Introduction to UNIX, computers, and key tools

Chris Paciorek

2024-08-28

Table of contents

1. UNIX command line basics **1**

2. Version control **2**

3. Parts of a computer **2**

4. Connecting to other machines **3**

5. Editors **3**

Summary of some useful editors . . . 3

(Optional) Basic emacs . . . 4

(Optional) Emacs keystroke sequence shortcuts (aka, *key bindings*). . . . 4

(Optional) Basic vim . . . 4

PDF

1. UNIX command line basics

By UNIX, I mean any UNIX-like operating system, including Linux and MacOS. 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.

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 and/or the material in last week's Computing Skills Workshop. 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 by Wednesday September 4 and (optionally) attend section on Friday August 30 to work on the problems at the end of the tutorial and be able to ask questions of the GSI and other students.

For an alternative introduction (which also includes some of the topics we'll discuss in Unit 3 on shell scripting), see this Software Carpentry tutorial.

2. Version control

Version control refers to the process of keeping track of changes to your materials on a computer, often code files but other files as well. It generally works well with text files because version control operates based on differences between two versions of a file, and those differences are kept track of on a line-by-line basis. Version control is a central concept/practice in modern scientific computing.

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

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. I've heard good things (albeit a few years ago) about GitHub Desktop, available for Mac and Windows.

1. To clone (i.e., copy) a repository (in this case from GitHub) (`berkeley-stat243` is the organization and `fall-2024` is the repository):

   ```
   git clone https://github.com/berkeley-stat243/fall-2024
   ```

2. To update a repository to reflect changes made in a remote copy of the repository:

   ```
   cd /to/any/directory/within/the/local/repository  ## e.g., cd fall-2024
   git pull
   ```

More information is available in the Computing Skills Workshop as well as this tutorial on the basics of using Git, as well as lots of information/tutorials online. We'll see a lot more about Git in Section/Lab sessions, where you'll learn to set up a repository, make changes to repositories and work with local and remote versons of the repository. In particular, you'll have a chance to work through the tutorial and practice with Git in the first section/lab (September 6). During the course, you'll make use of Git for submitting problem sets and for doing the final project. You should practice using it more generally while preparing your problem set solutions, and you'll need to use it extensively for the final group project.

3. Parts of a computer

We won't spend a lot of time thinking about computer hardware, but we do need to know some of the basics that relate to using a computer and programming effectively. There are many layers of technical detail one could get into, but the key things for now are to have a basic understanding of these components of a computer:

- CPU
  - cache (limited fast memory easily accessible from the CPU)
- main memory (RAM)
- bus
- disk

Depending on what your code is doing, the slow step in a computation might be:

- doing the actual calculations on the CPU,
- moving data back and forth from (main) memory (RAM) to the CPU, or

- reading/writing (I/O) data to/from disk.

Please read this overview from a class at CMU that is similar to this class. Don't worry about too much of the details in the "How Programs Run" section - just try to get the main idea.

4. Connecting to other machines

To connect to a remote machine, you generally need to use SSH. SSH is available as `ssh` when you are on the UNIX command line. There are also SSH clients for Windows. The SCF has more information on SSH client programs and on using SSH, including connecting without entering your password. 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*. Here's how we can copy a local file to a remote machine.

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

And here's how we can copy from a remote machine to the machine you are on.

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

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.

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

5. Editors

For scientific computing, we need a text editor, not a word processor, because we're going to be operating on code files, plain-text data files, and markup language documents (e.g., Markdown/Quarto) for which word processing formatting gets in the way. **Don't use Microsoft Word or Google Docs to edit code files or Markdown/Quarto/R Markdown/LaTeX.**

Summary of some useful editors

- various editors available on all operating systems:
  - traditional editors born in UNIX: *emacs*, *vim*
  - some newer editors: *Atom*, *Sublime Text* (Sublime is proprietary/not free)
- Windows-specific: *WinEdt*
- Mac-specific: *Aquamacs Emacs*, *TextMate*, *TextEdit*
- RStudio provides a built-in editor for R code and Quarto/R Markdown files. One can actually edit and run Python code chunks quite nicely in RStudio. (Note: RStudio as a whole is an IDE (integrated development environment. The editor is just the editing window where you edit code (and Markdown) files.)

- *VSCode* has a powerful code editor that is customized to work with various languages, and it has a Quarto extension.

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.

Be careful in Windows - file suffixes are often hidden.

(Optional) 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: Python code 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
  - If working with Python and R, one can start up a Python or R interpreter in an additional Emacs buffer and send code to that interpreter and see the results of running the code.
  - For working with R, ESS (emacs speaks statistics) mode is helpful. This is built into Aquamacs Emacs.
- 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`

(Optional) Emacs keystroke sequence shortcuts (aka, *key bindings*).

> **Note** Several of these (Ctrl-a, Ctrl-e, Ctrl-k, Ctrl-y) work in the command line, interactive Python and R sessions, and other places as well.

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

(Optional) Basic vim

vim is another option as an editor. Like emacs, it's been around for a long time, and some of the other options above are probably more user friendly. However, it can be helpful to know how to do some

basic things in vim.

For example, if you run `git commit` without the `-m` flag to add a message, you'll be put in a vim editor window by default (you can also modify what editor git uses).

vim has two modes: `normal` mode, which allows you to carry out various operations (such as navigation, saving files, moving and deleting lines) and `insert` mode, which allows you to actually insert text.

To get into `insert` mode from `normal` mode, type "i". To get back to `normal` mode, press `Esc`.

When in normal mode, you can type `:w` to save, `:x` to save and exit, and `:q` to exit. To search a document for a string (e.g., "python docstring", type `/python docstring` and return/enter. Type `Esc` to get out of the search.