Unit 1: Basics of UNIX

August 12, 2021

By UNIX, I mean any UNIX-like operating system, including Linux and Mac OS X. On the Mac you can access a UNIX terminal window with the Terminal application (via `Applications -> Utilities -> Terminal` from the Finder). Most modern scientific computing is done on UNIX-based machines, often by remotely logging in to a UNIX-based server.

There are a variety of ways to get access to a UNIX-style command line environment, outlined in `howtos/accessingUnixCommandLine.html` in this repository. If you're using your own Mac, you should install the Xcode developer tools.

Any tutorials mentioned below are available at
http://statistics.berkeley.edu/computing/training/tutorials.

1 UNIX basics

Please see the material in our tutorial on the *basics of UNIX* to get up to speed on working in a UNIX-style command line environment. We'll do a bit of demo in the first class. If what you see seems quite familiar, you can probably skip that tutorial. If what you see is not familiar, please work through the tutorial and (optionally) attend section on Friday August 27 to work in groups on the problems at the end of the tutorial.

For an alternative introduction (which also includes some of the topics we'll discuss in Unit 4 on shell scripting), see this Software Carpentry tutorial.

2 Version control

A very popular and powerful tool for version control is Git. We'll be using Git extensively to do version control.

More information is available in the tutorial on the basics of using Git. You'll have a chance to work through the tutorial and practice with Git in the first section/lab. During the course, you'll

make use of Git for submitting problem sets and for doing the final project. I recommend you practice using it more generally while preparing your problem set solutions.

Git stores the files for a project in a *repository*. Here are some basic Git commands you can use to access the class materials from the command line. There are also graphical interfaces to Git that you can explore. That said, I recommend Github Desktop, available for Mac and Windows.

1. To clone (i.e., copy) a repository (in this case from Github) [*berkeley-stat243* is the project and *stat243-fall-2020* is the repository]:
   ```
   >git clone https://github.com/berkeley-stat243/stat243-fall-2020
   ```

2. To update a repository to reflect changes made in a remote copy of the repository:
   ```
   >cd /to/any/directory/within/the/local/repository
   > git pull
   ```

We'll see a lot more about Git in section, where you'll learn to set up a repository, make changes to repositories and work with local and remote versons of the repository. The section material will follow our tutorial on the basics of Git, Github, and version control.

Some links to more resources:

- Git for Scientists: A Tutorial: http://nyuccl.org/pages/GitTutorial/
- Gitwash: workflow for scientific Python projects: http://matthew-brett.github.io/pydagogue/gitwash_build.html
- Git branching demo: http://pcottle.github.io/learnGitBranching/

3 Connecting to other machines

To connect to a remote machine, you need to use SSH. SSH is available as `ssh` when you are on the UNIX command line. There are also SSH clients for Windows. For more information on SSH client programs and on using SSH, please see this webpage. Here's an example of connecting to one of the SCF machines (this assumes you have an SCF account, which you may, and that your user name is paciorek, which it is not).

```
> ssh paciorek@radagast.berkeley.edu
```

To copy files between machines, we can use *scp*, which has similar options to *cp*. The first command here copies a local file to a remote machine. The second copies from a remote machine to the machine you are on. You can use relative paths to refer to locations on the local machine. On the remote machine all paths need to be absolute or to be relative to your home directory on that machine. Again, this assumes you have an SCF account.

```
> scp file.txt paciorek@radagast.berkeley.edu:~/research/.
> scp paciorek@radagast.berkeley.edu:/data/file.txt
~/research/renamed.txt
```

There are also client programs for file transfer; see this webpage.

4 Editors

For statistical computing, we need a text editor, not a word processor, because we're going to be operating on files of code and data files, for which word processing formatting gets in the way. Don't use Microsoft Word or Google Docs to edit code files or Markdown/R Markdown/LaTeX.

4.1 Some useful editors

- various editors available on all operating systems:
  - traditional editors born in UNIX: *emacs*, *vim*
  - some newer editors: *Atom*, *Sublime Text*
- Windows-specific: *WinEdt*
- Mac-specific: *Aquamacs Emacs, TextMate, TextEdit*
- Be careful in Windows - file suffixes are often hidden
- RStudio provides a built-in editor for R code files
- VSCode has a powerful code editor that is customized to work with various languages, including nicely handling JSON, YAML, and XML files.

As you get started it's ok to use a very simple text editor such as Notepad in Windows, but you should take the time in the next few weeks to try out more powerful editors such as one of those listed above. It will be well worth your time over the course of your graduate work and then your career.

4.2 Basic emacs

Emacs is one option as an editor. I use emacs a fair amount, so I'm including some tips here, but other editors listed above are just as good.

- *emacs* has special modes for different types of files: R code files, C code files, Latex files – it's worth your time to figure out how to set this up on your machine for the kinds of files you often work on

  - For working with R, ESS (emacs speaks statistics) mode is helpful. This is built into Aquamacs emacs. Alternatively, the Windows and Mac versions of R, as well as RStudio (available for all platforms) provide a GUI with a built-in editor.

- To open emacs in the terminal window rather than as a new window, which is handy when it's too slow (or impossible) to pass (i.e., tunnel) the graphical emacs window through ssh:

```
> emacs -nw file.txt
```

Table 1: Helpful *emacs* control sequences

Sequence	Result
C-x,C-c	Close the file
C-x,C-s	Save the file
C-x,C-w	Save with a new name
C-s	Search
ESC	Get out of command buffer at bottom of screen
C-a	Go to beginning of line
C-e	Go to end of line
C-k	Delete the rest of the line from cursor forward
C-space, then move to end of block	Highlight a block of text
C-w	Remove the highlighted block, putting it in the kill buffer
C-y *(*after using **C-k** or **C-w***)*	Paste from kill buffer ('y' is for 'yank')