Unix Tools	Lecture 2
CS114 Fall 2003	Wednesday, October 1, 2003

File and Directory Manipulation

The command `cd` is used to change your current directory. If you just type in `cd` by itself, it will change you back to your initial directory (usually, your home directory). If you specify a path, as in `cd /home/rp65/` or `cd ..`, it will change your current directory into the directory that you specify.

If you happen to lose track of what your current directory is, the command `pwd` will print it out for you.

Finally, the command `ls` will display the content of the current directory (i.e. it will list all the files and all the directories it contains). If you specify an argument, `ls` *`directory`*, then `ls` will display the content of *`directory`*.

The following commands are also useful when dealing with files and directories. Recall that whenever we talk about a file or a directory, it actually stands for a path to a file or directory.

- `mkdir` *`directory`*: creates *`directory`*
- `rmdir` *`directory`*: removes *`directory`* (only if it's empty and you are not inside it)
- `touch` *`file`*: creates an empty *`file`*
- `cat` *`file`*: dumps the content of *`file`*
- `rm` *`file`*: removes *`file`*
- `mv` *`oldname newname`*: renames a file (and moves it if names refer to different directories)
- `mv` *`file1 file2 ... directory`*: moves files into a directory
- `cp` *`oldname newname`*: copies a file
- `cp` *`file1 file2 ... directory`*: copies files into a directory

Some commands have *options*, that affect the way they behave. Options are typically a letter preceded by a `-`. For example, if you give the option `-i` to `rm`, that is, if you write `rm -i` *`file`*, the system will prompt you for confirmation before deleting the file. Some commands understand

many options. The command `ls` for instance, recognizes, among others, the option `-F`, which makes `ls` give you some indication of the type of each file (it adds a trailing `/` at the end of every directory), and also the option `-a`, which makes `ls` display *everything* in the directory, including so-called *hidden* files or directory. A file or directory is hidden if its name starts with a period, such as `.i-am-hidden`. Hidden files aren't special in any ways, except that `ls` by default will not list them. Typically, programs needed configuration files will make those files hidden, otherwise they'd clutter up your home directory.

How do you learn and remember the various options that each command understand? You can look at the online Unix documentation, available through a command `man`. If you type `man` *`command`*, it will search for and display documentation on *`command`*. This information is called the *man pages* of the command. Among other things, you will get a description of the arguments the command expects, and the options it understands. (The command `man` itself understanding different options, you can always do a `man man` to find out.)

Commands are often built-in. Programs (or applications) are essentially commands stored on the filesystem. You use a program just like you would a command, by specifying its name at the prompt, along with possible arguments. By default, Unix will search for the program in some directories it knows about (for instance, `/bin`, `/usr/bin/`, `/usr/local/bin`, etc) to attempt to find a file matching the name you specify. If it cannot find one (or if it is not marked as executable, which we will see in the next lecture), it reports an error. Otherwise, it starts the program.

To give you an example of a program, consider `lynx`, a web browser that you can use from a text-only console. To start it, do either `lynx` or `lynx` *`URL`*. (`lynx` can be found in `/usr/local/bin/`.) It can also be used noninteractively. If you try `lynx -dump` *`URL`*, it will dump the formatted content of the *`URL`* on the screen. For more information on `lynx`, consult the man pages.

Since programs are just files, you can specify exactly where you the program you want to execute is stored, by using a path. For example, consider `pine`, an email program you will be using. We have installed pine in the directory of the course, `~cs114/bin/`. To execute `pine`, you need to specify a path to `pine` (either absolute or relative). Therefore, you can invoke `pine` by writing `~cs114/bin/pine` at the Unix prompt. This by-passes the Unix search, and attempts directly to execute the program you specify.