File and Directory Management

Viewing Files and Directories: The ls command

To view a list of the filenames and subdirectories within a directory, we use the command ls (the name of the command stands for `list`). For instance, if we type

```
% ls /
```

at the prompt in the xterm, then Unix will display in the xterm window a list of all files and directories in the root directory. The contents of this directory will be listed in a form similar to the following example:

```
export    prevost1  ugrad0     home        prevost2
ugrad1    kernel    prevost3   ugrad2
```

When no argument is supplied to the ls command, it simply displays the contents of the current working directory. Thus, if we type the following in our home directory

```
% ls
```

then Unix might display the following list of files and directories:

```
Mail       Windows    cs250      hello.c    program
News       bin        cs300      home.gif
```

It is difficult, if not impossible, to discern from this output which of the names are associated with files, executable programs, or directories. To differentiate between these types of files, we can use the -F option:

```
% ls -F
Mail/       Windows/    cs250/      hello.c     program*
News/       bin/        cs300/      home.gif
```

As shown in this example, a slash / is appended to the end of each directory name. Also, an asterisk * character is added to the end of the names of the files that contain programs that can be run (such as the file program in the example). Such an asterix will therefore appear at the end of the name of every executable file that we create by compiling a C++ program. The * is not part of the filename.

Some file names have a dot . in front of them. These files are called hidden files or dot files. The ls command displays the dot files in a directory if the -a option is used. For instance:

```
% ls -aF
./          .emacs      News/       cs250/      home.gif
../         .newsrc     Windows/    cs300/      program*
.accept     Mail/       bin/        hello.c
```

The files .emacs, .newsrc, and .accept are all displayed. Note how we used more than one option at the same time.

As mentioned before, the files called . and .. are two special directories. The single dot represents the current working directory, while the double dots represent its parent directory. For example, if we execute this command in our home directory:

```
% ls -F ..
```

then Unix will list the files in the parent directory of our home directory, and produce an output similar to:

```
OLD/        NEW/        DORMANT/    TT_DB/      a1a1/
a1a2/       a1a3/       a1a4/       b1b1/       b1b2/
b1b3/       b1b4/       c1c1/       c1c2/       c1c3/
c1c4/
```

The actual output on the Undergraduate servers at SFU would be much longer. As can be seen from this listing, the parent directory mostly contains the home directories for undergraduate user accounts.

In the remainder of these notes, we will assume that ls is always given the -aF options, even if they are not written explicitly.

To get more information about files and directories, such as their sizes, ownerships and when they were last changed, we use the -l option. For instance, if user *a1a1* typed the following in the xterm window:

```
% ls -l
```

then the following information might be displayed:

```
-rw-r--r--  4 a1a1 guest    9512   May  10   15:21   banner.gif
drwx------  1 a1a1 guest     512   May  13   09:59   document
-rwx------  1 a1a1 guest 5275326   May  27   15:53   emacs
-rw-------  5 a1a1 guest   11512   May  28   15:28   hello.c
-rw-r--r--  4 a1a1 guest     512   May  19   15:27   home.html
drwx------  1 a1a1 guest   25194   Jun  16   09:59   temp
```

The meaning of the information contained in each column is the following (column 1 is the leftmost column):

- Column 1: The first character is either d indicating a directory, or - indicating a file. Next we have access permissions such as -rw-r--r--. These specify which users have permission to read (r), write (w) or execute (x) the file.

- Column 2: The number of hard links to the file.

- Column 3: The owner of the file (e.g. a1a1).
- Column 4: The name of the group the file belongs to (groups are collections of users, e.g. guest).
- Column 5: The size of the file in characters (bytes).
- Column 6: The date and time of the most recent change to the contents of the file (e.g. May 10 15:21).
- Column 7: The name of the file or directory (e.g. banner.gif or document).

Navigating Directories: The cd Command

When we start a Unix session, the current working directory is initially our home directory. We can use the pwd command (`print working directory`) to display the path of the current working directory. For example, user *a1a1* is initially placed in the home directory /home/a1a1. To display the name of the working directory, user *a1a1* would type:

```
% pwd
```

and Unix would display:

```
/home/a1a1/
```

By displaying the current working directory, we can verify whether or not we are in the correct directory. To get out of the current working directory and go into another directory, we use the cd command. This command has the following form:

```
% cd directory_name
```

where directory_name is the name or the full path of the desired directory. Suppose that we are in our home directory, and that executing ls gives us:

```
./          .emacs      News/       cs250/      home.gif
../         .newsrc     Windows/    cs300/      program*
.accept     Mail/       bin/        hello.c
```

To go to the cs300 directory, we would type:

```
% cd cs300
```

and then the current working directory would become /home/a1a1/cs300.

On the other hand, if user *c3po* is in its home directory and wants to go to a directory called Lab1, which is contained in subdirectory cs300, then the command:

```
% cd Lab1
```

would produce the error message:

```
Lab1: No such file or directory
```

Unix cannot find directory Lab1 because it is contained inside the subdirectory cs300. To get to this directory, we would need to use the command

```
% cd cs300/Lab1
```

or

```
% cd ~/cs300/Lab1
```

where the tilde ~ is a special path that refers to the current user's home directory, as mentioned before.

Recall that if we do not give a full path name, then cd treats the name as being relative to the current directory. So the command

```
cd ../r2d2
```

would change the current working directory to /home/r2d2.

Creating and Removing Directories

The mkdir Command

When we start a new project, we might want to create a new directory that will contain the files for that project. To create a new directory, we use the mkdir (ma ke `directory`) command. The directory will be created as a subdirectory of the current working directory. To create the directory temporary inside our home directory, we would type

```
% mkdir temporary
```

while we are in our home directory. If we wanted our new directory to be a subdirectory of a directory other then our current directory, then we could either

- Go to the directory that we want the new directory to be created in, and create the new directory there, or
- Specify the full path name for the new directory.

For example, if we type:

```
% mkdir ~/cs300/temporary
```

then Unix will create a new directory called temporary within the directory /home/user-id/cs219.

The rmdir Command

To remove a directory, we use the rmdir (re move directory) command. The command name is followed by the name of the target directory. For example,

```
% rmdir ~/cs300
```

will remove the cs300 directory located in the home directory. If the target directory contains files or subdirectories, rmdir will display the error message:

```
Directory not empty
```

Sometimes rmdir will report a more cryptic error, such as:

```
rmdir: temporary: File exists
```

We must delete all the files and subdirectories in the target directory before removing the directory. The remove command, rm, is used to remove files and is discussed in the section on Removing Files.

As usual, to remove a directory that is not contained in the current directory, we must specify its full path name.

Copying, Moving and Removing Files

Unix allows us to copy, move, rename, and remove files. We can make a copy of a document before experimenting with format changes. We can move files into or out of a directory, and rename files to better organize them. When a file is no longer needed, we can remove it.

The cp Command

To duplicate a file, we use the cp (co py) command. The format of this command is:

```
% cp source_filename destination_filename
```

where source_filename is the name of the original file and destination_filename is the name of the resulting copy. For example, if we type:

```
% cp hello.c example.c
```

while in our home directory, then Unix will make a copy of the file hello.c. The new file will be named example.c.

To copy files from a directory not contained in the current directory, we need to specify their full path. Similarly, if the copy of the file should not be stored in the current directory, then we must also specify its full path. For example, if we type:

```
% cp ~/cs300/Lab1/command.txt ~/cs300/Lab9/document.txt
```

then Unix will copy the file command.txt in directory ~/cs300/Lab1 to a new file named document.txt in the ~/cs300/Lab9 directory.

If destination_filename is a directory, then the copy of source_filename is stored in that directory, and has the same name as the original file. For example, by typing:

```
% cp ~/cs300/Lab1/command.txt ~/cs300/Lab9
```

we would tell Unix to copy the file command.txt to the Lab9 directory, and name the copy command.txt.

The mv Command

To move a file to a different location, we use the mv (mo ve) command. The format of the command is:

```
% mv source_filename destination_filename
```

where source_filename is the name of the original file and destination_filename is the name of the file or directory to move the source file to. For example, if we type:

```
% mv home.gif hello.c
```

in our home directory, then Unix will move, or rename, the file home.gif to the file hello.c. After moving, the original contents of the file hello.c no longer exists. The mv command overwrites the file that we are moving to if that file exists. Therefore, use the mv command carefully.

The destination_filename can be a directory. For example, if we type:

```
% mv home.gif cs300
```

then Unix will move the file home.gif to the directory cs300.

As for copying files, we need to use full path names as arguments in order to move a file from a directory not contained in the current directory, or to move a file to a directory that is not contained in the current directory.

The rm Command

To remove (delete) a file, we use the rm (re move) command. This command has the following form:

```
% rm filename [filename]
```

where each filename is the name of a file we want to remove. The rm command is very powerful and permanent; use it with extreme caution. There is no way to recover a removed file in a Unix system.

If we type:

```
% rm newHome.gif hello.c
```

while in our home directory, then Unix will remove both files. Unix does not usually request a confirmation, but we can use the -i option to require an interactive confirmation before a removal occurs. For example, if we type:

```
% rm -i newHome.gif hello.c
```

then Unix will request us to confirm the deletions:

```
rm: remove `newHome.gif`?
```

At that point, we should press the [y] key to remove the file. Any other key will cancel the removal.

The rm command can also be used to remove a directory and the files and subdirectories it contains, all in one step. This is done with the -r option. This is often used to bypass the rmdir requirement that a directory be empty before it can be removed. If we type:

```
% rm -r cs300
```

while in our home directory, Unix will attempt to remove the cs300 directory and all of its contents. The user must have the proper permissions to delete the affected files and directories. Confirmation may be asked of the user in the case of deleting a non-regular file.