Jawaharlal Nehru Engineering College

Laboratory Manual

of

UNIX AND SHELL PROGRAMMING

For

Second Year Students
Dept: Computer Science & Engineering

FOREWORD

It is my great pleasure to present this laboratory manual for second year engineering students for the subject of Unix and Shell Programming keeping in view the vast coverage required for understanding the concept of Shell Programming.

As a student, many of you may be wondering with some of the questions in your mind regarding the subject and exactly what has been tried is to answer through this manual.

As you may be aware that MGM has already been awarded with ISO 9000 certification and it is our endure to technically equip our students taking the advantage of the procedural aspects of ISO 9000 Certification.

Faculty members are also advised that covering these aspects in initial stage itself, will greatly relived them in future as much of the load will be taken care by the enthusiasm energies of the students once they are conceptually clear.

Dr. S.D.Deshmukh,

Principal

LABORATORY MANUAL CONTENTS

This manual is intended for the Second year students of Computer Science & Engineering in the subject of Unix and Shell Programming. This manual typically contains practical/Lab Sessions related Unix and Shell Programming covering various aspects related the subject to enhanced understanding.

Unix and Shell Programming provides students the idea of new operating system.It also helps to understands the concept of Shell programming along with the use of various filters including simple,advanced,etc.It also provides the idea of new scripting language like PERL.

Students are advised to thoroughly go through this manual rather than only topics mentioned in the syllabus as practical aspects are the key to understanding and conceptual visualization of theoretical aspects covered in the books.
Good Luck for your Enjoyable Laboratory Sessions

Prof. D .S. Deshpande
HOD,CSE

Mr. S.A.Kharat
Lecturer, CSE Dept.

SUBJECT INDEX

DOs and DON’’T DOs in Laboratory:

1. Make entry in the Log Book as soon as you enter the Laboratory.

2. All the students should sit according to their roll numbers starting from their left to right.

3. All the students are supposed to enter the terminal number in the log book.

4. Do not change the terminal on which you are working.

5. All the students are expected to get at least the algorithm of the program/concept to be implemented.

6. Strictly observe the instructions given by the teacher/Lab Instructor.

Instruction for Laboratory Teachers:

1. Submission related to whatever lab work has been completed should be done during the next lab session. The immediate arrangements for printouts related to submission on the day of practical assignments.

2. Students should be taught for taking the printouts under the observation of lab teacher.

3. The promptness of submission should be encouraged by way of marking and evaluation patterns that will benefit the sincere students.

1. ***Lab Exercise***

Exercise No 1: (2 Hours) – 1 Practical

Aim: -Execution of various file/directory handling commands.

TOOLS: UNIX operating system/any flavor of Linux.

STANDARD 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) **pwd COMMAND:**
   pwd - Print Working Directory. pwd command prints the full filename of the current working directory.

   SYNTAX:
   The Syntax is
   pwd [options]

2) cd COMMAND:
cd command is used to change the directory.

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

3) ls COMMAND:
ls command lists the files and directories under current working directory.

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

OPTIONS:

-l	Lists all the files, directories and their mode, Number of links, 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) rm COMMAND:
rm linux 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) mv COMMAND:

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] oldname newname
```

OPTIONS:

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

-i Prompts before overwriting another file.

6) cat COMMAND:

cat linux 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) cmp COMMAND:

cmp linux 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) cp COMMAND:

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) echo COMMAND:

echo command prints the given input string to standard output.

SYNTAX:

The Syntax is

```
echo [options..] [string]
```

11)mkdir COMMAND:

mkdir command is used to create one or more directories.

SYNTAX:

The Syntax is

mkdir [options] directories

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.

12) paste COMMAND:

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]

OPTIONS:

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

13) rmdir COMMAND:

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

SYNTAX:

The Syntax is

rmdir [options..] Directory

OPTIONS:

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

CONCLUSIONS:

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

2. Lab Exercise

Exercise No 2: (2 Hours) – 1 Practical

Aim: -Simple shell script for basic arithmetic and logical calculations.

TOOLS: UNIX operating system/any flavor of Linux.

STANDARD PROCEDURE:

- 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
- Relational operators.

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

Arithmetic Operators:

There are following arithmetic operators supported by Bourne Shell.

Assume variable a holds 10 and variable b holds 20 then:

Operator	Description	Example
+	Addition - Adds values on either side of the operator	\`expr $a + $b\` will give 30
Operator	**Description**	**Example**
-	Subtraction - Subtracts right hand operand from left hand operand	\`expr $a - $b\` will give -10
*	Multiplication - Multiplies values on either side of the operator	\`expr $a * $b\` will give 200
/	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.

Logical Operators:

There are following Boolean operators supported by Bourne Shell.

Assume variable a holds 10 and variable b holds 20 then:

Operator	Description	Example
!	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.

CONCLUSIONS:

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

3. Lab Exercise

Exercise No 3: (2 Hours) – 1 Practical

Aim: - Shell scripts to check various attributes of files and directories.

TOOLS: UNIX operating system/any flavor of Linux.

STANDARD PROCEDURE:

- 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 e**x**ecute.

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

```
chmod -w 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

```
chmod -rwx 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 you 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:

Operator	Description	Example
-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.

Operator	Description	Example
-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 sepcial 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
```

CONCLUSIONS:

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.

4. Lab Exercise

Exercise No 4: (2 Hours) – 1 Practical

Aim: - Shell scripts to perform various operations on given strings.

TOOLS: UNIX operating system/any flavor of Linux.

STANDARD PROCEDURE:

- 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 "$stringZ" 'abc[A-Z]*.2'`   # 8
echo `expr "$stringZ" : 'abc[A-Z]*.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.....
#       0-based 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*.

CONCLUSIONS:

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.

5. Lab Exercise

Exercise No 5: (2 Hours) – 1 Practical

Aim: - Shell scripts to explore system variables such as PATH, HOME etc.

TOOLS: UNIX operating system/any flavor of Linux.

STANDARD PROCEDURE:

- 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 UNIX-like 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 colon-separated 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 UNIX-like operating systems:

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

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

CONCLUSIONS:

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

6. Lab Exercise

Exercise No 6: (2 Hours) – 1 Practical

Aim: - Shell scripts to check and list attributes of processes.

TOOLS: UNIX operating system/any flavor of Linux.

STANDARD PROCEDURE:

- 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 non-volatile 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 **ps** command displays active processes.
The syntax for the **ps** command 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.
-g*list*	Displays data for the *list* of group leader IDs.
-j	Displays the process group ID and session ID.
-l	Displays a long listing
-p*list*	Displays data for the *list* of process IDs.
-s*list*	Displays data for the *list* of session leader IDs.
-t*list*	Displays data for the *list* of terminals.
-u*list*	Displays data for the *list* of usernames.

CONCLUSIONS:

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.

7. Lab Exercise

Exercise No 7: (2 Hours) – 1 Practical

Aim: -Execution of various system administrative commands.

TOOLS: UNIX operating system/any flavor of Linux.

STANDARD 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:

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
bozita cdrom cdwriter 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.

CONCLUSIONS:

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

8. Lab Exercise

Exercise No 8: (2 Hours) – 1 Practical

Aim: -Write awk script that uses all of its features.

TOOLS: UNIX operating system/any flavor of Linux.

STANDARD 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:

The AWK utility is an interpreted programming language typically used as a data extraction and reporting tool. It is a standard feature of most Unix-like operating Systems.

Awk is an excellent tool for building UNIX/Linux shell scripts. AWK is a Programming language that is designed for processing text-based 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 :-

```
 index(string,search)                    length(string)
split(string,array,separator)            substr(string,position)
substr(string,position,max)               tolower(string)
                      |toupper(string)
```

CONCLUSIONS:

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

9. Lab Exercise

Exercise No 9: (2 Hours) – 1 Practical

Aim: - Use sed instruction to process /etc/password file.

TOOLS: UNIX operating system/any flavor of Linux.

STANDARD PROCEDURE:

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]... {script-only-if-no-other-script} [input-file]...

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 script-file, --file=script-file

 add the contents of script-file to the commands to be executed

-i[SUFFIX], --in-place[=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, --line-length=N

 specify the desired line-wrap length for the 'l' command

--posix

 disable all GNU extensions.

-r, --regexp-extended

 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 case-insensitive manner.
- w file If substitution was made, write out the result to the given file.
- We can use different delimiters (one of @ % ; :) instead of /

CONCLUSIONS:

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

10. Lab Exercise

Exercise No 10: (2 Hours) – 1 Practical

Aim: - Write a shell script to display list of users currently logged in.

TOOLS: UNIX operating system/any flavor of Linux.

STANDARD PROCEDURE:

- 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:

who - show who is logged on

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

DESCRIPTION
-a, **--all**
  same as **-b -d --login -p -r -t -T -u**
-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 **-l**)

-l, **--lookup**
attempt to canonicalize hostnames via DNS (-l is deprecated, use **--lookup**)

-m
only hostname and user associated with stdin

-p, **--process**
print active processes spawned by init

-q, **--count**
all login names and number of users logged on

-r, **--runlevel**
print current runlevel

-s, **--short**
print only name, line, and time (default)

CONCLUSIONS:

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

11. Lab Exercise

Exercise No 11: (2 Hours) – 1 Practical

Aim: -Write a shell script to delete all the temporary files.

TOOLS: UNIX operating system/any flavor of Linux.

STANDARD PROCEDURE:

- 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:
rm - remove files or directories

SYNOPSIS
rm [*OPTION*]... *FILE*...

OPTIONS

Remove (unlink) the FILE(s).

-d, **--directory**
    unlink FILE, even if it is a non-empty directory (super-user only)
-f, **--force**
    ignore nonexistent files, never prompt
-i, **--interactive**
    prompt before any removal
-r, **-R**, **--recursive**

remove the contents of directories recursively

-v, **--verbose**

explain what is being done .

To remove a file whose name starts with a `-', for example `-foo', use one of these commands:

rm -- -foo
rm. /-foo

fsck :-
check and repair a Linux file system.

SYNOPSIS
fsck [**-sACVRTNP**] [**-t** *fstype*] *[filesys ...]* [--] [**fs-specific-options**]

DESCRIPTION
fsck is used to check and optionally repair one or more Linux file systems. *filesys* can be a device name (e.g. */dev/hdc1*, */dev/sdb2*), a mount point (e.g. */*, */usr*, */home*), or an ext2 label or UUID specifier (e.g. UUID=8868abf6-88c5-4a83-98b8-bfc24057f7bd or LABEL=root). Normally, the **fsck** program will try to run file systems on different physical disk drives in parallel to reduce total amount time to check all of the file systems.

CONCLUSIONS:

With the help of given procedure and information about the commands we can write shell program to delete all the temporary files.

12. Lab Exercise

Exercise No 12: (2 Hours) – 1 Practical

Aim: -Write a shell script to search an element from an array using binary searching..

TOOLS: UNIX operating system/any flavor of Linux.

STANDARD PROCEDURE:

- 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:

if condition

if condition which is used for decision making in shell script, If given condition is true then command is executed.
Syntax:

```
if condition
then
        command1 if condition is true or if exit status
        of condition is 0 (zero)
        ...
        ...
Fi
```

i)**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]

ii) eval:
The general format of the eval command is very simple:
    eval [*command_to_be_interpreted...*]

Algorithm for Binary Search:-

i.Binary search works by comparing an input value to the middle element of the array. The comparison determines whether the element equals the input, less than the input or greater.

ii. When the element being compared to equals the input the search stops and typically returns the position or number of searches of the element. If the element is not equal to the input and the element at the middle point is greater than the input being searched, the current middle point becomes the new high point and the array is cut in half again and re-tested.

iii. If the element at the middle point is less than the input being searched, the current middle point becomes the new low point and the array is cut in half again and retested. This cutting in half and adjusting either the high point or the low point is repeated until the item is found or the low point and the high point converge.

CONCLUSIONS:

With the help of given procedure and information about the commands we can write shell program to find an array element using Binary Search.