Basic UNIX Commands

Created by : M.S. Schmalz (Thank you Mark!)

This section reviews UNIX file, directory, and system commands, and is organized as follows:

Information in this section was compiled from a variety of text- and Web-based sources, and is not to be used for any commercial purpose.

UNIX implementations employ a *shell* that accepts user commands and invokes the appropriate Operating System (OS) processes. These OS routines can act on files, some of which can be executable, others of which are data or documentation files. Additionally, UNIX provides the `man` utility to help users view documentation about a given command or group of commands. UNIX file, directory, and basic system commands enable one to (a) navigate through a UNIX-based computer system and (b) locate, retrieve, modify, or store information organized in files or hierarchical structures called directories.

3.1. The UNIX Command Line Interface

UNIX users run one of a collection of programs, the most frequently used of which is called a "shell". Numerous shells have been developed over the years (e.g., Korn shell, C shell, etc.)

When you open a shell, you will see a prompt on the screen. The prompt can be the name of your computer followed by a special character, like "%" or ">". The whole prompt might look like this:

Example of UNIX Prompt: `hausdorff%` ,

which would mean that (a) you are using a computer called "hausdorff", and (b) the shell is waiting for your keyboard input (signified by "%"). Since prompt formats vary widely, your shell's prompt might be different from the example shown above.

Convention. Throughout these course notes, the symbol "%" indicates the end of the prompt, the symbol after which you type a command or statement into the C-shell.

At the prompt, if you type the name of a program to be run followed by the *Enter* key, then the shell will run the program. When the program is done, you are sent back to the shell. Many programs that the shell runs are similar to traditional operating system commands (e.g., the directory listing command `DIR` in MS-DOS). To get a directory listing in UNIX, type "ls" and press to list the contents of your directory.

Example. If you want to invoke a UNIX directory listing, the system prompt and the string you type (emboldened) would appear as follows:

```
% ls
```

Many UNIX programs or commands have *options* that change the command's functionality. In the UNIX command line, options are usually preceded by a dash.

Example. Detailed information about files in the current directory can be obtained by typing `ls -l`, i.e.,

Detailed File Information: `% ls -l`

Here, the "-" precedes an option ("l", which stands for *long*).

Screen output from the `ls -l` command should look similar to the following. In an actual UNIX session, a user's login-name would be substituted for "bsimpson":

```
total 6
-rw-r--r--  1 bsimpson      123 Jun 23  1998 -i
drwxr-S---  2 bsimpson      512 Apr  7 12:31 Articles/
drwxr-s---  1 bsimpson      512 May  2 14:01 News/
drwxr-s---  4 bsimpson     1024 Apr 11 14:17 programs/
-rw-------  1 bsimpson     1891 May  4 11:56 dead.letter
-rw-------  1 bsimpson    25247 May  4 11:33 mbox
```

In Section 3.3, we will describe in detail the meaning of the preceding display, as well as the function of UNIX file commands. Now that you know what the command line is and how to use it, we will digress briefly to discuss the UNIX file system.

3.2. Overview of UNIX File System

When a disk is formatted, the physical disk is divided into a number of partitions, which are abstractions, each of which has an associated file system. The file system consists of hierarchically-arranged subdirectories and files that can be conveniently represented by an abstraction called a *directory tree*. A directory tree that ***might*** be found in a UNIX file system is exemplified in Figure 3.2.1.

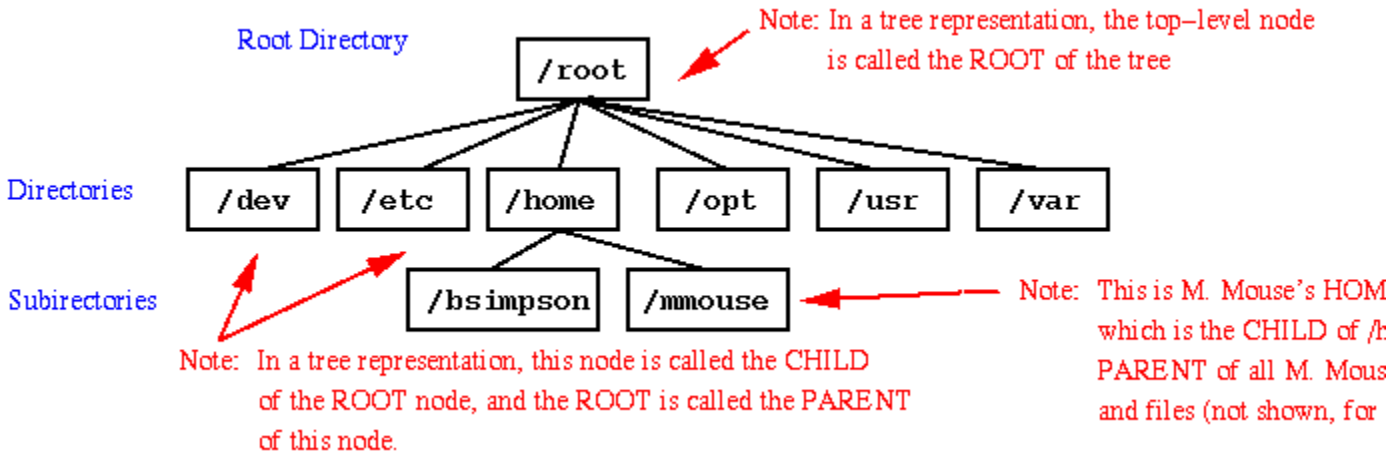

Figure 3.2.1. Tree representation of an example UNIX file system.

3.2.1. UNIX System Directories

Several standard directories typically appear in a UNIX filesystem, as follows:

`dev` directory contains special device files that are used to drive hardware objects such as CD-ROM, floppy disk, etc. These files will be discussed in a later portion of the course, if time permits.

`etc` directory files are required for operation of the specific machine or system that you are using (also called *machine-dependent* or *system-dependent* files

`home` directory houses a collection of user directories on a given system. This is where your personal files would be kept if you had an account on a UNIX system such as *grove*.

`opt` directory typically contains application-specific subdirectories and files (e.g., programs such as FrameMaker or a public domain package such as the image editor `xv` or graphing package `xvgr`).

`usr` directory contains files that can be shared by all users (recall data and program sharing introduced in the MULTICS operating system)

`var` directory is comprised of files whose size varies with time, such as incoming mail and spooler files (e.g., print spool files for printout).

3.2.2. UNIX File Description

A Unix file is specified by a parameter block called an **i-node**. An *i-node* exists on disk for every file that is on that disk, and there exists a copy of the i-node in kernel memory for every open file. All of the information that UNIX knows about a file, except the file name, is stored in the i-node, which includes:

File access and file type, known as the *mode*.

File ownership information, which is important for security

Time stamps that record date and time of last modification, last access, and last modification of mode

… and much much more!

3.3. UNIX Documentation via `man` command

The majority of operating systems have an on-line help facility. UNIX expresses its help information in terms of *man pages*, which are supposed to resemble pages from a software manual.

In UNIX, the `man` command displays information from specially-formatted reference manuals. These are useful, reasonably complete manual pages that the user can select by (a) command name, or (b) one-line summaries selected either by keyword (`-k` option), or by the name of an associated file (`-f` option). If no manual page can be found for a given command, option, or keyword, then the `man` program prints an error message.

The `man` command is invoked using the following syntax:

```
machine% man {keyword}
```

For example, on the SunOS system, entering `man exit` after the command line prompt yields the following information for the *exit* command:

```
Fortran Library Routines                                EXIT(3F)

NAME
     exit - terminate process with status

SYNOPSIS
     subroutine exit (status)
     integer*4 status

DESCRIPTION
     exit  flushes  and  closes all the files of the process, and
     notifies the parent process if it is executing a wait.   The
     low-order  8 bits of status are available to the parent pro-
     cess.  These 8 bits are shifted left 8 bits, and  all  other
     bits  are zero.  Therefore, status should be in the range of
     256 - 65280.

     This call never returns.

     The C function exit may cause  cleanup  actions  before  the
     final `sys exit'.

     If you call exit without an argument, you get a warning mes-
     sage, and a zero is automatically provided as an argument.

FILES
     libF77.a

SEE ALSO
     exit(2), fork(2), fork(3f), wait(2), wait(3f)

SunOS 5.6             Last change: 98/09/16                     1
```

This documentation has several categories, as follows:

NAME - Gives the name of the command or programming language keyword, together with a short description of functionality

SYNOPSIS - Expanded description of functionality with respect to the string that follows *NAME*

DESCRIPTION - Fully describes the command or programming language keyword, including options and functionality in detail, often with examples

FILES - Shows what system files are associated with the command or programming language keyword

SEE ALSO - Provides other commands or programming language keywords that can be used as arguments to related *man* calls.

Additionally, at the bottom of the man page is a date of most recent modification and a version number (in the above example, *SunOS 5.6*) that tells you how recent the help or documentation is. The man command further supports the *bugs* category, which tells the user or system analyst what known problems are associated with the command or programming language keyword. In a simple command like *exit* listed above, there are few if any known bugs. However, other commands might have bugs associated with them, as do public-domain programs that can be run from UNIX.

3.4. File Commands

The UNIX file system supports viewing of directories and file contents, moving and copying of files, renaming files and setting permissions, and other similar tasks. These operations are standard on modern computer file systems.

3.4.1. Viewing Directory File Information

Recall the example output of the `ls -l` command given in Section 3.1:

```
total 6
-rw-r--r--  1 bsimpson      123 Jun 23  1991 -i
drwxr-S---  2 bsimpson      512 Apr  7 12:31 Articles/
drwxr-s---  1 bsimpson      512 May  2 14:01 News/
drwxr-s---  4 bsimpson     1024 Apr 11 14:17 programs/
-rw-------  1 bsimpson     1891 May  4 11:56 dead.letter
-rw-------  1 bsimpson    25247 May  4 11:33 mbox
```

The first column (e.g. *-rw-r--r--*), pertains to file permissions, namely, who has access to your files and what they can do with them (r = "read", w = "write", etc.).

The name (*bsimpson*) is the login-i.d. of the person who owns the files (should be you). The number after that is how many bytes the file uses, followed by the creation date/time and finally the file name. (Some systems have different formats for `ls` output, so check with your consultant or system administrator if you have questions.)

Example. In the preceding directory listing, file *mbox* is readable and writeable by *bsimpson*, occupies 25,247 bytes on disk, and was created May 4 of the current year at 11:33am.

Another option for the `ls` command is `-a`, where *a* denotes *all*. Certain files in your directory are usally invisible to the `ls` or `ls -l` command. (A file is invisible if its name begins with ".") These *hidden files* are usually system-oriented files that are transparent to a user's day-to-day operations. If you had to look at these every day, they would probably clutter up your directory listing. By typing

% ls -a

you can obtain a listing of all your regular files, plus several files beginning with ".", for example, ., .., `.cshrc`, `.login`, `.newsrc` and several others.

One can usually combine options on the `ls` command. For example, to detailed information on all of your files can be produced by typing `ls -a -l` or `ls -l -a`.

Efficient Usage Tip. With some commands (like `ls`) you can put several options after just one dash, for example,

`ls -al` or `ls -la`.

3.4.2. Moving Files

The command `mv`, which means *move*, allows a user to move or rename files. For safety, `mv` should be used with the `-i` option, which asks you if you really want to overwrite a file. You are also prompted if you try to rename a file to a name that exists in the current directory. New users have the `-i` option set by default. The following discussion explains two different forms of `mv` that you can use.

mv *file1 file2*

This form of `mv` changes the name of a file (from *file1* to *file2*). This command also applies to directories. For example, to rename a directory, use the directory name instead of one or more filenames.

mv *file1 ... fileN directory*

This form of the `mv` command will move one or more files (separated by spaces) into the designated directory, which is the last argument on the command line. Note that *file1* ... *fileN* can be either file or directory names.

Example. Given the following UNIX session:

```
% ls

a.out*   cop3610/   emg3312/   private/   typescript

% mv a.out typescript

remove typescript? n

% ls

a.out*   cop3610/   emg3312/   private/   typescript

% mv a.out bogus

% ls

bogus*   cop3610/   emg3312/   private/   typescript

% mv bogus typescript cop3610

% ls

cop3610/   emg3312/   private/
```

```
% ls cop3610


bogus*   typescript
```

The first `mv` attempted to rename `a.out` to `` `typescript `` (a file which already exists). Since the `-i` option is used by default, the question `remove typescript?` was returned because the file `` `typescript `` already exists. By answering `y`, the target file `typescript` would have been overwritten by the file `a.out`. Answering `n`simply cancels the move process.

In the second use of `mv`, `a.out` was renamed to `bogus`.

The third `mv` command moved the `bogus` and `typescript` files into the `cop3610` directory.

3.4.3. Copying Files

To copy a file, use `cp`, which means *copy*. The `cp` command has two formats that are analogous to the `mv` command formats. The only difference between the these commands is that `cp` doesn't remove the original file. Thus we say that *renaming a file is a destructive form of copying*, because the source file is removed.

Similar to `mv`, `cp` also offers a `-i` option. New users have this option set by default. Again, note that `cp` and `mv` are almost identical in usage.

3.4.4. Removing Files

The command `rm` means *remove* and is used to erase files. For safety, this command should almost always be used with the interactive option `-i`. Thus, whenever you remove a file you will be asked to verify the removal of the file. New users have `rm` initialized to use the `-i` option by default.

Example.

```
% ls
```

```
a.out*   cop3610/   emg3312/   private/   tempfile   typescript

% rm -i tempfile

rm: remove tempfile? y

% ls

a.out*   cop3610/   emg3312/   private/   typescript
```

In this example, the file `tempfile` was removed. Note that `rm` asked if `tempfile` should really be removed (the corresponding prompt is emboldened).

Note: When you remove a file with `rm`, the file is no longer available *from the user's perspective* - you will **not** be able to get it back unless you request a tape backup dump from the systems staff. **Most UNIX systems do not have an *undelete* command** .

3.4.5. Changing File Permissions

On a multi-user system, keeping selected files out of the reach of prying eyes is important. As a (relatively weak) security measure, the Unix operating system has built-in *file permissions* feature.

By typing `ls -l`, we have seen that a long listing of the files in the current directory can be displayed. An example is displayed below. As noted previously, the letters and dashes on the left side of the listing represent the permissions set on each file or directory.

Example.

```
% ls -l

drwxr-xr-x 11 bsimpson        572 Nov 16 05:11 drafts/
```

```
-rw------- 1  bsimpson          1 Dec 10 20:04 termpaper
```

The letter in the first column describes the type of the file, while the other nine letters describe the file's permissions, which indicate who can access the file and how it can be accessed.

In order to understand the concept of file security at a basic level, the following permission codes are listed:

- : permission is not set

r : read permission is set

w : write permission is set

x : execute permission is set

d : the file represents a directory

The nine permission characters are partitioned into three sets of three characters each, where each set of three characters contains `r`, `w` and `x` codes. The three partitions comprise an *access control list*, and are described as follows:

user - Permissions in the first left-hand group of three characters control the user's access to the file. If the current user is not the owner of the file, then the user might not be able to access that file, depending on how the permissions are set. To find the owner of a file, use `ls -l`, and look at the username (e.g., `bsimpson` in the preceding example).

group - A collection of users can be formally aggregated in a *group*, which is a list of permissible usernames. For example, a group *cop3610* could contain the usernames of all students enrolled for this course in a given semester. User permissions thus control the access that people in the group(s) assigned to the file have. You can use the command `ls -lg` to list all groups associated with each file.

other - Users in the *other* partition comprise all users not in the `user` or `group` partitions. The *other* permissions control access that users in the rest of the world (who can login to your system) have to each file.

Examples.

```
-rw------- 1 instr cop3610 7830 Nov 19 15:06 hw2
```

Only the owner (*instr*) of the file *hw2* has permission to read and write to the file.

```
-rw-r----- 1 instr cop3610 17820 Nov 19 15:06 hw1
```

This is a file that is readable by both the owner (*instr*) and by users that are in the *cop3610* group. However, only the owner has write permission to the file (`rw` code in the left-hand group of three characters).

```
drwxr-xr-x 6 instr staff 512 Apr 19 22:27 /home/instr/410/
```

This is a directory (as shown by the left-hand `d`) that is readable and executable by everyone, but can be written to only by the owner (*instr*). The *groups* field contains *staff*, which is usually a group of privileged users.

```
-rw-rw-rw- 1 instr cop3610 783 Sep 18 15:06 temp
```

This file (`temp`) is readable and writable by all users.

When a file or directory is created, UNIX sets default file permissions according to the `umask` descriptor in your `.cshrc` file. To change the permissions on a file, the `chmod` command is used, which has the following form:

```
chmod mode file(s)
```

where *mode* specifies the change of permissions on the specified file(s). The *mode* is specified as follows:

Step 1. Choose one or more permission partitions by specifying `u` (user), `g` (group), `o` (other), or `a` (all).

Step 2. Type `+` (add permission) or `-` (delete permission).

Step 3. Specify the permissions to be changed using `r` (read), `w` (write), or `x` (execute).

Here follow three examples of `chmod` usage.

Example 1.

```
% ls -l
```

```
drwxr-xr-x 11 bsimpson         572 Nov 16 05:11 drafts/

-rw------- 1  bsimpson        5666 Dec 10 20:04 termpaper
```

```
% chmod go+r termpaper
```

This use of `chmod` gave *read* access for *group* and *others* to the file `termpaper`, as shown below:

```
% ls -l
```

```
drwxr-xr-x 11 bsimpson         572 Nov 16 05:11 drafts/

-rw-r--r-- 1  bsimpson        5666 Dec 10 20:04 termpaper
```

Example 2. Assume the file status shown at the end of the previous example.

```
% chmod o-rx drafts

% ls -l
```

```
drwxr-x--- 11 bsimpson         572 Nov 16 05:11 drafts/

-rw-r--r-- 1  bsimpson        5666 Dec 10 20:04 termpaper
```

This example removed *read* and *execute* access for *others* from the `drafts` directory.

Example 3. Assume the file status shown at the end of the previous example.

```
% chmod go-rwx termpaper
```

```
% ls -l
```

```
drwxr-x--- 11 bsimpson         572 Nov 16 05:11 drafts/

-rw------- 1  bsimpson        5666 Dec 10 20:04 termpaper
```

In this example, the `chmod` command is directed to remove all permissions for **group** and **others** from `termpaper`.

Note: To make a directory accessible to everyone, one must specify *group* and *others* read and execute permissions for the entire directory. For example:

```
chmod go+rx directory
```

.

3.4.6. Viewing File Contents

We begin with the `cat` command, then progress to the `more` command.

The command `cat` means *concatenate* and is often used to view short files. Supplying `cat` with multiple file names, as follows:

```
cat file1 file2 ... fileN
```

will display each file sequentially in a continuous stream of text. This is why the command is called *concatenate*. If a file is large and you want to use `cat` to view it, you will have to have quick reflexes, and use `{Ctrl}-s` and `{Ctrl}-q` to stop and restart scrolling of text so you can get a chance to view it. On modern computers, scrolling is usually so fast that you will likely lose the text you are trying to see. Thus, we recommend the use of commands such as `more` or `less` to view your files.

The commands `more` and `less` are commonly used to view files one screen at a time. When you use either of these commands, you will have an information bar at the bottom of the screen. For example, you can press the `{spacebar}` to go to the next screen, `b` to go back a page, or the `{Return}` to scroll the file forward a line at a time. When you finally get to the end of the file, `more` will return a Unix prompt, while `less` will wait for you to press `q` to quit. The following list of options will work for both `more` and `less`commands, unless otherwise indicated.

`{spacebar}` - Takes you to the next page.

`b` - Takes you to the previous page.

{Enter} key - Scrolls forward one line.

k - Scrolls backward one line (`less` only).

g - Takes you to the beginning of the file (`less` only).

G - Takes you to the end of the file (`less` only).

h - Shows you a help screen.

/pattern - Goes to the next occurrence of *pattern* in the file. When you finish typing in the pattern, you must press the `{Enter}` key. Here, *pattern* is a *regular expression*, which we will define later in this course.

n - Search forward for another occurrence of the pattern previously searched for with `/`.

r - Search backwards for a previous occurrence of the pattern previously searched for with `/`.

q - Quits the `more` or `less` program.

Either program works, but `less` is much more flexible. Backward scrolling is just just one of many features that `less` has, which `more` does not. We suggest using `less` if you want full-featured file display.

3.4.7. Wildcard Usage

In certain cases, UNIX supports application of a command to multiple files. The command length and complexity can be reduced via *wildcard characters* for efficient matching of filenames. Wildcard characters are:

1. `?` -- matches any *one* character.
2. `*` -- matches any *contiguous group* (string) of zero or more characters.

To better understand the use of wildcards, let us consider the following examples.

Example. Let a directory contain the files `file`, `file2`, `file3`, `fun`, `fun2`, `mbox`, and `readme`. Here follows a terminal interactive session, where brackets ([ ]) contain explanations that do not appear on the computer monitor:

```
% ls *          [ * matches all filenames]
file   file2   file3   fun   fun2   mbox   readme


% ls f*         [ f* matches all filenames beginning with f]

file   file2   file3   fun   fun2


% ls file?      [ file? matches all filenames of length 5
beginning with file]

file2  file3


% ls ????       [ ???? matches all filenames of length 4]

file   mbox
```

The first example shows `ls` with `*`, which matches all files in the directory (since all filenames have 0 or more characters).

In the second example, `f*` matches all the files beginning with an `f`, which are listed.

The third example uses `file?` to match all filenames that begin with the word `file` and have one character following that word.

In the fourth example, four `?` characters in a row match all filenames that are four and only four characters long.

Wildcards should be used with caution. For example, when used with a destructive command like `rm`, the wildcard "*" could help you remove all the files in your directory!

3.5. Directory Commands

Having covered file commands, we now turn to commands related to directories, which are collections of files and other directories.

3.5.1. Discovering your Location

It is often difficult to remember where you are within a given file system, due to the tree structure that usually has many levels. To make the location of the current directory clear to you, UNIX provides the `pwd` command, which means *present working directory*.

Example. Using the previous login i.d. of `bsimpson`, typing the command `pwd` in B. Simpson's home directory would yield the following interactive session:

```
% pwd



/home/bsimpson
```

This also holds for any location that you are at in the file system.

3.5.2. Changing the Directory

To reset the current directory, which is like moving from one directory to another directory, UNIX provides the `cd` command, which means *change directory*. Whenever you need to move to your home directory, just type `cd` with no arguments. If you specify a directory name as an argument, `cd` will attempt to located that directory, then set it as the current directory if it is a valid directory in the UNIX filesystem.

Example. Suppose you want to view files in *bsimpson*'s `` `cop3610' `` directory. Then, you would type:

```
% cd ~bsimpson/cop3610
% pwd



/home/bsimpson/cop3610



%ls -l
```

and the filenames would be displayed.

Important Note: the tilde preceding *bsimpson* is expanded by the Unix C-shell into the full pathname of that person's home directory, so you do not have to enter a potentially long pathname.

Suppose B. Simpson has an `assign1` directory under `cop3610`. To move there, type

```
% cd assign1
% pwd



/home/bsimpson/cop3610/assign1
```

If you use `` `..' `` as the argument, `cd` will bring you **up** a level in the directory tree.

Example. Assume the filesystem from the previous example.

```
% cd ..



% pwd



/home/bsimpson/cop3610
```

3.5.3. Viewing Directory Contents

We've already overviewed the `ls` command for listing directory contents, together with the "l" and "a" options. Several commonly-used options follow:

`-a` : Lists **all** the files in the directory, including hidden files.

`-F` : Appends a single character to filenames *on the display only* that aren't text files, to denote file type. For example, directories have a trailing **/** and executable files have a trailing *****. New users have this option set by default. Hence, the examples of `ls` in these course notes assume that this option is being used.

`-l` : Lists in *long* format, telling (from left to right) the file's permissions, number of links to the file, the file's owner, the file's size in bytes, and the time the file was last modified. Examples were provided previously.

`-g` : In conjunction with the `-l` option, the `-g` option of `ls` includes the file's group following the *owner field* for each file.

`-R` : Used to generate a **recursive** listing of all directories encountered *below the level of the current directory*.

Note that `ls` lists the current directory by default, if you do not specify the name of a file or directory you wish to list. Some interesting results can occur, as shown below.

Example. Suppose you are in *bsimpson*'s home directory, and you type `ls -a`. If the directory is not secured, you might see the following:

```
% ls -a
./      .cshrc     .login       .msgsrc     cop3610/     private/
../     .emacs     .logout      a.out*      emg3312/
typescript
```

Here, the two files called "." and ".." are *directory links*. In particular, "." is a link to the current directory, and ".." is a link to the *parent* of the current directory. You can backtrack upward through a directory structure by using "`cd ..`" to pop up one directory level.

3.5.4. Creating a New Directory

Users can create new directories using the `mkdir` command. Prior to this, one must determine where the directory is to be placed. For example, a directory can be located one level below one's home directory or subdirectory.

Example. Let *bsimpson* create a directory within his `emg3312` directory:

```
% pwd                    [Locate the parent directory of
the new directory]
/home/bsimpson/emg3312



% mkdir drafts           [Make the new directory]
```

```
% ls                          [Check to be sure that the new
directory is there]


drafts/    termpaper
```

Recall that, when you create a directory, its permissions and the default permissions of all its children (files and directories) are set according to the `umask` setting. It is always wise to view permissions with the `ls -l` command, then change them with the `chmod` command.

3.5.5. Removing an Existing Directory

Occasionally, a user may decide he wants to remove a directory. (Sometimes this needs to be done to make room for more files in your home directory.) In this case, one uses the `rmdir` command, which means *remove directory*. However, the directory to be removed must be empty of all files and subdirectories. Otherwise, `rmdir` will inform you that the directory is not empty and will not remove it.

Example. Assuming that *bsimpson* has a `drafts` directory, let us try to remove that directory using `rmdir`:

```
% pwd                                [check the current
directory]
/home/bsimpson/emg3312

          [this is the current directory]


% ls                                 [list contents of
current directory]

drafts/ termpaper


% rmdir drafts                       [try to remove the
drafts directory]

rmdir: drafts: Directory not empty [this is an error
message from rmdir]
```

```
% ls drafts                          [o.k., let's see what is
in "drafts"]

monologue notes

                  [contents of directory "drafts"]



% rm drafts/*                        [try to remove contents
of directory "drafts"]

rm: remove drafts/monologue? y       ["rm" asks do you really
want to do this?

rm: remove drafts/notes? y             ...and you answer "y"
for yes]



% ls drafts                          [check to see if files
have been removed]

                                     [no listing => directory
"drafts" is empty]



% rmdir drafts                       [now we can remove
"drafts"]

% ls                                 [so what is left under
emg3312?]

termpaper                            [...only the "termpaper"
directory remains]
```

Note that *bsimpson* first had to remove (or move) the files from the directory `drafts` before removing the directory itself. Note also that the files were removed with `rm` followed by a wildcard ("*").

Important Note: Upon the removal of a directory, `rmdir` will not tell you that the directory was removed. Instead, you are supposed to use `ls` to see that the directory is actually gone. This is another little idiosyncracy of UNIX that users love to hate.

3.5.6. Directory and File Command Summary

Here follows a summary of the UNIX file and directory commands covered in this section:

`pwd`

(present working directory) Display the full pathname of the current directory.

`cd` *`directory`*

(change directory) Changes the current directory. If you specify no arguments, then `cd` will bring you back to your home directory.

`ls` *`directory ...`*

(list directory) Lists the contents of a directory or directories. If the directory name is unspecified, then `ls` will list the contents of the current directory. You can also supply filename(s) instead of a directory name, to get more information about one or more files.

`mkdir` *`directory ...`*

(make directory) Create a directory or directories.

`rmdir` *`directory ...`*

(remove directory) Remove an empty directory or directories (the directories must not contain any files).

`cat` *`file1 ...`*

(concatenate) Used to view a file or files continuously on your terminal. If you use `cat` to view a long file, it is necessary to use `{Ctrl}-s` keys to pause the screen and `{Ctrl}-q` keys to unpause. Otherwise, file contents will be displayed too fast for you to read them. For files longer than one screen, it is recommended that you use either more or less to view the file.

`more` *`file`*
`less` *`file`*

These two programs display a file one screen at a time, and offer viewing options such as paging backwards through a file and pattern searching. In a typical UNIX idiosyncrasy, less is the more sophisticated of the two and has features that aren't found in more.

`rm` *`file1 ...`*

(remove) Removes the specified file(s).

`mv` *`file1 file2`*
`mv` *`file1 ... fileN directory`*

(move) Moves a file or directory. In the first form, *`file1`* will be moved to (renamed as) *`file2`*. The second form will move a number of files into a directory which you specify as the final argument on the command line. Directory names can be used in place of the filenames in either of the forms, to move or rename directories.

`cp` *`file1 file2`*
`cp` *`file1 ... fileN directory`*

(copy) Copies files. In the first form, *file1* will be copied to a file called *file2*. The second form will copy a number of files into a directory which you specify as the final argument on the command line.

chmod *mode file1 ...*
(change mode) Changes file permissions. The *mode* specifies how permissions are to be changed for the listed *file(s)*.

This terminates our overview of UNIX file and directory commands. We next discuss commands that affect the computer system directly.

3.6. Basic System Commands

When using a computer or networked workstation, it is occasionally useful to perform system functions such as determining who is using the computer, what is the date and time, and so forth. UNIX commands that affect the system without changing files or directories are called *system commands*. Several frequently-used system commands are reviewed, as follows.

3.6.1. N/A

3.6.2. Getting the Date and Time

UNIX has a convenient method for showing you what time it is. Just type date on the command line, and you will get a date and 24-hour time display like this:

Example.

```
% date
Tue Feb  8 19:36:33 EST 2000
```

This means that today is Tuesday, 8 February 2000, and the time is 7:36:33 pm Eastern Standard Time.

Finding Out Who is Logged In

Suppose you happen upon a UNIX workstation and someone is logged in. Your first inclination might be to ask around and see who is using the computer. Or, you can type whoami at an available prompt to see who the user is.

Example.

```
% whoami
hkmung
```

This concludes our overview of basic UNIX commands. We next discuss the software development process with a UNIX operating system.

References