Ph 20.2 – Unix Shell, Make Files, and Version Control

Due: Week 3 -v20190930-

This Assignment

The previous assignment have introduced the basic use of Python for scientific computing and numerical computing. In this assignment, we take a step back; instead of focusing on writing code we focus on the tools you might use to maintain or preserve your code and to automate the process of running it. In particular, this assignment should familiarize you with the basics of three important tools used in scientific computing on a Unix platform:

- Version Control (git)
- Makefiles
- The Unix Shell

Understanding how to work with these tools will be particularly important if you are working on code with a number of other people rather than on your own, as is the case in most large scientific collaborations.

This assignment will primarily refer you to tutorial material from the web site *Software Carpentry*. You're welcome to track down and use other resources, but your TAs will be most familiar with, and thus better able to answer questions about, the Software Carpentry tutorials.

Version Control

Collaborative projects, as well as some individual projects, benefit significantly from putting the project under version control. The most widely used version control system in Unix is probably *git*, especially due to the widespread useage of `https://github.com` in the open-source and academic communities as a place for hosting code and other materials, including by Software Carpentry itself. The Software Carpentry tutorial on Version Control with Git is located at `http://swcarpentry.github.io/git-novice/`.

There is no central repository in git, rather a git repository is created in each folder where git is initiated. Also, git can operate both from the command line or via any number of GUIs.

Numerous alternatives to Git exist. If you would prefer to use a different version control system, discuss it with your TA. An older alternative to *git* is *svn*. The site `http://maverick.inria.fr/~Xavier.Decoret/resources/svn/index.html`, "SVN Tutorial fo Unix" (sic), may also be useful for some of the details of setting up an svn repository. *mercurial* is another alternative and is python-based.

The Assignment - Part I

1. Make a git (or svn or mercurial) repository for your Ph2x work.
2. Import a "project", in this case your assignment Ph20.1, *Introduction to Python*. Discuss with your TA if you would rather define a different project, however see Part II of the assignment below.
3. Check out a working version of the Ph20.1 project into a new directory.

4. Make some changes (any changes) to the working version of the project.

5. Commit the changes using git (or alternative).

Reading the first five sections of the Software Carpentry tutorial should walk you through all of the steps needed to complete this part of the assignment. You now should be able to use a version control system for tracking versions of your own work as well as creating collaborative projects.

Reading the steps in Section 7 of the Software Carpentry tutorial, you will learn how to host your repository remotely on GitHub. For this class, we have set up a Gitlab at `https://bayard.caltech.edu` (you can register with a Caltech email address) and you can create your own projects there. From now on, you are recommended to upload your classwork of the following assignments to your Gitlab repository and share your TA with a link to your repository. Ask your TA for help if you find any difficulties in submitting classwork in this way.

Makefiles

As your programs become more complex, requiring compiling and linking of multiple subprograms, routines, and libraries, the shell command, *make* , can be very useful.

Make is a souped up version of shell scripts, and (through multiple implementations) is itself a Turing-complete language. Although it is normally used to build large C++ libraries, here we will use it to simply build an assignment solution by automating all the processes that you usually have to do by hand. The most prominent advantage to doing this is that if you have to fix one basic thing at an input file level, it's not necessary to redo everything else. Simply type 'make' and the pdf is automatically regenerated. Unfortunately, the language used by Make dates to 1976 (!) and can seem extremely unintuitive compared to, say, Python.

The Software Carpentry tutorial on Make is located at `http://swcarpentry.github.io/make-novice/`. You can also look at the Wikipedia entry for "makefile" or "Make (software)" for useful information and links to additional tutorials.

The Assignment - Part II

1. Write a makefile which produces a completed Ph20 assignment with the intent of using a single *make* command to generate the completed assignment. Use Assignment Ph20.1, *Introduction to Python* as the assignment, or discuss an alternative with your TA (see above). It is possible to write a very basic makefile using only the information from the first two sections of the Software Carpentry tutorial. However, we can do better than that:

2. The makefile should include pre-requisites and dependencies as discussed in the fourth section of the tutorial. To create a dependency structure it may be useful to alter your Python code from the last assignment so that it can produce one plot at a time depending on the command-line arguments it has given. Review the first assignment if you've forgotten how to take arguments from the command line.

3. And the makefile can be made less redundant and pithier using automatic variables, pattern rules, and variables, as discussed in the eponymous sections of the tutorial. In particular, if you have altered your Python code as just discussed, you can use these techniques to combine the rules for making each required image into a single rule.

4. Of course, put the makefile under version control in the repository from the previous part of the assignment.

5. Finally, use your completed makefile to produce the output for the entire assignment using a single *make* command. The completed assignment pdf should include the version control log, the makefile itself, the source code, and possibly command-line output, as well as some portion of the original 20.1 writeup including several plots. This should not be done by hand, but rather with a particular LaTeX package, such as `\verbatiminput{text.txt}`, `\inputminted`, and `\lstinputlisting`.

The Unix Shell (optional reading, for those who are interested)

You should already be familiar with some flavor of the Unix Shell as a means of navigating the Unix filesystem or running programs—it's what you've been using to run `ipython`, `emacs`, and `pdflatex`, for example, and which you have just used to set up version control and run makefiles. However, the Unix shell combined with some built-in Unix utilities such as `grep` can also be used as a powerful programming environment in its own right, for example to run an operation on an entire set of files simultaneously or to search a set of files for some particular string of text. If you're interested in learning how to do this, a good starting point is the Software Carpentry tutorial: `http://swcarpentry.github.io/shell-novice/`. The first several parts of the lesson should be very familiar to you at this point, but the last four sections are a good introduction to the use of the shell as a programming environment. Even if you have no particular interest in shell scripting you may want to skim the module on "Pipes and Filters", one of the most important capabilities of the Unix Shell, as well as the "Finding Things" module on the use of `grep`.