LABORATOTY MANUAL

CSE105

CREATIVE ENGINEERING WORKSHOP

Prepared By:

Baljit Singh Saini

Name of the Student: …………………………………………

Registration Number/Roll Number: ………………………….

Section and Group: ……………………………………………

School of Computer Science and Engineering

General Guidelines for the students

1. Completion of Lab manual exercises will be counted in the CA marks.
2. Sit according to roll number and occupy the designated system only.
3. Always use the same system throughout the semester.
4. Reporting any issues related to the designated system to the teacher is student’s responsibility.
5. Self-practice is the key to understand concepts of this course
6. Electronic devices should be used on a professional level.
7. Do not change the system settings without seeking prior approval from the teacher.

Lab 1- Installation

Aim: To know the installation process for Linux

Learning Objective: Student will learn how to install a Linux distribution in his computer system.

Theory: There are basically four methods by which you can install Linux distribution in your system:

1. Re-partitioning your hard disk to free up enough room to permit dual boot (side-by-side) installation of Linux, along with your present operating system.
2. Using a host machine hypervisor program (such as VMWare's products or Oracle Virtual Box) to install a client Linux Virtual Machine.
3. Booting off of and using a Live CD or USB stick and not writing to the hard disk at all
4. Activating Linux Bash Shell on Windows 10

Outline of Procedure:

A web link is provided below for each of the four methods. These are by no means the only links that explain the process, you can search for other videos or tutorial on the internet.

1. Re-partitioning
   https://www.youtube.com/watch?v=vt5Lu_ltPkU
2. Virtual machine
   https://www.lifewire.com/run-ubuntu-within-windows-virtualbox-2202098
3. Live CD
   https://help.ubuntu.com/community/LiveCD
4. Linux bash shell
   https://docs.microsoft.com/en-us/windows/wsl/install-win10

Learning Outcomes (What I have Learnt):

S.No	Parameter	Marks Obtained	Max. Marks
1	Understanding of the student about the Procedure		20
2	Observations and analysis including learning Outcomes		20
3	Completion of experiment		10
	Signature of Faculty		

<u>Lab 2 - General Purpose Utilities</u>

Aim: To teach the use of commands like cal, date, echo, printf, bc and script

Learning Objective: Students will learn about the use of simple commands that will help them get started and interact with the system using the CLI.

Theory:

cal – shows the calendar. By default, it displays the calendar of current month.

date – check system date and time

echo – to print a line of text

printf – to format and print data

bc – precision language to help to calculations

script – to record a terminal session

passwd – to change the password for the current user

who – to check which are the currently logged in users

Outline of the Procedure:

1. **cal**
   $ cal
   Output: will display the current month
   $ cal 2018
   Output: will show calendar for entire year (2018)
2. **date**
   $date
   Output: the current date. E.g.
   Wed Jul 18 10:26:57 DST 2018
3. **echo**
   $echo “Line of Text”
   Output: Line of Text
4. **printf**
   $printf “hello”
   Output: hello$*yourcommandprompt*
5. **bc**
   $ bc
   Now you can perform any calculation e.g.
   1+3
   Output: 4
   To exit press Ctrl + d
6. **script**
   $script *file1*
   (Now your session will start recording in file *file1*. Execute cal and date command)
   To stop recording press *Ctrl + d*
   To view the session
   $ cat file1

7. **passwd**
   $passwd
   The system will ask you to enter the current and the new passwd

Further Reading:

Exercises:

1. Display the calendar for June 2050.
2. What was the day on 5th Aug 2000?
3. Display todays date in the following format

   YYYYMMDD

4. Display the following text on screen using echo command

   First Line of Input

   Second Line of input

5. Display the following text on screen using printf command

   First Line of Input

   Second Line of input

6. Use script command to record a session showing the use of bc command

Learning Outcomes (What I have Learnt):

S.No	**Parameter**	**Marks Obtained**	**Max. Marks**
1	Understanding of the student about the Procedure		20
2	Observations and analysis including learning Outcomes		20
3	Completion of experiment		10
	Signature of Faculty		

Lab 3 – Basic commands

Aim: The aim of this lab is study commands like pwd, man and ls.

Learning Objective: Students will learn about how to get information about any command and use it using manual page.

Theory:

pwd – stands for *present working directory*. This command prints the location of the current directory in which the user is working

man – man command shows the manual page of any command. The manual page contains all the information related to a command. In case you want to understand how a command works always investigate its manual page.

ls – ls command is used to list directory contents. The contents refer to files as well as sub-directories.

Outline of the Procedure:

1. **pwd**
   $pwd
   Output : Full path of your current working directory (something like /home/user)
2. **man**
   $man pwd
   Output: shows the manual page of pwd command
   Note: press "*q*" to exit the manual page
3. **ls**
   $ls
   Output: displays the list of files and sub-directories
   $ls -l
   Output: Displays the long-list of files and sub-directories. Something like below

   ```
   drwxrwxrwx 1 root   root     4096 Nov 16  2017 udacity
   -rwxrwxrwx 1 root   root       78 Jun 22 14:47 until
   ```

   Note: all entries which start with 'd' are for directories while all entries which start with '-' represent files.

Exercises:

1. How many options can be used with pwd command?
2. What is the command to see hidden files?
3. How can you check the size of a file?
4. What is the command to display the contents of a directory in reverse order.
5. Write the command to see the contents to current directory as well as the all the sub-directories.

Learning Outcomes (What I have Learnt):

S.No	**Parameter**	**Marks Obtained**	**Max. Marks**
1	Understanding of the student about the Procedure		20
2	Observations and analysis including learning Outcomes		20
3	Completion of experiment		10
	Signature of Faculty		

Lab 4 – File Related Commands

Aim: To study about commands to create, delete and modify files

Learning Objective: To teach about the working of touch, mkdir, rm, rmdir, cat, cp, mv and cd commands.

Theory:

touch - command is used to update the time stamp of a file. But it can also be used to create new files.

mkdir - is used create new directories.

rm - command is used to delete files.

rmdir - command is used to delete directories.

cat – cat command is used to concatenate files and print on the monitor. It can also be used to write some content into a file.

cp – is used to copy files from one directory to another. This is analogous to copy-paste operation.

mv – is used to move files from one directory to another. This is analogous to cut-paste operation. mv can also be used to rename files.

cd - command is used to change the current working directory.

Outline of the Procedure:

1. **touch**
   -to update timestamp

   Consider any file in your current directory.
   ```
   $ ls -l test1
   -rwxrwxrwx 1 root root 35 Nov 16  2017 test1
   ```
   Note the date
   ```
   $ touch test1
   $ ls -l test1
   -rwxrwxrwx 1 root root 35 Jul 18 14:17 test1
   ```
   The date is changed i.e. the timestamp is updated

   -to create a new file
   ```
   $touch xyz
   ```
   It will create a new file with the name xyz if no file with this name exists. Any number of files can be created by separating the file names with spaces
   ```
   $touch file1 file2 file3
   ```
2. **mkdir**
   ```
   $mkdir dir
   ```
   Output: a directory with the name *dir* will be created
   ```
   $mkdir dir1 dir2 dir3
   ```
3. **rm**
   ```
   $ls
   File1    file2     file3     Dir1
   $rm File1
   ```

```
$ls
file2     file3     Dir1
```
Output: the command rm deletes the file File1

4. **rmdir**

```
$ls
file2     file3     Dir1
$rmdir Dir1
$ls
file2     file3
```
Output: rmdir deletes the directory Dir1

5. **cat**

- to write content into a file

```
$cat > file1

Write the content

You want

Press Ctrl+d to end
```

- to view the content of a file

```
$cat file1
```

Output: will display the contents of file *file1*

- to concatenate two files

```
$cat file1 file2
```

Output: Will show the contents of file1 followed by file2

```
$cat file1 file2 > file3
```

Output: Will join the contents of file1 and file2 and redirect it to file3

6. **cp**

```
$ls
File1    file2     Dir1
```
Suppose you want to copy File1 to Dir1
```
$cp File1 Dir1
$ls Dir1
File1
$ls
File1    file2     Dir1
```

7. **mv**

- to move file from one directory to another

```
$ls
File1    file2     Dir1
```
Suppose you want to move File1 to Dir1
```
$mv File1 Dir1
$ls Dir1
File1
```

```
$ls
file2     Dir1
```

Note: File1 is not in your current directory now.

- To rename a file

```
$ls

File1    file2     Dir1
```

Suppose you want to rename File1 to XYZ

```
$mv File1 XYZ

$ls

XYZ    file2     Dir1
```

8. cd

```
$ls
File1    file2     Dir1
$pwd
```

This will show the path of current directory. Let us suppose /home/user. Now Dir1 is a directory here

```
$cd Dir1
$pwd
```

Output: /home/user/Dir1

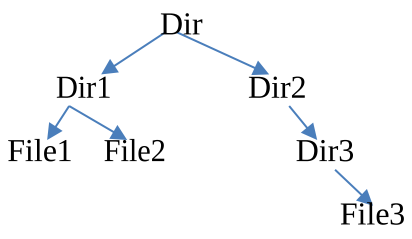
The current directory is changed and is now Dir1.

```
$cd..
```

Output: The parent directory will become the current directory

Exercises:

1. Use touch in such a manner that it does not create a new file (if it does not already exists).
2. Create a hierarchy of directories A, B and C such that directory C is inside B and B is inside A by using mkdir only once.
3. Create the following structure



4. Suppose your current working directory is /home/user. It has two file f1 and f2 and two directories d1 and d2. Write the commands to do the following without changing your directory
   a. Create a file inside directory d1.
   b. Copy the file from d1 to directory d2

    c. Delete the directory d2

5. Write the command to append some contents to a file.
6. Create two files F1 and F2. Create a directory D1. Copy both the files inside D1 by using cp command only once
7. Write the command to one directory into another directory
8. Can a directory be renamed using mv?
9. Differentiate between the use of soft and hard links
10. Consider a directory D1 having sub-directory D2 and files F1 and F2. The current working directory is also /home/user/D1. What will happen if the following command is issued:

```
$rm -r /home/user/D1
```

Learning Outcomes (What I have Learnt):

S.No	**Parameter**	**Marks Obtained**	**Max. Marks**
1	Understanding of the student about the Procedure		20
2	Observations and analysis including learning Outcomes		20
3	Completion of experiment		10
	Signature of Faculty		

Lab 5 – File Permissions

Aim: To study the use of chmod and umask commands

Learning Objective: To teach the concepts of permission associated with files and how to modify the permissions.

Theory:

Every file has its own access rights for different groups of users. Access rights are – Read, Write and Execute. These rights are provided to different groups of users which are- Owner, Groups, Other Users. A file's permission appears to be in format of three group of letters. Here three groups represent the permissions given to groups of the user.

chmod command is used to change the permissions of files and directories. umask command is used to set the default permissions that will be given to files and directories.

Outline of the Procedure:

We can view the permissions of a file by using 'ls -l' command in the terminal. For example let's say we have output in terminal as follows:

```
$ ls –l

rwx-r--r-- 1 root root 415 Feb 19 21:04 file1.txt

-rwxrx-r-- 5 root root 21 Feb 19 21:09 file2.txt
```

Then to deduce the permissions of a file: we pick up the data 'rwxr--r--' printed in front of each file. Here we will break down the data into three parts.

First part: rwx : Represents the access rights of the owner of the file. Second part: r-- : Represents the access rights of group.

Third part: r-- : Represents the access rights of other users.

Here, meaning of symbols is as following: r- readable, w- writeable and x- executable.

Now in order to change the permissions we use 'chmod' command.

Suppose we want the permissions of file1.txt to be Readable, Writeable and Executable for Owner.

Readable and Executable for Group.

No Permissions for others.

```
$ chmod 750 file1.txt

$ls -l

rwx-r-x--- 1 root root 415 Feb 19 21:04 file1.txt
```

Here Octal Code represents permissions to a file. When in this Octal Code we use:

7- We are granting all rights- reading, writing and executing a file

6- We are granting two rights- reading and writing on file.

5- We are granting two rights- read and execute file.

4- We are granting one right- read a file.

3- We are granting two rights – write and execute a file. 2- We are granting one right- write on a file.

1- We are granting one right- execute a file. 0- We are granting no rights at all.

While doing so we need to add three of these to permissions Octal Code i.e. – one for the owner, second for the group and third for other users.

The above method is called the absolute method. chmod can also be used in relative method. In this method we tell which permissions is to be granted and which permission is to be denied.

```
$ls -l
```

```
rwx-r--r-- 1 root root 415 Feb 19 21:04 file1.txt
```

Now suppose that you want to deny execute permissions to owner.

```
$chmod u-x file1.txt
```

```
$ls -l
```

```
rw--r--r-- 1 root root 415 Feb 19 21:04 file1.txt
```

Now suppose that you want to allow write permissions to group.

```
$chmod g+w file1.txt
```

```
$ls -l
```

```
rw--rw-r-- 1 root root 415 Feb 19 21:04 file1.txt
```

In the similar manner chmod can be used to change permissions of directories also.

chmod can change permissions of existing files and directories. But if you want to change the default permissions that are given when you create a new file or directory then the command is *umask.*

Suppose you want that every newly created file be given the following permissions

Read and write to the owner

Read to the group

None to others

```
$umask 137
```

The interpretation of the octal number is opposite to chmod. Here 1 means execute permission is denied to the owner, 3 means write and execute and denied to group and 7 means no permission to others.

Note: A regular can never be given execute permission by default.

Exercises:

1. Create a file F1. Check the existing permissions. Change the permissions to Read and write to owner, write to group and read and execute to others.
2. Suppose the existing permissions of a file F are rw-rw-r—
   a) Using relative method allow execute permission to all.
   b) Deny write permission to group members
3. Change the default permissions of directories to “rwx” for owner, “rw-“ for group and “r—” for others.
4. Change the default permissions of files to “rwx” for owner, “rw-“ for group and “r—” for others.
5. Create a directory D1. Check the existing permissions. Change the permissions to Read, write and execute for owner, write for group and none for others.

Learning Outcomes (What I have Learnt):

S.No	Parameter	Marks Obtained	Max. Marks
1	Understanding of the student about the Procedure		20
2	Observations and analysis including learning Outcomes		20
3	Completion of experiment		10
	Signature of Faculty		

Lab 6 – grep command

Aim: To study the working of grep command

Learning Objective: Student will learn how to search content within a file

Theory: grep command is used to search pattern within a file. The output of this command is the entire line containing the desired pattern. Note that pattern can be a word or a part of word. E.g. if you are looking for the pattern ‘is’ then grep will look for ‘is’ as well as ‘th**is**’ since ‘is’ appears in the word ‘this’.

Outline of the Procedure:

Create a file **sample.txt (tab separated)** having the content

```
        This is a sample

        File containing

        Three lines.
```

```
$grep “sample” sample.txt
```

Output:

```
This is a sample
```

i.e. line which contains the pattern “sample”

```
$grep -c “in” sample.txt
```

Output:

```
2
```

-c options with grep displays the count of lines containing the pattern.

Regular Expression with grep

Regular expression is a pattern that illustrates a string. Regular expression are constructed with the help of character classes explained below:

. (period) – matches a single character

*(Asterisk) – matches the previous character zero or more times

[a1v] – any one character out of a or 1 or v

[^a1v] – any character except a, 1 and v

^ - lines starting with e.g. ^a will match lines starting with a

$ - lines ending with e.g. a$ will match lines ending with a

{n} – the preceding character will be matched n times

Extended expression:

+ – matches the previous character one or more times

? – matches the previous character zero or one time

Note: Use -E option with grep for extended expressions

Outline of the Procedure:

Create a file **sample.txt (tab separated)** having the content

```
	This is a sample
	File iss containing
	Three lines.
```

```
$grep “is*” sample.txt
```

Output:

```
	This is a sample
	File iss containing
	Three lines.
```

```
$grep -E “is+” sample.txt
```

Output:

```
	This is a sample
	File iss containing
```

```
$grep “i[ln]” sample.txt
```

Output:

```
	File iss containing
	Three lines.
```

```
$grep “^T” sample.txt
```

Output:

```
	This is a sample
	Three lines.
```

Exercises:

Create a file *test.t*

```
Sample file

File for testing

1Understanding of grep command

2 Grep is used to filter

Text IN fileee

Ample content written
```

1. How many lines contain the pattern “file”?
2. Write command to find out lines that start with a digit.
3. Display lines which contain the pattern ‘an’ or ‘am’
4. Display the lines along with line numbers that contain “in” (case insensitive)
5. What is the pattern to search for empty lines?
6. How many lines have a word with exactly 2 characters?
7. Write the command to search for either ‘of’ or ‘IN’.

Learning Outcomes (What I have Learnt):

S.No	Parameter	Marks Obtained	Max. Marks
1	Understanding of the student about the Procedure		20
2	Observations and analysis including learning Outcomes		20
3	Completion of experiment		10
	Signature of Faculty		

Lab 7 – Redirection and Pipes

Aim: To study about the concepts of input/output redirection and pipes.

Learning Objective: Student will learn how to redirect output of any command into a file and how to make output of one command to act as input of another command.

Theory:

Redirection – Redirection means changing the standard input or output device. It is of two types : input (<) and output (>). With input redirection the input can be read from a file rather than from keyboard. With output redirection the output can be directed into a file rather than to monitor.

Pipes – Pipes are used to connect two processes (commands). The symbol for pipe is '|'. With pipes the output of one command can act as an input for another command.

Outline of the Procedure:

Suppose that you want to save the contents of a directory into a file. To list the content of directory the command used is 'ls'

```
$ls
```

But this will display the contents on screen. Now to save them we have to redirect the output of 'ls' into file

$ls>*filename*

Output:

The output of 'ls' command is saved in a file *filename*

Now consider that you want the list of those files which contain the pattern 'test'. For this we need two commands: ls to list the files and grep to search for pattern. Also, grep needs to work on the output of ls command. Hence, we use pipes (|) to make the output of ls work as input of grep

```
$ ls | grep "test"
```

Exercises:

1. Find the number of files whose size is more than 500bytes
2. Count the total number of files in your current working directory
3. Count the total number of sub-directories in your current working directory
4. Copy the contents of file F1 into file F2.
5. Copy the contents of file F1 into file F2 without overwriting the original contents of F2.

Learning Outcomes (What I have Learnt):

S.No	Parameter	Marks Obtained	Max. Marks
1	Understanding of the student about the Procedure		20
2	Observations and analysis including learning Outcomes		20
3	Completion of experiment		10
	Signature of Faculty		

Lab 8 – System Admin commands

Aim: To study about user management and group management commands

Learning Objective: To teach how to create new users and groups and how to manage existing users and groups.

Theory: An administrator must continuously add new users and groups. Also, he needs to manage existing users as well. Linux provides certain commands which help in the same. The commands for managing users are useradd, userdel and usermod. Commands related to group management are groupadd, groupdel and groupmod. To run these commands one need to login as an administrator.

Outline of the Procedure:

Login as admin

```
$su
```

Enter password for admin account. Your command prompt will change to '#'

Create a new User

```
#useradd new_user
```

Output: This will create a user with name new_user. Open the file /etc/passwd. The last entry in this file will be for the recent user that you have created.

Delete an existing User

```
$userdel new_user
```

Modify User account

If you want to change the existing UID of the user

```
$usermod -u 1199 new_user
```

Output: 1199 is the new UID assigned to the user new_user.

Create a new Group

```
#groupadd new_group
```

Output: This will create a group with name new_group. Open the file /etc/group. The last entry in this file will be for the recent user that you have created.

Delete an existing Group

```
$groupdel new_group
```

Modify Group account

If you want to change the existing GID of the group

```
$groupmod -u 1199 new_group
```

Output: 1199 is the new GID assigned to the group new_group.

Exercises:

1. Create a new group GG having GID 1000.
2. Create a new user UU having UID 100 and assign it to group GG.
3. Create a new user UU having UID 101 and assign it to group GG. Use single command only
4. Assign a new password to the user UU
5. Delete all members of group GG.

Learning Outcomes (What I have Learnt):

S.No	**Parameter**	**Marks Obtained**	**Max. Marks**
1	Understanding of the student about the Procedure		20
2	Observations and analysis including learning Outcomes		20
3	Completion of experiment		10
	Signature of Faculty		

Lab 9 – Shell Script

Aim: To study how to create a shell script

Learning Objective: Student will learn how to write and run simple shell scripts

Theory: To execute a sequence of commands we can create a script and run it from the shell. To execute any script make sure that execute permission is allowed. Use chmod to allow execute permission.

Variable Declaration: All variables in shell script are considered and stored as strings, even if numeric value is passed to them.

X=12

Y=”hello”

Are valid variable declarations.

Outline of Procedure:

Create a file *script1*

$nano script1

```
X=12

Y=”hello”

echo $X

echo $Y
```

$chmod u+x script1

$./script1

Output:

12

hello

Description: echo command displays the output. $ tells echo to display the value of the variable (X/Y)

Use single-quotes to display the string as such i.e. if you want the output to be $X then write

echo ‘$X’

Reading input

Input from the user can be either taken by using *read* command or by using command line arguments. Arguments passed while running the script are called command line arguments. There are predefined environment variables which store these arguments.

$0 – The name of the script

$# - the number of arguments passed

$* - List of all parameters

$1, $2 .. – the parameters given to the script

Outline of Procedure:

Problem 1: To create a file asking the name from the user and then printing a confirmation message.

Sol: $nano script2

```
echo “Enter the name of the file”

read name

touch $name

echo “File Created Successfully”
```

$ ./script2

Problem 2: To create a file by taking file name as command line argument and then printing a confirmation message.

Sol: $nano script3

```
touch $1

echo “File Created Successfully”
```

$ ./script3 F1

Exercises:

1. Execute the following script and understand the output
   ```
   x=1
   y="Hello"
   echo "value of x is $x"
   echo "value of y is $y"
   echo "value of \$x is $x"
   echo "value of $x is $x"
   echo 'value of $x is $x'
   echo value of $x is $x
   ```
2. Write a shell script to print the total number of arguments passed and create a directory with the name given as second argument
3. Write a shell script to read one file and one directory (present in current working directory) from the user. Copy the file into the directory.
4. Write a shell script to read one file and one directory path (present in any directory) from the user. Copy the file into the directory. Show the contents of the directory before and after copying the file.
5. Write a shell script to enter a file/directory name at command line. Change its permissions to

   user - rwx
   group - rw-
   others - r--

Learning Outcomes (What I have Learnt):

S.No	Parameter	Marks Obtained	Max. Marks
1	Understanding of the student about the Procedure		20
2	Observations and analysis including learning Outcomes		20
3	Completion of experiment		10
	Signature of Faculty		

Lab 10 – Shell script (conditional constructs and loops)

Aim: To study about conditional constructs.

Learning Objective: To teach students how to make use of if, if-else, for loop and while loop.

Theory:

The syntax for if condition is

```
if [ condition ]  // note the space around the square brackets
then
        statements..
fi
```

The syntax for if -else is

```
if [ condition ]
then
        statements..
else
        statements..
fi
```

The syntax for for-loop is

```
for variable in values
do
        statements..
done
```

or

```
for ((initialization;condition;inc/dec))
do
        statements..
done
```

The syntax for while loop is

```
while [ condition ]
do
        statements..
done
```

Outline of Procedure:

Program 1: Write a shell script to check if the named passed at command line is of an existing file or not.

```
if [ -e $1 ]
then
        echo “File exists”
else
        echo “File does not exist”
```

Program 2: Write a shell script to create three files

```
for x in file1 123 file3
do
        touch $x
done
```

Program 3: Write a program to create directories with all the names passed at command line

```
for x in $*

do

        mkdir $x

done
```

Exercises:

1. Write a shell script to read a name from command line. Check whether the name is of a file or directory.
2. Write a shell script to read 3 names. Check how many names are of file and how many are of directories.
3. Write a shell script to print the total number of arguments passed at command line. If the arguments passed are > 3 then create a file with the name passed as 1st argument and a directory with the name passed as 2nd argument.
4. Write a shell script to read 5 file names from the user. Count how many names start with "f".
5. Write a shell script to read a directory path from command line. Count how many files are there in this directory.
6. Write a shell script to create a file in a directory. Ask the names of the file and directory from the user. Do not create the file if it already exists or if the directory does not exists(also display an appropriate message).
7. Write a shell script to count the number of files having read permission on them in a directory.

Learning Outcomes (What I have Learnt):

S.No	**Parameter**	**Marks Obtained**	**Max. Marks**
1	Understanding of the student about the Procedure		20
2	Observations and analysis including learning Outcomes		20
3	Completion of experiment		10
	Signature of Faculty		

<u>Lab 11 – Shell Script (Functions)</u>

Aim: To teach how to write functions in shell script

Learning Objective: Student will learn how to modularize code by using functions.

Theory:

The syntax for function is

```
        function_name ( )
        {
                Statements..
        }
```

Outline of Procedure:

Program 1: Write a shell script to show the use of functions

```
Fun1()
{
        echo “Inside function”
}
echo “Before calling function”
Fun1()
echo “After function call”
```

Output:

```
Before calling function
Inside function
After function call
```

Program 2: Write a shell script to create a function which creates a file.

```
Fun1()

{

        touch $*

        echo “Files created with first and second
argument only”

}

Fun1 $1 $2
```

Output:

```
./scriptname file1 file2 file3 file4
```

Files with name file1 and file2 are created because these are the only arguments passed while the function Fun1 is called

Exercises:

1. Write a shell script to create a function to add two numbers.

2. Write a shell script to create two functions – one to create files and another to delete files.

3. Write a shell script to create functions to
   a. Display long list of files – dir()
   b. Perform a search within a file – ser()

   If the 1st argument at command line is a directory name then a call to dir() should be made to display that directories content else if the name if of a file then call ser() to find if the pattern “help” exists in that file or not.
4. Write a shell script that has a function to copy files. The first two arguments while running the script should be names of files to be copied and the third argument should be directory where the files are to be copied.
5. Write a shell script having a function which returns some value.

Learning Outcomes (What I have Learnt):

S.No	**Parameter**	**Marks Obtained**	**Max. Marks**
1	Understanding of the student about the Procedure		20
2	Observations and analysis including learning Outcomes		20
3	Completion of experiment		10
	Signature of Faculty		

Lab 12 - RCS (Revision Control System)

Aim: To teach how to track changes made to a file with the use of Revision Control System

Learning Objective: The objective of this practical is to create different revisions of a file so that the changes made to the file at different times can be tracked.

Theory: RCS is used to create versions of a file so that you can compare your current version of the file with a previous version. Suppose you create a file and write some content into it on 13th Jan, 2018. Then on 20th of the same month you do some editing and save it. Then on 2nd Feb again you do some editing and save it. Now, can you see the original content of the file? How much content you added or deleted? The answer is "No".

But RCS is a method which helps us to see the content in each revision that we did and also the difference between any revisions.

Outline of the Procedure:

```
$ touch Test

$ rcs -i Test

$ nano Test

// write some content in the file

$ ci Test

// ci (check in) will make the first revision of the file

// Now to change the content

$ co -l Test

$ nano Test

//modify the content

$ ci Test

// This will create revision 1.2
```

In similar fashion you can create n number of revisions.

To see all the revisions associated with the file you can use **rlog** command

```
$ rlog Test
```

To see the difference between two revisions associated with the file you can use **rcsdiff** command

```
$ rcsdiff -r1.1 -r1.2 Test
```

Exercises:

1. Create a file "F1". Create 4 revisions of the file.
2. Show the summary of all the revisions of the file F1.
3. Check the difference between 2nd and 4th revision of file F1.
4. Create a file "F2". Create 2 revisions. Now create the third revision but making changes into the 1st revision.
5. Differentiate between the working of co and co -l commands.

Learning Outcomes (What I have Learnt):

S.No	Parameter	Marks Obtained	Max. Marks
1	Understanding of the student about the Procedure		20
2	Observations and analysis including learning Outcomes		20
3	Completion of experiment		10
	Signature of Faculty		

Lab 13 – Multiple Source Files

Aim: To study about the use of make command

Learning Objective: Students will learn how handle file compilation issues in large projects.

Theory:

Whenever we are working on a small project the entire code is contained in a single ".c" file. If there is any change in the code the programmer recompiles that ".c" file and reruns the program.

But if the project is very large the entire code is split into multiple files. Generally, each file contains the code for one function. Suppose the code is changed in one of the files and you do not know in which file then all the files have to be recompiled so that the project runs correctly. This process is very time consuming as it requires compiling those files also in which there was no change of code. Also, if a single file in which the code was changed is not recompiled then the project will not work correctly.

Outline of the Procedure:

Let us consider an example to understand the situation.

Problem: Design a calculator which can perform addition and subtraction.

Solution: Create three files

cal.c – the main file which takes the input and calls the addition and subtraction functions.

add.c – contains the function add() to perform the addition.

sub.c – contains the function sub() to perform the subtraction.

cal.c

```
#include<stdio.h>

extern void add(int x, int y);

extern void sub(int x, int y);

int main()

{

        int x=5, y=3;

        add(x,y);

        sub(x,y);

        printf("Executed the
cal.c file\n");

}
```

add.c

```
#include<stdio.h>

void add(int a, int b)

{

        int sum;

        sum=a+b;

        printf("Sum is %d
\n", sum);

}
```

sub.c

```
#include<stdio.h>

void sub(int a, int b)

{

        int sub;

        sub=a-b;

        printf("Subtracted
value is %d\n",sub);

}
```

To run compile all the three files individually

```
$ gcc -c cal.c

$ gcc -c add.c

$ gcc -c sub.c
```

Note: -c options generates a “.o” file from a “.c” file i.e. cal.c gets compiled into cal.o file

```
$ gcc -o calculator cal.o add.o sub.o

$ ./calculator
```

Output:

```
Sum is 8

Subtracted value is 2

Executed the cal.c file
```

Now suppose the programmer who manages the sub.c files makes some changes in the “printf” line. (Make some changes at your own). The manager of the project is aware that some changes have been done but don’t know to which file. So he has to recompile all the files.

```
 $ gcc -c cal.c

$ gcc -c add.c
```

(Note: add.c is recompiled which had no change made to it and sub.c is missed by the project manager by mistake)

```
$ gcc -o calculator cal.o add.o sub.o

$ ./calculator
```

Output:

```
Sum is 8

Subtracted value is 2

Executed the cal.c file
```

The output remains the same and the manager has no idea what went wrong and why no changes have been reflected in the output.

Makefile and the make command

This is where the makefile and the make command comes in handy. The make command keeps track of the changes made in any of the files and recompiles only the desired file. To tell make how the project is managed you are required to create a **makefile**.

The makefile has two important parts: dependencies and rules. Dependencies tell how every file of the project is related to the source files. Rules tell how the target file is created.

For clarity lets us create a makefile for our project

```
$nano makefile1
```

(Note: makefile1 is the name of the **makefile**)

The content of makefile1 are as below

```
calculator:	cal.o	add.o	sub.o
	gcc	-o	calculator	cal.o	add.o	sub.o
cal.o:	cal.c
	gcc	-c	cal.c
add.o:	add.c
	gcc	-c	add.c
sub.o:	sub.c
	gcc	-c	sub.c
```

All the spaces in the file are "tab". The makefile tell make that calculator is dependent on cal.o, add.o and sub.o. The rule says that whenever there is change in any of these execute

```
gcc -o calculator cal.o add.o sub.o
```

Similarly, dependencies and rules are defined for rest of the files.

To invoke the make command write

```
$ make -f makefile1
$ ./calculator
```

This time do some changes in the add.c file

```
$ touch add.c
$ make -f makefile
```

Output:

```
gcc -c add.c
gcc -o calculator cal.o add.o sub.o
```

Automatically only the affected files are recompiled.

```
$./calculator
```

Exercises:

Q1. To the above project add the following dependencies:

add.c is dependent on a user defined header file “a.h”

sub.c is dependent on a user defined header file “s.h”

Q2. Create a project having the following dependencies.

$$\begin{tikzcd} & \text{Main.c} \arrow[dl] \arrow[d] \arrow[dr] & \\ \text{A.c} \arrow[d] & \text{B.c} & \text{C.c} \\ \text{D.c} & & \end{tikzcd}$$

Add relevant code to each file.

Learning Outcomes (What I have Learnt):

S.No	Parameter	Marks Obtained	Max. Marks
1	Understanding of the student about the Procedure		20
2	Observations and analysis including learning Outcomes		20
3	Completion of experiment		10
	Signature of Faculty		

Lab 14 – Manual Page

Aim: To study creation of manual pages

Learning Objective: To teach how to create a manual page for a new command created by the user.

Theory: An additional task while creating a new command is writing a manual page for that command. The layout of any manual page follows somewhat same pattern, which is:

- Header
- Name
- Synopsis
- Description
- Examples
- Authors
- Reporting bugs
- Copyright
- See Also

Manual pages in Linux are formatted using *groff.*

Outline of the Procedure:

Create a new file *mycommand.txt* and write the below mentioned content in the file

```
.TH NEWCOMMAND

.SH NAME

MyCommand \− the purpose of the command goes here.

.SH SYNOPSIS

.B mycommand

[\−option ...]

.SH DESCRIPTION

.PP

The detailed description about the command

.PP

It was written for demonstration purposes.

.SH OPTIONS

.PP

It doesn't have any, but let's pretend, to make this template complete:

.TP
```

```
.BI \−option

If there was an option, it would not be −option.

.SH COPYRIGHT

This program is free software and is under the terms of the GNU General Public License as published by the Free Software Foundation

.SH AUTHORS

Baljit Singh Saini
```

Save the file and exit.

The manual page is processed using *groff* command.

```
$groff -Tascii -man mycommand.txt
```

Exercises:

Q1. Create a manual page for *cat* command.

Q2. Create a manual page for any command that you have created at your own.

Learning Outcomes (What I have Learnt):

S.No	Parameter	Marks Obtained	Max. Marks
1	Understanding of the student about the Procedure		20
2	Observations and analysis including learning Outcomes		20
3	Completion of experiment		10
	Signature of Faculty		