Department of Computer & Software Engineering – ITU
CE301L: Operating Systems Lab

Course Instructor: Dr. Rehan Ahmed	**Dated:**
Lab Engineer: Muhammad Umair Shoaib	**Semester: Fall 2024**
Batch: BSCE22	

Lab 1 – Introduction to Linux Command Line Interface

Name	**Roll number**	**Total (out of 35)**

Checked on: _____

Signature: _____

1 Introduction to Linux Command Line Interface

Introduction

In this lab we will have a look at the command line interface in Linux using the bash terminal in the Ubuntu Linux distribution. We will learn commands with which we will be able to navigate the Linux file system, create new files and manipulate existing files.

Objectives

This lab will introduce students to the command line interface in Ubuntu-Linux. The key learning objectives of this lab are listed below:

- Learn about the bash terminal and command line interface in Linux
- Navigate the Linux file system and view files and contents of directories inside the terminal
- Execute commands that can manipulate files and directories in Linux
- Use I/O redirection and command pipelining

Background

1.1 Navigating the Linux file system

In this section, we will introduce five commands that can be used to navigate the Linux file system, `pwd`, `ls`, `cd`, `less` and `file`. First, let's have a look at the structure of the file system in Ubuntu.

Ubuntu file system has a hierarchical directory structure similar to Windows. The first directory in the hierarchy is called root directory. The root directory contains files and subdirectories which contain more files and subdirectories and so on. Figure 1.1 shows a representation of Ubuntu file system as viewed inside a file manager.

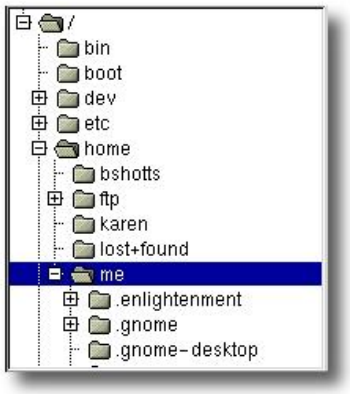

Figure 1.1: File system in Ubuntu as viewed in a file manager

1.1.1 Opening the terminal

The terminal can be opened in more than one ways. You can either search inside the apps by writing any of "terminal", "bash", "console", "gnome-terminal" etc., or you can use the keyboard shortcut of Ctrl + Alt + T. The terminal window is shown in Figure 1.2.

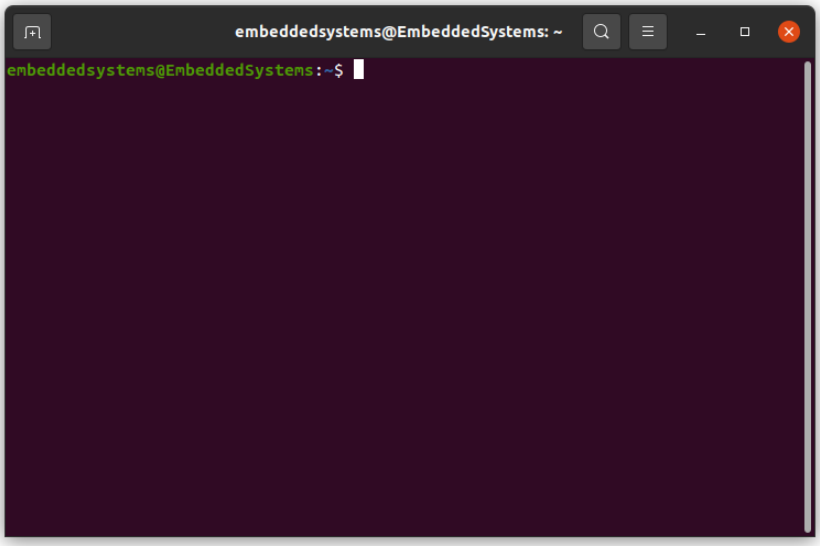

Figure 1.2: gnome-terminal in Ubuntu

1.1.2 pwd

While working on Command Line Interface (CLI), the directory in which we are currently resided is called the working directory. To view the name of the working directory, we use the `pwd` command.

When terminal is opened, the working directory is by default set to our home directory as shown in Figure 1.3.

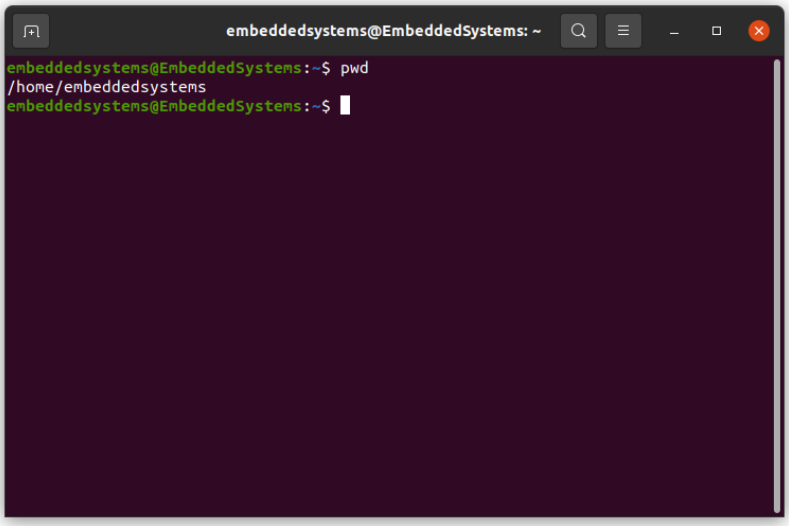

Figure 1.3: Output of pwd command

1.1.3 ls

The `ls` command is used to list the contents of a directory. It is probably the most commonly used Linux command. It can be used in a number of different ways. Here are some examples:

Table 1.1: Examples of ls command

Command	Result
`ls`	List the files in the working directory
`ls /bin`	List the files in the /bin directory (or any other directory we care to specify)
`ls -l`	List the files in the working directory in long format
`ls -l /etc /bin`	List the files in the /bin directory and the /etc directory in long format

`ls –la ..`	List all files (even ones with names beginning with a period character, which are normally hidden) in the parent of the working directory in long format

These examples also point out an important concept about commands. Most commands operate like this:

```
command -options arguments
```

Where `command` is the name of the command, `-options` is one or more adjustments to the command's behavior, and `arguments` is one or more “things” upon which the command operates.

In the case of `ls`, we see that `ls` is the name of the command, and that it can have one or more options, such as `-a` and `-l`, and it can operate on one or more files or directories.

1.1.3.1 A closer look at long format

If we use the `-l` option with `ls`, you will get a file listing that contains a wealth of information about the files being listed. Here's an example:

```
-rw-------   1 me       me            576 Apr 17  2019 weather.txt
drwxr-xr-x   6 me       me           1024 Oct  9  2019 web_page
-rw-rw-r--   1 me       me         276480 Feb 11 20:41 web_site.tar
-rw-------   1 me       me           5743 Dec 16  2018 xmas_file.txt

----------     -------  -------  -------- ------------ -------------
    |             |        |         |         |             |
    |             |        |         |         |         File Name
    |             |        |         |         |
    |             |        |         |         +---  Modification Time
    |             |        |         |
    |             |        |         +-------------   Size (in bytes)
    |             |        |         |
    |             |        |        +------------------------        Group
    |             |        |
    |             |       +--------------------------------        Owner
    |             |
    |            +-----------------------------------------   File Permissions
```

In file permissions, the first character is the type of file. A “-” indicates a regular (ordinary) file. A “d” indicates a directory. The second set of three characters represent the read, write, and execution rights of the file’s owner. The next three represent the rights of the file’s group, and the final three represent the rights granted to everybody else.

1.1.3.2 cd

To change the working directory, we use the `cd` command. To do so, we type `cd` followed by the pathname of the desired working directory. A pathname is the route we take along the branches of the tree of Linux file system to get to the directory we want. Pathnames can be specified in two different ways: absolute pathnames or relative pathnames. Let’s look at absolute pathnames first.

An absolute pathname begins with the root directory and follows the tree branch by branch until the path to the desired directory or file is completed. For example, there is a directory on your system in which most programs are installed. The pathname of the directory is `/usr/bin`. This means from the root directory (represented by the leading slash in the pathname) there is a directory called “usr” which contains a directory called “bin”.

Let's try this out:

```
embeddedsystems@EmbeddedSystems:~$ cd /usr/bin
embeddedsystems@EmbeddedSystems:/usr/bin$ pwd
/usr/bin
embeddedsystems@EmbeddedSystems:/usr/bin$ ls
'['                                  mformat
 aa-enabled                          migrate-pubring-from-classic-gpg
 aa-exec                             mimeopen
 aconnect                            mimetype
 acpi_listen                         min12xxw
 add-apt-repository                  minfo
```

```
addpart                         mkdir
alsabat                         mkfifo
alsaloop                        mkfontdir
alsamixer                       mkfontscale
alsatplg                        mkisofs
alsaucm                         mkmanifest
amidi                           mk_modmap
amixer                          mknod
amuFormat.sh                    mksquashfs
apg                             mktemp
apgbfm                          mkzftree

and many more...
```

An absolute pathname starts from the root directory and leads to its destination, whereas a relative pathname starts from the working directory. To do this, it uses a couple of special notations to represent relative positions in the file system tree. These special notations are "." (dot) and ".." (dot dot). The "." notation refers to the working directory itself and the ".." notation refers to the working directory's parent directory. Here is how it works: Let's change the working directory to `/usr/bin` again:

```
embeddedsystems@EmbeddedSystems:~$ cd /usr/bin
embeddedsystems@EmbeddedSystems:/usr/bin$ pwd
/usr/bin
```

Now let's say that we wanted to change the working directory to the parent of `/usr/bin` which is `/usr`. We could do that in two different ways. First, with an absolute pathname:

```
embeddedsystems@EmbeddedSystems:~$ cd /usr
embeddedsystems@EmbeddedSystems:~$ pwd
/usr
```

Or, with a relative pathname:

```
embeddedsystems@EmbeddedSystems:~$ cd ..
embeddedsystems@EmbeddedSystems:~$ pwd
/usr
```

Likewise, we can change the working directory from `/usr` to `/usr/bin` in two different ways. First using an absolute pathname:

```
embeddedsystems@EmbeddedSystems:~$ cd /usr/bin
embeddedsystems@EmbeddedSystems:~$ pwd
/usr/bin
```

Or, with a relative pathname:

```
embeddedsystems@EmbeddedSystems:~$ cd ./bin
embeddedsystems@EmbeddedSystems:~$ pwd
/usr/bin
```

In most cases, we can omit the "./". It is implied. Typing the following would do the same thing:

```
embeddedsystems@EmbeddedSystems:~$ cd bin
```

1.1.4 less

`less` is a program that lets us view text files. This is very handy since many of the files used to control and configure Linux are human readable.

The `less` program is invoked by simply typing:

```
less text_file
```

This will display the file.

1.1.4.1 Controlling less

Once started, `less` will display the text file one page at a time. We can use the Page Up and Page Down keys to move through the text file. To exit `less`, we type "q". Here are some commands that `less` will accept:

Table 1.2: Keyboard commands for the less program

Command	Action
Page Up or b	Scroll back one page
Page Down or space	Scroll forward one page
G	Go to the end of the text file
1G	Go to the beginning of the text file
/characters	Search forward in the text file for an occurrence of the specified characters
n	Repeat the previous search
h	Display a complete list of less commands and options
q	Quit

1.1.5 file

The `file` command helps to determine what kind of data a file contains before we try to view it. `file` will examine a file and tell us what kind of file it is. To use the file program, we just type:

```
file name_of_file
```

The `file` program can recognize most types of files. Some examples are listed below:

Table 1.3: Various kinds of files in Linux

File Type	Description	Viewable as text?
ASCII text	An ASCII text file	yes
Bourne-Again shell script text	A bash script	yes
ELF 64-bit LSB executable	An executable binary program	no
ELF 64-bit LSB shared object	A shared library	no
GNU tar archive	A tape archive file. A common way of storing groups of files.	no, use `tar tvf` to view listing.
gzip compressed data	An archive compressed with gzip	no
HTML document text	A web page	yes
JPEG image data	A compressed JPEG image	no
PostScript document text	A PostScript file	yes
Zip archive data	An archive compressed with zip	no

1.2 Manipulating files

This section will introduce the following commands:

- touch – create files and update access and modification timestamps of files
- cat – create, display and concatenate files
- cp – copy files and directories
- mv – move or rename files and directories
- rm – remove files and directories
- mkdir – create directories

These four commands are among the most frequently used Linux commands. They are the basic commands for manipulating both files and directories.

1.2.1 touch

The `touch` command can be used to create a single or multiple files:

```
embeddedsystems@EmbeddedSystems:~$ touch file1
```

This creates a file named "file1" in the current working directory.

```
embeddedsystems@EmbeddedSystems:~$ touch file1 file2
```

This creates two files named “file1” and “file2” in the current working directory.

Other useful examples of `touch` and its options are given below:

Table 1.4: Examples of the touch command

Command	Result
`touch -a file1`	Changes the access time of file1 to current time. If file1 does not exist, a new file is created.
`touch –m file1`	Changes the modification time of file1 to current time. If file1 does not exist, a new file is created.
`touch file1 –r file2`	Copies the access and modification times of file2 to the access and modification times of file1.
`touch –t YYMMDDHHMM.SS file1`	Creates a new file with specified timestamp.
`touch –c -t YYMMDDHHMM.SS file1`	Changes the timestamp of an existing file.

1.2.2 cat

The `cat` command is a frequently used command in Linux. The cat command coupled with I/O redirection operators can be used to perform several tasks, such as to create new files, concatenate existing files, to view contents of files, etc. Some examples of `cat` command are listed below:

Command	Result
`cat file1`	Displays the contents of file1 file
`cat file1 file1`	Displays contents of both files
`cat > file1`	Creates new file named file1
`cat –n file`	Displays contents with the line numbers in the output terminal
`cat file1 > file2`	Output contents of file1 in a new or existing file “file2”

1.2.3 cp

The `cp` program copies files and directories. In its simplest form, it copies a single file:

```
embeddedsystems@EmbeddedSystems:~$ cp file1 file2
```

It can also be used to copy multiple files (and/or directories) to a different directory:

```
embeddedsystems@EmbeddedSystems:~$ cp file... directory
```

Other useful examples of `cp` and its options include:

Table 1.5: Examples of the cp command

Command	Results
`cp file1 file2`	Copies the contents of file1 into file2. If file2 does not exist, it is created; otherwise, file2 is silently overwritten with the contents of file1.
`cp -i file1 file2`	Like above however, since the “-i” (interactive) option is specified, if file2 exists, the user is prompted before it is overwritten with the contents of file1.
`cp file1 dir1`	Copy the contents of file1 (into a file named file1) inside of directory dir1.
`cp -R dir1 dir2`	Copy the contents of the directory dir1. If directory dir2 does not exist, it is created. Otherwise, it creates a directory named dir1 within directory dir2.

1.2.4 mv

The `mv` command moves or renames files and directories depending on how it is used. It will either move one or more files to a different directory, or it will rename a file or directory. To rename a file, it is used like this:

```
embeddedsystems@EmbeddedSystems:~$ mv filename1 filename2
```

To move files (and/or directories) to a different directory:

```
embeddedsystems@EmbeddedSystems:~$ mv file... directory
```

Examples of `mv` and its options include:

Table 1.6: Examples of mv command

Command	Results
`mv file1 file2`	If file2 does not exist, then file1 is renamed file2. If file2 exists, its contents are silently replaced with the contents of file1.
`mv -i file1 file2`	Like above however, since the "-i" (interactive) option is specified, if file2 exists, the user is prompted before it is overwritten with the contents of file1.
`mv file1 file2 dir1`	The files file1 and file2 are moved to directory dir1. If dir1 does not exist, mv will exit with an error.
`mv dir1 dir2`	If dir2 does not exist, then dir1 is renamed dir2. If dir2 exists, the directory dir1 is moved within directory dir2.

1.2.5 rm

The `rm` command removes (deletes) files and directories.

```
embeddedsystems@EmbeddedSystems:~$ rm file...
```

Using the recursive option `-r`, `rm` can also be used to delete directories:

```
embeddedsystems@EmbeddedSystems:~$ rm -r directory...
```

Examples of `rm` and its options include:

Table 1.7: Examples of rm command

Command	Results
`rm file1 file2`	Delete file1 and file2.
`rm -i file1 file2`	Like above however, since the "-i" (interactive) option is specified, the user is prompted before each file is deleted.
`rm -r dir1 dir2`	Directories dir1 and dir2 are deleted along with all of their contents.

Linux does not have an undelete command so it is advised to be careful with the use of `rm`. Once you delete something with `rm`, it's gone.

1.2.6 mkdir

The `mkdir` command is used to create directories. To use it, you simply type:

```
embeddedsystems@EmbeddedSystems:~$ mkdir directory...
```

1.2.7 Wildcards

Since the shell uses filenames so much, it provides special characters to help you rapidly specify groups of filenames. These special characters are called wildcards. Wildcards allow you to select filenames based on patterns of characters. The table below lists the wildcards and what they select:

Table 1.8: Summary of wildcards and their meanings

Wildcard	Meaning
*	Matches any characters
?	Matches any single character
`[characters]`	Matches any character that is a member of the set characters. The set of characters may also be expressed as a POSIX character class such as one of the following: POSIX Character Classes [:alnum:] Alphanumeric characters [:alpha:] Alphabetic characters [:digit:] Numerals [:upper:] Uppercase alphabetic characters [:lower:] Lowercase alphabetic characters
`[!characters]`	[!characters] Matches any character that is not a member of the set characters

Using wildcards, it is possible to construct very sophisticated selection criteria for filenames. Here are some examples of patterns and what they match:

Table 1.9: Examples of wildcard matching

Pattern	Matches
*	All filenames
g*	All filenames that begin with the character “g”
b*.txt	All filenames that begin with the character “b” and end with the characters “.txt”
Data???	Any filename that begins with the characters “Data” followed by exactly 3 more characters
[abc]*	Any filename that begins with “a” or “b” or “c” followed by any other characters
[[:upper:]]*	Any filename that begins with an uppercase letter. This is an example of a character class.
BACKUP.[[:digit:]] [[:digit:]]	Another example of character classes. This pattern matches any filename that begins with the characters “BACKUP.” followed by exactly two numerals.
*[![:lower:]]	Any filename that does not end with a lowercase letter.

We can use wildcards with any command that accepts filename arguments.

1.2.7.1 Using commands with wildcards

Since the commands we have covered here accept multiple file and directories names as arguments, you can use wildcards to specify them. Here are a few examples:

Table 1.10: Command examples using wildcards

Command	Results
cp *.txt text_files	Copy all files in the current working directory with names ending with the characters “.txt” to an existing directory named text_files.
mv dir1 ../*.bak dir2	Move the subdirectory dir1 and all the files ending in “.bak” in the current working directory's parent directory to an existing directory named dir2.
rm *~	Delete all files in the current working directory that end with the character “~”. Some applications create backup files using this naming scheme. Using this command will clean them out of a directory.

1.3 I/O redirection

In this section, we will explore a powerful feature used by command line programs called input/output redirection. As we have seen, many commands such as `ls` print their output on the display. This does not have to be the case, however. By using some special notations, we can redirect the output of many commands to files, devices, and even to the input of other commands.

1.3.1 Standard output

Most command line programs that display their results do so by sending their results to a facility called standard output. By default, standard output directs its contents to the display. To redirect standard output to a file, the “>” character is used like this:

```
embeddedsystems@EmbeddedSystems:~$ ls > file_list.txt
```

In this example, the `ls` command is executed and the results are written in a file named file_list.txt as shown in F. Since the output of `ls` was redirected to the file, no results appear on the display.

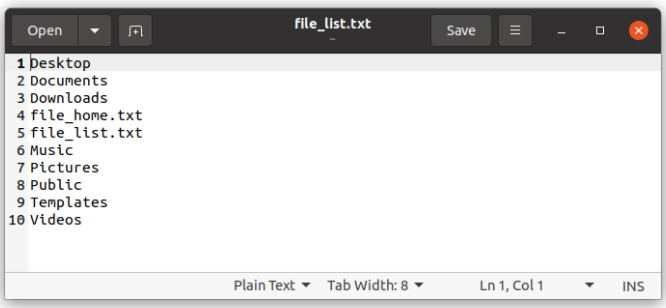

Figure 1.4: Output of ls redirected to file_list.txt

Each time the command above is repeated, file_list.txt is overwritten from the beginning with the output of the command `ls`. To have the new results appended to the file instead, we use “>>” like this:

```
embeddedsystems@EmbeddedSystems:~$ >> file_list.txt
```

When the results are appended, the new results are added to the end of the file, thus making the file longer each time the command is repeated. If the file does not exist when we attempt to append the redirected output, the file will be created.

1.3.2 Standard input

Many commands can accept input from a facility called standard input. By default, standard input gets its contents from the keyboard, but like standard output, it can be redirected. To redirect standard input from a file instead of the keyboard, the "<" character is used like this:

```
embeddedsystems@EmbeddedSystems:~$ sort < file_list.txt
```

In the example above, we used the sort command to process the contents of file_list.txt. The results are output on the display since the standard output was not redirected. We could redirect standard output to another file like this:

```
embeddedsystems@EmbeddedSystems:~$ sort < file_list.txt > sorted_file_list.txt
```

As we can see, a command can have both its input and output redirected. Be aware that the order of the redirection does not matter. The only requirement is that the redirection operators ("<" and ">") must appear after the other options and arguments in the command.

1.3.3 Pipelines

The most useful and powerful thing we can do with I/O redirection is to connect multiple commands together to form what are called pipelines. With pipelines, the standard output of one command is fed into the standard input of another. Here is a very useful example:

```
[me@linuxbox me]$ ls -l | less
```

In this example, the output of the `ls` command is fed into less. By using this "`| less`" trick, we can make any command have scrolling output.

1.3.3.1 Filters

One kind of program frequently used in pipelines is called a filter. Filters take standard input and perform an operation upon it and send the results to standard output. In this way, they can be combined to process information in powerful ways. Here are some of the common programs that can act as filters:

Table 1.11: Common filter commands

Program	What it does
`sort`	It sorts standard input then outputs the sorted result on standard output.
`uniq`	Given a sorted stream of data from standard input, it removes duplicate lines of data (i.e., it makes sure that every line is unique).
`grep`	Examines each line of data it receives from standard input and outputs every line that contains a specified pattern of characters.
`fmt`	Reads text from standard input, then outputs formatted text on standard output.
`pr`	Takes text input from standard input and splits the data into pages with page breaks, headers and footers in preparation for printing.
`head`	Outputs the first few lines of its input. Useful for getting the header of a file.
`tail`	Outputs the last few lines of its input. Useful for things like getting the most recent entries from a log file.
`tr`	Translates characters. Can be used to perform tasks such as upper/lowercase conversions or changing line termination characters from one type to another (for example, converting DOS text files into Unix style text files).
`sed`	Stream editor. Can perform more sophisticated text translations than `tr`.
`awk`	An entire programming language designed for constructing filters. Extremely powerful.

1.3.3.2 Examples of performing tasks with pipelines

Linux provides a program called lpr that accepts standard input and sends it to the printer. It is often used with pipes and filters. Here are a couple of examples:

```
cat poorly_formatted_report.txt | fmt | pr | lpr
```

And,

```
cat unsorted_list_with_dupes.txt | sort | uniq | pr | lpr
```

In the first example, we use `cat` to read the file and output it to standard output, which is piped into the standard input of `fmt`. `fmt` formats the text into neat paragraphs and outputs it to standard output, which is piped into the standard input of `pr`. `pr` splits the text neatly into pages and outputs it to standard output, which is piped into the standard input of `lpr`. `lpr` takes its standard input and sends it to the printer.

The second example starts with an unsorted list of data with duplicate entries. First, `cat` sends the list into `sort` which sorts it and feeds it into `uniq` which removes any duplicates. Next `pr` and `lpr` are used to paginate and print the list.

Lab tasks

This lab will be evaluated based on your in-lab performance and submitted reports according to the rubric presented at the end of this manual. Your solutions of tasks will be used to assess the performance metrics of Realization of Experiment, Conduction of Experiment, Data Collection and Computer Use. The metric of Analysis will be graded according to the questions in the Analysis Section. The grading of Lab Safety and Discipline and Teamwork metrics will be based on the lab engineer's observations (detailed criteria given in Appendix A).

1.4 Task 1: Navigation and manipulation [Marks: 10]

1. Power up your Ubuntu virtual machine and log in using your username and password.
2. Unzip the given folder "lab1" and paste the unzipped folder on your Ubuntu Desktop.
3. Open terminal by pressing Ctrl+Alt+T.
4. Change working directory to the copied folder lab1. Use `pwd` to ensure that the working directory has been changed successfully.
5. Use `ls` to list all files in the lab1 folder. Use options so that long format is used and hidden files are also shown. Paste screenshot of the output of `ls`.
6. Give answer of question 1 of Analysis.
7. Identify rights of owner, group and everybody else by observing the output of `ls`. Also find out the type of each file using `file` command. Copy the following table in your report and fill it with your observations:

Table 1.12: Identification of user rights from ls output

File / dir name	File type	Owner rights	Group rights	Others' rights

8. Now create a folder named "txtfiles" using `mkdir` command inside the lab1 folder.
9. Use `cp` command along with an appropriate *wildcard* to copy all files with .txt extension to the txtfiles folder. After the copy has successfully completed, use `ls` command to list the contents of txtfiles folder. Paste screenshot showing the `cp` command and `ls` command output in your report.
10. Remove the files that have a .txt file extension from the lab1 folder using `rm` command. Paste command.
11. Create another folder named "BSCE" inside lab1 folder and *move* all files ending with two digits using `mv` command and an appropriate wildcard. Again, use `ls` to view folder contents and paste screenshot of `mv` command and `ls` command output in your report.

1.5 Task 2: Command pipelining and I/O redirection [Marks: 10]

1. The files that you have moved inside the BSEE folder contain roll numbers of students of each batch. These roll numbers are not sorted inside these files and some are also duplicate. Sort the BSCE19 file using `sort` command and remove duplicates using `uniq` command. Use `touch` command to create a new file "BSCE_sorted". Then use standard output so that this file would contain the sorted roll numbers from the file BSCE19. Paste screenshot of all commands used in this step in your report.

2. Display the contents of BSCE_sorted file on the terminal using the `cat` command. Paste screenshot of result.
3. Now, use `sort` and `uniq` to sort and remove duplicates from the other file. Output these roll numbers to the same BSCE_sorted file without changing its previous contents. Paste screenshot of the command in your report.
4. Display the contents of BSCE_sorted file on the terminal using `cat` command. Paste screenshot.
5. Now, change working directory to txtfiles folder.
6. Using `grep` command, standard output and pipelining, write a single-line command with pipelining that would search in all files in txtfiles folder for lines that contain the phrase “Operating Systems Lab” and output the result to a new file named “Search_result”. Use `grep` so that the search is not case-sensitive. Paste screenshot of your command.
7. Give answer of questions 2 and 3 of Analysis.
8. Use `cat` and `less` commands to display the contents of the Search_result file. Paste screenshot in your report.

1.6 Analysis [Marks: 5]

1. Which directories do the first two rows of the ls output represent? Note down your answer in lab report. [2]

2. What option did you use with `grep` to make sure that the search is not case-sensitive? [1]

3. What would happen if the order in which commands are pipelined is changed? Explain. [2]

Assessment rubric for Lab 1

Performance	**CLO**	**Able to complete the tasks over 80% (4 – 5)**	**Able to complete the tasks 50 - 80% (2 – 3)**	**Tasks completion below 50% (0 – 1)**	**Marks**
1. Realization of experiment	1	Conceptually understands the topic under study and develops the experimental setup accordingly	Needs guidance to understand the purpose of the experiment and to develop the required setup	Incapable of understanding the purpose of the experiment and consequently fails to develop the required setup	
2. Conducting experiment	1	Sets up the required software, writes and executes bash/C/C++ programs according to the requirement of task and examines the program output carefully	Needs assistance in setting up the software, makes minor errors in writing codes according to task requirements	Unable to set up the software and to write and execute program according to task requirements	
3. Data collection	1	Interprets program output and completes data collection as required in the lab task, ensures that the data is entered in the lab report according to the specified instructions	Completes data collection with minor errors and enters data in lab report with slight deviations from provided guidelines	Fails at observing output states of experimental setup and collecting data, unable to fill the lab report properly	
4. Data analysis	1	Analyzes the data obtained from experiment thoroughly and accurately verifies it with theoretical understanding, accounts for any discrepancy in data from theory with sound explanation, where asked	Analyzes data with minor error and correlates it with theoretical values reasonably. Attempts to account for any discrepancy in data from theory	Unable to establish a relationship between practical and theoretical results and lacks in-depth understanding to justify the results or to explain any discrepancy in data	
5. Computer use	1	Possesses sufficient hands-on ability to work with Linux OS, GNU toolchain and other relevant software	Is able to work on Linux OS with GNU toolchain or other relevant software with some assistance	Cannot operate the Linux OS and other software without significant assistance	
6. Teamwork	3	Actively engages and cooperates with other group members in an effective manner	Cooperates with other group members in a reasonable manner	Distracts or discourages other group members from conducting the experiments	
7. Lab safety and disciplinary rules	3	Observes lab safety rules; and adheres to the lab disciplinary guidelines aptly	Observes safety rules and disciplinary guidelines with minor deviations	Disregards lab safety and disciplinary rules	
				Total (out of 35)	