Displaying Files

Now that we have the basics of directories, we will learn some useful commands to manipulate regular files. We will start with commands to display the contents of files: *cat, more, less, head,* and *tail.*

The first of these, *cat* , reads one or more files, concatenates them together (which is where it gets its name), and prints them out. As you may have guessed by now, *cat* determines which file(s) to read and print based on its command line arguments. It will print out each file you name, in the order that you name them.

If you do not give *cat* any command line arguments, then it will read *standard input* and print it out. Typically, standard input is the input of the terminal that you run a program from—meaning it is usually what you type. If you just run *cat* with no arguments, this means it will print back what you type in. While that may sound somewhat useless, it can become more useful when either standard input or standard output (where it prints: typically the terminal's screen) are *redirected* or *piped* somewhere else.

While you can use *cat* to display the contents of a file, you typically want a bit more functionality than just printing the file out. The more command displays one screenfull and then waits until you press a key before displaying the next screenfull. It gets its name from the fact that it prompts *---More--* to indicate that you should press a key to see more text. The *less* command supercedes *more* and provides more functionality: you can scroll up and down with the arrow keys, and search for text. Many systems actually run less whenever you ask for more.

There are also commands to show just the start (*head*) or just the end (*tail)* of a file. Each of these commands can take an argument of how many lines to display from the requested file. Of course, for full details on any of these commands, see their *man* pages.

Note that these commands just let you view the contents of files.

Moving, Copying, and Deleting

Another task you may wish to perform is to move (*mv)* , copy (*cp*), or delete (*rm* — stands for "remove") files. The first two take a source and a destination, in that order. That is where to move (or copy) the file from, followed by where to move (or copy) it to. If you give either of these commands more than 2 arguments, they assume that the first N-1 are sources, and the last is the destination, which must be a directory. In this case, each of the sources is moved (or copied) into that directory, keeping its original filename.

The *rm* command takes any number of arguments, and deletes each file that you specify. If you want to delete a directory, you can use the *rmdir* command instead. If you use *rmdir* , the directory must be empty—it must contain no files or subdirectories (other than . and ..). You can also use *rm* to *recursively* delete all files and directories contained within a directory by giving it the *-r* option. Use rm with care: once you delete something, it is gone.

Pattern Expansion: Globbing and Braces

You may (frequently) find yourself wishing to manipulate many files at once that conform to some pattern—for example, removing all files whose name ends with ~ (editors typically make backup files while you edit by appending ~ to the name). You may have many of these files, and typing in all of their names would be tedious.

Because these names follow a pattern, you can use *globbing* —paterns which expand to multiple arguments based on the file names in the current directory—to describe them succinctly. In this particular case, you could do *rm * ~* (note there is no space between the * and the *~;* doing r *m * ~* would expand the * to all files in the directory, and then ~ would be a separate argument after all of them). Here, * is a pattern which means "match anything". The entire pattern *~ matches any file name (in the current directory) whose name ends with ~ . The shell expands the glob before passing the command line arguments to *rm*— that is, it will replace *~ with the appropriately matching names, and rm will see all of those names as its command line arguments.

There are some other UNIX globbing patterns besides just *. One of them is ? which matches any one character. By contrast, * matches any number (including 0) of characters. You can also specify a certain set of characters to match with [...] or to exclude with [!...]. For example, if you were to use the pattern file0[123].txt it would match file01.txt, file02.txt, and file03.txt. If you did file0[!123].txt, then it would not match those names, but would match names like file09.txt, file0x.txt, or file0..txt (and many others).

Sometimes, you may wish to use one of these special characters literally—that is, you might want to use * to mean just the character *. In this case, you can escape the character to remove its special meaning. For example, rm * will remove exactly the file named *, whereas rm * will remove all files in the current directory.