LAB MANUAL OF
LINUX & X-WINDOWS PROGRAMMING
ETCS 355

Maharaja Agrasen Institute of Technology,
PSP area, Sector – 22, Rohini, New Delhi – 110085
(Affiliated to Guru Gobind Singh Indraprastha University, Dwarka, New Delhi)

INDEX OF THE CONTENTS

1. Introduction to UNIX and LINUX Lab

The UNIX operating system has been an influential force for decades. It is the foundation of industry's movement towards "rightsizing" and Open Systems. This course introduces the basic capabilities common to both the Linux and UNIX systems and provides an extensive exposure to the common commands, file system organization, standard applications, and editors.

This hands-on course will provide an introduction to the Linux and UNIX operating systems and common commands, and lays the foundation for further learning and use of these systems.

Who Should Attend

Computer novices and computer professionals who want to learn more about the Linux and UNIX operating systems.

Prerequisites

None. Some experience with any other operating system or a programming language will be helpful.

Benefits of attending the lab

Upon completion of this course, students should be able to:

- Log on and off the system
- Manage the password
- Retrieve help information about Linux and UNIX commands
- Use a variety of common commands
- Access networked systems using standard applications
- Recognize different types of files
- Utilize a hierarchical file system with full and relative path names
- Create and manage files and directories
- Use the vi editor to create and modify file contents
- Format and print information
- Monitor and control program processes
- Use simple shell features such as pipes and I/O redirection

2. LAB REQUIREMENTS

H/W Detail	Intel CPU (P-IV 1.6MHz.)/256 MB RAM/ 60GB HDD/ Intel 845GLL M.B./On board Sound & 3D Graphics card/ Key Board/ Mouse/ CD Drive/ FDD/15" Color Monitor/ UPS LAN Card	12 Nos.
	Dot Matrix Printer	1 No.
	LaserJet Printer	1 No.
S/W Detail	RedHat Linux, C/C++	

3. LIST OF EXPERIMENTS
(As prescribed by G.G.S.I.P.U)

Paper Code: ETCS 355 **P** **C**
Paper: Linux & X-Windows Programming **3** **1**

Q1 Write script to print the message "Hello" on the console.
Q2 Write script to perform following basic math operation as
a) addition
b) substraction
c) multiplication
d) division
Note : i) take input from keyboard
ii) take input as command line parameter
Q3 Write script to display current date, time, username and current directory
Q4 Write shell script to show various system configuration like
a) Currently logged user and his long name
b) Current shell
c) Your home directory
d) Your operating system type
e) Your current path setting
f) Your current working directory
g) Show all available shells
Q5 Write a script to convert the contents of a given file from uppercase to lowercase and also count the number of lines, words and characters of the resultant file. Also display the resultant file in descending order
Q6 Write a shell script to list the files arranged in descending order of their size.
Q7 Write a shell script to find the reverse of a given number.
Q8 Write a shell script to print the contents of the file from given line numbers to next given number line.
Q9 Write a shell script to print the pattern

```
1
2   2
3   3   3
4   4   4   4
```

Q10 Write a script to determine whether given file exist or not, file name is supplied as command line argument, also check for sufficient number of command line argument.
Q11 Write a script to determine whether given command line argument ($1) contains "*" symbol or not, if $1 does not contains "*" symbol add it to $1, otherwise show message "Symbol is not required".
Q12 Write a script to check whether the given string is palindrome or not.

4. LIST OF ADVANCE EXPERIMENTS (Beyond the syllabus prescribed by G.G.S.I.P.U)

List of Advance experiments given to the students is summarized as below:

Experiment 1

Task 1 to 10 on *command line file system browsing*

Experiment 2

Task 1 to 3 on *file permissions*
Task 1 to 4 on *creating groups and users*
Task 1 to 5 on *setting up shared directories*

Experiment 3

Task 1 to 8 on *local user logins*
Task 1 to 6 on *switching user accounts*
Task 1 to 4 on *umask*

Experiment 4

Task 1 to 6 on *symbolic and hard links to files*
Task 1 to 3 on *file system*

Experiment 5

Task 1 to 10 on *Linux General purpose utilities*

5. PROJECTS TO BE ALLOTED (Beyond the syllabus prescribed by G.G.S.I.P.U)

The project is to be submitted at the end of the semester along with a project report by the individual student.

Students are to develop a new utility to be added to the current Linux shell.

Students can select project work of his own choice subject to the permission of concern faculty.

List of few projects given to the students is summarized as below:

- Check for leap year
- Search and replace string in a file
- Welcome script on each login
- Count the users logged using same id
- Report logged user status
- File compression
- Directory Backup
- Search & Replace word in a file
- File comparison
- Sorting of files (size)

NOTE: The project is to be made using shell programming.

6. FORMAT OF THE LAB RECORD TO BE PREPARED BY THE STUDENTS

1. The front page of the lab record prepared by the students should have a cover page as displayed below.

NAME OF THE LAB

Font should be(Size 20”, italics bold, Times New Roman)

Faculty name

should be (12”, Times Roman)

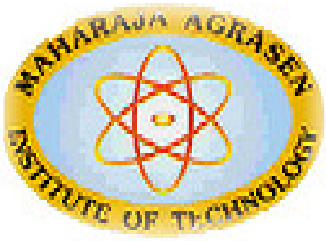
Student name Font

Roll No.:

Semester:

Group:

Font should be (12”, Times Roman)



Maharaja Agrasen Institute of Technology, PSP Area,

Sector – 22, Rohini, New Delhi – 110085

Font should be (18”, Times Roman)

2. The second page in the record should be the index as displayed below.

LINUX & X-WINDOWS PROGRAMMING

PRACTICAL RECORD

PAPER CODE : ETCS 307

Name of the student :

University Roll No. :

Branch :

Section/ Group :

PRACTICAL DETAILS

Experiments according to the lab syllabus prescribed by GGSIPU

Exp. no	Experiment Name	Date of performance	Date of checking	Remarks	Marks

PROJECT DETAILS

1. TITLE :
2. MEMBERS IN THE PROJECT GROUP :
3. PROJECT REPORT ATTACHED :
   a) YES b) NO
4. SOFT COPY SUBMITTED :
   a) YES b) NO

Signature of the lecturer Signature of the student

( ) ( )

3. Each practical which student is performing in the lab should have the following details :
   a) Topic Detail
   b) AIM
   c) Algorithm
   d) Source Code
   e) Output
   f) Viva questions

4. Project report should be added at last page.

7. MARKING SCHEME FOR THE PRACTICAL EXAMS

There will be two practical exams in each semester.

- Internal Practical Exam
- External Practical Exam

INTERNAL PRACTICAL EXAM

It is taken by the concerned lecturer of the batch.

MARKING SCHEME FOR THIS EXAM IS:

Total Marks: 40

Division of 40 marks is as follows

1. Regularity: 25
   - Performing program in each turn of the lab
   - Attendance of the lab
   - File
2. Viva Voice: 10
3. Project: 5

NOTE: For the regularity, marks are awarded to the student out of 5 for each experiment performed in the lab and at the end the average marks are giving out of 25.

EXTERNAL PRACTICAL EXAM

It is taken by the concerned lecturer of the batch and by an external examiner. In this exam student needs to perform the experiment allotted at the time of the examination, a sheet will be given to the student in which some details asked by the examiner needs to be written and at the last viva will be taken by the external examiner.

MARKING SCHEME FOR THIS EXAM IS:

Total Marks: 60

Division of 60 marks is as follows

1. Sheet filled by the student: 20
2. Viva Voice: 15
3. Experiment performance: 15
4. File submitted: 10

NOTE:

- Internal marks + External marks = Total marks given to the students
  (40 marks) (60 marks) (100 marks)
- Experiments given to perform can be from any section of the lab.

8. DETAILS OF EACH SECTION

ALONGWITH

EXAMPLES, EXERCISES

&

EXPECTED VIVA QUESTIONS

SECTION 1

Administration: Fundamentals Installation and Configuration

This is a core system administration module, focusing on Linux installation and broad aspects of its configuration

Getting Started

- Login/Logout
- Login-Logout Problems
- Start Terminal Session

Shell

- The UNIX Shell
- Commands
- Examples of Commands
- Command Execution Sequence
- Standard Input and Output
- Redirection (>,>>,<)
- Redirection (>,>>,<) Examples
- Pipelines: Connecting Command Processes
- Pipe Examples
- Paging Output
- Background Processing
- Shell Wild-card Matching
- Shell Wild-card Matching Examples

Information About User Machine

- Shell Variables
- User Environment
- Working With Your Prompt
- Secondary Prompt
- hostname – Display System Name
- Today Is?
- Which Version of UNIX Am I Using?
- who – List Who Is Logged On
- tty – Display Terminal Device Name
- pwd – Print Working Directory
- ps – Process Status

- nice – Run a Command at Low Priority
- kill – Kill a Process
- kill Examples
- df – Report Number of Free Disk Blocks and i-nodes
- df/bdf Example
- du – Summarize Disk Usage

Files

- Types of Files
- Symbolic and hard Links
- File Names
- Hierarchical System
- Tree Structured File System
- File Protection
- File Details
- File Pathnames
- Full Pathnames
- Relative Pathnames
- Home Directory

File Commands

- Command Examples
- cat – Concatenate and Display
- echo – Output a String
- ls – List Directory Contents
- mkdir – Make Directories
- cd and pwd
- cp – Copy files and directories
- rm – Remove Files or Directories
- ln – make links between files
- Symbolic Links
- mv – Move or Rename Files
- rmdir – Remove Empty Directories
- tail – Output the Last Part of Files
- file – Determine File Type
- find – Find Files
- chmod – Change Mode
- chmod and Directories

Vi Editor

- Creating a File Using vi
- Special Functions
- Inserting Text
- Moving Around in the File
- Scrolling, Paging and Moving
- Searching
- Substituting Text
- Deleting and Changing Text
- Moving and Copying Text
- Correcting Text Summary
- Sentences and Paragraphs
- Manipulating Text
- Transferring Data
- Writing a File
- Reading from File or Command
- Leaving the Editor
- vi Commands Summary

Information Retrieval

- man ---Displays Manual Pages Onlines
- info Files

Information Processing

- diff ---Difference Between Two Files
- comm ---Display Common Lines
- grep ---Search Patterns In Files
- sort ---Sort Files
- cut ---Cut Out Selected Field
- paste ---Paste Files Together
- join ---Join Files on a Common Field
- tr ---Translate Characters
- wc ---Word Count
- tee ---Replicate Standard Output to a File
- pr ---Prepare Files For Printing

Batch Jobs

- at and batch
- cron , crontab

Common vi editor command list

For this Purpose	Use this vi Command Syntax
To insert new text	esc + i (You have to press 'escape' key then 'i')
To save file	esc + : + w (Press 'escape' key then 'colon' and finally 'w')
To save file with file name (save as)	esc + : + w "filename"
To quit the vi editor	esc + : + q
To quit without saving	esc + : + q!
To save and quit vi editor	esc + : + wq
To search for specified word in forward direction	esc + /word (Press 'escape' key, type /word-to-find, for e.g. to find word '**shri**', type as **/shri**)
To continue with search	n
To search for specified word in backward direction	esc + ?word (Press 'escape' key, type word-to-find)
To copy the line where cursor is located	esc + yy
To paste the text just deleted or copied at the cursor	esc + p
To delete entire line where cursor is located	esc + dd
To delete word from cursor position	esc + dw
To Find all occurrence of given word and Replace then globally without confirmation	esc + :$s/word-to-find/word-to-replace/g For. e.g. :$s/mumbai/pune/g Here word "mumbai" is replace with "pune"
To Find all occurrence of given word and Replace then globally with confirmation	esc + :$s/word-to-find/word-to-replace/cg
To run shell command like ls, cp or date etc within vi	esc + :!shell-command For e.g. :!pwd

NOTE that following commands are for new users or for Beginners only. The purpose is if you use this command you will be more familiar with your shell and secondly, you need some of these command in your Shell script. If you want to get more information about any of the following command, use man page or info pages as

Syntax:
man {command-name}
Example:
$ man cat

Syntax:
info {command-name}
Example: $ info date

Even some command like mv, ls give command help on command line by typing command as follows

Syntax:
{command-name} --help
Example:
$ mv --help
$ ls --help | more

To see help or options related with ls command; here you can get screen-by-screen help, since help of ls command is quite big that can't fit on single screen.

cat
To display text file.
Syntax
cat {file name}
Examples
$cat foo

cfdisk
To create/modify/delete partition table interactively.
Syntax
Cfdisk

cd
To change current directory.
Syntax
cd {directory name}
Options
- (minus) Change to previous directory location.

Examples
$ cd /usr/local/apache
$ pwd
/usr/local/apache
$ cd /var/log
$ pwd
/var/log
$ cd –
$ pwd
/usr/local/apache
$ cd (Change to home directory)

chfn
Change the finger information of user.
Syntax
chfn [user-name]
Examples
$ chfn vivek
$ chfn # change the currently log on users finger information

chgrp
Change the group owner of a file.
Syntax
chgrp {group name} {file/directory name}
Options
-R Recursively change group owner of files/directory
Examples
chgrp oracle /usr/database
chgrp msc /home/cstudents

chmod
Change the file/directory permission directory.
Syntax
chmod permission-options file/directory name.
Options
(a)permission-options -
rwx rwx rwx
| | |
1 2 3
1 – owner
2 – group
3 – others

For r,w,x octal value is 4,2,1 respectively.
So if you want to give r,w,x permission to owner, for group r,x, and for w,x to others. Then,
rwx rwx rwx
421 401 021
For owner add this number as 4+2+1 = 7
For group add this number as 4+0+1 = 5
For others add this number as 0+2+1 = 3
So command will be $ chmod 753 filename

Examples

$chmod 755 first.sh

(b)permission-options -

chmod {u|g|o|a} {+|-} {r|w|x} {filename}

u - User who owns the file
g - Group file owner
o - User classified as other
a - All other system user
+ Set permission
- Remove permission

r - Read permission
w - Write permission
x - Execute permission

Examples

$ chmod u+x,g+wx,o+x myscript

Above command set permission for file called 'myscript' as User (Person who creates that file or directory) has execute permission (u+x) Group of file owner can write to this file as well as execute this file (g+wx) Others can only execute file but can not modify it, Since we have not given w (write permission) to them. (o+x).

chown

Change the owner of file/directory.

Syntax

chown {username}[.group-name] {file/directory-name}

Options

-R Recursively change group owner of files/directory

Examples
$chown vivek.users *.jsp
All files ending with .jsp are now owned by user vivek of the group users.

clear
Clear the screen.
Syntax
Clear

cp
To Copy files.
Syntax
cp {source} {destination}
Options
-f force the cp to copy files even if file exist (overwrite the files).
-i Ask y/n confirmation before coping each file.
Examples
$ cp -f /mnt/floppy/* /home/vivek

cmp
Syntax
cmp {file1} {file2}
Examples
$ cmp myfile myfile.old

date
Change or set current date and time.
Syntax
date [date or time string]
Examples
$date # will show current date & time
$date --date="2001-3-15" # will set date to 2001-Mar-15
$date --date="2001-3-15 11:59 AM" # date as well as time

To display file contains
Syntax
less filename

ln
To Create a link.
Syntax
ln -s {file/directory} {file/directory}

Examples
$ln -s p longprogname
$ls -l

locate
To quickly (securely also) search file.
Syntax
locate {file name}
Options
-R Recursively display the files & directory.
-l Long listing of files & directory.
Note
If you are running locate first time then run following command, which will create database file.
(Run as root user)
$ su
password
#/etc/cron.daily/slocate.cron
Examples
$locate -l foo

Ls
To Show list of files and dirs
Syntax
ls [file/dir name]
Options
-R Recursively display the files & directory.
-l Long listing of files & directory.
-a Show all files start with . (DOT) # for e.g.: .bash_profile
-N Name wise sorting.
Examples
$ ls
$ ls -lR / > alist &
$ ls –a

To view text file
Syntax
more {file name}

mv
To move the file(s)/directory.
Syntax
mv {source} {destination}

pr
To print file.
Syntax
pr {file-name}
Examples
$pr myfile

passwd
To change the password.
Syntax
passwd {username}
Options
-d Delete the password. (Use with very carefully)
Examples
$ passwd
To remove the files.
Syntax
rm {filename}
Options
-rf Remove all files/directory with subdirectories. (DOS's – deltree)
Examples
$ rm foo
$ rm -rf /home/vivek/oldfiles

sort
To sort files.
Syntax
sort {file-name}
Options
-r Reverse normal order
-n Sort in numeric order
-nr Sort in reverse numeric order
Examples
$sort myfile

su
Become super user if user name is not given, or change the user ID.
Syntax
su [username]

tar
Linux archive program.

Syntax
tar options {tar-file-name} {dir-name-to-archive}
Options
-c Create the file.
-f {filename} Name of archive file
-z Compress the file.
-x Extract the files from archive
Examples
# To create archive file of /home/vivek directory
$ tar -czf mybackup.tar.gz /hove/vivek
#If you want to extract file from above
$ tar -xf mybackup.tar.gz

top
To see process information in neat format.
Syntax
top
Options
? Get more information about top command.

touch
Create empty file or change/update time stamp of file.
Syntax
touch {filename}
Examples
$ touch /etc/dhcpd.leases #create file for dhcpd demon
$ touch /etc/modules # update time stamp of file

umask
Specify the permission for files when files are created for owner(u),group(g), and others(o).
Syntax
umask –S {u=xx,g=xx,o=xx}
Where xx can be (r)ead,(w)rite.
Or you can use 022 for read only file permission for others and 077 for read and write permission.
Examples
$umask -S u=rw,g=,o=
$umask 022

Useradd/userdel
To Add/delete new user to your system
Syntax
useradd -g {group-name} {user-name-to-add}

userdel {user-name-to-delete}

Examples

#add rani user to system (must be root)
#adduser -g oracle rani
#passwd rani
#Delete the user kaju
#userdel kaju

w

Shows who logged on and what they are doing

Syntax

w

who am i

About your self.

Syntax

who am i

which

Show the location of file from which it is executed when you type the name.

Syntax

which {file name}

Examples

$ which cc
$ ls –l `which cc`

SECTION 2

Shell Programming

- What Is The Shell and What Does It Do?
- Which Shell?
- Bourne Shell Programming
- Shell Features
- Bourne Shell Program
- Sample Shell Script

How to write shell script
Now we write our first script that will print "Knowledge is Power" on screen. To write shell script you can use Linux's text editor such as vi or mcedit or even you can use cat command. Here we are using cat command you can use any of the above text editor. First type following cat command and rest of text as its

$ cat > first
#
 # My first shell script
#
clear
echo "Knowledge is Power"

Press Ctrl + D to save. Now our script is ready. To execute it type command
$ ./first

This will give error since we have not set Execute permission for our script first; to do this type command
$ chmod +x first
$ ./first

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

We will write script to print command line argument and we will see how to access them

```
$ cat > demo
#!/bin/sh
# Script that demos, command line args
echo "Total number of command line argument are $#"
echo "$0 is script name"
echo "$1 is first argument"
echo $2 is second argument"
echo "All of them are :- $*"
```

Save the above script by pressing ctrl+d, now make it executable

```
$ chmod +x demo
$ ./demo Hello World
$ cp demo ~/bin
$ demo
```

Shell Script Exercise – 1

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(>0) for false.

Syntax: *test expression* OR *[expression]*

Now will write script that determine whether given argument number is positive. Write script as follows

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

Run it as follows
$ chmod +x ispostive
$ ispostive 5
Here o/p : 5 number is positive

$ispostive -45
Here o/p : Nothing is printed

$ispostive
Here o/p : ./ispostive: test: -gt: unary operator expected

The line, if test $1 -gt 0 , test to see if first command line argument($1) is greater than 0. If it is true(0) then test will return 0 and output will printed as 5 number is positive but for -45 argument there is no output because our condition is not true(0) (no -45 is not greater than 0) hence echo statement is skipped. And for last statement we have not supplied any argument hence error ./ispostive: test: -gt: unary operator expected is generated by shell , to avoid such error we can test whether command line argument is supplied or not.
test or [expr] works with
1.Integer (Number without decimal point)
2.File types
3.Character strings

Exercise

Q1 Write script to print the message “Hello” on the console.

Q2 Write script to perform following basic math operation as

e) addition
f) substraction
g) multiplication
h) division

Note : i) take input from keyboard
ii) take input as command line parameter

Q3 Write script to display current date, time, username and current directory

Q4 Write shell script to show various system configuration like

h) Currently logged user and his long name
i) Current shell
j) Your home directory
k) Your operating system type
l) Your current path setting
m) Your current working directory
n) Show all available shells

Q5 Write a script to convert the contents of a given file from uppercase to lowercase and also count the number of lines, words and characters of the resultant file. Also display the resultant file in descending order

Shell Script Exercise – 2
(Conditional statements and Loops)

if...else...fi

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

Syntax:

if condition
then
command1 if condition is true or if exit status
of condition is 0(zero)
...
...
else
command2 if condition is false or if exit status
of condition is >0 (nonzero)
...
...
Fi

For eg. Write Script as follows

```
$ cat > isnump_n
#!/bin/sh
#
# Script to see whether argument is positive or negative
#
if [ $# -eq 0 ]
then
echo "$0 : You must give/supply one integers"
exit 1
fi
if test $1 -gt 0
then
echo "$1 number is positive"
else
echo "$1 number is negative"
fi
```

Try it as follows

$ chmod +x isnump_n
$ isnump_n 5
Here o/p : 5 number is positive

$ isnump_n -45
Here o/p : -45 number is negative

$ isnump_n
Here o/p : ./ispos_n : You must give/supply one integers

$ isnump_n 0
Here o/p : 0 number is negative

Here first we see if no command line argument is given then it print error message as "./ispos_n : You must give/supply one integers". if statement checks whether number of argument ($#) passed to script is not equal (-eq) to 0, if we passed any argument to script then this if statement is false and if no command line argument is given then this if statement is true. The echo command i.e. echo "$0 : You must give/supply one integers"
| |
| |
1 2
1 will print Name of script
2 will print this error message
And finally statement exit 1 causes normal program termination with exit status 1 (nonzero means script is not successfully run), The last sample run $ isnump_n 0 , gives output as "0 number is negative", because given argument is not > 0, hence condition is false and it's taken as negative
number. To avoid this replace second if statement with if test $1 -ge 0.

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

For eg. Write Script as follows

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

Run it as,

```
$ chmod +x testfor
$ ./testfor
```

The for loop first creates i variable and assigned a number to i from the list of number from 1 to 5, The shell execute echo statement for each assignment of i. (This is usually know as iteration) This process will continue until all the items in the list were not finished, because of this it will repeat 5 echo statements.

for e.g. Now try script as follows

```
$ cat > mtable
#!/bin/sh
#
#Script to test for loop
##
if [ $# -eq 0 ]
then
echo "Error - Number missing form command line argument"
echo "Syntax : $0 number"
echo " Use to print multiplication table for given number"
exit 1
fi
n=$1
for i in 1 2 3 4 5 6 7 8 9 10

do
echo "$n * $i = `expr $i \* $n`"
done
```

Save and Run it as

```
$ chmod +x mtable
$ ./mtable 7
$ ./mtable
```

For first run, Above program print multiplication table of given number where i = 1,2 ... 10 is multiply by given n (here command line argument 7) in order to produce multiplication table as
7 * 1 = 7
7 * 2 = 14
...
..
7 * 10 = 70
And for Second run, it will print message - Error - Number missing form command line argument Syntax : ./mtable number
Use to print multiplication table for given number
This happened because we have not supplied given number for which we want multiplication table, Hence we are showing Error message, Syntax and usage of our script. This is good idea if our program takes some argument, let the user know what is use of this script and how to used it. Note that to terminate our script we used 'exit 1' command which takes 1 as argument (1Indicates error and therefore script is terminated)

while loop

Syntax:

while [condition]
do
command1
command2
command3
..
....
Done

Loop is executed as long as given condition is true. For eg. Above for loop program can be written using while loop as

$cat > nt1
#!/bin/sh
#
#Script to test while statement
##
if [$# -eq 0]
then
echo "Error - Number missing form command line argument"
echo "Syntax : $0 number"
echo " Use to print multiplication table for given number"
exit 1

```
fi
n=$1
i=1
while [ $i -le 10 ]
do
echo "$n * $i = `expr $i \* $n`"
i=`expr $i + 1`
done
```

Save it and try as

```
$ chmod +x nt1
$./nt1 7
```

Above loop can be explained as follows

n=$1	Set the value of command line argument to variable n. (Here it's set to 7)
i=1	Set variable i to 1
while [$i -le 10]	This is our loop condition, here if value of i is less than 10 then, shell execute all statements between do and done
do	Start loop
echo "$n * $i = `expr $i \* $n`"	Print multiplication table as 7 * 1 = 7 7 * 2 = 14 .... 7 * 10 = 70, Here each time value of variable n is multiply be i.
i=`expr $i + 1`	Increment i by 1 and store result to i. (i.e. i=i+1) Caution: If we ignore (remove) this statement than our loop become infinite loop because value of variable i always remain less than 10 and program will only output 7 * 1 = 7 ... ... E (infinite times)
done	Loop stops here if i is not less than 10 i.e. condition of loop is not true. Hence loop is terminated.

From the above discussion not following points about loops
(a) First, the variable used in loop condition must be initialized, Next 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.

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 eg. Create 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
```

Save it by pressing CTRL+D

```
$ chmod +x car
$ car van
$ car car
$ car Maruti-800
```

Here first we will check, that if $1(first command line argument) is not given set value of rental variable to "*** Unknown vehicle ***",if value given then set it to given value. The $rental is compared against the patterns until a match is found. Here for first run its match with van and it will show output For van Rs.10 per k/m. For second run it print, "For car Rs.20 per k/m". And for last run, there is no match for Maruti-800, hence default i.e. *) is executed and it prints, "Sorry, I can not gat a Maruti-800 for you". Note that esac is always required to indicate end of case statement.

Exercise

Q1 Write a shell script to list the files arranged in descending order of their size.
Q2 Write a shell script to find the reverse of a given number.
Q3 Write a shell script to print the contents of the file from given line numbers to next given number line.
Q4 Write a shell script to print the pattern

```
1
2   2
3   3   3
4   4   4   4
```

Shell Script Exercise – 3

Shell Built in Variables

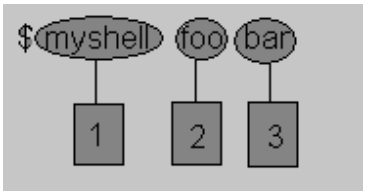
Shell Built in Variables	Meaning
$#	Number of command line arguments. Useful to test no. of command line args in shell script.
$*	All arguments to shell
$@	Same as above
$-	Option supplied to shell
$$	PID of shell
$!	PID of last started background process (started with &)

Why Command Line arguments required

1. Telling the command/utility which option to use.
2. Informing the utility/command which file or group of files to process (reading/writing of files).

Let's take rm command, which is used to remove file, but which file you want to remove and how you will tail this to rm command (even rm command don't ask you name of file that you would like to remove). So what we do is we write command as follows:
$ rm {file-name}
Here rm is command and filename is file which you would like to remove. This way you tail rm command which file you would like to remove. So we are doing one way communication with our command by specifying filename Also you can pass command line arguments to your script to make it more users friendly. But how we access command line argument in our script.

Lets take ls command
$ ls -a /*
This command has 2 command line argument -a and /* is another. For shell script,
$ myshell foo bar



1 Shell Script name i.e. myshell

2 First command line argument passed to myshell i.e. foo

3 Second command line argument passed to myshell i.e. bar

In shell if we wish to refer this command line argument we refer above as follows

1 myshell it is $0

2 foo it is $1

2 bar it is $2

Here **$#** (built in shell variable) will be 2 (Since foo and bar only two Arguments), Please note at a time such 9 arguments can be used from $1..$9, You can also refer all of them by using $* (which expand to `$1,$2...$9`). Note that $1..$9 i.e command line arguments to shell script is know as "*positional parameters*".

Following script is used to print command ling argument and will show you how to access them:

```
$ vi demo
#!/bin/sh
#
# Script that demos, command line args
#
echo "Total number of command line argument are $#"
echo "$0 is script name"
echo "$1 is first argument"
echo "$2 is second argument"
echo "All of them are :- $* or $@"
```

Run it as follows

Set execute permission as follows:
$ chmod 755 demo

Run it & test it as follows:
$ ./demo Hello World

If test successful, copy script to your own bin directory (Install script for private use)
$ cp demo ~/bin

Check whether it is working or not (?)
$ demo
$ demo Hello World

Note that you ***can't assigne the new value to command line arguments i.e positional parameters***. So following all statements in shell script are invalid:
$1 = 5
$2 = "My Name"

Exercise

Q1 Write a script to determine whether given file exist or not, file name is supplied as command line argument, also check for sufficient number of command line argument.

Q2 Write a script to determine whether given command line argument ($1) contains "*" symbol or not, if $1 does not contains "*" symbol add it to $1, otherwise show message "Symbol is not required".

Q3 Write a script to check whether the given string is palindrome or not.

ADVANCE EXPERIMENTS

Advance Experiments 1

Tasks : (Command Line Filesystem Browsing)

1. Login on tty as user "**root**" with the password of "**test123**"
2. Immediately after logging into the system, you should be in your home directory. Verify this using the "**print working directory**" command.
3. Check to see if you have any files in your home directory using each of the following commands:
   $ ls
   $ ls –a
   $ ls –al
   Why do the first and second command return different numbers of files?
   What is the size of the largest file and the largest directory currently in your home directory as reported by the third command?
4. Now use "touch" command to create the files needed for the remaining tasks.
   Files needed are {**report, graph}_{jan,feb,mar**}.
5. Use the "**ls**" command to examine the results of the last command.
6. In order to organize your files, you must first create some new directories in the order given below. As you change directories, below, be sure to check that your working directory is as expected.
   $ mkdir projects
   $ mkdir projects/graphs
   $ cd projects
   $ pwd
   $ mkdir reports
   $ cd reports
   $ pwd
   $ mkdir ../backups
   Use "**ls**" to examine your work.
7. Begin by moving all of the "**graph**" files into the **graphs** subdirectory of the **projects** directory. Do this in two steps: in the first step, move one file; in the second step, move two files. Use "**ls**" to examoine your work.
8. Next, move two of the "**report**" files into the **reports** subdirectory of the **projects** directory. Move the files in one command.
9. Remove the remaining **report** file.
10. Change into the **backups** directory and copy the **januray** files into this directory. Copy one using an absolute pathname and the other using a relative pathname.

Advance Experiments 2

Task1: File permissions

1. Symbolically (e.g., rwx_wx__x), what are the permissions
   644:
   755:
   000:
   711:
   700:
   777:
   555:
   111:
   600:
   731:
2. Given a file with permissions 755, what commands would change the permissions to r_xr_ _ r _ _?
3. Given an executable file to download what step must be performed to run it. List two different ways to perform that step.

Task2: creating the groups and users

1. Using the **userdd** command, add accounts for the following users **john, alex, dave, bob, zak, eddy** and **manager**. Remember to give each user a password.
2. Using the **groupadd** command, add the following groups to your system. Use the **–g** option to set the correct GID.

Group	**GID**
Sales	10000
hr	10001
web	10002

   Why should you set the Gid in this manner instead of allowing the system to set the GID by default?
3. Using the **usermod** command add **john** and **alex** to the **sales** group, **dave** and **bob** to the **hr** group. Add **zak** and **eddy** to the **web** group. Add **manager** to all groups. The **–G** option can be used to add users to supplemental groups.
4. Login as each user and use the **id** command to verify that they are in the appropriate groups. How else might you verify this information?

Task3: setting up shared directories

1. Create a directory called **/depts**. With a **sales**, **hr** and **web** directory within the **/depts** directory.
   mkdir –p /depts/{sales,hr,web}
2. Using the **chgrp** command, set the group ownership of each diretory to the group with the matching name, for example **sales -> /depts./sales**
3. Set the permissions on the **/depts** directrory to **755**, and each **subdirectory** to **770**.
4. Set the **sgid bit** on each **departmental** directory so that files created within those directories will be owned by the appropriate group.
5. Experiment by logging in as each user and creating or altering files in each of the directories. You can also use **su -** , but make sure to include the dash and to exit one **su** session before starting another.

Advance Experiments 3

Task1: Local user logins

1. Switch to virtual terminal **tty1**
2. Login into your workstation
3. Determine information about this specific login. Run the commands:
   a. $**whoami**
   b. $**groups**
   c. $**id**
   Examine the output of the commands
4. Determine information about all concurrent logins on the workstation. Examine the output of the commands
   a. $ **users**
   b. $ **who**
   c. $ **w**
5. Switch to virtual terminal **tty2**
6. Login into your workstation
7. Determine information about this specific login. Run the commands:
   a. $ **whoami**
   b. $ **groups**
   c. $ **id**
   Examine the output of the commands
8. Determine information about all concurrent logins on the workstation
   a. $ **users**
   b. $ **who**
   c. $ **w**
   Examine the output of the commands

Task2: Switching user accounts

1. Switch to virtual terminal
2. Run the command to determine your user information and your current directory. Record the results **id= , pwd=**
3. Use **su –** student to switch to user student. Run id and pwd to determine your current directory. Record the results.
4. Exit out of the student account, returning to the visitor account.
5. Use **su** student without the **–** to switch to user student. Run **pwd** and **id**, Record the results.
   How do the results differ from step 3? Why?
6. Logout of all the shells that you used during this sequence.

Task3: Using umask to set default permissions on newly-created files

1. Log in
2. View your current **umask**
3. Create a couple of files (**umtest1, umtest2)** using touch command and a directory **umtestdir1**
4. Cange your umask to a move secure setting. Create new files and a directory.
   a. $ **umask 027**
   New files are **umtest3, umtest4**
   Directory is **umtestdir2**

   Before looking at the permissions, what would you expect them to be?
   umtest1: _____
   umtest2: _____
   umtestdir1: _____
   umtest3: _____
   umtest4: _____
   umtestdir2: _____
   List the files to see if you are correct:
   $ **ls –ld um***

Task4: setting a umask

1. Switch to **tty1**
2. Log in a user **visitor** with a password of visitor.
3. Display your current **umask**.
4. Below is a table of umasks. Fill in the table with the permissions of files and directories given the umask.

Umask	**Directory permissions**	**File permissions**
002		
022		
007		
027		
077		

Advance Experiments 4

Task 1: Types of files

1. Create both a symbolic and hard link in your home directory that point to the ***words*** file in your home directory.
2. Test that your new links both function as pointer to the words data (use **head** command)
3. Examine the links that you have created with the commands that follow, then answer the question below

   $ **ls –il hard soft**
   $ **stat hard soft**

   a) Reported file size of Hard _____ and soft_____.
   b) Actual number of data blocks used by hard_____and soft_____.
   c) How would you explain the discrepancy in blocks used by the two links?
   d) Link count listed for Hard _____ and soft_____.
   e) Ownership (UID\GID) for Hard _____ and soft_____.

4. Can you make a hard link to a target that does exist? why or why not?
5. Can you make a symbolic link to a "**target**" that does not exist? Does the output of **ls** give you any indication of this condition?
6. Can you make a hard link to a symbolic link? What happen when you try?

Task 2

1. Use **df** to determine the amount of free space on your filesystem.
2. Try using the **–h** and **–H** option of df command to report total in "human Readable" sizes instead, what is the difference between the two switch?
3. Use the command from your home directory to determine how much space all Of your files are consuming. Provide more readable output.

Question Bank

(Linux and X - windows Programming)

Q.1.How to Define variable x with value 10 and print it on screen.
$ x=10
$ echo $x

Q.2.How to Define variable xn with value Rani and print it on screen
$ xn=Rani
$ echo $xn

Q.3.How to print sum of two numbers, let's say 6 and 3
$ echo 6 + 3
This will print 6 + 3, not the sum 9

$ expr 6 + 3
Now It will print sum as 9 , But
$ expr 6+3
will not work because space is required between number and operator

Q.4.How to define two variable x=20, y=5 and then to print division of x and y (i.e. x/y)
$x=20
$ y=5
$ expr x / y

Q.5.Modify above and store division of x and y to variable called z
$ x=20
$ y=5
$ z=`expr x / y`
$ echo $z

Q6. There is file called foo, on your disk and you give command, $ ./trmfi foo what will be output.
foo file will be deleted, and message "foo file deleted" on screen will be printed.

Q7. If bar file not present on your disk and you give command, $ ./trmfi bar what will be output.
Message "rm: cannot remove `bar': No such file or directory" will be printed because bar file does not exist on disk and we have called rm command, so error from rm commad

Q8. If you type $ ./trmfi, What will be output.
Message “rm: too few arguments” will be shown by rm command, because rm is called from script without any parameters.

Q9. If you want to print your home directory location then you give command:
(a) $ echo $HOME or

(b) $ echo HOME

Which of the above command is correct & why?

(a) command is correct, since we have to print the contains of variable (HOME) and not the HOME. You must use **$** followed by variable name to print variables cotaines.

Q10. "Currently only two Process are running in your Linux/PC environment", Is it True or False?, And how you will verify this?
No its not true, when you start Linux Os, various process start in background for different purpose. To verify this simply use **top** or **ps aux** command.

Q11. What is Shell Script ?

*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*

Q12. 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.

Q13. 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`

Q14. 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
```

Q15. How do I find all file have *.c ?
$ find / -name *.c -print

Q16. How do I find all core file and delete them ?
$ find / -name core* -ok rm {} \;

Q17. How do I find all *.c files having 2kb file size?
Syntax: find -size number[ckwb]
number - It's 2 byte or 1 kilobyte etc
c - bytes
k - kilobytes
w - words (2-byte)
b - 512-byte block
Example:
$ find / -name *.c -size 2k -print

Q18. How do I find all files access 2 days before?
Syntax: find -type f -atime -days
Example:
$ find /home/alok -type f -atime -2

Q19. How do I find all files NOT accessed in a given period (2 days)?
Syntax: find -type f -atime +days
Example:
$ find /home/alok -type f -atime +2

Q20. How do I find all special block files in my system?
Syntax: find -type [bcdpfls]
b block device
c character device
d directory
p name pipe (FIFO)
f regular file
l symbolic link
s socket
Example:
$ find / -type b -print

Using whereis

Use to find binary, source and man pages files for command.
Syntax: whereis {filename}

Q21. How do I find location of ls (binary program) and it's man page location?
$ whereis ls

Q22. What is a File?
File are collection of data items stored on disk. Or it's device which can store the information/data/music/picture/movie/sound/book; In fact what ever you store in computer it must be inform of file. Files are always associated with devices like hard disk ,floppy disk etc. File is the last object in your file system tree.

In Linux files have different types like

- Executable files (stored in /bin, /usr/bin)
- Special files (stored in /dev)
- Configuration files (stored in /etc)
- Directory
- User files or ordinary files etc

Q23. How do I rename all my .htm files to .html files?
Opps! mv will not work, it works with two files only. Try ***rename*** as
Syntax: rename from to files-to-rename
Example:**$ rename .htm .html *.htm**