CET-225
Operating Systems
Experiment # 02

Experiment Title

Managing files and Directories in Linux

Assessment of CLO(s): 04

Performed on ____________________

Student Name:			
Roll No.		**Group**	
Semester		**Session**	

Total (Max)	**Performance (03)**	**Viva (03)**	**File (04)**	**Total (10)**
Marks Obtained				
Remarks (if any)				

Experiment evaluated by

Instructor's Name	**Engr Bushra Aziz**		
Date		**Signature**	

THEORY

Files & Filenames

The most basic concept of a file defines it as a distinct chunk of information that is found on the hard drive. Distinct means that there can be many different files, each with its own particular contents. To keep files from getting confused with each other, every file must have a unique identity. In Linux, you identify each file by its name and location. In each location or directory, there can be only one file by a particular name. Linux allows filenames to be up to 256 characters long. These characters can be lower- and uppercase letters, numbers, and other characters, usually the dash (-), the underscore (_), and the dot (.). They can't include reserved meta characters such as the asterisk, question mark, backslash, and space, because these all have meaning to the shell.

Directories

Linux, like many other computer systems, organizes files in directories. You can think of directories as file folders and their contents as the files. However, there is one absolutely crucial difference between the Linux file system and an office filing system. In the office, file folders usually don't contain other file folders. In Linux, file folders can contain other file folders. In fact, there is no Linux "filing cabinet"—just a huge file folder that holds some files and other folders. These folders contain files and possibly other folders in turn, and so on.

The Root Directory

In Linux, the directory that holds all the other directories is called the root directory. This is the ultimate parent directory; every other directory is some level of subdirectory. From the root directory, the whole structure of directory upon directory springs and grows like some electronic elm. This is called a tree structure because, from the single root directory, directories and subdirectories branch off like tree limbs.

Naming Directories

Directories are named just like files, and they can contain upper- and lowercase letters, numbers, and characters such as -, ., and _. The slash (/) character is used to show files or directories within other directories.

The Home Directory

Linux provides each user with his or her own directory, called the home directory. Within this home directory, users can store their own files and create subdirectories. Users generally have complete control over what's found in their home directories. Because there are usually no Linux system files or files belonging to other users in your home directory, you can create, name, move, and delete files and directories as you see fit. The location of a user's home directory is specified by Linux and can't be changed by the user. This is both to keep things tidy and to preserve system security.

Important Directories in the Linux File System

Most of the directories that hold Linux system files are "standard." Other UNIX systems will have identical directories with similar contents. This section summarizes some of the more important directories on a Linux system.

/

This is the root directory. It holds the actual Linux program, as well as subdirectories. Do not clutter this directory with your files!

/home

This directory holds users' home directories. In other UNIX systems, this can be the /usr or /u directory.

/bin

This directory holds many of the basic Linux programs. bin stands for binaries, files that are executable and that hold text only computers could understand.

/usr

This directory holds many other user-oriented directories. Some of the most important are described in the following sections. Other directories found in /usr include

docs	Various documents, including useful Linux information
man	man The man pages accessed by typing man <command>
games	games The fun stuff!

/usr/bin

This directory holds user-oriented Linux programs.

/var/spool

This directory has several subdirectories. Mail holds mail files, spool holds files to be printed, and uucp holds files copied between Linux machines.

/dev

Linux treats everything as a file! The /dev directory holds devices. These are special files that serve as gateways to physical computer components. For instance, if you copy to /dev/fd0, you're actually sending data to the system's floppy disk. Your terminal is one of the /dev/tty files. Partitions on the hard drive are of the form /dev/hd0. Even the system's memory is a device!

A famous device is /dev/null. This is sometimes called the bit bucket. All information sent to /dev/null vanishes—it's thrown into the trash.

/usr/sbin

This directory holds system administration files. If you do an ls -l, you see that you must be the owner, root, to run these commands.

/sbin

This directory holds system files that are usually run automatically by the Linux system.

/etc

This directory and its subdirectories hold many of the Linux configuration files. These files are usually text, and they can be edited to change the system's configuration (if you know what you're doing!).

Creating Files

Linux has many ways to create and delete files. In fact, some of the ways are so easy to perform that you have to be careful not to accidentally overwrite or erase files!

Return to your home directory by typing cd. Make sure you're in your /home/<user> directory by running pwd. A file can be created by typing ls -l /bin > test. Remember, the > symbol means "redirect all output to the following filename." Note that the file test didn't exist before you typed this command. When you redirect to a file, Linux automatically creates the file if it doesn't already exist. What if you want to type text into a file, rather than some command's output? The quick way is to use the command cat.

The cat Command

The cat command is one of the simplest, yet most useful, commands in Linux. The cat command basically takes all its input and outputs it. By default, cat takes its input from the keyboard and outputs it to the screen. Type cat at the command line:

```
$ cat
```

The cursor moves down to the next line, but nothing else seems to happen. Now cat is waiting for some input:

```
Hello
```

Everything you type is repeated on-screen as soon as you press Enter.So how do you use cat to create a file? Simple! You redirect the output from cat to the desired filename:

```
$ cat > newfile

Hello world

Here's some text
```

Type as much as you want. When you are finished, press ^D by itself on a line; you will be back at the Linux prompt. Now you want to look at the contents of newfile. You could use the more or less commands, but instead, let's use cat. Yes, you can use cat to look at files simply by providing it with a filename:

```
$ cat newfile

Hello world

Here's some text
```

You can also add to the end of the file by using >>. Whenever you use >>, whether with cat or any other command, the output is always appended to the specified file. (Note that the ^D character does not appear on-screen)

```
$ cat >> newfile Some

more lines

$ cat newfile

Hello world
```

```
Here's some text

Some more lines
```

Now, try this:

```
$ cat newfile anotherfile> thirdfile

$ cat thirdfile

Hello world

Here's some text

Some more lines

Different text
```

cat stands for concatenate; cat takes all the specified inputs and regurgitates them in a single lump. This by itself would not be very interesting, but combine it with the forms of input and output redirection available in Linux and you have a powerful and useful tool.

Moving and Copying Files

You often need to move or copy files. The mv command moves files, and the cp command copies files. The mv command is much more efficient than the cp command. When you use mv, the file's contents are not moved at all; rather, Linux makes a note that the file is to be found elsewhere within the file system's structure of directories.

When you use cp, you are actually making a second physical copy of your file and placing it on your disk. This can be slower (although for small files, you won't notice any difference), and it causes a bit more wear and tear on your computer. Don't make copies of files when all you really want to do is move them! The syntax for the two commands is similar:

```
mv <source> <destination>

cp <source> <destination>
```

In the Linux environment renaming a file is just a special case of moving a file. To move a file to /tmp use

this:

```
$ mv fileone /tmp

$ mv fileone /tmp/newfilename
```

By using the above a file can be renamed. Simply move a file from its existing name to a new name in the same directory:

```
$ mv fileone newfilename
```

Because the mv command can accept more than two arguments so more than one file can be moved. To move all files in the current directory with the extension .bak, .tmp, .old to /tmp use this:

```
$ mv *.bak *.tmp *.old /tmp
```

The cp command found at /bin/cp is used for copying and provides a powerful tool for copy operations.

The most basic uses for cp command are to copy a file from one place to another or to make a duplicate file in the same directory. For instance, to copy a file this file in the current directory to a second file to be called this file-copy in the same directory enter the following command:

```
$ cp thisfile thisfile-copy
```

Using ls –1 to look at the directory listing of the files, you would find two files with identical sizes but different date stamps. The new file has a date stamp indicating when the copy operation took place: it is a new, separate file. Changes to thisfile-copy do not affect the original this file file. Similarly to make a copy of this file in the /tmp directory use the following command:

```
$ cp thisfile /tmp
```

and if you want to copy this file to /tmp but give the new file a different name, enter

```
$ cp thisfile /tmp/newfilename
```

Also to avoid overwriting a file accidentally use the –i flag of the cp command which forces the system to confirm any file it will overwrite when copying. Then a prompt like the following appears:

```
$ cp –i thisfile newfile

cp: overwrite thisfile?
```

Copying Multiple Files in One Command

In DOS only one file or file expression can be copied at a time. To copy three separate files then three commands must be issued. The Linux cp command makes this a bit easier. The cp command can take more than two arguments. If more than two arguments are passed to the command then the last one is treated as the destination and all preceding files are copied to this destination.

For example to copy fileone ,filetwo and filethree in the current directory to /tmp then the following

commands can be issued:

```
$ cp fileone /tmp

$ cp filetwo /tmp

$ cp filethree /tmp
```

All this can be bundled into one command like this:

```
$ cp fileone filetwo filethree /tmp
```

Similarly wildcards can be used to mix and copy a large number of files in one command. For instance, this command copies all files with any one of the three extensions in one command.

```
$ cp *.txt *.doc *.bak /tmp
```

When copying multiple files in this way it is important to remember that the last argument must be a directory, since it is impossible to copy two or more files into a single file.

To copy an entire directory and all its subdirectories the –R flag of the cp command is used. This command indicates that a directory is to be recursively copied. For example if a subdirectory called "somedir" exists in the current directory and is to be copied to /tmp then the following command can be used:

```
$ cp –R somedir /tmp
```

Creating a Directory

To create a new directory, use the mkdir command. The syntax is mkdir <name>, where <name> is replaced by whatever you want the directory to be called. This creates a subdirectory with the specified name in your current directory:

```
$ ls anotherfile newfile thirdfile

$ mkdir newdir
```

Moving Directories

To move a directory, use the mv command. The syntax is mv <directory> <destination>. In the following example, you would move the newdir subdirectory found in your current directory to the /tmp directory:

```
$ mv newdir /tmp

$ cd /tmp

$ ls

/newdir
```

The directory newdir is now a subdirectory of /tmp.

Note: When you move a directory, all its files and subdirectories go with it.

Removing Files and Directories

Now that you know how to create files and directories, it's time to learn how to undo your handiwork.

Removing Files

To remove (or delete) a file, use the rm command found at /bin/rm. (rm is a very terse spelling of remove).

The syntax is rm <filename>. For instance:

`$ rm myfile` removes the file myfile from your current directory.

`$ rm /tmp/myfile` removes the file myfile from the /tmp directory.

`$ rm *` removes all files from your current directory. (Be careful when using wildcards!)

`$ rm /tmp/*`files removes all files ending in "files" from the /tmp directory.

Note: As soon as a file is removed, it is gone! Always think about what you're doing before you remove a file. You can use one of the following techniques to keep out of trouble when using wildcards.

Run ls using the same file specification you use with the rm command. For instance:

$ ls *files

myfiles newfiles samefiles

$ rm *files

In this case, you thought you wanted to remove all files that matched *files. To verify that this indeed was the case, you listed all the *files (wildcards work the same way with all commands). The listing looked okay, so you went ahead and removed the files.

Use the i (interactive) option with rm:

```
$ rm -i *files

rm: remove 'myfiles'? y

rm: remove 'newfiles'? n

rm: remove 'samefiles'? y
```

Note: When you use rm -i, the command goes through the list of files to be deleted one by one, prompting you for the OK to remove the file. If you type y or Y, rm removes the file. If you type any other character, rm does not remove it. The only disadvantage of using this interactive mode is that it can be very tedious when the list of files to be removed is long.

Removing Directories

The command normally used to remove (delete) directories is rmdir. The syntax is rmdir <directory>.

Before you can remove a directory, it must be empty (the directory can't hold any files or subdirectories).

Otherwise, you see

```
rmdir: <directory>: Directory not empty
```

This is as close to a safety feature as you will see in Linux! This one might mystify you (in your home directory)

```
$ rmdir zippy

rmdir: zippy: Directory not empty
```

You will most often come across this situation in a system administrator role. Sometimes you want to remove a directory with many layers of subdirectories. Emptying and then deleting all the subdirectories one by one would be very tedious. Linux offers a way to remove a directory and all the files and subdirectories it contains in one easy step. This is the r (recursive) option of the rm command.

The syntax is rm -r <directory>. The directory and all its contents are removed.

Note: You should use rm -r only when you really have to.

File Permissions and Ownership

All Linux files and directories have ownership and permissions. You can change permissions, and sometimes ownership, to provide greater or lesser access to your files and directories. File permissions also determine whether a file can be executed as a command. If you type ls -l or dir, you see entries that look like this:

```
-rw-r—r— 1 fido users 163 Dec 7 14:31 myfile
```

The -rw-r—r— represents the permissions for the file myfile. The file's ownership includes fido as the owner and users as the group.

File and Directory Ownership

When you create a file, you are that file's owner. Being the file's owner gives you the privilege of changing the file's permissions or ownership. Of course, once you change the ownership to another user, you can't change the ownership or permissions anymore! File owners are set up by the system during installation. Linux system files are owned by IDs such as root, uucp, and bin. Do not change the ownership of these files.

The chown command

Use the chown (change ownership) command to change ownership of a file. The syntax is chown <owner> <filename>. In the following example, you change the ownership of the file myfile to root:

```
$ ls -l myfile

$ chown root myfile

$ ls -l myfile
```

To make any further changes to the file myfile, or to chown it back to fido, you must use su or log in as root.

The chgrp command

Files (and users) also belong to groups. Groups are a convenient way of providing access to files for more than one user but not to every user on the system. For instance, users working on a special project could all belong to the group project. Files used by the whole group would also belong to the group project, giving those users special access. Groups normally are used in larger installations. You may never need to worry about groups.

The chgrp command is used to change the group the file belongs to. It works just like chown. the syntax of this command is as follows: chgrp [-Rcfv] [--recursive] [--changes] [-silent] [--quiet] [--verbose] group filename…

File Permissions

Linux lets you specify read, write, and execute permissions for each of the following: the owner, the group, and "others" (everyone else).

read permission enables you to look at the file. In the case of a directory, it lets you list the directory's contents using ls.

write permission enables you to modify (or delete!) the file. In the case of a directory, you must have write permission in order to create, move, or delete files in that directory.

execute permission enables you to execute the file by typing its name. With directories, execute permission enables you to cd into them.

For a concrete example, let's look at myfile again:

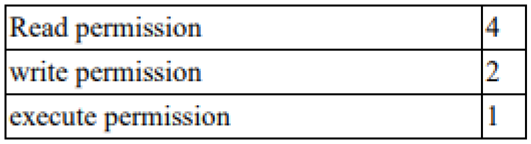
```
-rw-r—r— 1 fido users 163 Dec 7 14:31 myfile
```

The first character of the permissions is -, which indicates that it's an ordinary file. If this were a directory, the first character would be d.

The next nine characters are broken into three groups of three, giving permissions for owner, group, and other. Each triplet gives read, write, and execute permissions, always in that order. Permission to read is signified by an r in the first position, permission to write is shown by a w in the second position, and permission to execute is shown by an x in the third position. If the particular permission is absent, its space

is filled by -.

In the case of myfile, the owner has rw-, which means read and write permissions. This file can't be executed by typing myfile at the Linux prompt.



Read permission	4
write permission	2
execute permission	1

The group permissions are r—, which means that members of the group "users" (by default, all ordinary users on the system) can read the file but not change it or execute it. Likewise, the permissions for all others are r—: read-only.

File permissions are often given as a three-digit number—for instance, 751. It's important to understand how the numbering system works, because these numbers are used to change a file's permissions. Also, error messages that involve permissions use these numbers. The first digit codes permissions for the owner, the second digit codes permissions for the group, and the third digit codes permissions for other (everyone else).

The individual digits are encoded by summing up all the "allowed" permissions for that particular user as

follows:

Therefore, a file permission of 751 means that the owner has read, write, and execute permission (4+2+1=7), the group has read and execute permission (4+1=5), and others have execute permission (1). If you play with the numbers, you quickly see that the permission digits can range between 0 and 7, and that for each digit in that range there's only one possible combination of read, write, and execute permissions.

Note: If you're familiar with the binary system, think of rwx as a three-digit binary number. If permission is allowed, the corresponding digit is 1. If permission is denied, the digit is 0. So r-x would be the binary number 101, which is 4+0+1, or 5. —x would be 001, which is 0+0+1, which is 1, and so on. The following combinations are possible:

0 or —-: No permissions at all

4 or r—: read-only

2 or -w-: write-only (rare)

1 or —x: execute

6 or rw-: read and write

5 or r-x: read and execute

3 or -wx: write and execute (rare)

7 or rwx: read, write, and execute

Changing File Permissions

To change file permissions, use the chmod (change [file] mode) command. The syntax is chmod

<specification> file.

There are two ways to write the permission specification. One is by using the numeric coding system for permissions:

Suppose that you are in the home directory.

```
$ ls -l myfile

-rw-r—r— 1 fido users 114 Dec 7 14:31 myfile

$ chmod 345 myfile
```

You can also use letter codes to change the existing permissions. To specify which of the permissions to change, type u (user), g (group), o (other), or a (all). This is followed by a + to add permissions or a - to remove them. This in turn is followed by the permissions to be added or removed. For example, to add execute permissions for the group and others, you would type:

```
$ chmod go+r myfile
```

Changing Directory Permissions

You change directory permissions with chmod, exactly the same way as with files. Remember that if a directory doesn't have execute permissions, you can't cd to it.

EXERCISE

1. What do the following do (assuming ch1 and ch2 contains some text)?

cat ch1 ch2 ch3 > “your-practical-group”

2. Differentiate between cp and mv command?

3. What is the difference between the permissions 777 and 775 of the chmod command?

4. Discuss the Linux Directory Structure (File System Hierarchy). Also Draw directory Tree structure.