENT/FLAGS' filename
```

s is substitute command

/ is a delimiter

REGEXP is regular expression to match

REPLACEMENT is a value to replace

FLAGS can be any of the following

g Replace all the instance of REGEXP with REPLACEMENT

n Could be any number, replace nth instance of the REGEXP with REPLACEMENT.

p If substitution was made, then prints the new pattern space.

i match REGEXP in a caseinsensitive manner.

w file If substitution was made, write out the result to the given file.


We can use different delimiters ( one of @ % ; : ) instead of /

## **Outcome:**

Withthe help of given procedure and information about the commands we can process sed instructions on /etc/passwd file.

# DEV BHOOMI INSTITUTE OF TECHNOLOGY

## LAB MANUAL

	<b>Course Name:</b> System Administration Lab	<b>Experiment No. 10</b>	
	<b>Course Code :</b> PCS-402 <b>Faculty :</b> Ms. RituRawat	<b>Branch:</b> CSE	<b>Semester:</b> IV

**Objective:** Write a shell script to display list of users currently logged in.

**Tools:** UNIX operating system/any flavor of Linux.

### **Algorithm:**

Step 1: start UNIX OS in your computer and login in it and enter Username and Password.

Step 2: Create a folder with your Id Number or Name Followed by Roll No.

Step 3: now go to your folder from the terminal and after that open the VI editor with the desired program name with extension .sh.

Step 4: now write your program and quit back to the terminal.

Step 5: After writing program make it executable by using \$chmod + x program name.

Step 6: Now run the program using any one of the method...

sh program name

./program name

### **Theory:**

whoshow who is logged on

SYNOPSIS who [OPTION]... [ FILE | ARG1 ARG2 ]

DESCRIPTION a, all same as b d login p r t Tu b, boot time of last system boot d, dead print  
dead processes H, heading print line of column headings i, idle

add idle time as HOURS:MINUTES, . or old (deprecated, use u) login print system login processes (equivalent to SUS 1) l, lookup attempt to canonicalize hostnames via DNS (l is deprecated, use lookup) m

only hostname and user associated with stdinp, process print active processes spawned by initq, - count all login names and number of users logged on r, runlevelprint current runlevels, short print only name, line, and time (default)

**Outcome:**

With the help of given procedure and information about the commands we can write shell program to display list of users currently logged in.