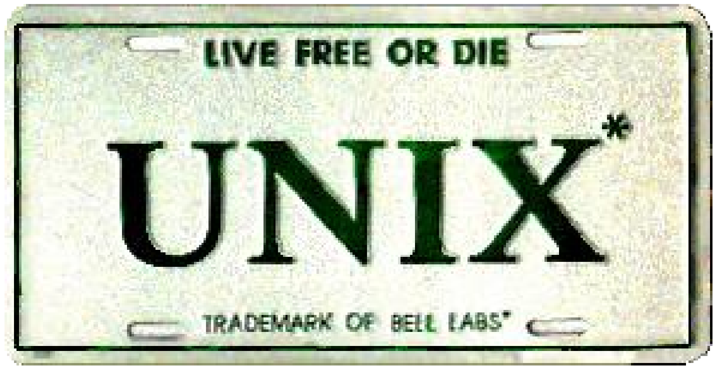

UNIX for Programmers and Users

“UNIX for Programmers and Users”
Third Edition, Prentice-Hall, GRAHAM GLASS, KING ABLES

Displaying a File: cat

- cat with the name of the file that you wanted to display:

```
$ cat  heart                    -->  list the contents of the "heart" file.
I  hear her breathing.
I'm  surrounded by the sound.
Floating in this secret place,
I never shall be found.
$ _
```

- cat is good for listing the contents of small files, but it doesn't pause between full screens of output.

Displaying a File: more

more -f +lineNumber fileName

- The more utility allows you to scroll a list of files, one page at a time.
- By default, each file is displayed starting at line 1, although the +option may be used to specify the starting line number.
- The -f option tells more not to fold (or wrap) long lines.

- After each page is displayed, more displays the message “--more--” to indicate that it’s waiting for a command.

- To list the next page, press the space bar.
- To list the next line, press the Enter key.
- To quit from more, press the “q” key.
- ^B will display the previous page
- H will display help page

- Try:

```
$ ls –la /usr/bin > myLongFile
$ more myLongFile
```

Displaying a File: head and tail

head -n fileName

- The head utility displays the first n lines of a file. If n is not specified, it defaults to 10. If more than one file is specified, a small header identifying each file is displayed before its contents.

tail -n   fileName

- The tail utility displays the last n lines of a file. If n is not specified, it defaults to 10. If more than one file is specified, a small header identifying each file is displayed before its contents.

- The first two lines and last two lines of my “heart” file.

```
$ head -2 heart                        -->  list the first two lines.
I hear her breathing,
I’m surrounded by the sound.
$ tail -2 heart                        -->  list the last two lines.
Floating in this secret place,
I never shall be found.
$ head -15 myLongFile                  --> see what happens
```

Renaming/Moving a File: mv

```
mv  -i  oldFileName  newFileName
mv  -i  fileName directoryName
mv  -i  oldDirectoryName newDirectoryName
```

- The first form of mv renames oldFileName as newFileName.
- The second form allows you to move a collection of files to a directory.
- The third form allows you to move an entire directory.
- The -i option prompts you for confirmation if newFileName already exists so that you do not accidentally replace its contents. You should learn to use this option (or set a convenient shell alias that replaces “mv” with “mv –i”; we will come back to this later).

Renaming/Moving Files: mv

- Here’s how to rename the file using the first form of the mv utility:

```
$ mv  heart   heart.ver1                    --> rename to “heart.ver1”.
$ ls
heart.ver1
$ _
```

Making Directory: mkdir

mkdir -p newDirectoryName

- The mkdir utility creates a directory. The -p option creates any parent directories in the newDirectoryName pathname that do not already exist.

- If newDirectoryName already exists, an error message is displayed and the existing file is not altered in any way.

```
$ mkdir lyrics            --> creates a directory called “lyrics”.
$ ls –lF                  --> check the directory listing in order
                          --> to confirm the existence of the
                          --> new directory.
-rw-r--r--    1   glass   106   Jan  30  23:28    heart.ver1
drwxr-xr-x    2   glass   512   Jan  30  19:49    lyrics/
$ _
```

Moving Files

- Once the “lyrics” directory is created, we can move the “heart.ver1” file into its new location. To do so, used mv and confirm the operation using ls:

```
$ mv heart.ver1  lyrics     --> move into “lyrics”.

$ ls                        --> list the current directory.
lyrics/                     --> “heart.ver1” has gone.

$ ls lyrics                 --> list the “lyrics” directory.
heart.ver1                  --> “heart.ver1” has moved.
$ _
```

Changing Directories: cd

- **cd directoryName**

- The following might be inconvenient; especially if we deal with large hierarchy:

```
$cat lyrics/heart.ver1                    --> display
```

- Instead, change directory:

```
$ cd  lyrics                              --> change directory
$ cat   heart.ver1                        --> display
```

- The cd shell command changes a shell's current working directory to be directoryName.

- If the directoryName argument is omitted, the shell is moved to its owner's home directory.

Reorganizing Directories

```
$ pwd                                --> display where I am
/home/glass

$ cd lyrics                          --> move into the "lyrics" directory
$ pwd
/home/glass/lyrics

$ cd ..                              --> move up one level
$ pwd                                --> display new position
/home/glass

$ cd lyrics                          --> move into the "lyrics" directory
$ pwd
/home/glass/lyrics

$ ls ~/                              --> "~/" refers to home directory
/home/glass
$ _
```

Copying Files: cp

- To copy the file, I used the cp utility, which works as follows:

```
cp -i oldFileName newFileName
cp -ir fileName directoryName
```

- The first form of cp copies the contents of oldFileName to newFileName.
- If the label newFileName already exists, its contents are replaced by the contents of oldFileName.
- The -i option prompts you for confirmation if newFileName already exists so that you do not accidentally overwrite its contents. Like with mv, it is a good idea to use this option or create an alias.

Copying Files: cp

- The -r option causes any source files that are directories to be recursively copied, thus copying the entire directory structure.

- cp actually does two things
  - It makes a physical copy of the original file’s contents.
  - It creates a new label in the directory hierarchy that points to the copied file.

```
$ cp heart.ver1  heart.ver2                --> copy to “heart.ver2”.
$ ls -l heart.ver1  heart.ver2             --> confirm the existence of both files.
-rw-r--r--   1   glass   106   Jan  30  23:28   heart.ver1
-rw-r--r--   1   glass   106   Jan  31  00:12   heart.ver2
$ cp -i heart.ver1  heart.ver2             --> what happens?
```

Deleting a Directory: rmdir

rmdir directoryName

- The rmdir utility removes all of the directories in the list of directory names provided in the command. A directory must be empty before it can be removed.
- To recursively remove a directory and all of its contents, use the rm utility with the -r option.
- Here, we try to remove the "lyrics.draft" directory while it still contains the draft versions, so we receive the following error message:

```
$ rmdir  lyrics.draft
rmdir : lyrics.draft : Directory not empty.
$ _
```

Deleting Directories: rm -r

- The rm utility allows you to remove a file's label from the hierarchy.
- Here's a description of rm:

rm -fir fileName

- The rm utility removes a file's label from the directory hierarchy.
- If the filename doesn't exist, an error message is displayed.
- The -i option prompts the user for confirmation before deleting a filename. It is a very good idea to use this option or create a shell alias that translates from "rm" to "rm –i". If you don't, you will loose some files one day – you have been warned!
- If fileName is a directory, the -r option causes all of its contents, including subdirectories, to be recursively deleted.
- The -f option inhibits all error messages and prompts. It overrides the –i option (also one coming from an alias). This is dangerous!

Removing Directories with Files

- The -r option of rm can be used to delete the “lyrics.draft” directory and all of its contents with just one command:

```
$ cd                           --> move to my home directory.
$ rm  -r   lyrics.draft        --> recursively delete directory.
$ _
```

Counting Lines, Words and Characters in Files: wc

wc -lwc fileName

- The wc utility counts the number of lines, words, and/or characters in a list of files.
- If no files are specified, standard input is used instead.

- The -l option requests a line count,
- the -w option requests a word count,
- and the -c option requests a character count.

- If no options are specified, then all three counts are displayed.
-
- A word is defined by a sequence of characters surrounded by tabs, spaces, or new lines.

Counting Lines, Words and Characters in Files: wc

- **For example, to count lines, words and characters in the “heart.final” file, we used:**

```
$ cd  ~/lyrics.final
$ wc  heart.final          --> obtain a count of the number of lines,
                           --> words, and characters.
9     43    213   heart.final
$ _
```

Determining Type of a File: file

file fileName

- The file utility attempts to describe the contents of the fileName argument(s), including the language in which any of the text is written.
- file is not reliable; it may get confused.
- When file is used on a symbolic-link file, file reports on the file that the link is pointing to, rather than on, the link itself.
- For example,

```
$ file heart.final                          --> determine the file type.
heart.final: ascii text
$ _
```

File Permissions (Security)

- File permissions are the basis for file security. They are given in three clusters. In the example, the permission settings are "rw-r--r--":

```
-rw-r--r--   1  glass    cs    213   Jan  31  00:12   heart.final
```

User (owner)	Group	Others
rw-	r--	r--

← clusters

Each cluster of three letters has the same format:

Read permission	Write permission	Execute permission
r	w	x

File Permission

The meaning of the read, write, and execute permissions depends on the type of file:

	Regular file	Directory file
Read	**read the contents**	**read the directory (list the names of files that it contains)**
Write	**change the contents**	**Add files to the directory**
Execute	**execute the file if the file is a program**	**access files in the directory**