Computing Systems - Reading List

Module Name: Introduction to an Operating System

Lesson 3: Text Processing Commands

Recommended Reading Material

Brief: Notes attached with this document

Book:

Remote access link (Optional):

Duration: 30 minutes

Table of Contents

1. Introduction
2. The ‘cat’ command
    2.1. Creating Files
    2.2. Displaying Files
    2.3. Concatenating Files
    2.4. Exercises
3. Managing Files
    3.1. Copying Files
    3.2. Renaming / Moving of Files and Directories
    3.3. Removing Files and Directories
    3.4. Exercises
4. File Permissions
    4.1. Exercises
5. IO Redirection
    5.1. Exercises
6. Filters
    6.1. wc
    6.2. sort
    6.3. head
    6.4. tail
7. The grep command
    7.1. Exercises
8. The find command
9. Piping
    9.1. Exercises
10. Links
    10.1. Hard Links

10.2. Symbolic Links

1. INTRODUCTION

In Lab 0, we discussed some introductory aspects of the UNIX Operating system. We also discussed some basic UNIX commands like passwd [to change the password of the account], pwd [to print present working directory], ls [to list files and directories], man [to display manual pages], info [to display information pages], mkdir [to create directories], cd [to change directory], date [to get/set date and time], and cal [to display calendar]. We also discussed navigation through the directory structure using absolute and relative addressing.

In Lab 1, we will go through some more basic UNIX commands that helps in file related operations like creating, modifying, displaying, deleting and searching for contents in files. We will also discuss about file permissions in this lab. In a multi-user system like Linux where different people would be using variety of programs, files, etc., there is a need for organizing and keeping things secured. File permissions session in this lab sheet will help us securing our files and directories in the system.

In this lab sheet, we will also explore a powerful feature used by many command line programs called input/output redirection. We will also look at pipes, a form of redirection used in Linux and other UNIX-like operating systems to send the output of one program to another program for further processing.

2. The ‘cat’ command

cat is one of the most frequently used commands in UNIX-like operating systems. It has three related functions with regard to text files: creating new ones, displaying them and combining copies of them.

2.1. Creating Files

To create a file ‘**will**’, type the command at the command prompt

```
cat > will
```

and press the enter key. Terminal will wait for input from user. Now type

Take up one idea. Make that one idea your life - think of it, dream of it, live on that idea. Let the brain, muscles, nerves, every part of your body, be full of that idea, and just leave every other idea alone. This is the way to success.

After typing the paragraph above, press Ctrl-d. Once this key combination is typed, UNIX saves the contents typed so far and saves in the file named ‘will’.

To create a file with empty contents, we can use the **‘touch’** command. To create a file, type the command at the command prompt

```
touch > emptywill
```

UNIX creates a file by the name ‘emptywill’ with no contents in it. You can create multiple empty files using a single touch command by providing a list of file names separated by spaces.

2.2. Displaying Files

To see the contents of the file created, you can use cat command (“gedit” will also do) as

```
cat will
```

2.3. Concatenating (Combining) Files

Create one more file named “first” using ‘cat’ or ‘gedit’ command. Type the following

The famous Swami Vivekananda addressed the youth of India and said the following words.

To concatenate the contents of the two files (will and first) and store the result in the file named ‘firstwill’, the cat command is issued as follows:

```
cat first will > firstwill
```

‘firstwill’ contains the content of the file ‘first’ followed by the content of the file ‘will’.

2.4. Exercises

Create a directory **“Exercises”** under your home directory. Create two files **file1** and **file2** under **“Exercises”** directory. (Don’t forget to add some text in your files! ☺).

(1) Display the contents of both **file1** and **file2** in terminal with **“$”** at the end of each line.

(2) Display the contents of **file2** with the line numbers.

(3) Concatenate contents of **file1** and **file2** to **file3**. To distinguish the contents of combining files, include the line numbers for **file1** and end each lines of **file2** with a “$”.

3. Managing Files

3.1. Copying Files ('cp' command)

Copy command can be used to copy the contents of one file into another file or copy files from one location to another.

Syntax: cp [OPTION1] [OPTION2] ... SOURCE DESTINATION

*OPTIONS are optional

Options	Purpose
-f	if an existing destination file cannot be opened, remove it and try again
-i	prompt before overwrite
-l	link files instead of copying
-u	copy only when the source file is newer than the destination file or when the destination file is missing
-R or –r	copy directories recursively

- Example: To copy the contents of the file **firstwill** (source file) into the file **finalwill** (destination file), we use the **cp** command as follows.

cp firstwill finalwill

If the file **finalwill** doesn't exist, a new file **finalwill** is created in the current directory with the content of **firstwill**. And if the file exists and is used as the destination of **cp** command, the contents of this file will be overwritten.

- If the destination is a directory, the source file is copied into the destination directory with the same name as source. The source and destination can refer to absolute or relative path specification of the file location.

3.2. Moving (and Renaming) File/Directory ('mv' command)

Moving a file/directory means removing it from its current location and copying it to the new location.

Syntax: mv [OPTION1] [OPTION2] ... **SOURCE DESTINATION**

*OPTIONS are optional

Options	Purpose
-f	This option replaces the file/directory if it exists already in the destination without prompting to the user. Note that this is the default if the standard input is not a terminal.
-i	This option prompts us, if we are trying to replace a file/directory in the destination.

- **Examples:**
- **mv finalwill finalwish**

Moves file named **finalwill** to **finalwish**. The file named **finalwill** will not be available in the system after this operation.

- **mv myfile exercises/**

Moves the file named **myfile** to the directory **exercises**. The file named **myfile** will be available inside exercise directory (not in current location).

- **mv file2 ../**
  Moves the file named **file2** to the **parent** directory (if write permission is available).
- **mv –f first third**

Moves the file **first** to **third** in the same directory (location) or it is equivalent to removing **first** from the current directory and writes into the current directory as **third**. **This command has the effect of renaming the file first to third**.

But what if a file named **third** already exists in the current directory?
The –f specified in the command will replace the existing file **third** with the new one. But, the command

mv –i first third

will notify you before it attempts to replace the existing file, and will prompt you if you are sure in replacing the existing one. If you type **n**, it would skip replacing.

This command can be issued without –f or –i options. In this case, the UNIX system assumes the option is –f. That is, option –f is the default option.

3.3. Removing Files and Directories ('rm' command)

To remove a file we use **rm** command.

Syntax: rm [OPTION1] [OPTION2] ... [filenames | directory]

Options	Purpose
-f	Remove all files (whether write-protected or not) in a directory without prompting the user. In a write-protected directory, however, files are never removed (whatever their permissions are), but no messages are displayed. If the removal of a write-protected directory is attempted, this option will not suppress an error message.
-i	Interactive. With this option, rm prompts for confirmation before removing any files. It overrides the -f option and remains in effect even if the standard input is not a terminal.
-r or -R	Recursively remove directories and subdirectories in the argument list. The directory will be emptied of files and removed. The user is normally prompted for removal of any write-protected files which the directory contains. The write-protected files are

	removed without prompting, however, if the -f option is used, or if the standard input is not a terminal and the -i option is not used. Symbolic links that are encountered with this option will not be traversed. If the removal of a non-empty, write-protected directory is attempted, the utility will always fail (even if the -f option is used), resulting in an error message.

- **Examples**

rm help	To remove a file named **help**
rmdir exercises	To remove a directory named **exercises**

To avoid inadvertently deleting a file, use the **rm** command with `-i` option.

rm -i filename

This will prompt you to confirm that you want to remove a file from the current directory. Answering **y** will delete the file. The file is not deleted if any other response is given.

- To remove a directory, we can use the **rm** command, but we specify the name of the directory to be deleted instead. **rm -r directory_name**

This deletes all the contents of the directory including any subdirectories.

To avoid inadvertently removing a directory, always use the **rm** command with `-i` option.

rm -ir directory_name

But if the directory we want to delete is empty, we may use **rmdir** command.

3.4. Exercise

Create two sub-directories **"CP1"** and **"CP2"** under your home directory. Create a sub-directory **"SLIDES"** and two text files **"lecture.txt"** and **"lab.txt"** under **"CP1"** directory.

Perform the following operations with home as your working directory.

(1) Rename **"lecture.txt"** file to **"lec.txt"**.

(2) Copy all text files of **"CP1"** directory to **"CP2"** directory.

(3) Move **"CP2"** directory to a newly created directory **"Others"** under your home directory.

(4) Rename **"CP1"** directory to **"CP"**.

(5) Create replica of **"CP"** directory in your home directory and name it as **"Backup"**.

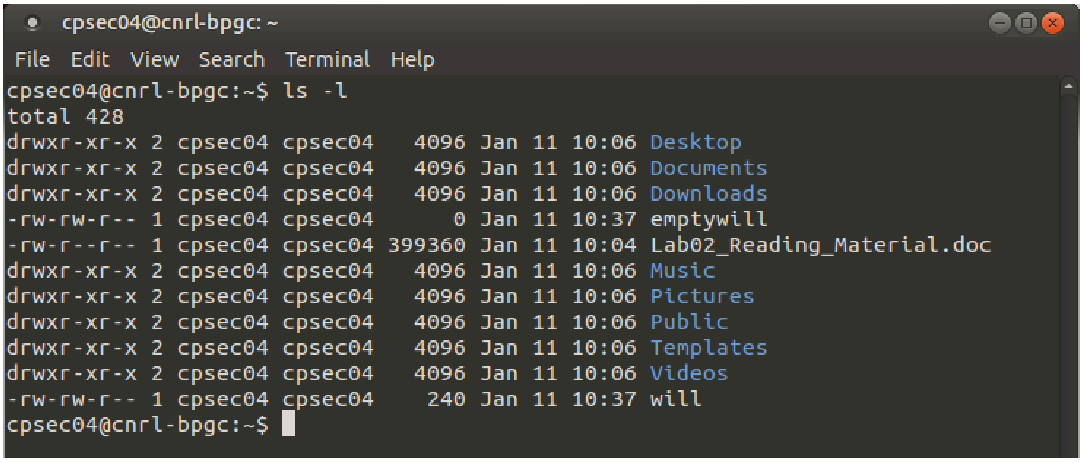
(6) Delete **"Others"** directory from your home directory.

(7) Create **"CP2018"** directory under your home directory. Create a file inside **"CP2018"** named **"info.txt"** with merged contents from **"lec.txt"** and **"lab.txt"** files of your **"CP"** directory.

4. File Permissions

We have seen from Lab 0 that **ls –l** gives files in long listing format as shown in Figure 1.

The listing includes the total block size (1 block=1024 bytes of memory) of the files in the directory and subdirectories, type and permissions, owner, group, size, date and time of last modification and the names of the files or directories in the current directory.



```
cpsec04@cnrl-bpgc:~$ ls -l
total 428
drwxr-xr-x 2 cpsec04 cpsec04   4096 Jan 11 10:06 Desktop
drwxr-xr-x 2 cpsec04 cpsec04   4096 Jan 11 10:06 Documents
drwxr-xr-x 2 cpsec04 cpsec04   4096 Jan 11 10:06 Downloads
-rw-rw-r-- 1 cpsec04 cpsec04      0 Jan 11 10:37 emptywill
-rw-r--r-- 1 cpsec04 cpsec04 399360 Jan 11 10:04 Lab02_Reading_Material.doc
drwxr-xr-x 2 cpsec04 cpsec04   4096 Jan 11 10:06 Music
drwxr-xr-x 2 cpsec04 cpsec04   4096 Jan 11 10:06 Pictures
drwxr-xr-x 2 cpsec04 cpsec04   4096 Jan 11 10:06 Public
drwxr-xr-x 2 cpsec04 cpsec04   4096 Jan 11 10:06 Templates
drwxr-xr-x 2 cpsec04 cpsec04   4096 Jan 11 10:06 Videos
-rw-rw-r-- 1 cpsec04 cpsec04    240 Jan 11 10:37 will
cpsec04@cnrl-bpgc:~$ 
```

Figure 1: The output of ls -l command. Notice the permissions

The words "total 428", indicates that total of 428 blocks is occupied by the files in the disk. Each block contains 1024 bytes of memory.

First column of list identifies the type and permissions associated with each file. Type is identified by first character; it is hyphen (**'-'**) for files, **'d'** for directories and **'l'** for symbolic links(discussed later). Following the type, a series of 9 characters that can take the values **'r'** (read), **'w'** (write), **'x'** (execute), and **'-'** tells about file permissions for three types of users given below respectively.

Each file has a specific permission

- For the user/owner (u)
- For the group (g)
- Others (o)

Each one of them has three permissions

- To read (r) – 4 (100)
- To write (w) – 2 (010)
- To execute (x) – 1 (001)

Note: A hyphen in any position means that you don't have that particular permission.

The **chmod** command is used to set the three permissions for all three categories of users (owner, group and others) for the file. Only owner of the file can set the access permissions.

Syntax:

```
chmod [OPTION]... MODE[,MODE]... FILE...

chmod [OPTION]... OCTAL-MODE FILE...
```

Options	**Purpose**
-c	like verbose but report only when a change is made
-R	change files and directories recursively

Examples:

Command	**Explanation**
chmod u+x will	Execute permission is added to the owner for the file **will**
chmod ugo+x will or chmod a+x will	Execute permission is added to all for the file **will**
chmod o-x will	Execute permission is removed from the others for the file **will**
chmod g+rx, o+x will	Read and execute permission to group and execute permission to others for the file **will**

Alternative way to set the file permissions by using numbers:

Command	**Explanation**
chmod 777 will	Read, write and execute permissions to all for the file **will**
chmod 774 will	Owner and group have all three permissions and others have only read permission for the file **will**
chmod 400 will	Read permission to the owner for the file **will** is set. Group and others don't have any access to the file.

4.1. Exercises

With **"CP"** as your working directory, perform the following operations:

(1) Deny all permissions for others to the contents of **"CP2018"** directory.

(2) Prevent group and others from modifying the contents of your **"Backup"** directory.

(3) Set all permissions for the contents of **"CP"** directory to all three types of users.

5. IO Redirection

Every operating system defines a standard input device and a standard output device. UNIX defines keyboard to be the standard input device and the monitor to be the standard output device. UNIX allows us to temporarily change the standard input and standard output by means of what is called as **Indirection** and **Piping**. The symbol > means indirection of output (to a file or printer) and the symbol < means indirection of input for a command (from a file).

Create a file named **testfile** with the following contents:

Happiness is when what you think, what you say, and what you do are in harmony.

Execute the following command: **cat testfile > file2**

The above command declares **file2** as the temporary standard output causing the contents of **testfile** to be redirected to **file2**. Open file2 and check if the contents are the same as that in **testfile**.

Now re-open the file **testfile**, change the contents to

A person who is repeatedly committing the same mistake is a fool.

Execute the following command: **cat testfile > file2**

Open file2. Do you still see the earlier contents? If the file file2 is not empty, it will be overwritten. To avoid this, use >>. This appends to the old contents of file2.

Change the contents of **testfile** again with the following text

Man lives in fear of death his entire life, but alas: death is the ultimate surprise. So, live life King Size.

and execute the command: **cat testfile >> file2**

Open **file2** and check if the earlier contents and the currently modified contents are seen in file2. That was about output indirection.

Let us now turn to input indirection.

What does the command **cat testfile** do?

This command (cat) takes its input from the file named **testfile** and displays the result into the standard output. Please note that this command does not take its input from standard input, rather from a file.

Now try the command: **cat < testfile**

What is the result? No surprise. No difference from the previous command. You are getting the contents of the file **testfile** in the standard output, but the file **testfile** is temporarily the standard input and the command cat gets its input from standard input.

Now try the command: **cat < testfile > op**

What does this command perform? How does it differ from the command **cat testfile > op** ?

The cat command considers **testfile** as input and directs it to the file **op**. Open the file **op** and check if the contents are the same as **testfile**.
(Note: File **op** is implicitly created when the indirection command is used.)

The indirection operators are:

Indirection Operators	Function
> file	make file as the standard output
< file	make file as the standard input
>> file	make file as the standard output, append to it if it exists
<< word	take the shell input up to the first line containing 'word'
command1 | command2	make the output of command1 as the input to command2

5.1. Exercises

Commands	What does the command do?
$ ls > filelist	
$ date ; who > op	
$ date ; who; ls >op	
$ (date ; who) > op	
$ date; (who ; ls) > op	

6. Filters

File Related (filter) Commands

Many UNIX programs read some input, perform a transformation on it, and write it to some output. These programs are called filters and when used together with pipes can produce powerful programs. Some useful filters are explained below.

6.1. wc (word count):

This command helps in counting the number of lines, words and characters in a specified file.

To get the count for **file1**, the wc command is: **wc file1**

This command will give output in the format: **<l> <w> <c> file1**

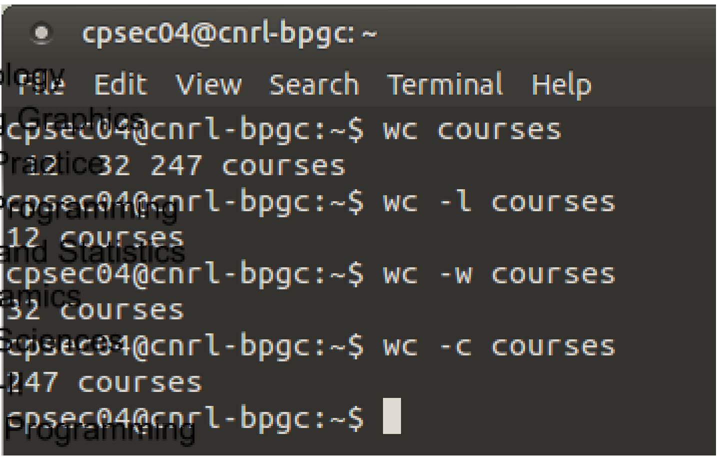
This means the file **file1** has 'l' number of lines, 'w' number of words and 'c' number of characters**.**
This command can use options like –l, -w, -c to get the number of lines, words characters individually.

The usage will be as follows:

wc –option filename

Example: Create a file named **courses** with the following contents:

1. Physics-I
2. Mathematics-I
3. Chemistry-I
4. General Biology
5. Engineering Graphics
6. Workshop Practice
7. Computer Programming
8. Probability and Statistics
9. Thermodynamics
10. Electrical Sciences
11. Chemistry-II
12. Computer Programming



```
cpsec04@cnrl-bpgc:~$ wc courses
12  32 247 courses
cpsec04@cnrl-bpgc:~$ wc -l courses
12 courses
cpsec04@cnrl-bpgc:~$ wc -w courses
32 courses
cpsec04@cnrl-bpgc:~$ wc -c courses
247 courses
cpsec04@cnrl-bpgc:~$ 
```

Figure 2: wc command variations

The wc command outputs for the file **courses** are as shown in figure 2.

6.2. sort :

This command is used to sort the contents of the file. While sorting the files, this command bases the comparison on the first character in each of the lines. If the first character for two lines is same, then the second character is used in comparison.

To sort the contents in the file **file1**, the sort command is

sort file1

Example:

Create a file named **file1** with the following contents:

9
7
5
1
2

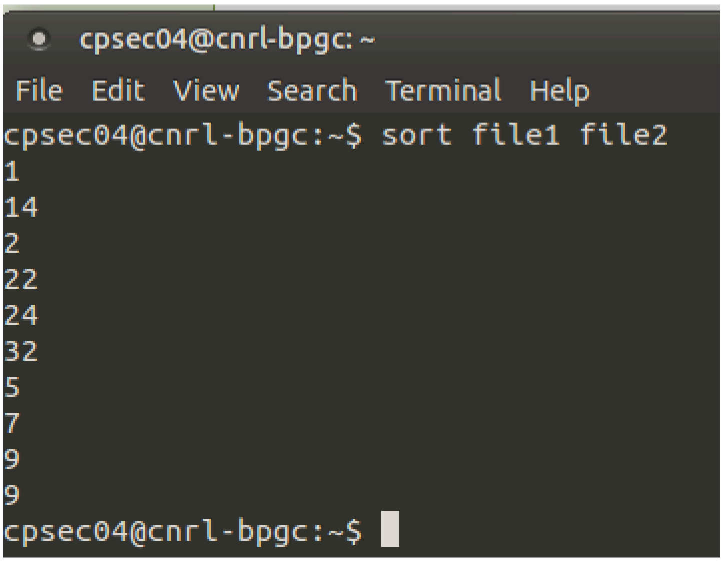
Create another file named **file2** with the following contents:

24
22
14
32
9

Execute the following command:

sort file1 file2

This command will sort the contents of both files at a time (Figure 3).



```
cpsec04@cnrl-bpgc:~$ sort file1 file2
1
14
2
22
24
32
5
7
9
9
cpsec04@cnrl-bpgc:~$ 
```

Figure 3: Results of a sort command

The general syntax of the sort command is

sort [OPTION] ... [file name]**...**

The most common flags are as follows:

Option	Comment
-b	**Ignore leading blanks**
-d	**Consider only blanks and alphanumeric characters**
-f	Fold lowercase to uppercase characters before sorting (i.e., "Bill", "bill" and “BILL” are treated the same).
-r	Reverse the result of comparisons.
-u	**Ignore repeating lines**

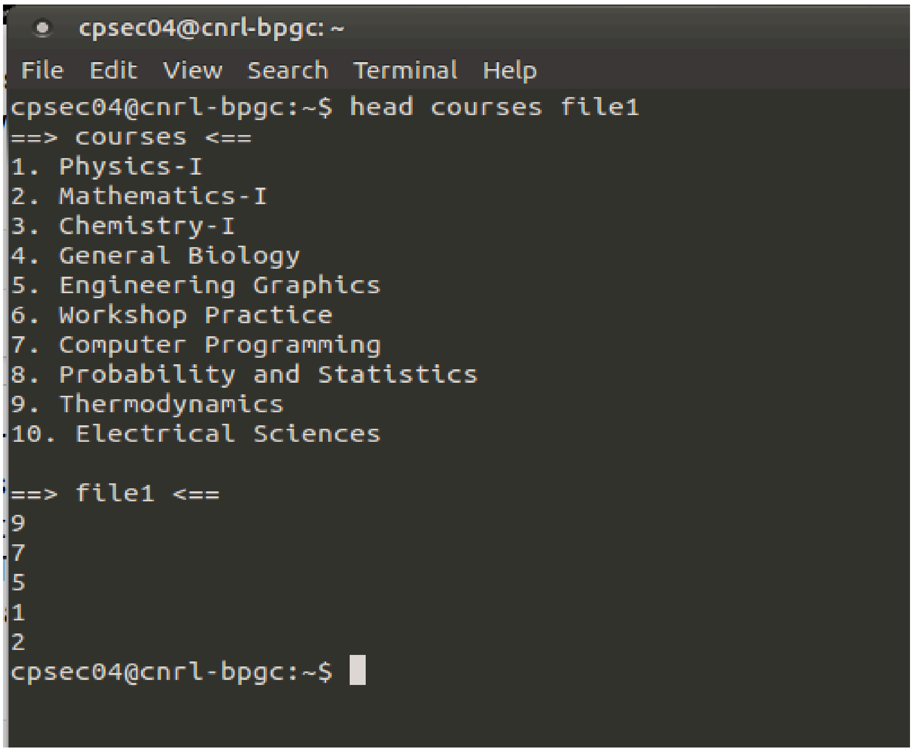
6.3. head:

The head command reads the first few lines of any text given to it as an input and writes them to the display screen.Syntax is:

head [options] [file(s)]

By default, head returns the first ten lines of each file name specified.

Try the following head commands for the files created above. (Figure 4) head courses file1



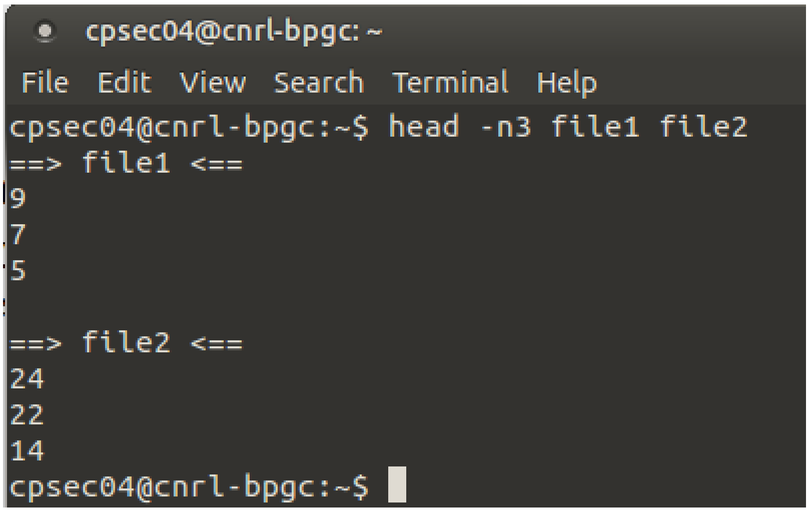
```
cpsec04@cnrl-bpgc:~$ head courses file1
==> courses <==
1. Physics-I
2. Mathematics-I
3. Chemistry-I
4. General Biology
5. Engineering Graphics
6. Workshop Practice
7. Computer Programming
8. Probability and Statistics
9. Thermodynamics
10. Electrical Sciences

==> file1 <==
9
7
5
1
2
cpsec04@cnrl-bpgc:~$
```

Figure 4: Output of the head command with default(no) options

We can specify the number of lines to be displayed from the file by using –n option. The -n option is used followed by an integer indicating the number of lines desired. For example, the above example could be modified to display the first 3 lines from each file:

head –n3 file1 file2



```
cpsec04@cnrl-bpgc:~$ head -n3 file1 file2
==> file1 <==
9
7
5

==> file2 <==
24
22
14
cpsec04@cnrl-bpgc:~$
```

Figure 5: Output of head command with -n option

6.4. tail:

The tail command is similar to the head command except that it reads the final lines in files rather than the first lines.

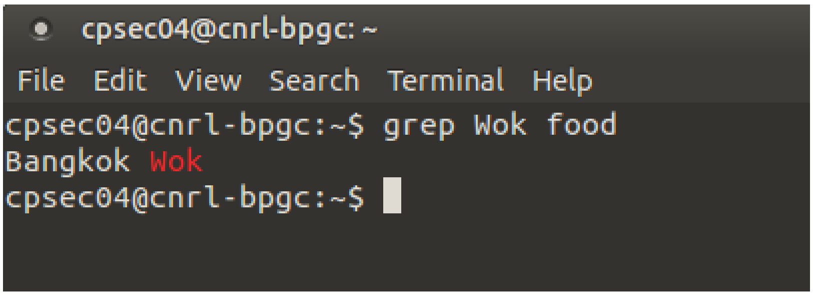
7. Grep Command

The UNIX grep command helps you search for strings in a file.
Create a file named “**food**” with the following contents:

```
Afghani Cuisine
Mandalay
Big Apple Deli
Isle of Java
Tio Pepe's Peppers
Sushi and Sashimi
Sweet Tooth
Bangkok Wok
```

To find the lines that contain the string **“Wok”** from the file food and display those lines to the standard output, the command is

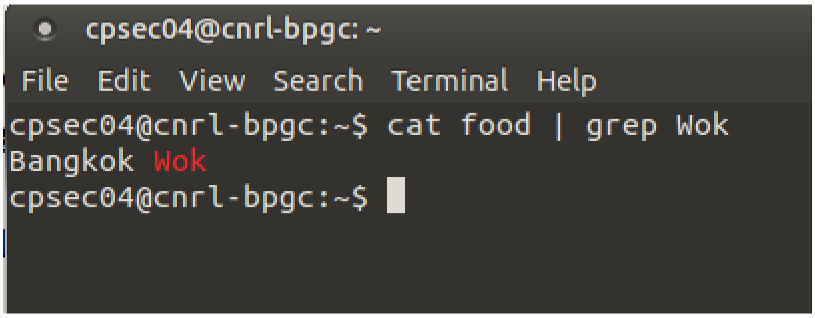
grep Wok food



```
cpsec04@cnrl-bpgc:~$ grep Wok food
Bangkok Wok
cpsec04@cnrl-bpgc:~$
```

Figure 6: Execution of the grep command

It will display the line(s) containing **“Wok”** in the file **food**. It is done by matching the pattern **“Wok”** in the file food. (Figure 6).
What is the result of the command **cat food | grep Wok** ?



```
cpsec04@cnrl-bpgc:~$ cat food | grep Wok
Bangkok Wok
cpsec04@cnrl-bpgc:~$
```

Figure 7: Using grep on the output of cat command

Here, grep acts as a filter. (Figure 7)

Checking for the given string in multiple files:

Syntax: **grep [OPTIONS] "string" FILE_PATTERN**

The grep output will include the file name in front of the line that matches the specific pattern. When the Linux shell sees the meta-character, it does the expansion and gives all the files as input to grep.

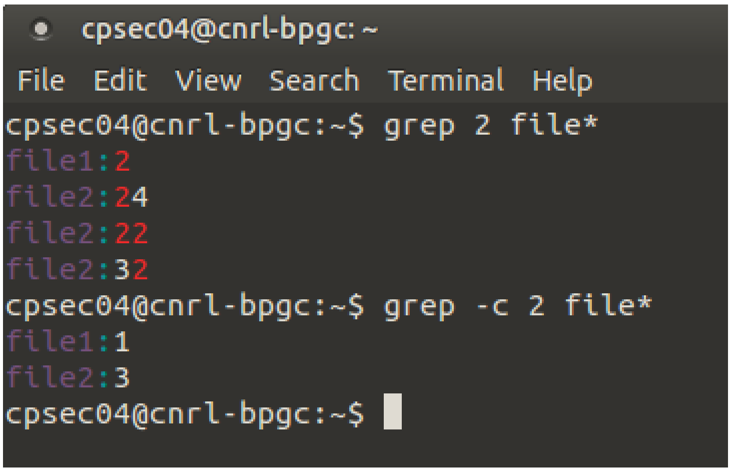
Some options with 'grep' command include:

Option	Comment
-c	Prints only a count of the lines that contain the pattern.
-i	Ignores upper/lower case distinction during comparisons.
-l	Prints only the names of files with matching lines, separated by NEWLINE characters. Does not repeat the names of files when the pattern is found more than once.
-n	Precedes each line by its line number in the file (first line is 1).
-v	Prints all lines except those that contain the pattern.

Try the following grep commands: (Figure 8)

grep 2 file*

grep –c 2 file*



```
cpsec04@cnrl-bpgc:~$ grep 2 file*
file1:2
file2:24
file2:22
file2:32
cpsec04@cnrl-bpgc:~$ grep -c 2 file*
file1:1
file2:3
cpsec04@cnrl-bpgc:~$ 
```

Figure 8: Using grep with -c option

7.1. Exercises

(1) Print the last 8 bytes of the file **"food"**
(2) Create a file **"file4"** with the first 3 lines of **"file1"** and the last 2 lines of **"file2"**
(3) Print numerically sorted contents of "**file2"**

(4) Check if the contents of “**file4**” are in sorted order or not. If not, save the contents sorted in reverse order with no duplicate entries to “**file5**”.
(5) Display the lines of file “**courses**” which has the string “Computer” along with the line number.
(6) Print the length of the longest line in the file “**courses**”.

8. Find

The ‘find’ command is used to search for files in a directory hierarchy. If no ‘starting point’ is given, ‘.’ is assumed. The main difference between find and grep is that find matches the pattern in the file names and grep matches the pattern in the file contents.

The syntax of find is:

```
find [-H] [-L] [-P] [-Olevel] [-D help|tree|search|stat|rates|opt|exec] [path...] [expression]
```

*Items in [ ] are optional

This varies from the ls command in the way that you can set advanced conditions for the files to be listed and can also run actions on them.

Example:

Go to the exercises directory, and create 4 empty files: file1, file2, file3 and file4.

Remove the write permission to the owner for file1.

To find all files which have write permission and begin with the string “file”, the find command is used as follows:

```
find file* -writable
```

You can also use the find command to do actions on the matched files. The actions are generally file operations.

For example, if we need to delete all files that are writable from the previous command, we can modify the above command as follows:

```
find file* -writable –delete
```

Running this command will delete file2, file3 and file4. Many more options are available, you can go through them in the man page for find.

9. Piping

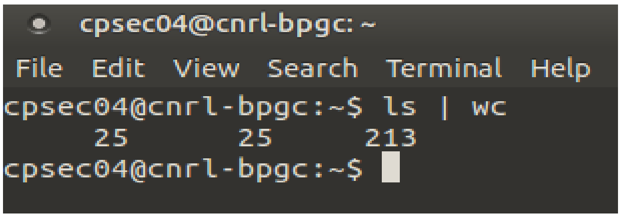
The indirection operator (|) or pipe symbol helps in joining two commands. Now the output from one command becomes the input of the other. Two or more commands connected in this way forms a **pipe**.

When a command takes its input from another command, performs some operation on that input, and writes the result to the standard output (which may be piped to yet another program), it is referred to as a filter. Hence the commands that make up the pipe are called **filters**. (Imagine a sequence of pipes, each connected with the adjacent ones by means of a filter).

One of the most common uses of filters is to modify output. Just as a common filter culls unwanted items, the UNIX filters can be used to restructure output. Putting a vertical bar (|) on the command line between two commands makes up a pipe. When a pipe is set up between two commands, the standard output of the command to the left of the pipe symbol becomes the standard input of the command to the right of the pipe symbol.

So, what commands are qualified to become a filter? Simple, if it can take input from standard input and can send its output to standard output, and then it can be a filter.

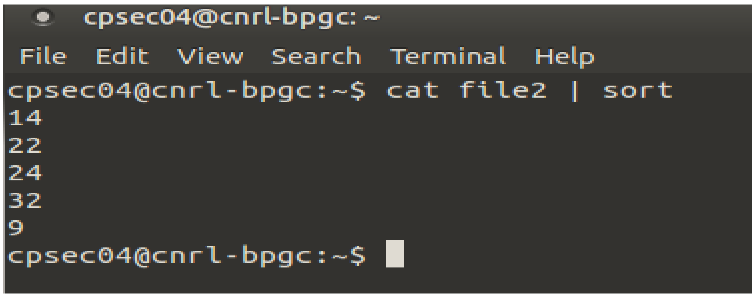
The command: **ls | wc**
Will execute the ls command first, direct the output to the next command wc. The command wc is executed and the output is shown in the terminal as shown in figure 9.



```
cpsec04@cnrl-bpgc:~$ ls | wc
     25      25     213
cpsec04@cnrl-bpgc:~$ 
```

Figure 9: Using output of ls as input to wc

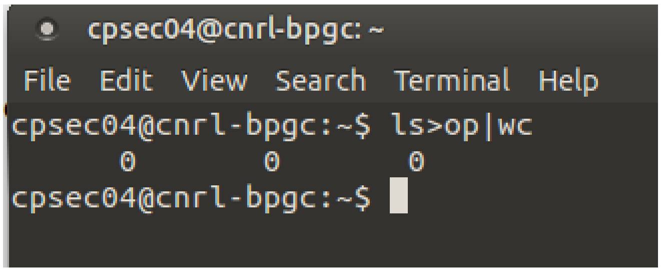
Now execute the following command: **cat file2 | sort**
The output of cat command is given as input to the sort command. The result is shown in figure 10.



```
cpsec04@cnrl-bpgc:~$ cat file2 | sort
14
22
24
32
9
cpsec04@cnrl-bpgc:~$ 
```

Figure 10: Using sort on the output of cat command

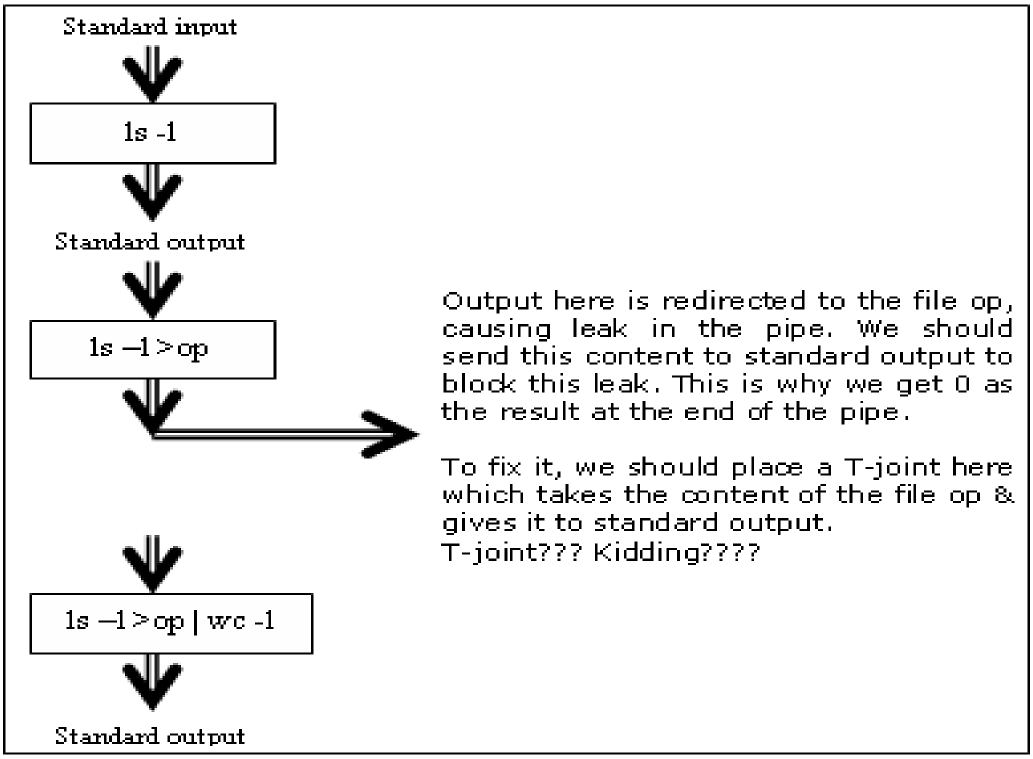
Try executing the following command: **ls > op | wc**



```
cpsec04@cnrl-bpgc:~$ ls>op|wc
      0       0       0
cpsec04@cnrl-bpgc:~$ 
```

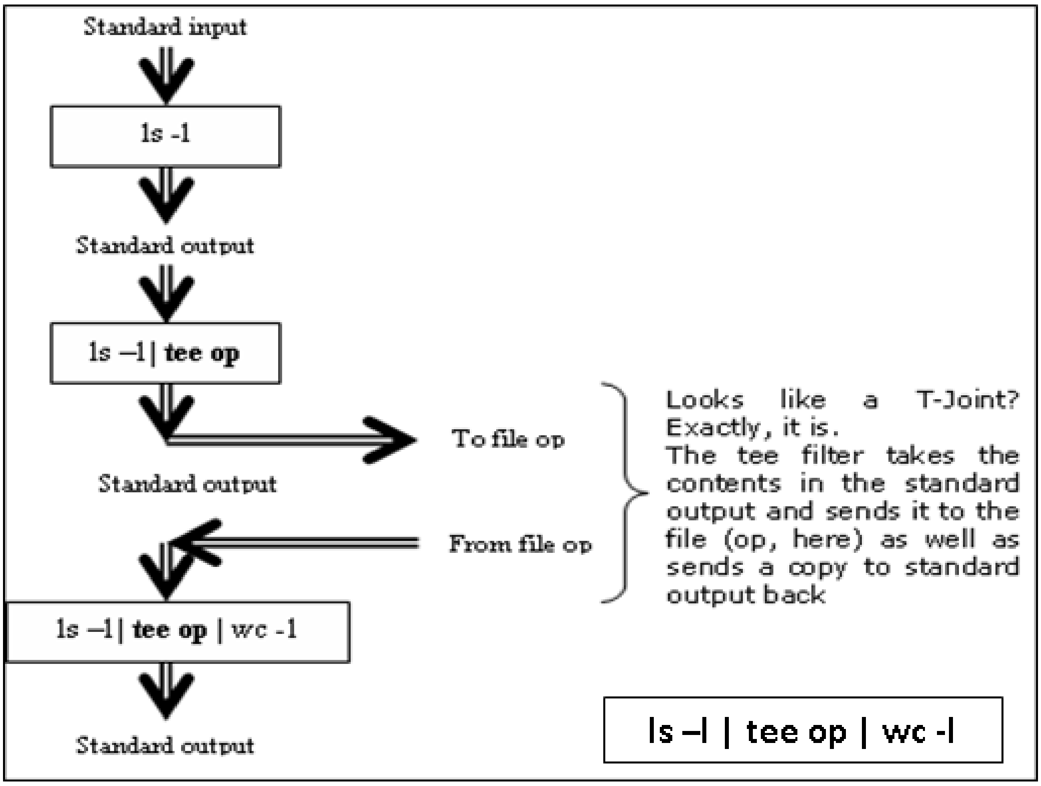
Figure 11: Using redirection with pipe

Why is the output zero?



Output here is redirected to the file op, causing leak in the pipe. We should send this content to standard output to block this leak. This is why we get 0 as the result at the end of the pipe.

To fix it, we should place a T-joint here which takes the content of the file op & gives it to standard output.
T-joint??? Kidding????



Looks like a T-Joint? Exactly, it is.
The tee filter takes the contents in the standard output and sends it to the file (op, here) as well as sends a copy to standard output back

ls –l | tee op | wc -l

Thus the tee command is used to direct the output to a file as well as the standard output which is the terminal. Now try the command **ls -l |tee op | wc –l**

Does it display the number of lines (wc -l) in the directory listing (ls -l) on the terminal?

9.1. Exercises

Execute the following:

Commands	What does the command do?
$ ls | wc	
$ ls > op | wc	
$ ls | wc > op	
$ cat file file file | sort	
$ cat file | sort | head -1 | wc -w	

10. Hard Links and Symbolic Links

As you have seen in the ls –i command, every file and directory is assigned a unique number called as the inode number by the Operating System. This inode number represents an inode record which holds the information about the file like its name, permission, owner details, group details etc.

A link is a simple way to refer to the contents of a file. There are two types of links:

10.1. Hard Links

A hard link is a pointer to the file's i-node.

For example, create a file called "test.txt" with the following contents:

This is the test file.

You can create a hard link to the file using the following command:

ln test.txt htest.txt

The two file names refer to the same content. Editing one file will also modify the other file because both filenames refer to the same file content.

Test this by opening "test.txt" and modifying the content. Then open "htest.txt".

10.2. Symbolic Links

Symbolic link is a file that contains the name of another file. We can then access the contents of the other file through this name. To create a symbolic link for the file "test.txt", type the following command:

ln -s test.txt stest.txt

Now the file "test.txt" can be accessed using "stest.txt" as well.

10.3. The Difference

A hard link works just like any other file pointer. Given a file “test.txt” and its hard link “htest.txt”, there is no way to differentiate which file is the hard link. Also, if we remove “test.txt” using the rm command, the file contents will still be accessible using the “htest.txt” file.

A symbolic link has the following indications:

1. Observe the permissions of a symbolic link after ls –l.
2. Observe the filename section of the symbolic link.
3. ls -F can be used to identify which file is a symbolic link. (If the filename ends with a ‘@’ symbol, it is a symbolic link)

The most important point to note about symbolic links is that unlike hard links, a symbolic link can be used only until the original file exists. If the original file is removed, the symbolic link is not valid anymore and the file contents cannot be accessed in any manner.

For example, if you remove the “test.txt” file using the rm command, using the symbolic link to open the file will result in an error.

However, in this particular case, the contents are still accessible using “htest.txt”.

<u>Solution to Exercise Questions</u>

<u>Exercise 2.4</u>

```
cd ~
mkdir Exercises
cd Exercises
cat > file1
cat > file2
```

```
(1)     cat –e file1 file2
(2)     cat –n file2
(3)     cat –n file1 > file3; cat –e file2 >> file3
```

<u>Exercise 3.4</u>

```
cd ~
mkdir CP1
mkdir CP2
mkdir CP1/SLIDES
cat > CP1/lecture.txt
cat > CP1/lab.txt

(1)     mv CP1/lecture.txt CP1/lec.txt
(2)     cp CP1/*.txt CP2/
(3)     mkdir Others; mv CP2 Others/CP2
(4)     mv CP1 CP
(5)     cp –r CP Backup
(6)     rm –r Others
(7)     mkdir CP2018
        cat CP/lec.txt CP/lab.txt > CP2018/info.txt
```

Exercise 4.1

```
(1)     chmod o-rwx ../CP2014/*
(2)     chmod go-w ../Backup/*
(3)     chmod 777 *
```

Exercise 7.1

```
(1)     tail -8c food
(2)     (head -n3 file1; tail -n2 file2) > file4
(3)     sort -g file2
(4)     sort -c file4
(5)     (sort -ru file4) > file5
(6)     grep -nw Computer courses
(7)     wc -L courses
```