1 The Unix Command Line

Unix is a computer operating system that exists in many variants, most notably Linux and (loosely) Mac OS X. All of these Unix-based systems have one thing in common - the Unix command line interface (CLI), which is the foundation of these modern operating systems.

Unix CLIs such as Bash are text-based as opposed to graphical. Although the CLI can seem "hacker"-ish or intimidating, it's safe to think of it as simply an alternative version of a graphical user interface (GUI). Instead of clicking on icons to launch programs, you type the name of the program you want to launch. Instead of navigating through folders by clicking on folders, we use commands like `cd` and `ls`. Practically everything you do in a GUI and more is doable in a CLI, from opening and downloading files, editing text, checking the time, or even browsing the web or playing games. In fact, there are tasks that an average computer user will occasionally come across that can be more efficiently done with a CLI and its tools.

2 Getting Started

When you first open the terminal, a natural response is "help", and if you type that, you've just tried your first Unix program:

```
$ help
```

You can also use the command `man` to access the manual of any command:

```
$ man ls
```

Press `q` to exit any manual.

All CLI programs are run by typing in the name of the program, followed by Enter. Many programs, in addition, take arguments, just like functions in your favorite programming language - the `man` command, for example, takes in the name of a program whose manual you want to access (indeed, you can also run `man man`). Argument(s) are provided after the initial program, and are separated by spaces. Arguments are akin to, for example, specifying a file to open in a text editor, or function parameters in your favorite programming language.

Many programs will also have "flags". These are options that further specify the behavior of a program, and are typically specified by another name preceded by a dash or two, such as `-f` or `--name`. Command line flags are very similar to toggling options on and off in GUI-based programs, such as changing the display font and size of Microsoft Word.

3 Navigation

The most important skill in the Unix command line is moving around.

- `pwd` - Print Working Directory. This shows where you are in your filesystem.

- `ls DIR` - Lists the contents of directory `DIR`. If `DIR` is not specified (i.e. you just type in `ls`), this lists the contents of your current directory.
- `cd DIR` - Change Directory. This command takes an argument `DIR`, which is the name of the folder you would like to navigate into. You can provide this as an absolute path (e.g. `/home/username/Documents`) or a relative path (e.g. the name of a folder inside your current directory). For example, if you're located inside a directory, and there is a folder called `Pictures` inside your current directory, the command `cd Pictures` will move you to that directory.
  - The command `cd ..` will take you *up* one directory. In general, the string `..` refers to the directory one level up, and the string `.` refers to the current directory.
  - **Note:** Almost everything in the Terminal is case-sensitive. If there are spaces in your folder names, you will either need to wrap the name of your folder in quotes (e.g. `cd ''My Pictures''`) or use a backslash to "escape" the space (e.g. `cd My\ Pictures`).

Exercise: Getting Ready Navigate to the `Home` directory inside the `starter` folder in this lab. Once there, list the contents of the directory. You may need to find where you have saved the `unix_lab` folder inside your system.

4 Files

- `cp FILE1 FILE2` - copies the file `FILE1` into a new filename or location `FILE2`. This will overwrite any existing file located at `FILE2`!
  - `cp image1.jpg Pictures/` - copies the file image1.jpg into the folder Pictures (so that there will be another `image1.jpg` located in the directory `Pictures`
  - `cp image1.jpg image1_copy.jpg` - makes a copy of the file `image1.jpg` called `image1_copy.jpg` in the same directory.
- `mv FILE1 FILE2` - moves the file `FILE1` into a new filename or location `FILE2`. This command is the same as `cp` except the original file no longer exists. This command is also used to rename files, by simply moving a file within one directory, e.g. `mv oldname.txt newname.txt`. This will overwrite any existing file located at `FILE2`!
- `rm FILE` - removes the file `FILE`. This does not work for directories unless the `-r` (recursive) flag is specified. Be careful with this command! Unlike deleting files in most GUIs, there is no trash can or recycle bin that stores deleted files. Files removed this way are lost permanently.
  - A common idiom is `rm -r DIR` to remove a directory and all of its files, as well as `rm *` to remove all files in the current directory. Again, *be careful with these!*
- `mkdir DIR` and `rmdir DIR` - makes and removes directories, respectively. `rmdir` only works if the directory is empty, i.e. there are no files in it. Thus, it may be necessary to remove the files inside `rmdir` with `rm` first.

Exercise: Tidying Up Time to organize your `Home` folder. Make a `Pictures` directory inside `Home`, and move the image `Stanford1.jpg` there. Because you love Stanford so much, make two more copies of `Stanford1.jpg` called `Stanford2.jpg` and `Stanford3.jpg` in the same folder. Last, the folder `junk` contains some spam email. Try getting rid of it.

Tab Completion, Wildcards Most systems support a very handy feature, where if you begin to type the name of file/directory/command and press `<TAB>`, so long as it is unambiguous as to what file you are referring to, the CLI will automatically complete the name. For example, typing `py<TAB>` will autocomplete your command to `python`. Similarly, typing `cd Pic<TAB>` when in your `Home` folder will autocomplete your command to `cd Pictures`.

5 File IO

Besides text editing and file compilation, another useful application for CLIs in the context of CS is modifying text files.

- `less FILE` - A file viewer that displays the contents of file `FILE`. In this viewer, `j` or `k`, or the arrow keys, page up and down. `q` quits.
- `cat FILE1 FILE2 FILE3...` - Short for concatenate, `cat` combines all of the files supplied as arguments into one file and outputs the result. However, it is also commonly used for displaying the contents of one file, if only one file is specified, such as `cat essay.txt`.
  - If no files are specified, `cat` will output whatever you manually input into the terminal. Although this seems useless, it will actually come in handy for several applications.
- `grep PHRASE FILE1 FILE2 FILE3...` - Searches for the text string `PHRASE` in any of the files `FILE1, FILE2, FILE3, ....` For example, `grep pear fruitlist.txt` searches for occurrences of the word `pear` in the file `fruitlist.txt`.
  - If no files are specified, `grep` will search whatever text you input manually into the terminal. This will also become useful later on.
- `find DIR -name NAME` - The most common usage of `find`, this command searches the directory `DIR` for files with the name `NAME`. Often, it is common to use wildcards (*) when searching for names. For example, `find . -name ''*.txt''` searches for all files ending in `.txt` in the current directory and subdirectories.
- `wc FILE` - This counts the number of lines, words, and characters in the given file. You can also have it output only one of the statistics with any of the flags `-l`, `-w`, and `-m`, respectively. For example, `wc -w essay.txt` will output the number of words in `essay.txt`.
  - Once again, if no files are specified, `wc` will count the words you manually type into the terminal.

- `sort FILE` - Sorts the given `FILE` in ascending alphabetical order by line, and outputs the result. The flag `-n` sorts numerically. For example, `sort fruitlist.txt` will sort the list of fruits in the file by name.
- `curl URI` - Retrieves the file located at the given internet URI `URI` and outputs the result. In essence, this "downloads" a file.

Exercise: Looking for Stuff

- `my_languages.txt` contains all of the programming languages you know. Do you know Lisp? Try using both a file viewing method (`less` or `cat`) with the searching method (`grep`).
- How many fruits are listed in `fruitlist.txt`?
- Try displaying the contents of `Stanford1.jpg`. Is it what you expected?
- Like baseball? If you have a favorite baseball player, try searching for him in `Players.csv` file in the `Baseball` directory[1]. Search for the string `FirstName,LastName`, as that is how the name is organized in the file. If you don't have a favorite player, try David Ortiz. To see what all of the fields in the row mean, look at the first line of `Players.csv`, which tells you what each column means.
- According to `2013salaries.csv`, who was the highest-paid baseball player in 2013? `2013salaries.csv` only contains baseball reference IDs - if you don't recognize the name, you may need to search for the specific ID in `Players.csv`.

6 Input/Output Redirection

This is where stuff gets really interesting. Unix commands generally take some kind of input, and produce some kind of output. We can control where the commands get their input and output using *I/O Redirection*.

Although this discussion can get technical, a very basic explanation of this is that many commands' output can be *redirected* to other places besides just spitting out text in the terminal. Most notably, we can give the output of one command into a new file, or the input of another command. We have several tools available at our disposal to do this.

This is also the explanation to the initially odd default behavior of commands like `cat` and `grep`. Using these operators, we can feed in input to these commands without having to specify a specific file.

- `>` is the *output redirection* operator. After writing a command, we can use `>` followed by a filename to feed the output of a command into a file.
  - `curl URI > FILE` will retreive the file located at the given `URI` and save it or overwrite into a new file `FILE`

[1]Grabbed from http://www.seanlahman.com/baseball-archive/statistics/.

  - `grep pear fruitlist.txt > pearlist.txt` will grab all of the lines `grep` finds with the string `pear`, but instead of outputting back to the terminal, the content will be stored in a new file `pearlist.txt`.

- `<` is the *input redirection* operator, which allows many programs to take in the contents of files as output. This is perhaps less useful on its own, since many commands can take files directly. But, there are still many, many uses for it.

- `|` is the *pipeline* character, which essentially combines `<` and `>`. Basically, `command1 | command2` runs `command1`, then feeds the output of that command as the input of `command2`.

  - `grep pear fruitlist.txt | wc -l` will output the number of lines found in `fruitlist.txt` that contain the phrase `pear`. This is essentially a combined version of creating `pearlist.txt` as above and then running `wc -l pearlist.txt`.
  - `help | grep if` makes `grep` search for the phras with first or last names containing the string `Alex`. `if`, showing the syntax of `if` statements in the command line.

- `>>` is the *append* operator, which appends the output of any program onto the end of any file.

  - `cat >> file.txt` is thus a very common idiom. Since `cat` outputs whatever input you give, running this command, then manually typing in some text and then pressing `Ctrl+D` to end input will add the inputted text onto the end of `file.txt`.

Bonus Exercises

- In `2013Salaries.txt`, which Boston Red Sox (`BOS`) baseball players had the greatest and smallest salary in 2013? Can you use redirection to find this in one command? You may need to use the `sort -n` flag for numerical sort.

- Create a file named `AlexPlayers.csv` which contains all of the players and their statistics from `Players.csv` with first or last names containing the string `Alex`. Out of these players with `Alex` in their name, determine how many of them were born or had passed away in 1962.

- **Free Audiobooks** Mac OS X users have a very cool program called `say`, which, well, says anything you give it. You're tired of paying ridiculous amounts of money for eBooks (let alone real textbooks) so you've decided to take matters into your own hand. Find a way to get a free eBook from a public source like Project Gutenberg (www.gutenberg.org). Better yet, with the `say` command, turn this book into a free audiobook! (Quality is somewhat questionable)[2].

[2]http://www.maclife.com/article/columns/terminal_101_making_your_mac_talk_%E2%80%9Csay%E2%80%9D