Basics of Unix

Please read the tutorials from the Berkeley Statistical Computing Facility (SCF):

- **Unix Basics:** https://github.com/berkeley-scf/tutorial-unix-basics/
- **Using Bash:** https://github.com/berkeley-scf/tutorial-using-bash/

Also, please follow the software carpentry workshop **Shell Novice:**

https://github.com/swcarpentry/shell-novice

that we will be using during lecture.

History of Unix

Unix is an operating system. It was created during 1969 to 1971 by AT&T employees working at Bell Labs, while involved in designing a mainframe operating system called MULTICS, which was short for *Multiplexed Information and Computing Service*. Back then, mainframes were expensive so users were granted time-sharing slots to do their work and to use the power for the mainframe, often by logging in from a remote terminal that had far less computing power.

MULTICS was designed to manage that remote login time-sharing process. But MULTICS became such a huge and complex project that Bell Labs decided to pull out of it. The Bell Labs employees who'd worked on MULTICS still like the project's goals and so, almost as a side project they worked on building a smaller and simpler version of it. Originally, their version only supported a single user, so that as a play on the name they called it UNICS, *Uniplexed Information and Computing Service* instead of Multiplexed. Very quickly UNICS became able to support multiple users and so they renamed it **UNIX** (with an X), which doesn't actually stand for anything anymore.

Nowadays when people refer to "UNIX" it basically means any UNIX-like operating system, including Linux and Mac OS X. Unfortunately, MS Windows is not based on UNIX. Because most modern scientific computing is done on UNIX-based machines, it is important that you acquire a basic knowledge on how to work and perform tasks in these type of operating systems.

The Terminal Application

The way we interact with Unix is with the **command line**. If you have Mac OS X, you have this application. It's going to be located inside your `Applications` folder, inside the folder called `Utilities`, and there you'll see the program `Terminal`. In addition to the terminal, you should install the *Xcode developer tools* from https://developer.apple.com/xcode/

For Windows users, you can find a built-in program called Power-Shell. Better alternatives to Power-Shell are **Git-Bash** or **Cygwin**.

Command Structure

Using the command line involves typing commands. So you need to learn the command structure in Unix. Commands are always going to be the command, followed by the options, followed by the arguments; always in that order. You can't put the options after the arguments. They have to come in the middle. The command is always a single word. Here's one example:

```
echo "Hellow World"
```

In the previous example the command was `echo`. In this case there are no options. The argument is the string `"Hello World"`.

Options modify the behavior of the command in some way.

Here's another example:

```
ls -l Desktop
```

In the previous example the command is `ls` with the `-l` option, and the argument `Desktop`.

We can also have a list of options `ls -l -a -h` and then the argument `Desktop`:

```
ls -l -a -h Desktop
```

When there are multiple options, you can simply smash them altogether. We could mix and match these options in any way we want:

```
ls -lah Desktop
```

There is an exception to this, which is that sometimes an option wants an argument of its own. So for example with the `banner` command, the option `-w` specifies the width of the banner.

```
banner -w 50 Hello
```

To make that less confusing, most users eliminate the space between the option and its argument:

```
banner -w50 Hello
```

and that makes it clear that 50 belongs to the `w` option followed by the string.

Kernel and Shells

Related with UNIX and the command lines there are two other fundamental concepts: the *kernel* and the *shells*. The kernel is the core of the operating system in Unix. It's what takes care of allocating time and memory to program, doing sort of very fundamental root level management of how programs go about doing their thing.

Mac OS X uses the Mach kernel inside Darwin to do that. If you ever hear someone refer to Mach kernel, this is the very central part of the Unix operating system that really just takes care of how the operating system handles very basic tasks like memory management.

Outside of the kernell is the shell. The shell is the outer layer of the operating system. That's what we see when we open up a Terminal window. We're working in the shell. It interacts with the user and we can think of it as our working environment. The shell will send requests to the kernel. The kernel will then do its thing and results will then be returned back to the shell so that we can interact with them again. Mac OS X by default uses what's called the Bash shell, but it includes other choices as well.

The popular shells are `sh`, which is the very first one. It's called the *Thompson Shell*, created in 1971, and it was the main working environment that people had for a long time in Unix. In 1977 the *Bourne Shell* was created and it replaced the Thompson Shell.

We have `csh` for *C Shell*. Then we have the *Tabbed C Shell* in 1979, `tcsh`, the *Korn Shell*, `ksh`, then we have the *Bourne-Again Shell*, `bash`, that's the one we have by default. And then the *Z Shell*. It's one of the most recent. There are a lot of others. They almost all end in `sh`.

Keep in mind that the shell is not the operating system, it is just the working environment that we can choose. Actually, you can move from inside one shell into another shell without leaving the old one. We can nest them inside each other. It's just like moving into another working layer. To find out what the login shell is, type:

```
echo $SHELL
```

Files and directories

Anytime you are in a UNIX terminal, you have a *working directory*, which is your current location in the file system.

The filesystem is basically an upside-down tree. The top, or root of the tree, is the `/` directory. Within `/` there are subdirectories, such as `home`, or `Users` in Mac, (which contains users home directories where all of the files owned by a user are stored) and `bin` (containing UNIX programs, aka 'binaries').

Here's how you can see where you are

```
pwd
```

and here's how you list the files in the working directory

```
ls
```

You can use the long format option `-l` with `ls`

```
ls -l  # this lists files on 'long' format
```

Note the use of the `-l` flag to list the long-format information. As mentioned, options for UNIX commands are generally provided via this kind of syntax.

Also note that anything that follows `#` is a comment and is ignored by UNIX.

When using the `-l` flag to `ls`, you'll see information about each file (or directory) on a number of things, of which the most important are:

- (column 1) file permissions (more later)
- (column 3) the owner of the file
- (column 5) the size of the file in bytes
- (column 6-8) the last time the file was modified
- (column 9) name of the file

Now suppose I want to be in a different directory so I can see what is there or do things to the files in that directory.

The command you need is `cd` and the concept you need to become familiar with is the notion of relative and absolute *path*s. The path is the set of nested directories that specify a subdirectory of interest.

```
cd /Users/john  # go to home directory
cd stat243 # go to subdirectory based on relative path
pwd
```

Text Editors

For statistical computing and other tasks, we need a text editor that allows us to edit and modify the contents of text files. When I say text editor I don't mean a *word processor* (e.g. MS Word, Mac Pages, Google Docs). Instead, we are referring to text editors such as:

- vim, nano, emacs (traditional UNIX text editors)
- Windows: notepad, WinEdt, notepad++
- Mac: Aquamacs Emcas, TextMate, TextEdit, TextWrangler
- some more recent editors: Atom, Sublime Text
- RStudio provides a built-in editor for several types of files

Most Mac and Windows users tend choose text editors available for their platforms. If you are not used to work with text editors, you may probably want to try out a couple of editors and see which one you feel most comfortable working with. However, if you are seriously interested in *Data Science* or computing in general, you should spend some time learning one of the unix text editors (vim, nano, emacs). Why? Because those may be the only available text editors in many (remote) servers.