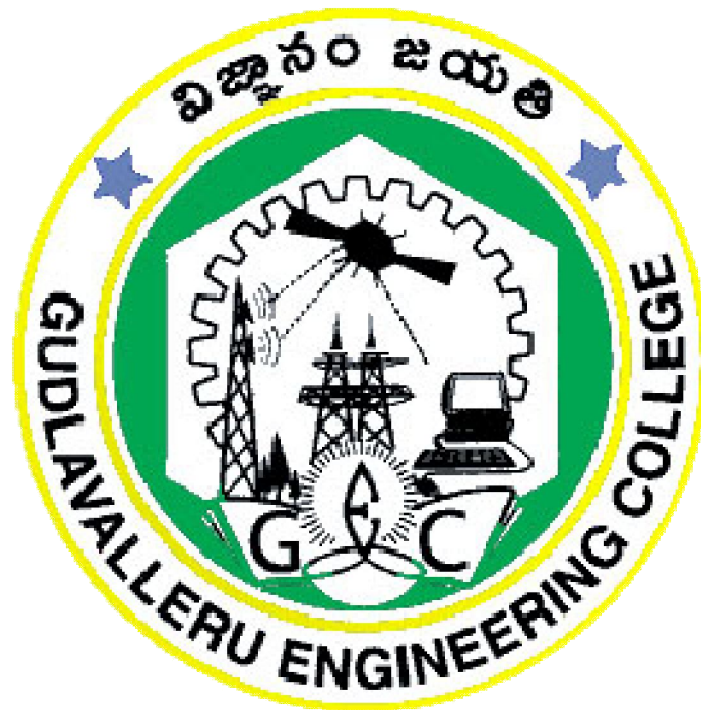


UNIX PROGRAMMING LAB
FACULTY MANUAL
II Year I Semester



Prepared by

Siva Naga Prasad Mannem
Assistant Professor

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
GUDLAVALLERU ENGINEERING COLLEGE

(An Autonomous Institute with Permanent Affiliation to JNTUK, Kakinada)

Seshadri rao Knowledge Village, Gudlavalleru – 521356

GUDLAVALLERU ENGINEERING COLLEGE

(An Autonomous Institution with Permanent Affiliation to JNTUK, Kakinada)

Seshadri Rao Knowledge Village, Gudlavalleru – 521356

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

INSTITUTE VISION & MISSION

INSTITUTE VISION:

To be a leading institution of engineering education and research, preparing students for leadership in their fields in a caring and challenging learning environment.

INSTITUTE MISSION:

- To produce quality engineers by providing state-of-the-art engineering education.
- To attract and retain knowledgeable, creative, motivated and highly skilled individuals whose leadership and contributions uphold the college tenets of education, creativity, research and responsible public service.
- To develop faculty and resources to impart and disseminate knowledge and information to students and also to society that will enhance educational level, which in turn, will contribute to social and economic betterment of society.
- To provide an environment that values and encourages knowledge acquisition and academic freedom, making this a preferred institution for knowledge seekers.
- To provide quality assurance.
- To partner and collaborate with industry, government, and R&D institutes to develop new knowledge and sustainable technologies and serve as an engine for facilitating the nation's economic development.
- To impart personality development skills to students that will help them to succeed and lead.
- To instil in students the attitude, values and vision that will prepare them to lead lives of personal integrity and civic responsibility.
- To promote a campus environment that welcomes and makes students of all races, cultures and civilizations feel at home.
- Putting students face to face with industrial, governmental and societal challenges.

DEPARTMENT VISION & MISSION

VISION

To be a Centre of Excellence in Computer Science and Engineering education and training to meet the challenging needs of the industry and society.

MISSION

- To impart quality education through well-designed curriculum in tune with the growing software needs of the industry.
- To serve our students by inculcating in them problem solving, leadership, teamwork skills and the value of commitment to quality, ethical behavior & respect for others.
- To foster industry-academia relationship for mutual benefit and growth.

PROGRAMME EDUCATIONAL OBJECTIVES (PEOs):-

PEO1: Identify, analyze, formulate and solve Computer Science and Engineering problems both independently and in a team environment by using the appropriate modern tools.

PEO2: Manage software projects with significant technical, legal, ethical, social, environmental and economic considerations

PEO3: Demonstrate commitment and progress in lifelong learning, professional development, leadership and communicate effectively with professional clients and the public.

PROGRAM OUTCOMES (POs)

Engineering students will be able to:

1. **Engineering knowledge:** Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems.
2. **Problem analysis:** Identify, formulate, review research literature, and analyze complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences.
3. **Design/development of solutions:** Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety, and the cultural, societal, and environmental considerations.
4. **Conduct investigations of complex problems:** Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions.
5. **Modern tool usage:** Create, select, and apply appropriate techniques resources, and modern engineering and IT tools including prediction and modeling to complex engineering activities with an understanding of the limitations.
6. **The engineer and society:** Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the professional engineering practice.
7. **Environment and sustainability:** Understand the impact of the professional engineering solutions in societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development.
8. **Ethics:** Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.
9. **Individual and team work:** Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.
10. **Communication:** Communicate effectively on complex engineering activities with the engineering community and with society at large, such as, being able to comprehend and write effective reports and design documentation, make effective presentations, and give and receive clear instructions.
11. **Project management and finance:** Demonstrate knowledge and understanding of the engineering and management principles and apply these to one's own work, as a

member and leader in a team, to manage projects and in multidisciplinary environments.

12. **Life-long learning:** Recognize the need for, and have the preparation and ability to engage in independent and life-long learning in the broadest context of technological change.

PROGRAM SPECIFIC OUTCOMES

Students will be able to

PSO1: Design, develop, test and maintain reliable software systems and intelligent systems.

PSO2: Design and develop web sites, web apps and mobile apps.

Course Objectives:

- To familiarize with various UNIX utilities.
- To impart knowledge on developing shell scripts.

Course Outcomes:

Upon successful completion of the course, the students will be able to

- develop shell scripts in order to perform shell programming.
- demonstrate the UNIX file system.

Mapping Of Course Outcomes With Program Outcomes

Unix Programming Lab	1	2	3	4	5	6	7	8	9	10	11	12	PSO1	PSO2
CO1: develop shell scripts in order to perform shell programming	1	1	2		1				1	1				
CO2: demonstrate the UNIX file system	1													
Unix Programming Lab	1	1	1		1				1	1				

LIST OF EXPERIMENTS

S. No	Program Name	Mapping Of Co's	Page No
1.	Practice session on UNIX commands and vi editor	CO1	7
2.	Write a shell script to print the factorial of first n natural numbers.	CO1	22
3.	Write a shell script to generate a multiplication table of the given number.	CO1	23
4.	Write a shell script to list the files in the current directory to which the user has read, write and execute permissions.	CO1	24
5.	Write a shell script to compare two strings by reading strings from the command line.	CO1	25
6.	Write shell script to read username & to find whether the user currently logged in or not.	CO1	26
7.	Write shell scripts to find the length of a given string and to extract a substring from a given string.	CO1	27
8.	Write a shell script that counts the number of lines and words present in a given file.	CO1	28
9.	Write a shell script that displays the list of all files in the given directory.	CO1	29
10.	Write a shell scripts that copies multiple files to a directory.	CO1	30
11.	Write a shell script (small calculator) that adds, subtracts, multiplies and divides the given two integers. There are two division options: one returns the quotient and the other returns remainder. The script requires 3 arguments: The operation to be used and two integer numbers. The options are add (-a), subtract (-s), multiply (-m), quotient (-c) and remainder (-r).	CO1	31
12.	Write a C program that illustrates uses of the opendir, readdir, and closedir APIs.	CO2	32
13	Write a program that takes one or more file/directory names as command line input and reports the following information on the file: <ul style="list-style-type: none"> • File type. • Time of last access. • Number of links. • Read, Write and Execute permissions. 	CO2	34
14	Write a C program that illustrates the creation of child process using fork system call.	CO2	36

Aim:-

1.Practice session on UNIX Commands and vi editor.

Login Unix

When you first connect to a Unix system, you usually see a prompt such as the following:

```
login:
```

To log in

- Have your userid (user identification) and password ready. Contact your system administrator if you don't have these yet.
- Type your userid at the login prompt, then press ENTER. Your userid is case- sensitive, so be sure you type it exactly as your system administrator has instructed.
- Type your password at the password prompt, then press ENTER. Your password is also case-sensitive.

If you provide the correct userid and password, then you will be allowed to enter into the system. Read the information and messages that comes up on the screen, which is as follows.

```
login : amrood
```

```
amrood' s
```

```
password:
```

```
Last login: Sun Jun 14 00:22:22 2009 from 129.11.164.72
```

You will be provided with a command prompt (sometime called the \$ prompt) where you type all your commands. For example, to check calendar, you need to type the cal command as follows

```
$ cal
```

```

    June 2009
Su M Tu We Th Fr Sa
 0
 1  2  3  4  5  6
 7  8  9 10 11 12 13
14 15 16 17 18 19 20
21 22 23 24 25 26 27
```

Change Password

All Unix systems require passwords to help ensure that your files and data remain your own and that the system itself is secure from hackers and crackers. Following are the steps to change your password –

Step 1: To start, type password at the command prompt as shown below.

Step 2: Enter your old password, the one you're currently using.

Step 3: Type in your new password. Always keep your password complex enough so that nobody can guess it. But make sure, you remember it.

Step 4: You must verify the password by typing it again.

```
$ passwd
Changing password
for amrood
(current) Unix
password: *****
New Unix
```

Note – We have added asterisk (*) here just to show the location where you need to enter the current and new passwords otherwise at your system. It does not show you any character when you type.

Listing Directories and Files

All data in Unix is organized into files. All files are organized into directories. These directories are organized into a tree-like structure called the filesystem. You can use the **ls** command to list out all the files or directories available in a directory. Following is the example of using **ls** command with **-l** option.

```
$ ls -l
total 19621
drwxrwxr-x  2 amrood amrood
                                4096 Dec 25 09:59 uml
```



```

1 Amrood amrood      5341 Dec 25 08:38 uml.jpg

-rw-rw-r--
drwxr-xr-x  2 Amrood amrood      4096 Feb 15  2006 Univ

drwxr-xr-x  2 Root   root        4096 Dec  9  2007 Urlspedia
-rw-r--r--  1 Root   root       276480 Dec  9  2007 urlspedia.tar
drwxr-xr-x  8 Root   root        4096 Nov 25  2007 Usr

-rwxr-xr-x  1 Root   root        3192 Nov 25  2007 webthumb.php
-rw-rw-r--  1 amrood amrood     20480 Nov 25  2007 webthumb.tar

-rw-rw-r--  1 amrood amrood      5654 Aug  9  2007 yourfile.mid
-rw-rw-r--  1 amrood amrood    166255 Aug  9  2007 yourfile.swf

```

␣

Here entries starting with **d...** represent directories. For example, uml, univ and urlspedia are directories and rest of the entries are files.

Who Are You?

While you're logged into the system, you might be willing to know : Who am I? The easiest way to find out "who you are" is to enter the *whoami* command.

```

$
whoam
i

```

Try it on your system. This command lists the account name associated with the current login. You can try **who am i** command as well to get information about yourself.

Who is Logged in?

Sometime you might be interested to know who is logged in to the computer at the same time.

There are three commands available to get you this information, based on how much you wish to know about the other users: **users**, **who**, and **w**.

```

$ users
amrood bablu qadi r

```

```
$ who
amrood tty0 Oct 8 14:10 (limbo)
bablu tty2 Oct 4 09:08 (calliope)
qadir tty4 Oct 8 12:09 (dent)
```

Try the **w** command on your system to check the output. This lists down information associated with the users logged in the system.

Logging Out

When you finish your session, you need to log out of the system. This is to ensure that nobody else accesses your files.

To log out

Just type the logout command at the command prompt, and the system will clean up everything and break the connection.

System Shutdown

The most consistent way to shut down a Unix system properly via the command line is to use one of the following commands –

Command	Description
halt	Brings the system down immediately
init 0	Powers off the system using predefined scripts to synchronize and clean up the system prior to shutting down
init 6	Reboots the system by shutting it down completely and then restarting it
poweroff	Shuts down the system by powering off

reboot	Reboots the system
Shutdown	Shuts down the system

File management:

All data in Unix is organized into files. All files are organized into directories. These directories are organized into a tree-like structure called the file system.

When you work with Unix, one way or another, you spend most of your time working with files. This tutorial will help you understand how to create and remove files, copy and rename them, create links to them, etc.

In Unix, there are three basic types of files –

Ordinary Files – An ordinary file is a file on the system that contains data, text, or program instructions. In this tutorial, you look at working with ordinary files.

Directories – Directories store both special and ordinary files. For users familiar with Windows or Mac OS, Unix directories are equivalent to folders.

Special Files – Some special files provide access to hardware such as hard drives, CD-ROM drives, modems, and Ethernet adapters. Other special files are similar to aliases or shortcuts and enable you to access a single file using different names.

Listing Files:

To list the files and directories stored in the current directory, use the following command:

```
$! s
bin      Hosts  lib    res.03
ch07     hw1   pub    test_results
ch07.bak hw2   res.01 users
docs     hw3   res.02 work
```

The command **ls** supports the **-l** option which would help you to get more information about the listed files

```
$ls -l
```

```
total drwxrwxr-x  2 amrood amrood    4096 Dec  25  09:59 Uml
      -rw-rw-r--  1 amrood amrood    5341 Dec  25  08:38 uml.jpg
      drwxr-xr-x  2 amrood amrood    4096 Feb  15  2006 Univ
      drwxr-xr-x  2 Root   root      4096 Dec   9  2007 Urlspedia
      -rw-r--r--  1 Root   root     276480 Dec   9  2007 urlspedia.tar
      drwxr-xr-x  8 Root   root      4096 Nov 25  2007 Usr
      drwxr-xr-x  2    200   300     4096 Nov 25  2007 webthumb-1.01
      -rwxr-xr-x  1 Root   root      3192 Nov 25  2007 webthumb.php
      -rw-rw-r--  1 amrood amrood   20480 Nov 25  2007 webthumb.tar
      -rw-rw-r--  1 amrood amrood    5654 Aug   9  2007 yourfile.mid
      -rw-rw-r--  1 amrood amrood  166255 Aug   9  2007 yourfile.swf
      drwxr-xr-x 11 amrood amrood    4096 May 29  2007 zlib-1.2.3
      $
```

Here is the information about all the listed columns –

First Column: Represents the file type and the permission given on the file. Below is the description of all type of files.

Second Column: Represents the number of memory blocks taken by the file or directory.

Third Column: Represents the owner of the file. This is the Unix user who created this file.

Fourth Column: Represents the group of the owner. Every Unix user will have an associated group.

Fifth Column: Represents the file size in bytes.

Sixth Column: Represents the date and the time when this file was created or modified for the last time.

Seventh Column: Represents the file or the directory name.

In the **ls -l** listing example, every file line begins with a **d**, **-**, or **l**.

Prefix	Description
-	Regular file, such as an ASCII text file, binary executable, or hard link
b	Block special file. Block input/output device file such as a physical hard drive
c	Character special file. Raw input/output device file such as a physical hard drive
d	Directory file that contains a listing of other files and directories
l	Symbolic link file. Links on any regular file
p	Named pipe. A mechanism for interprocess communications
s	Socket used for interprocess communication

Metacharacters

Metacharacters have a special meaning in Unix. For example, * and ? are metacharacters.

We use * to match 0 or more characters, a question mark (?) matches with a single character. For Example

```
$! s ch*.doc
```

Displays all the files, the names of which start with **ch** and end with **.doc**

Here, * works as meta character which matches with any character. If you want to display all the files ending with just **.doc**, then you can use the following

```
$! s *.doc
```

```
ch01-1.doc  ch010.doc  ch02.doc  ch03-2.doc
ch04-1.doc  ch040.doc  ch05.doc  ch06-2.doc
ch01-2.doc  ch02-1.doc  c
```

command –

To list the invisible files, specify the **-a** option to **ls** –

```
$ ls -a
.          .profile      docs      lib        test_results
..         .rhosts       hosts     pub        users
emacs     Bin           hw1       res.01     work
exrc      ch07          hw2       res.02
kshrc     ch07.bak      hw3       res.03
$
```

Single dot (.) – This represents the current directory.

Double dot (..) – This represents the parent directory.

Creating Files

You can use the vi editor to create ordinary files on any Unix system. You simply need to give the following command –

```
$ vi filename
```

The above command will open a file with the given filename. Now, press the key **i** to come into the edit mode. Once you are in the edit mode, you can start writing your content in the file as in the following program –

Once you are done with the program, follow these steps –

```
This is unix file....I created it for the first time.....
```

- Press the key **esc** to come out of the edit mode.
 - Press two keys **Shift + Z** together to come out of the file completely.
- You will now have a file created with filename in the current directory.

Editing Files

You can edit an existing file using the vi editor. We will discuss in short how to open an existing file –

Once the file is opened, you can come in the edit mode by pressing the key **i** and

```
$ vi filename
```

then you can proceed by editing the file. If you want to move here and there inside a file, then first you need to come out of the edit mode by pressing the key **Esc**. After this, you can use the following keys to move inside a file –

- **l** key to move to the right side.

- **h** key to move to the left side.
- **k** key to move upside in the file.
- **j** key to move downside in the file.

So using the above keys, you can position your cursor wherever you want to edit. Once you are positioned, then you can use the **i** key to come in the edit mode. Once you are done with the editing in your file, press **Esc** and finally two keys **Shift + ZZ** together to come out of the file completely.

Display Content of a File:

You can use the **cat** command to see the content of a file. Following is a simple

```
$ cat filename
This is unix file....I created it for the first time.....
```

example to see the content of the above created file.

You can display the line numbers by using the **-b** option along with the **cat** command as follows

```
$ cat -b filename
1 This is unix file....I created it for the first time.....
2 I'm going to save this content in this file.
$
```

Counting Words in a File:

You can use the **wc** command to get a count of the total number of lines, words, and characters contained in a file. Following is a simple example to see the information about the file created above

```
$ wc filename
2 19 103 filename
$
```

Here is the detail of all the four columns –

First Column: Represents the total number of lines in the file.

Second Column: Represents the total number of words in the file.

Third Column: Represents the total number of bytes in the file. This is the actual size of the file.

Fourth Column: Represents the file name.

You can give multiple files and get information about those files at a time.

```
$ wc filename1 filename2 filename3
```

Following is simple syntax

If we want to get number of **lines** in a file we can use **-l** option for wc.

\$ wc -l filename

If we want to get number of **words** in a file we can use **-w** option for wc.

\$ wc -w filename

If we want to get number of **characters** in a file we can use **-c** option for wc.

\$ wc -c filename

Copying Files:

To make a copy of a file use the **cp** command. The basic syntax of the command is

\$ cp source_file destination_file

Following is the example to create a copy of the existing file **filename**.

\$ cp filename copyfile

You will now find one more file **copyfile** in your current directory. This file will exactly be the same as the original file **filename**

Renaming Files:

To change the name of a file, use the **mv** command. Following is the basic syntax

\$ mv old_file new_file

The following program will rename the existing file **filename** to **newfile**.

\$ mv filename newfile

The **mv** command will move the existing file completely into the new file. In this case, you will find only **newfile** in your current directory.

Deleting Files:

To delete an existing file, use the **rm** command. Following is the basic syntax

\$ rm filename

Caution: A file may contain useful information. It is always recommended to be careful while using this **Delete** command. It is better to use the **-i** option along with **rm** command.

Following is the example which shows how to completely remove the existing file filename.

\$ rm filename

You can remove multiple files at a time with the command given below

\$ rm filename1 filename2 filename3

Listing Directories

To list the files in a directory, you can use the following syntax –

\$ ls dirname

Following is the example to list all the files contained in /usr/local directory

\$ls /usr/local

X11	bin	gimp	jikes	sbin
ace	doc	include	lib	share
atalk	etc	info	man	ami

Creating Directories

We will now understand how to create directories. Directories are created by the following command –

\$mkdir dirname

Here, directory is the absolute or relative pathname of the directory you want to create. For example, the command –

\$mkdir mydir

Creates the directory **mydir** in the current directory. Here is another example –

\$mkdir /tmp/test-dir

This command creates the directory **test-dir** in the **/tmp** directory. The **mkdir** command produces no output if it successfully creates the requested directory. If you give more than one directory on the command line, **mkdir** creates each of the directories. For example, –

\$mkdir docs pub

Creates the directories **docs** and **pub** under the current directory

Removing Directories

Directories can be deleted using the **rmdir** command as follows –

\$rmdir dirname

Note – To remove a directory, make sure it is empty which means there should not be any file or sub-directory inside this directory.

You can remove multiple directories at a time as follows –

\$rmdir dirname1 dirname2 dirname3

The above command removes the directories **dirname1**, **dirname2**, and **dirname3**, if they are empty. The **rmdir** command produces no output if it is successful.

Changing Directories

You can use the **cd** command to do more than just change to a home directory. You can use it to change to any directory by specifying a valid absolute or relative path. The syntax is as given below –

\$cd dirname

Here, **dirname** is the name of the directory that you want to change to. For example, the command –

\$cd /usr/local/bin

Changes to the directory **/usr/local/bin**. From this directory, you can **cd** to the directory **/usr/home/amrood** using the following relative path

\$cd ../../home/amrood

Renaming Directories

The **mv** (move) command can also be used to rename a directory. The syntax is as follows:

\$mv olddir newdir

You can rename a directory **mydir** to **yourdir** as follows –

\$mv mydir yourdir

File permissions and access modes:

In this we will discuss in detail about file permission and access modes in Unix. File ownership is an important component of Unix that provides a secure method for storing files. Every file in Unix has the following attributes –

- **Owner permissions** – The owner's permissions determine what actions the owner of the file can perform on the file.
- **Group permissions** – The group's permissions determine what actions a user, who is a member of the group that a file belongs to, can perform on the file.
- **Other (world) permissions** – The permissions for others indicate what action all other users can perform on the file.

The Permission Indicators

While using **ls -l** command, it displays various information related to file permission as follows–

Here, the first column represents different access modes, i.e., the permission

```
$ls -l /home/amrood
-rwxr-xr--      1 amrood  users 1024 Nov 2 00:10  myfile
drwxr-xr---    1 amrood          users 1024 Nov 2 00:10  mydir
```

associated with a file or a directory.

The permissions are broken into groups of threes, and each position in the group denotes a specific permission, in this order: read (**r**), write (**w**), execute (**x**)

- The first three characters (2-4) represent the permissions for the file's owner.

For example, `-rwxr-xr--` represents that the owner has read (r), write (w) and execute(x) permission.

- The second group of three characters (5-7) consists of the permissions for the group to which the file belongs. For example, `-rwxr-xr--` represents that the group has read(r) and execute (x) permission, but no write permission.
- The last group of three characters (8-10) represents the permissions for everyone else. For example, `-rwxr-xr--` represents that there is read (r) only permission.

File Access Modes

The permissions of a file are the first line of defense in the security of a Unix system. The basic building blocks of Unix permissions are the read, write, and execute permissions, which have been described below –

Read

Grants the capability to read, i.e., view the contents of the file.

Write

Grants the capability to modify, or remove the content of the file.

Execute

User with execute permissions can run a file as a program.

Directory Access Modes

Directory access modes are listed and organized in the same manner as any other file. There are a few differences that need to be mentioned:

Read

Access to a directory means that the user can read the contents. The user can look at the filenames inside the directory.

Write

Access means that the user can add or delete files from the directory.

Execute

Executing a directory doesn't really make sense, so think of this as a traverse permission.

A user must have execute access to the `bin` directory in order to execute the `ls` or the `cd` command.

Changing Permissions

To change the file or the directory permissions, you use the **chmod** (change mode) command. There are two ways to use `chmod` — the symbolic mode and the absolute mode.

Using chmod in Symbolic Mode

The easiest way for a beginner to modify file or directory permissions is to use the symbolic mode. With symbolic permissions you can add, delete, or specify the permission set you want by using the operators in the following table.

chmod Operator	Description
+	Adds the designated permission(s) to a file or directory.
-	Removes the designated permission(s) from a file or directory.
=	Sets the designated permission(s).

Here's an example using testfile. Running **ls -l** on the testfile shows that the file's permissions are as follows –

Then each example chmod command from the preceding table is run on the

```
$ls -l testfile
-rwxrwxr--      1 amrood  users 1024 Nov 2 00:10  testfile
```

```
$chmod o+wx testfile
$ls -l testfile
-rwxrwxrwx      1 amrood  users 1024 Nov 2 00:10  testfile
$chmod u-x testfile
$ls -l testfile
-rw-rwxrwx      1 amrood  users 1024 Nov 2 00:10  testfile
$chmod g=rx testfile
$ls -l testfile
```

testfile, followed by **ls -l**, so you can see the permission changes

Using chmod with Absolute Permissions

The second way to modify permissions with the **chmod** command is to use a number to specify each set of permissions for the file.

Each permission is assigned a value, as the following table shows, and the total of each set of permissions provides a number for that set

Number	Octal Permission Representation	Ref
--------	---------------------------------	-----

0	No permission	---
1	Execute permission	--x
2	Write permission	-w-
3	Execute and write permission: 1 (execute) + 2 (write) = 3	-wx
4	Read permission	r--
5	Read and execute permission: 4 (read) + 1 (execute) = 5	r-x
6	Read and write permission: 4 (read) + 2 (write) = 6	rw-
7	All permissions: 4 (read) + 2 (write) + 1 (execute) = 7	rwx

Here's an example using the testfile. Running `ls -l` on the testfile shows that the file's permissions are as follows –

```
$ls -l testfile
-rwxrwxr--      1 amrood users 1024 Nov 2 00:10 testfile
```

Then each example `chmod` command from the preceding table is run on the testfile, followed by `ls -l`, so you can see the permission changes –

```
$ chmod 755 testfile
$ls -l testfile
-rwxr-xr-x      1 amrood users 1024 Nov 2 00:10 testfile
$chmod 743 testfile
```

```
$ls -l testfile
-rwxr---wx      1 amrood users 1024 Nov 2 00:10 testfile
$chmod 043 testfile
$ls -l testfile
----r---wx      1 amrood users 1024 Nov 2 00:10 testfile
```

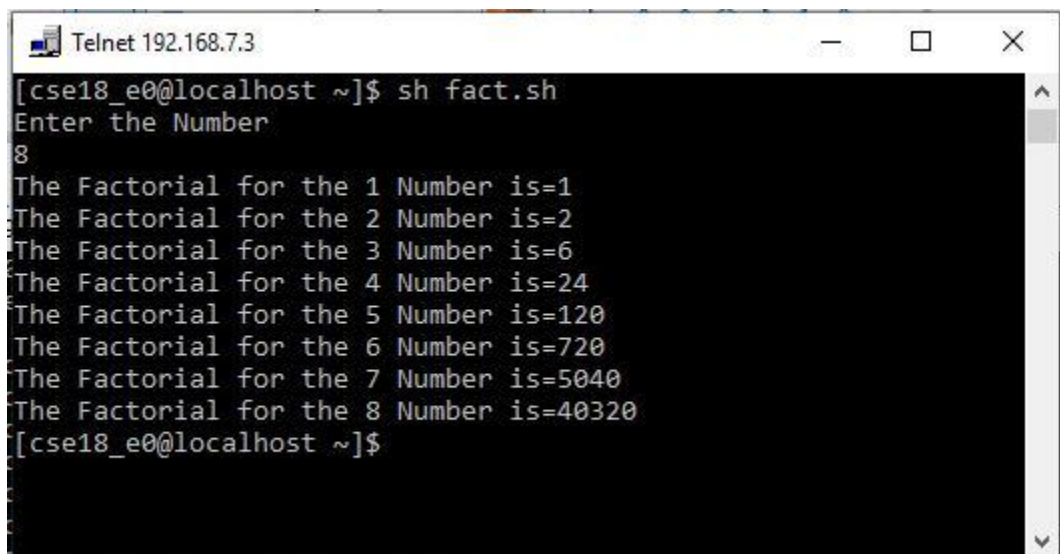
Aim:-

2. Write a shell script to print the factorial of first n natural numbers.

Source code:-

```
echo "Enter the Number"
read num
i=1
j=1
res=1
if test $j -eq 1
then
echo "The Factorial for the $j Number is="$j
j=`expr $j + 1 `
fi
while [ $j -le $num ]
do
while [ $i -le $j ]
do
res=`expr $res \* $i `
i=`expr $i + 1 `
done
echo "The Factorial for the $j Number is="$res
res=1
i=1
j=`expr $j + 1 `
done
```

Output:-



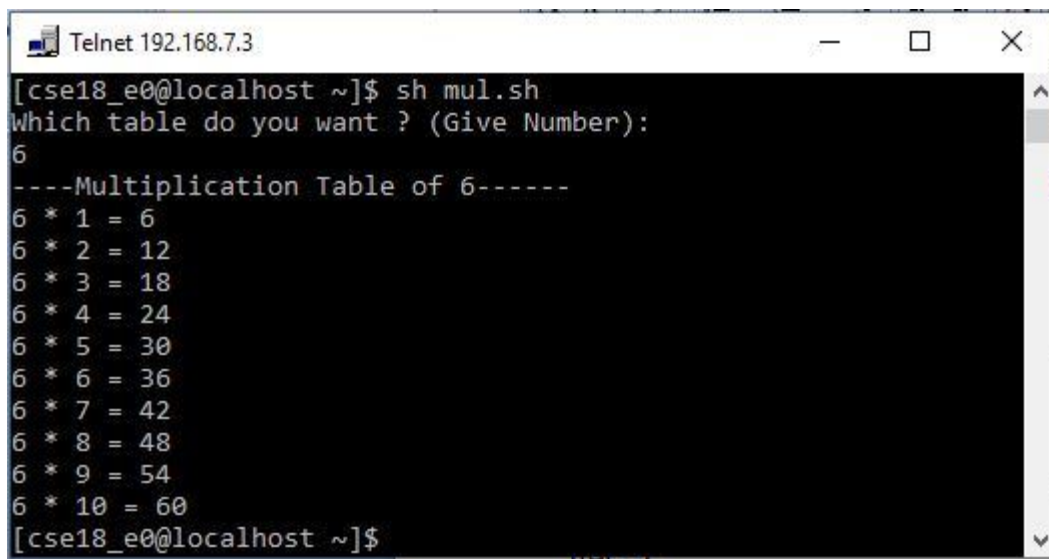
```
Telnet 192.168.7.3
[cse18_e0@localhost ~]$ sh fact.sh
Enter the Number
8
The Factorial for the 1 Number is=1
The Factorial for the 2 Number is=2
The Factorial for the 3 Number is=6
The Factorial for the 4 Number is=24
The Factorial for the 5 Number is=120
The Factorial for the 6 Number is=720
The Factorial for the 7 Number is=5040
The Factorial for the 8 Number is=40320
[cse18_e0@localhost ~]$
```

3. Write a shell script to generate a multiplication table of the given number.

Source code:-

```
echo "Which table do you want ? (Give Number):"  
  
read num  
  
echo "----Multiplication Table of $num-----"  
  
iter=1  
  
while [ $iter -le 10 ]  
  
do  
  
    res=`expr $num \* $iter`  
    echo "$num * $iter = $res"  
    iter=`expr $iter + 1`  
  
done
```

Output:



```
Telnet 192.168.7.3  
[cse18_e0@localhost ~]$ sh mul.sh  
Which table do you want ? (Give Number):  
6  
----Multiplication Table of 6-----  
6 * 1 = 6  
6 * 2 = 12  
6 * 3 = 18  
6 * 4 = 24  
6 * 5 = 30  
6 * 6 = 36  
6 * 7 = 42  
6 * 8 = 48  
6 * 9 = 54  
6 * 10 = 60  
[cse18_e0@localhost ~]$
```

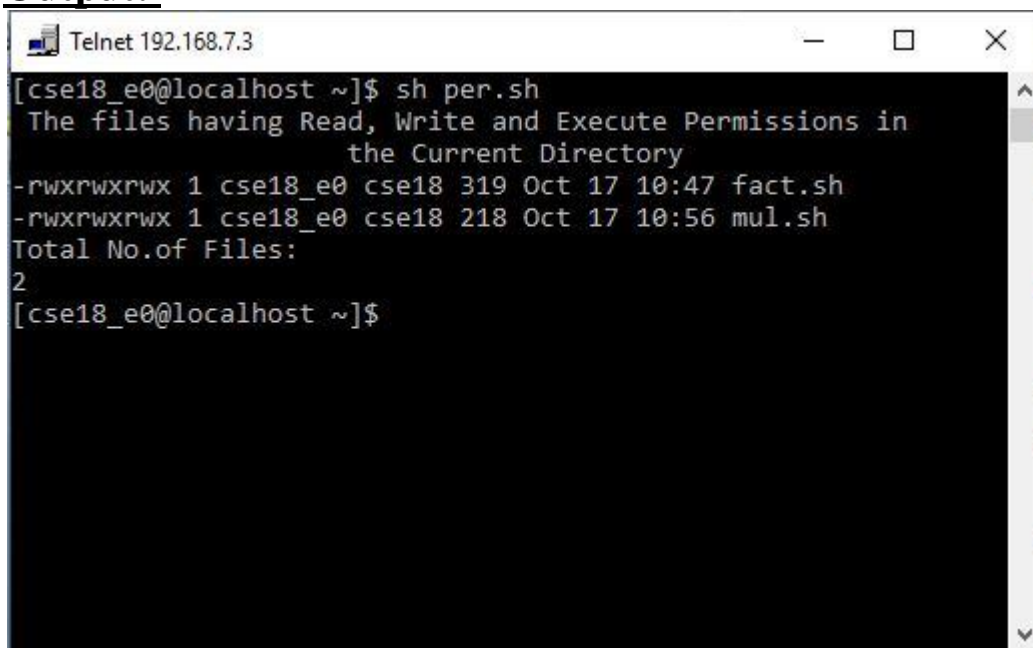
4. Write a shell script to list the files in the current directory to which the user has read, write and execute permissions.

Source code:-

```
echo " The files having Read, Write and Execute Permissions in
      the Current Directory"
ls -l | grep "^.rwx"
echo "Total No.of Files:"
ls -l | grep -c "^.rwx"
```

Note: grep -c displays the count of number of lines.

Output:



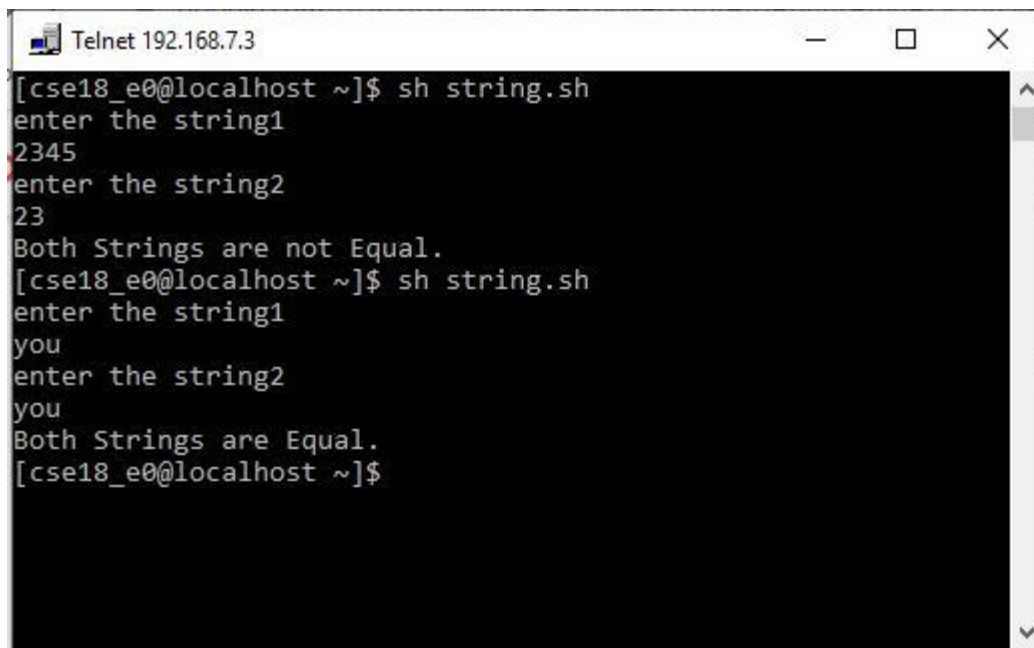
```
Telnet 192.168.7.3
[cse18_e0@localhost ~]$ sh per.sh
The files having Read, Write and Execute Permissions in
      the Current Directory
-rwxrwxrwx 1 cse18_e0 cse18 319 Oct 17 10:47 fact.sh
-rwxrwxrwx 1 cse18_e0 cse18 218 Oct 17 10:56 mul.sh
Total No.of Files:
2
[cse18_e0@localhost ~]$
```


5. Write a shell script to compare two strings by reading strings from the Command line.

Source code:-

```
echo enter the string1
readstr1
echo enter the string2
read str2
if [ "$str1" == "$str2" ]; then
    echo "Both Strings are Equal."
else
    echo "Both Strings are not Equal."
fi
```

Output:-



```
Telnet 192.168.7.3
[cse18_e0@localhost ~]$ sh string.sh
enter the string1
2345
enter the string2
23
Both Strings are not Equal.
[cse18_e0@localhost ~]$ sh string.sh
enter the string1
you
enter the string2
you
Both Strings are Equal.
[cse18_e0@localhost ~]$
```

6. Write shell script to read username & to find whether the user currently logged in or not.

Source code:-

```
echo "the user ---$1--- is logged on to the sytem on"  
who|grep "$1"| cut -c22-38
```

Output:

Telnet 192.168.7.3

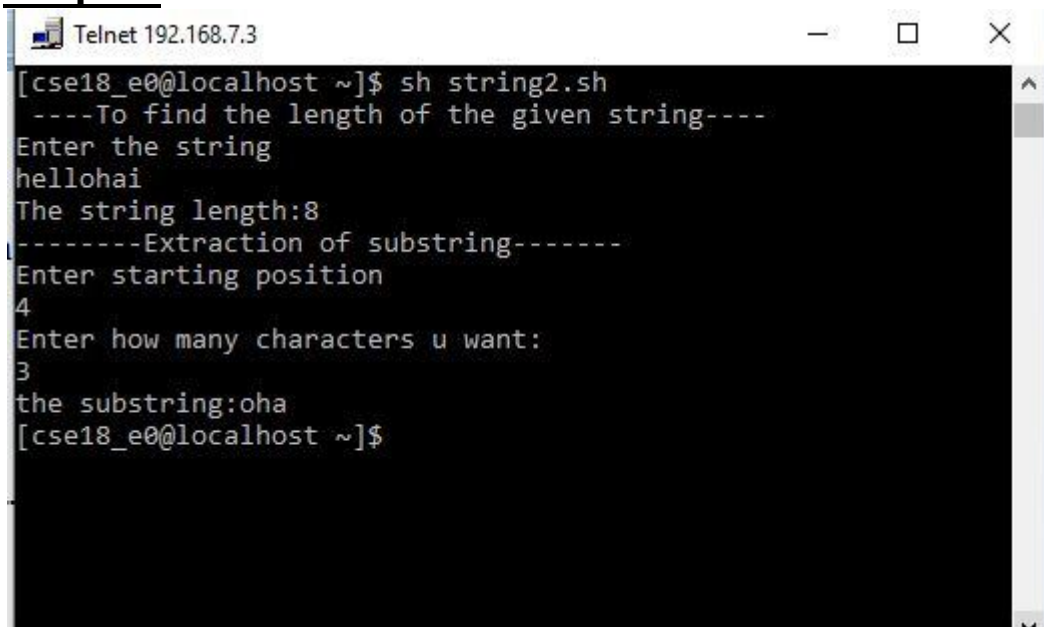
```
[cse18_e0@localhost ~]$ sh progrm6.sh  
the user ----- is logged on to the sytem on  
2020-10-17 10:02  
2020-10-17 10:02  
2020-10-17 12:15  
[cse18_e0@localhost ~]$
```

7. Write shell scripts to find the length of a given string and to extract a substring from a given string.

Source code:

```
echo " ----To find the length of the given string----"  
echo "Enter the string"  
read str  
strlen=${#str}  
echo "The string length:$strlen"  
echo "-----Extraction of substring-----"  
echo "Enter starting position"  
read pos1  
echo "Enter how many characters u want:"  
read pos2  
substr=${str:$pos1:$pos2}  
echo "the substring:$substr"
```

Output:



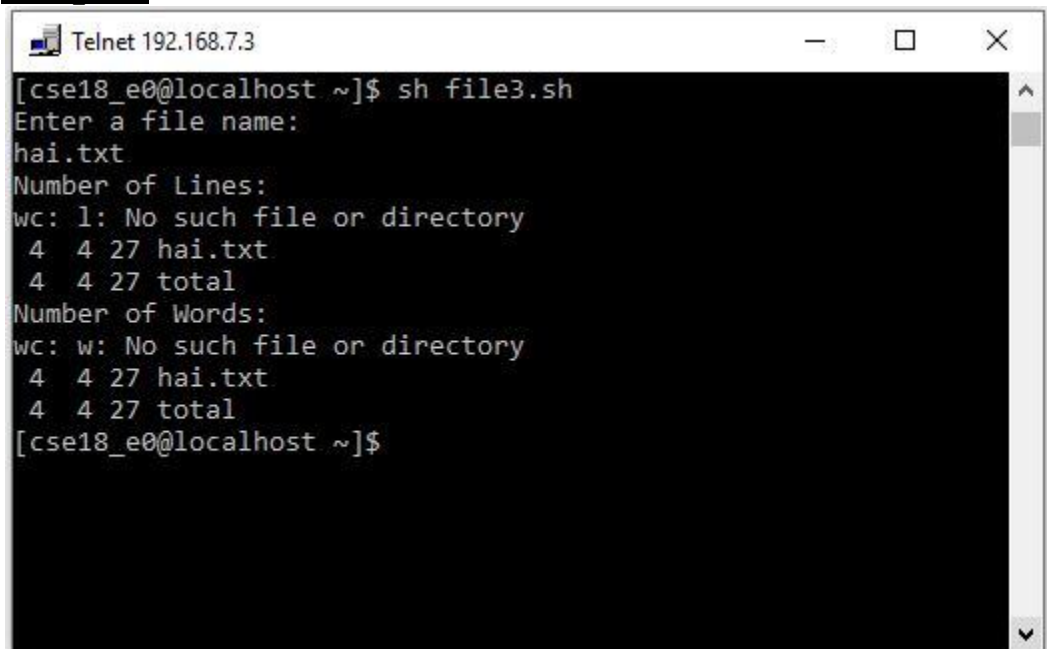
```
Telnet 192.168.7.3  
[cse18_e0@localhost ~]$ sh string2.sh  
----To find the length of the given string----  
Enter the string  
hellohai  
The string length:8  
-----Extraction of substring-----  
Enter starting position  
4  
Enter how many characters u want:  
3  
the substring:oha  
[cse18_e0@localhost ~]$
```

8. Write a shell script that counts the number of lines and words present in a given file.

Source code;-

```
echo Enter a file name:
read fn
echo Number of Lines:
wc -l $fn
echo Number of Words:
wc -w $fn
```

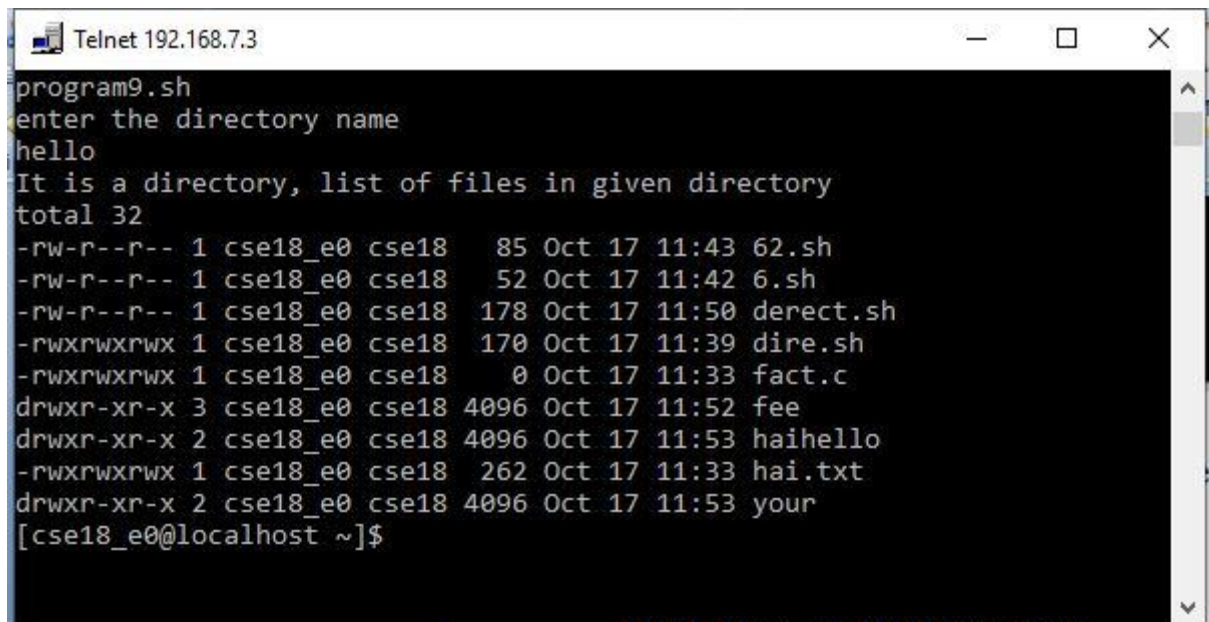
Output:



```
Telnet 192.168.7.3
[cse18_e0@localhost ~]$ sh file3.sh
Enter a file name:
hai.txt
Number of Lines:
wc: l: No such file or directory
 4  4 27 hai.txt
 4  4 27 total
Number of Words:
wc: w: No such file or directory
 4  4 27 hai.txt
 4  4 27 total
[cse18_e0@localhost ~]$
```

9. Write a shell script that displays the list of all files in the given directory.**Source code:-**

```
echo "enter the directory name"  
read dir  
if [ -d $dir ]  
then  
echo "It is a directory, list of files in given directory"  
ls -l $dir  
fi
```

Output:

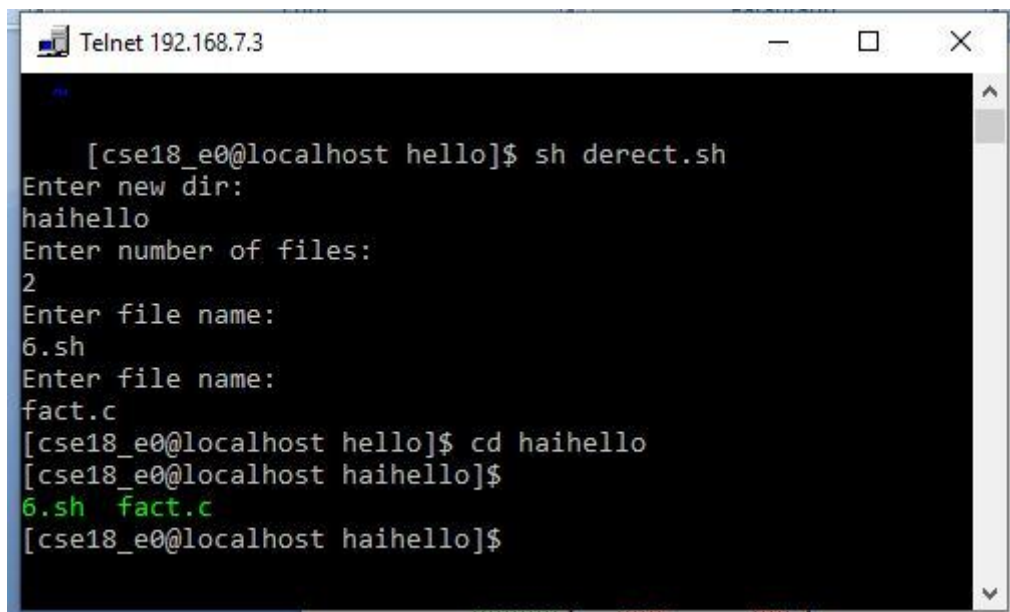
```
Telnet 192.168.7.3  
program9.sh  
enter the directory name  
hello  
It is a directory, list of files in given directory  
total 32  
-rw-r--r-- 1 cse18_e0 cse18 85 Oct 17 11:43 62.sh  
-rw-r--r-- 1 cse18_e0 cse18 52 Oct 17 11:42 6.sh  
-rw-r--r-- 1 cse18_e0 cse18 178 Oct 17 11:50 derect.sh  
-rwxrwxrwx 1 cse18_e0 cse18 170 Oct 17 11:39 dire.sh  
-rwxrwxrwx 1 cse18_e0 cse18 0 Oct 17 11:33 fact.c  
drwxr-xr-x 3 cse18_e0 cse18 4096 Oct 17 11:52 fee  
drwxr-xr-x 2 cse18_e0 cse18 4096 Oct 17 11:53 haihello  
-rwxrwxrwx 1 cse18_e0 cse18 262 Oct 17 11:33 hai.txt  
drwxr-xr-x 2 cse18_e0 cse18 4096 Oct 17 11:53 your  
[cse18_e0@localhost ~]$
```

10. Write a shell scripts that copies multiple files to a directory

Source code:-

```
iter=1
echo Enter new dir:
read nn
mkdir $nn
echo Enter number of files:
read na
while [ $iter -le $na ]
do
echo Enter file name:
read fn
cp $fn $nn
iter=`expr $iter + 1`
done
```

Output:



```
Telnet 192.168.7.3
[cse18_e0@localhost hello]$ sh direct.sh
Enter new dir:
haihello
Enter number of files:
2
Enter file name:
6.sh
Enter file name:
fact.c
[cse18_e0@localhost hello]$ cd haihello
[cse18_e0@localhost haihello]$
6.sh fact.c
[cse18_e0@localhost haihello]$
```

11. Write a shell script (small calculator) that adds, subtracts, multiplies and divides the given two integers. There are two division options: one returns the quotient and the other returns remainder. The script requires 3 arguments:

The operation to be used and two integer numbers. The options are add (-a), subtract (-s), multiply (-m), quotient (-c) and remainder (-r).

Source code:-

```

echo "Enter First Value "
read x
echo "Enter Second Value "
read y
while [ $q -ne 0 ]
do
echo "Enter -a for adding"
echo "Enter -s for subtraction"
echo "Enter -m for multiplication"
echo "Enter -c for Quotient"
echo "Enter -r for remainder"
read s
case $s in
-a) p=`expr $x + $y`
    Echo "Sum = $p"
;;
-b) p=`expr $x - $y`
    Echo "difference = $p"
;;
-m) p=`expr $x \* $y`
    Echo "Product = $p"
;;
-c) p=`expr $x / $y`
    Echo "quotient = $p"
;;
-r) p=`expr $x % $y`
    Echo "remainder = $p"
;;

```

12. Write a C program that illustrates uses of the opendir, readdir, and closedir APIs.

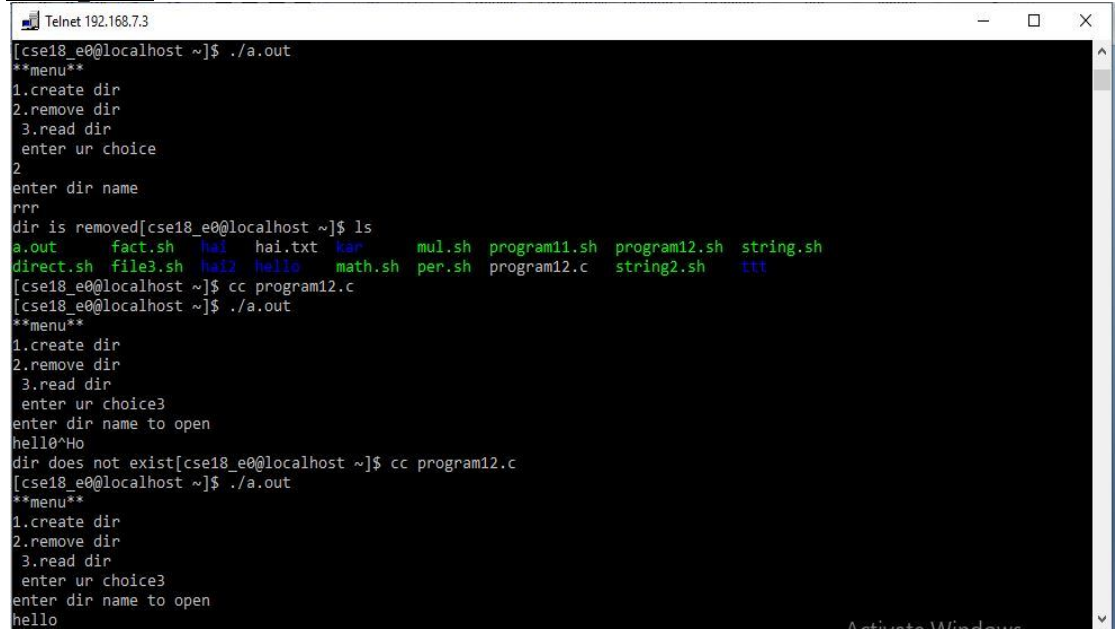
Source code:-

```
#include<stdio.h>
#include<fcntl.h>
#include<dirent.h>
main()
{
char d[10]; int c,op; DIR *e;
struct dirent *sd;
printf("**menu**\n1.create dir\n2.remove dir\n 3.read dir\n enter ur choice");
scanf("%d",&op);
switch(op)
{
case 1: printf("enter dir name\n");
scanf("%s",&d);
c=mkdir(d,777);
if(c==1)
printf("dir is not created"); else
printf("dir is created"); break;
case 2: printf("enter dir name\n");
scanf("%s",&d);
c=rmdir(d);
if(c==1)
printf("dir is not removed"); else
printf("dir is removed"); break;
case 3: printf("enter dir name to open");
scanf("%s",&d);
e=opendir(d);
if(e==NULL)
printf("dir does not exist"); else
{
printf("dir exist\n");
while((sd=readdir(e))!=NULL)
printf("%s\t",sd->d_name);
}
}
closedir(e);
```



```
break;  
}  
}
```

Output:



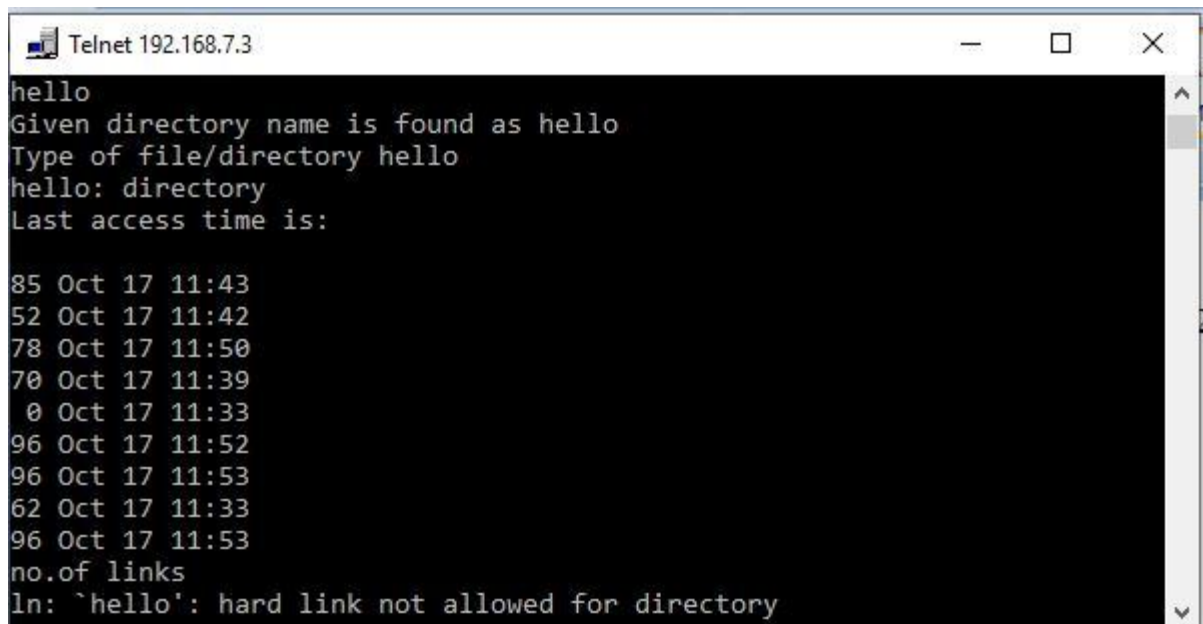
13. Write a program that takes one or more file/directory names as command line input and reports the following information on the file:

- **File type.**
- **Number of links.**
- **Time of last access.**
- **Read, Write and Execute permissions.**

Sourcecode:-

```
for i in $*
do
if [ -d $i ]
then
echo "Given directory name is found as $i"
fi
if [ -f $i ]
then
echo "Given name is a file as $i "
fi
echo "Type of file/directory $i"
file $i
echo "Last access time is:"
ls -l$i | cut-c 31-46
echo "no.of links"
ln $i
if [ -x $i -a -w $i-a -r $i ]
then
echo "$i contains all permission"
else
echo "$i does not contain all permissions"
fi
done
```

Output:



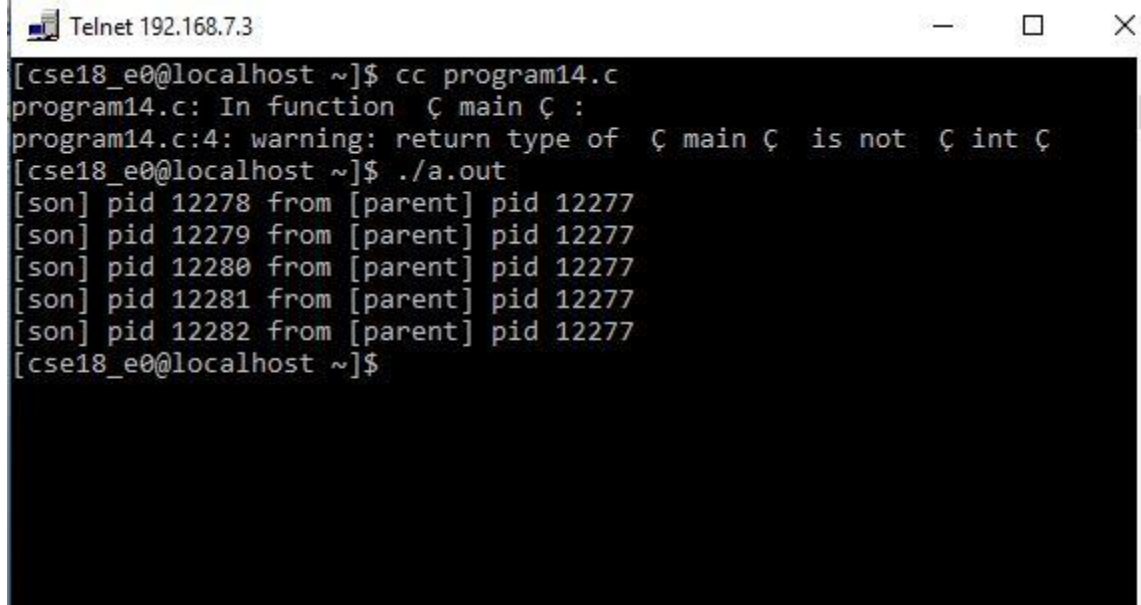
```
Telnet 192.168.7.3
hello
Given directory name is found as hello
Type of file/directory hello
hello: directory
Last access time is:
85 Oct 17 11:43
52 Oct 17 11:42
78 Oct 17 11:50
70 Oct 17 11:39
 0 Oct 17 11:33
96 Oct 17 11:52
96 Oct 17 11:53
62 Oct 17 11:33
96 Oct 17 11:53
no.of links
ln: `hello': hard link not allowed for directory
```

14. Write a C program that illustrates the creation of child process using fork system call.

Source code:-

```
#include<stdio.h>
int main()
{
    int i;
    for( i=0;i<5;i++)
    {
        if(fork() == 0)
        {
            printf("[son] pid %d from [parent] pid %d\n", getpid(), getppid());
            exit(0);
        }
    }
    for(i=0;i<5;i++)
    wait(NULL);
}
```

Output:



```
Telnet 192.168.7.3
[cse18_e0@localhost ~]$ cc program14.c
program14.c: In function   main   :
program14.c:4: warning: return type of   main   is not   int  
[cse18_e0@localhost ~]$ ./a.out
[son] pid 12278 from [parent] pid 12277
[son] pid 12279 from [parent] pid 12277
[son] pid 12280 from [parent] pid 12277
[son] pid 12281 from [parent] pid 12277
[son] pid 12282 from [parent] pid 12277
[cse18_e0@localhost ~]$
```