An Introduction to Version Control with Git

Activity for the 2015 EarthScope Data Processing Workshop

Charles J. Ammon, Penn State

Follow along with this activity on your own.

Introduction

Goal: Demonstrate the utility of ***git*** sufficiently for you to recognize the importance of learning more about it.

As a scientist is essential that you work to produce reproducible results. An important part of that process is keeping reords of your work. A scientific notebook is probably the most important part of that effort. For computational work, having a record of code changes is just as important. The importance of tracking changes to computer software has led to the development of source-version-control packages that allow you to document and record changes made to software - or at this point, any text files that you might be modifying.

Whether you are writing C, python, fortran, or matlab scripts, these computer tools provide you with an opportunity to keep a history of your code modifications. They also allow you to "roll back" any changes that may not have been a good idea.

Work through the following exercise to become familiar with the basics of git (and to gain some more experience with the Atom editor).

An Introduction to Version Control with Git

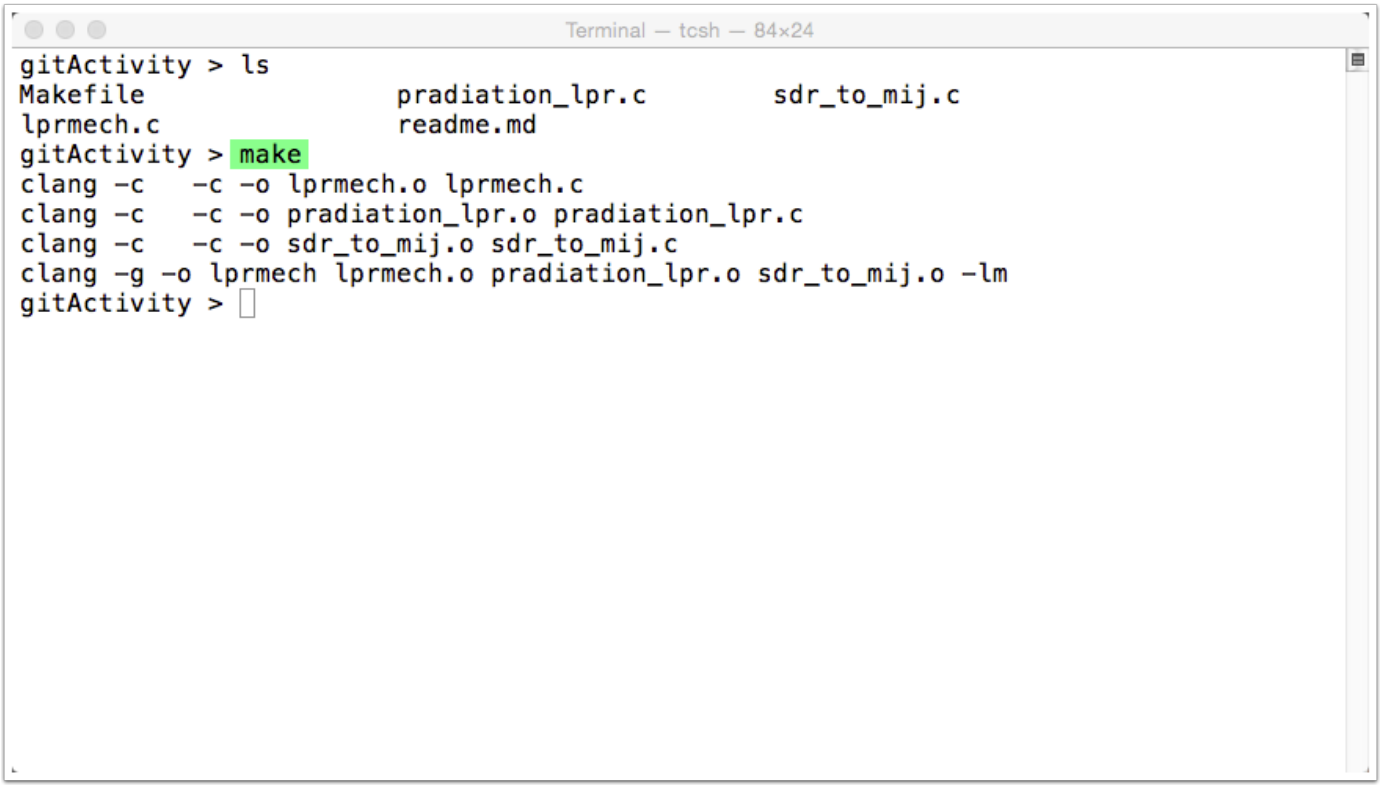
Activity Setup

Open a terminal and "***cd***" to the folder containing the files for this exercise (.../Editors_and_Codes/gitActivity). The folder contains a *readme.md* file in markdown format and three C source files and a makefile that are used to build a simple command-line utility that displays a line-printer version of an earthquake focal mechanism.

An Introduction to Version Control with Git

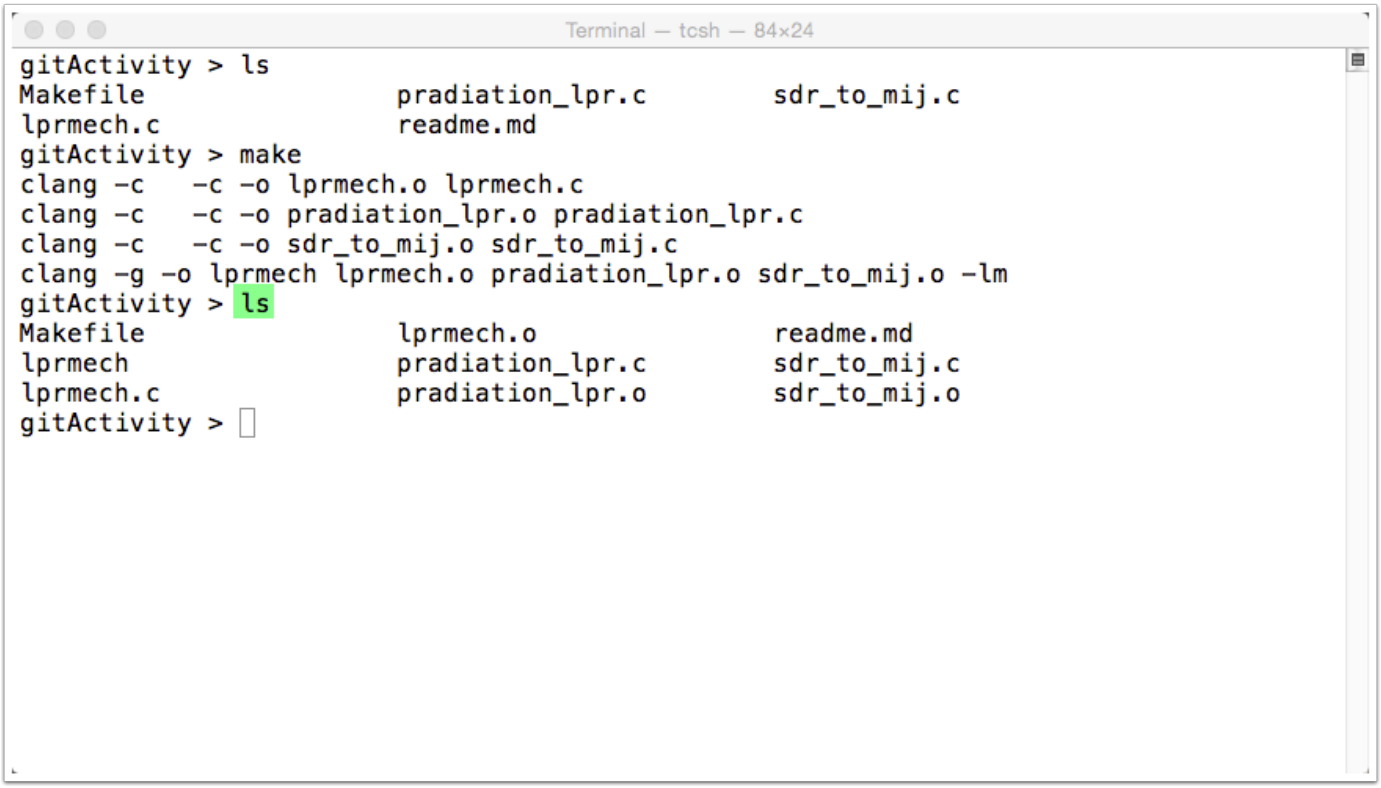
Build the Exceuteable

You can build the executable file using the ***make*** command. ***Make*** is a powerful tool for building codes containing multiple files (but not too many, ie. hundreds or thousands). We don't have time to get into it, but it is worth learning.



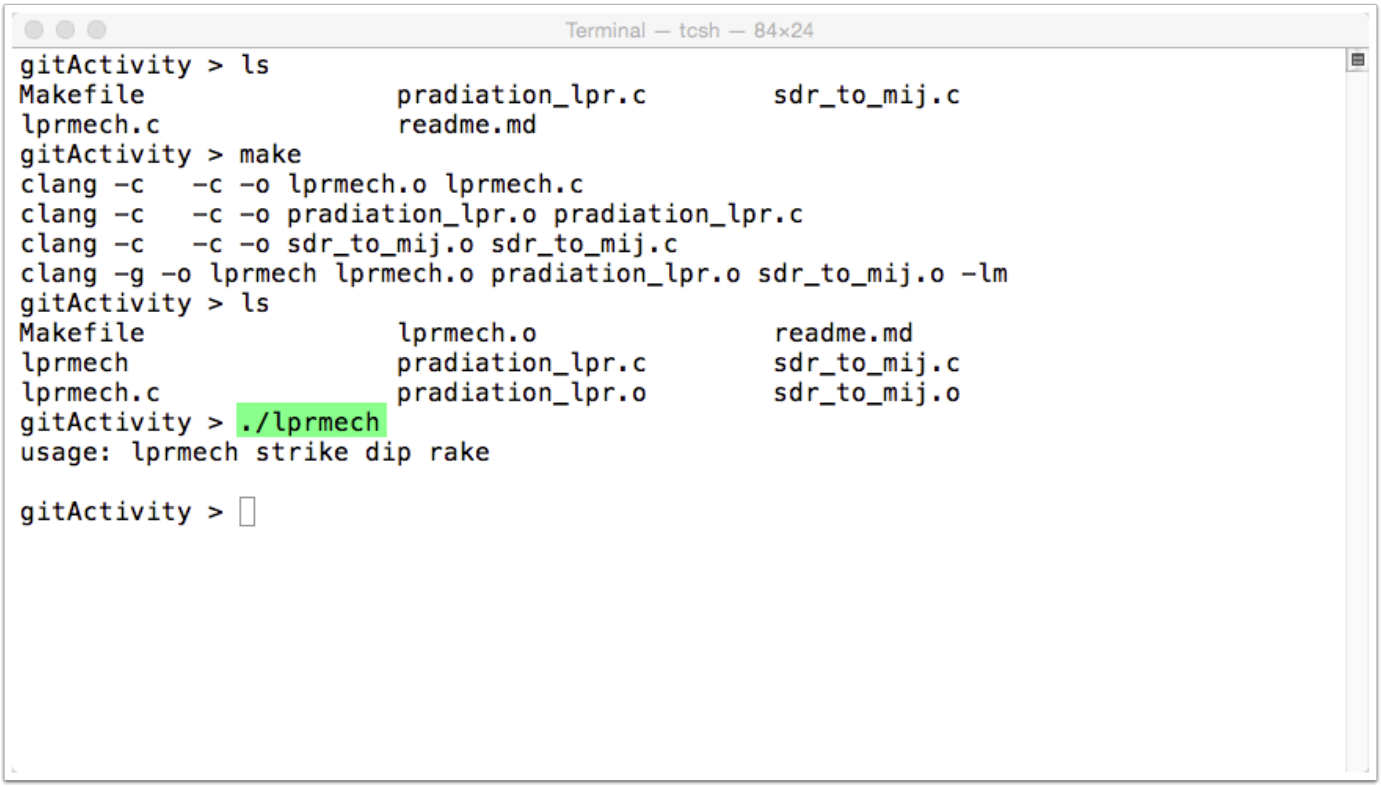
An Introduction to Version Control with Git

You can list the files again to see what make did. The executeable is called ***lprmech***, the files that end in ".o" are compiled (binary object files) versions of the C code created with the ***clang*** compiler. The last line in the make output is the linking step that linked the object files together to create the executable.

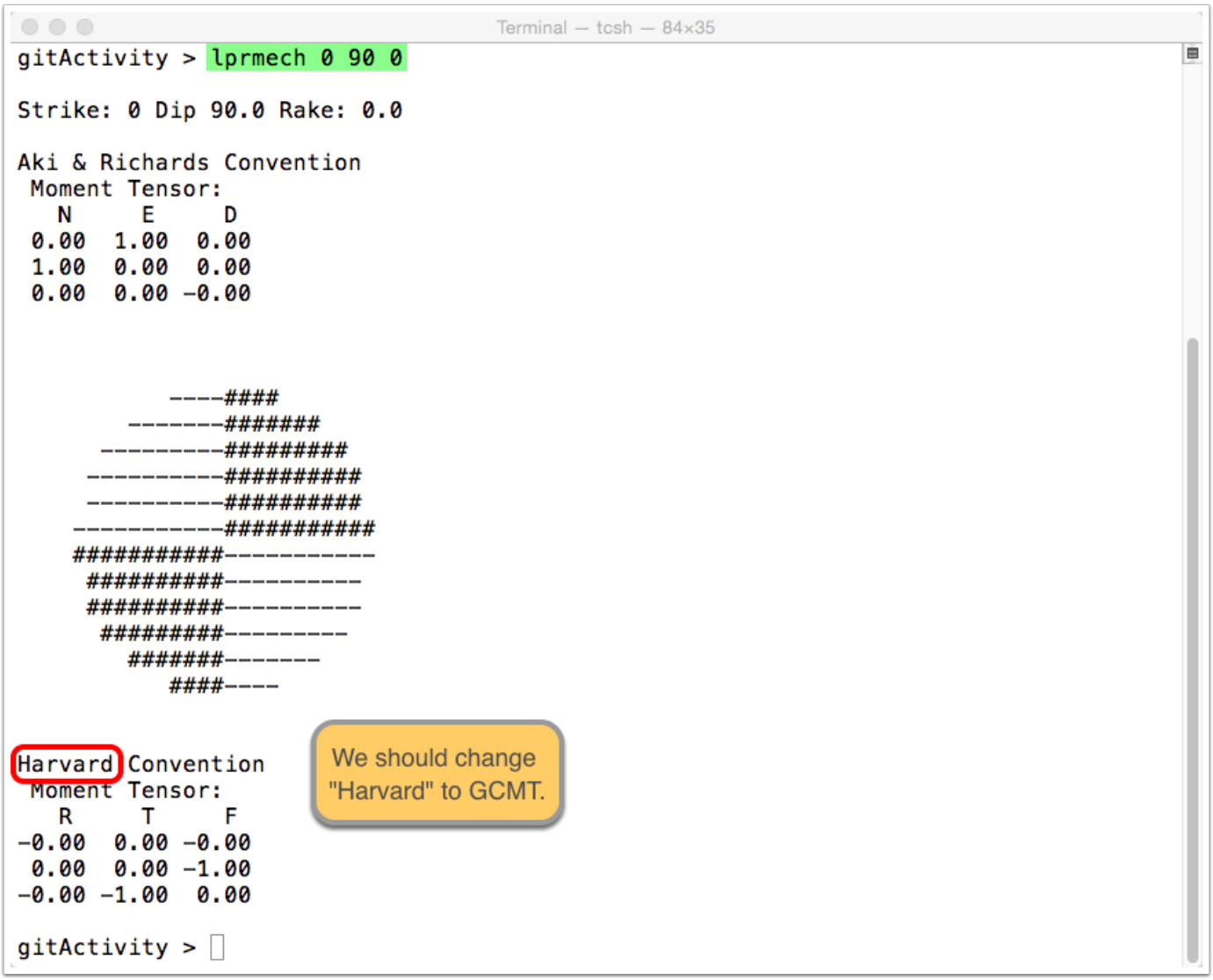


Test the Executable

To run the program, enter ***./lprmech***, you'll see a short usage statement showing you that the command accepts three command-line arguments, strike, dip, and rake.



If you provide values for the strike, dip, and rake, you can see what this simple utility does. Here's a sample output.



The code is quite useful for a quick display of focal mechanisms (that you can easily paste into email messages or notes). There are some things that we should change, however. One of the changes that we want to make is highlighted above. The new

name of the the centroid-moment-tensor project is the Global Centroid Moment-Tensor project, no longer the Harvard Centroid Moment-Tensor project. Before we tinker with a working code, however, we should use git to save the working version and track the changes that we make in a way that we can undo our changes, in case we make an error.

Using Git

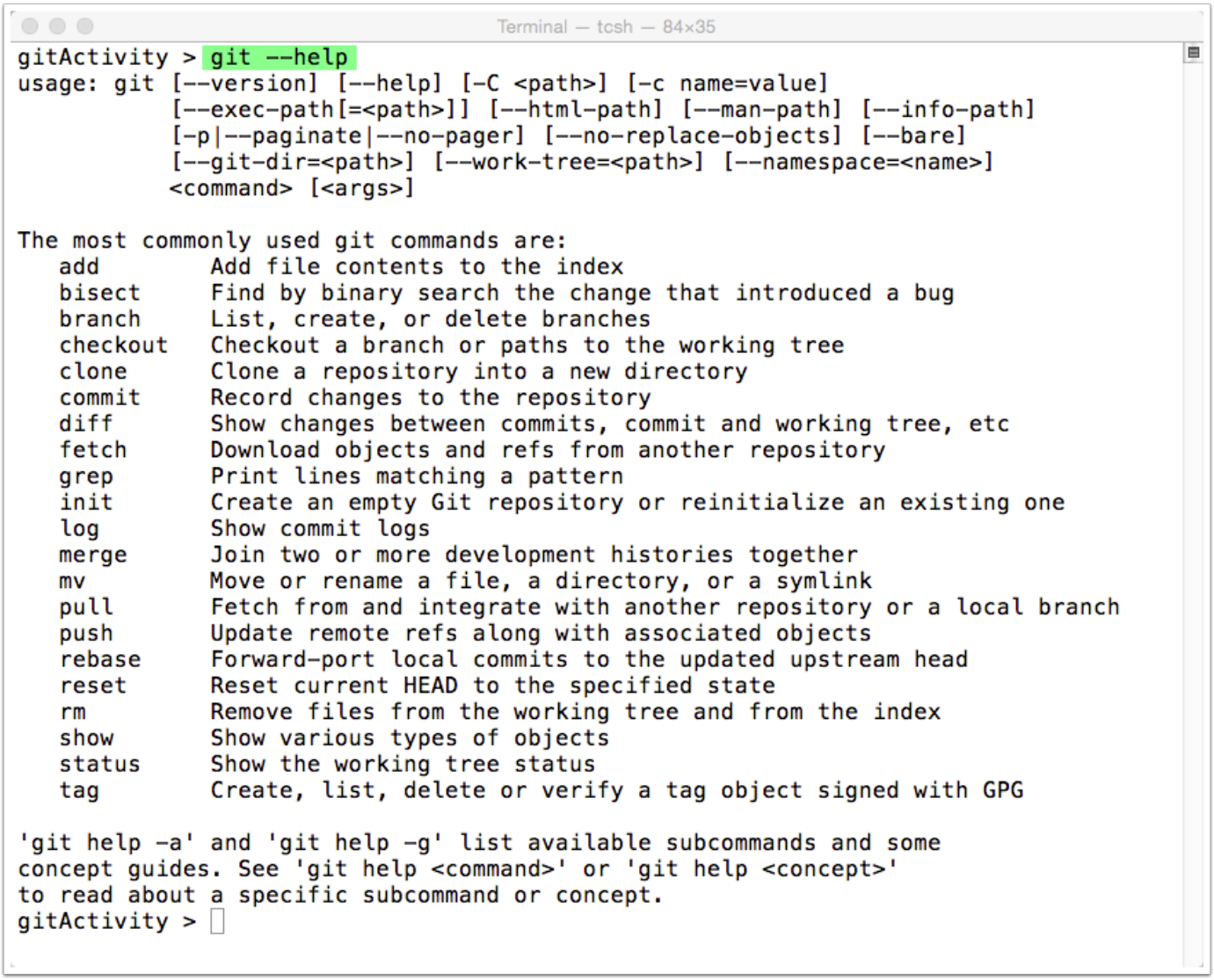
What we want to do next is create a local git repository to track changes that we are going to make in the code. ***git*** should be installed on the computers that you are using. If it's not on your computer, you can install it relatively simply. Follow the instructions at http://git-scm.com.

To see if ***git*** is live on the machine you are using, enter ***which git*** at the unix prompt.

An Introduction to Version Control with Git

git Help

You can get help on git from the unix man page, enter ***man git*** at the unix prompt or entering ***git --help*** at the prompt. There are also many web tutorials and at least one free ***git*** book available online.



We are not going to use most of those commands in this brief activity. We will focus on some basic frequently used commands.

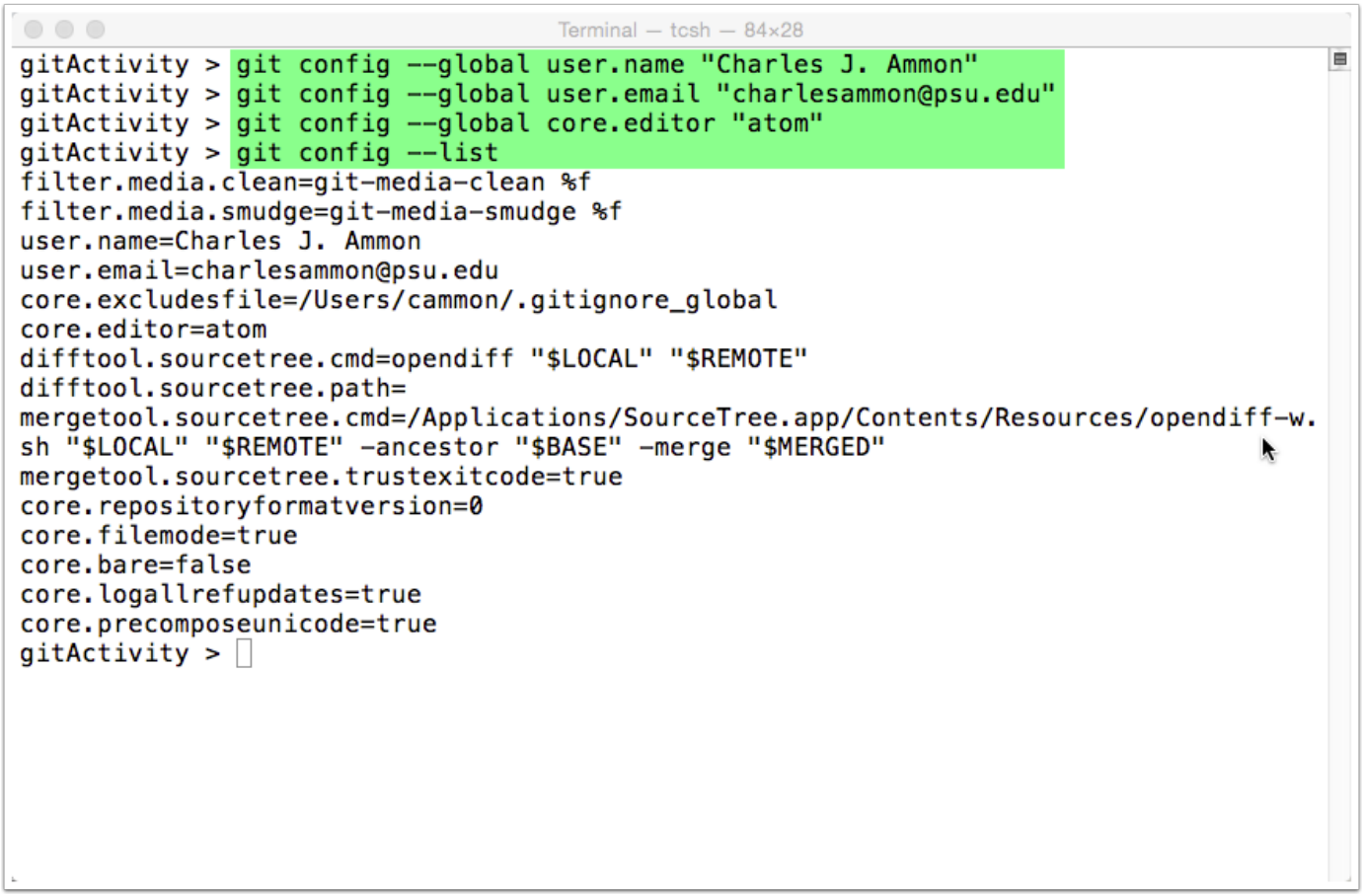
An Introduction to Version Control with Git

Git Configuration

We'll want git to know whow made changes and how to contact that person, so before we get too deep into the example, we'll configure that information into git. You'll want to substitute your information in place of mine. We use the git config command to set three items

- Our name
- Our email
- Our preferred editor

In the terminal, enter the commands as shown below.

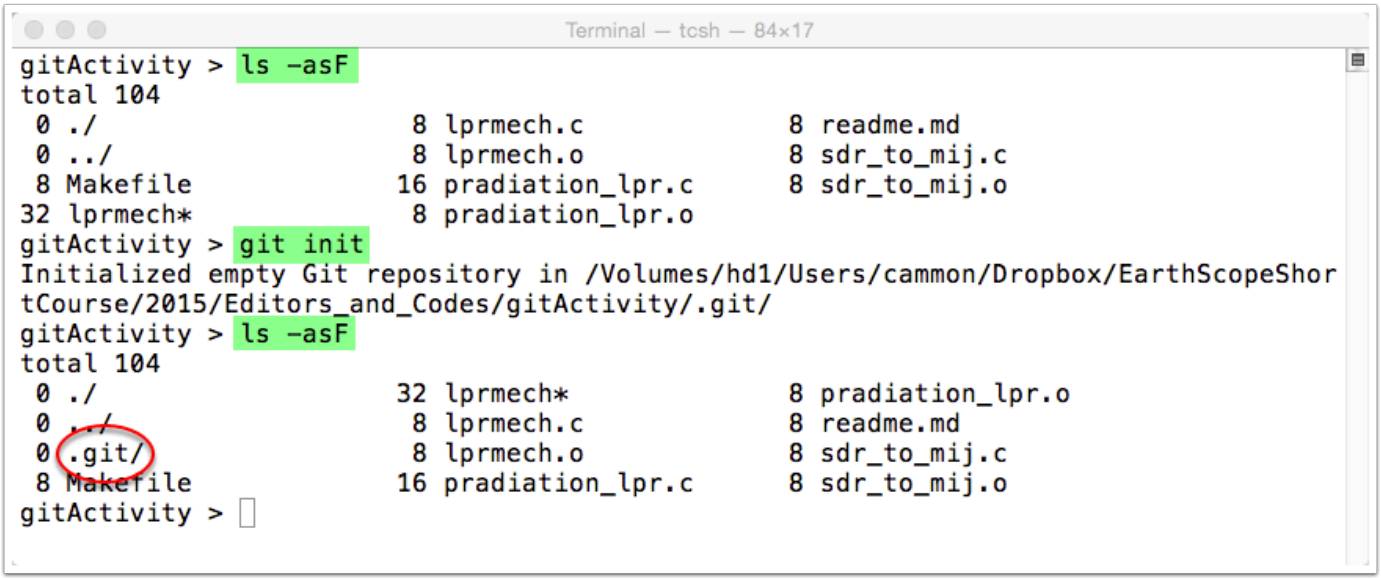


An Introduction to Version Control with Git

Creating the Git Repository

To create an empy git repository we simply use the git init command in the directory containing the files fpr which we plan to track changes. In our cse, that's the folder that we've been working in (with the C files, makefile, and readme.md). Enter the ***git init*** command in the terminal and hit return.

Here is the result that you should see. I started with an ***ls -asF*** command so that we could see hidden files and folders in the file list. The I entered ***git init***, and again listed the files. git created a subdirectory called ***.git*** inside the directory.

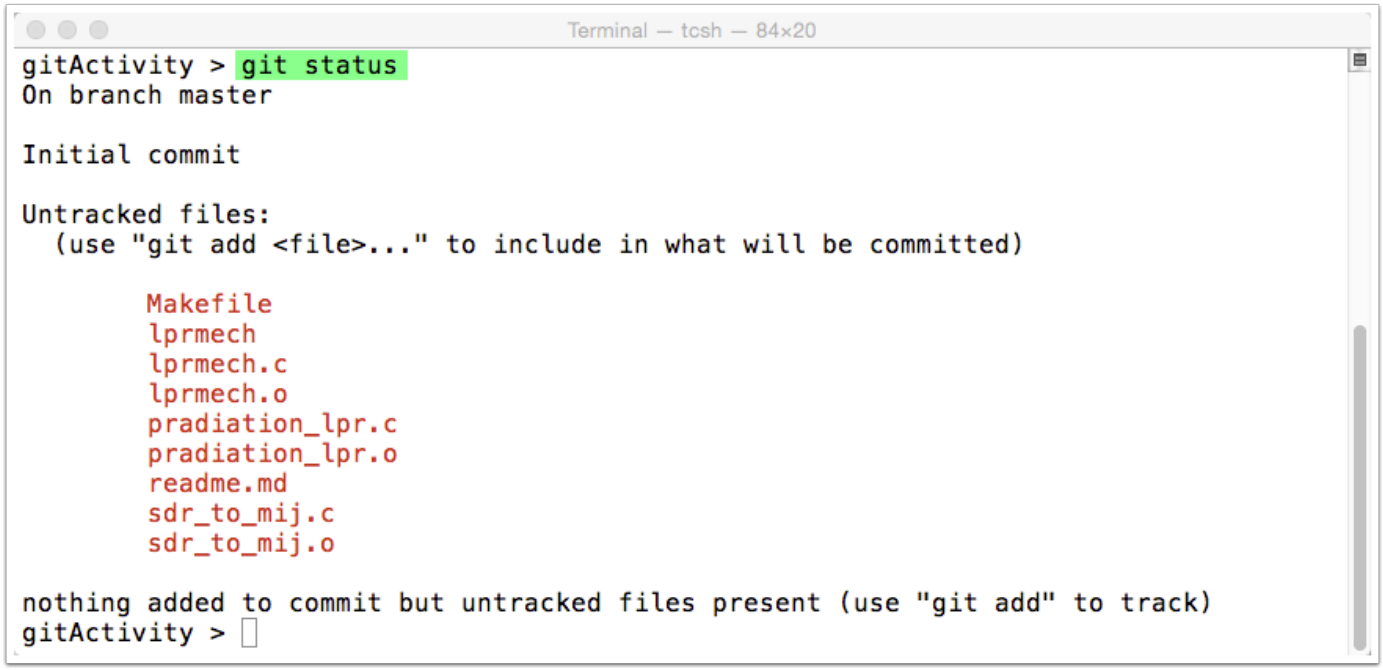


All the information needed by ***git*** will be placed inside the ***.git*** subdirectory. You can move the entire directory containing the source files and .git without affecting git's ability to track changes.

Right now, the ***.git*** folder is empty, nothing is stored within it, our first step using git is to add the files that we want to track to the repository. Adding changes to the

repository is a two-step process. First we ***stage*** the changed files (and right now all files are changes since the repository is empty), then we ***commit*** the changes.

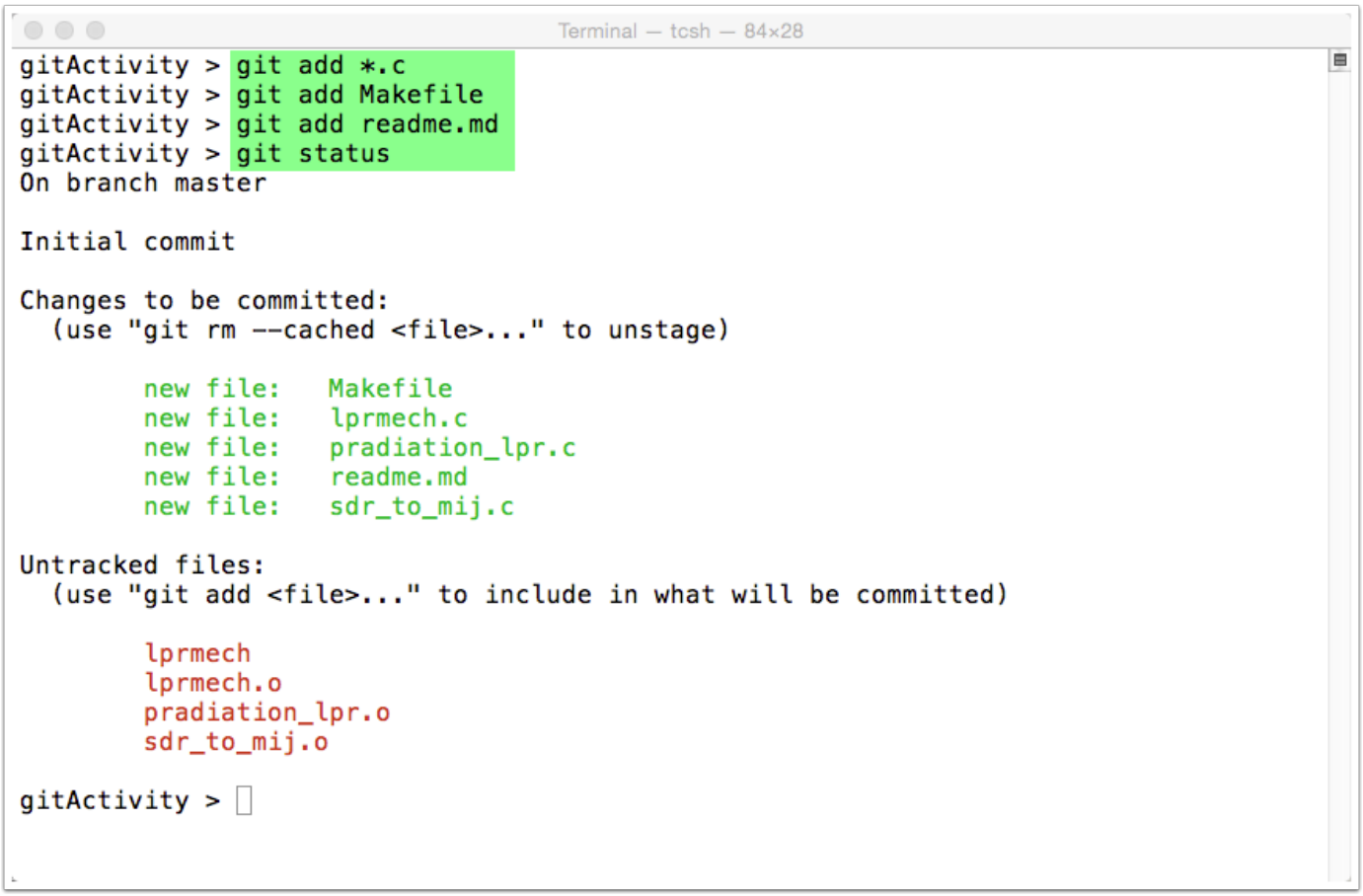
We'll start with an oft-used git command, git status. That will show us what files git recognizes in our project.



Staging Files for a Commit

git tells use that there's nothing to commit at this point, but the directory contains untracked files. We want git to track the C codes, the Makefile, and the readme.md file. There's no point in tracking the *.o files since they are created from the C codes and the make file.

We stage or ready the files of interest for commit using the ***git add*** command.

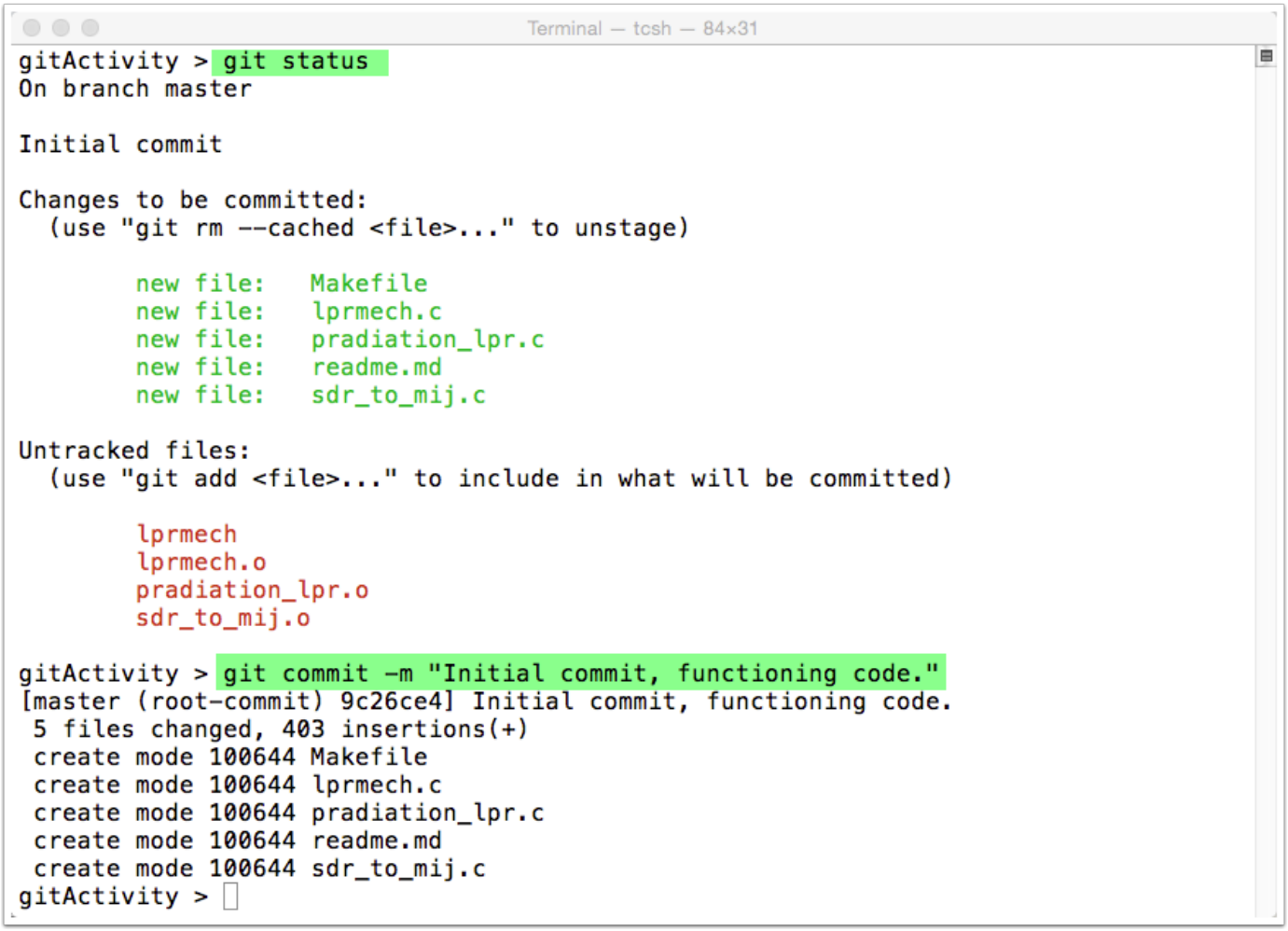


Again, ***git*** provides some hints for what we might want to do next in the status command. If we wanted to unstage a file we can use ***git rm --cached <file>*** or we can add some of the untracked files.

Commiting Changes to the Repository

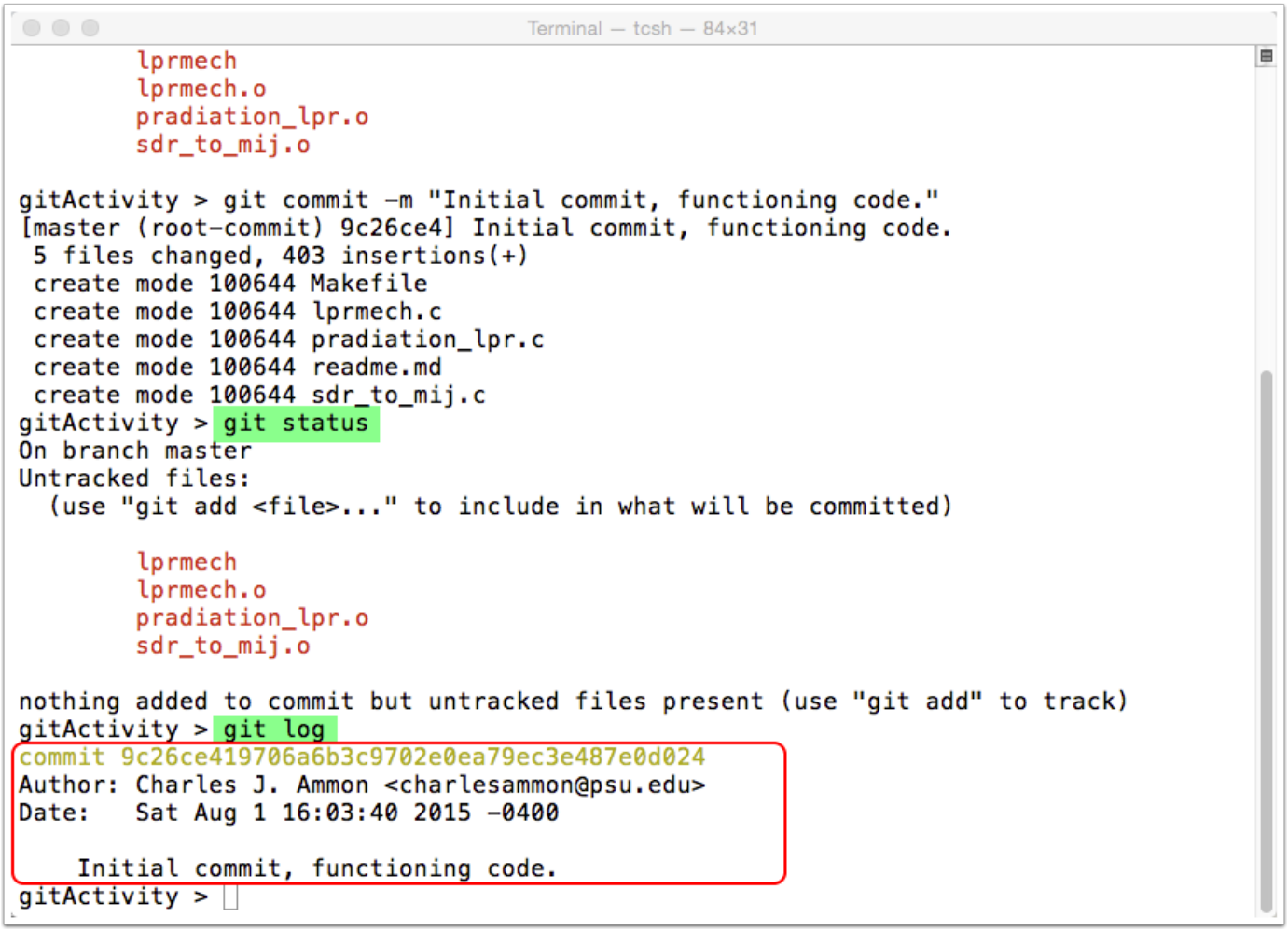
We are now ready to create the initial commit of the source codes to the repository. An important part of committing changes is describing those changes. This can be done on the command line using

git commit -m "description ..."



An Introduction to Version Control with Git

We can check the status of our repository using the ***git status*** command, and then check the history of the respository using the ***git log*** command.

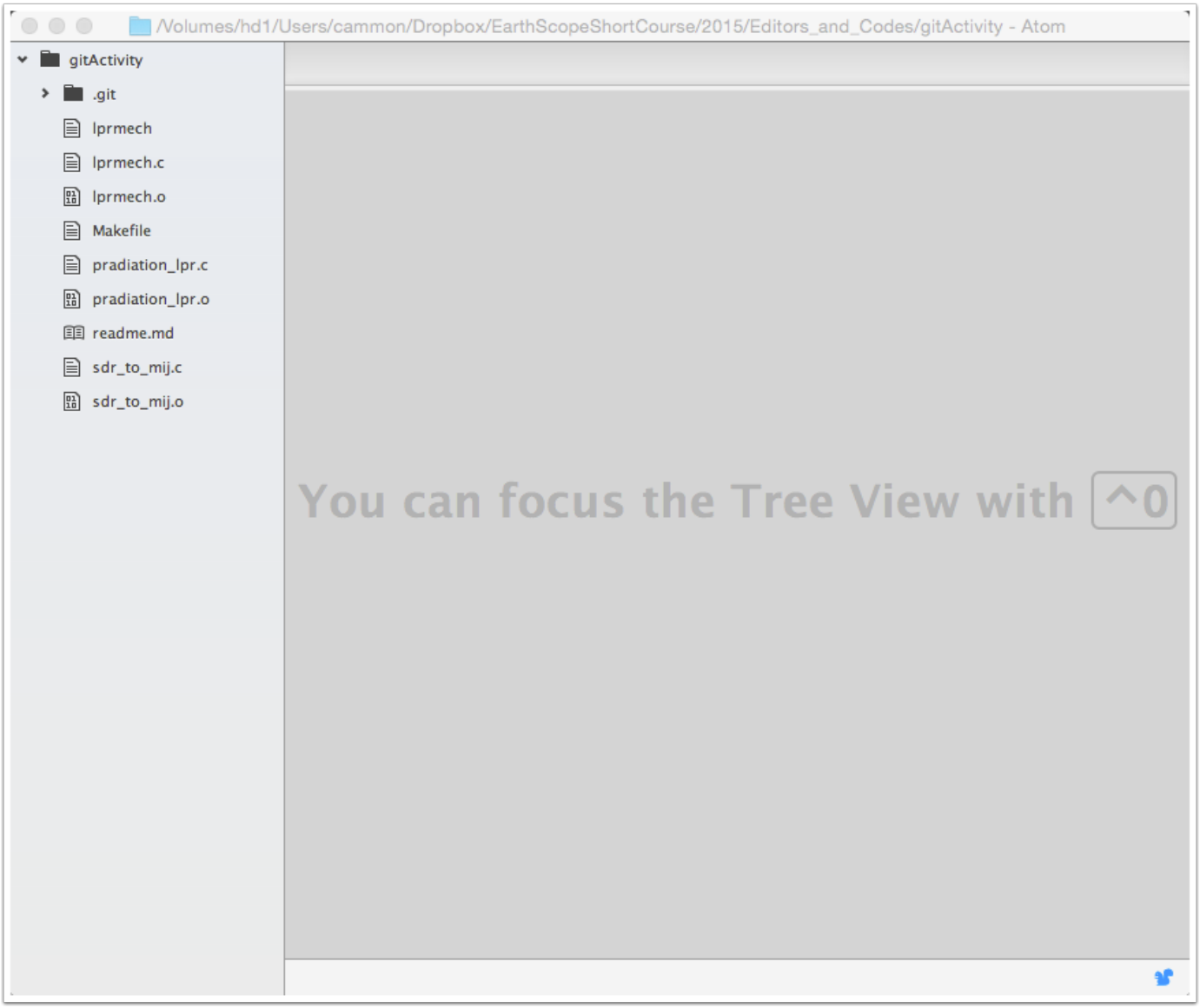


The ***git log*** command shows us the "hash" (the long string of hexidecimal numbers), the author, the date, and the message describing this commit. If at some point we want to revert back to this state of code, we would use the hash to identify this particular snapshot of the codes.

Changing the Codes

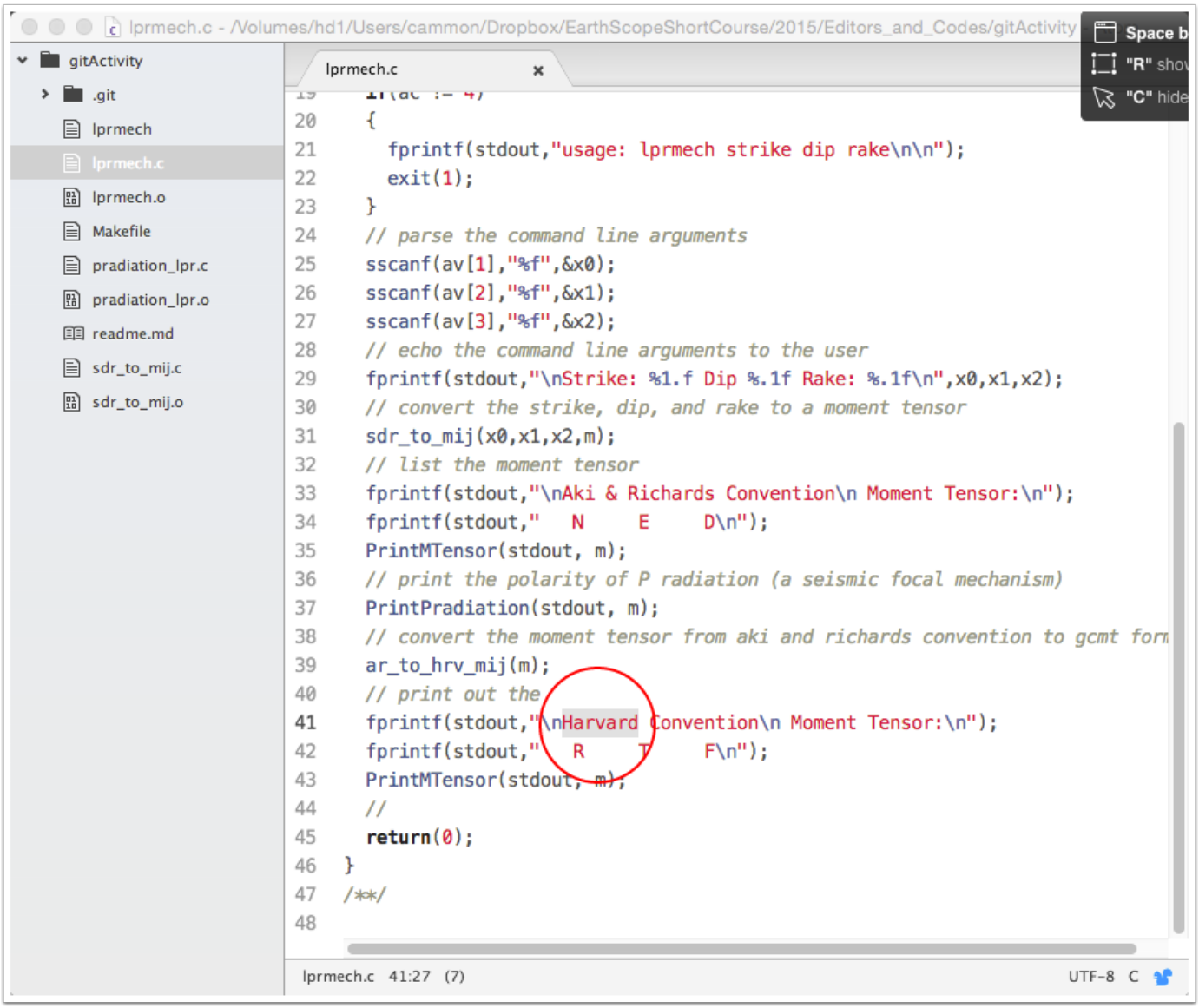
Now that we have the original codes safely recorded in the git repository, we can begin to modify the them without fear of losing the original working version (and without producing a suite of directories called "original", "new", "newer", etc.

Start up Atom and open the directory containing the ***lprmech*** source codes.



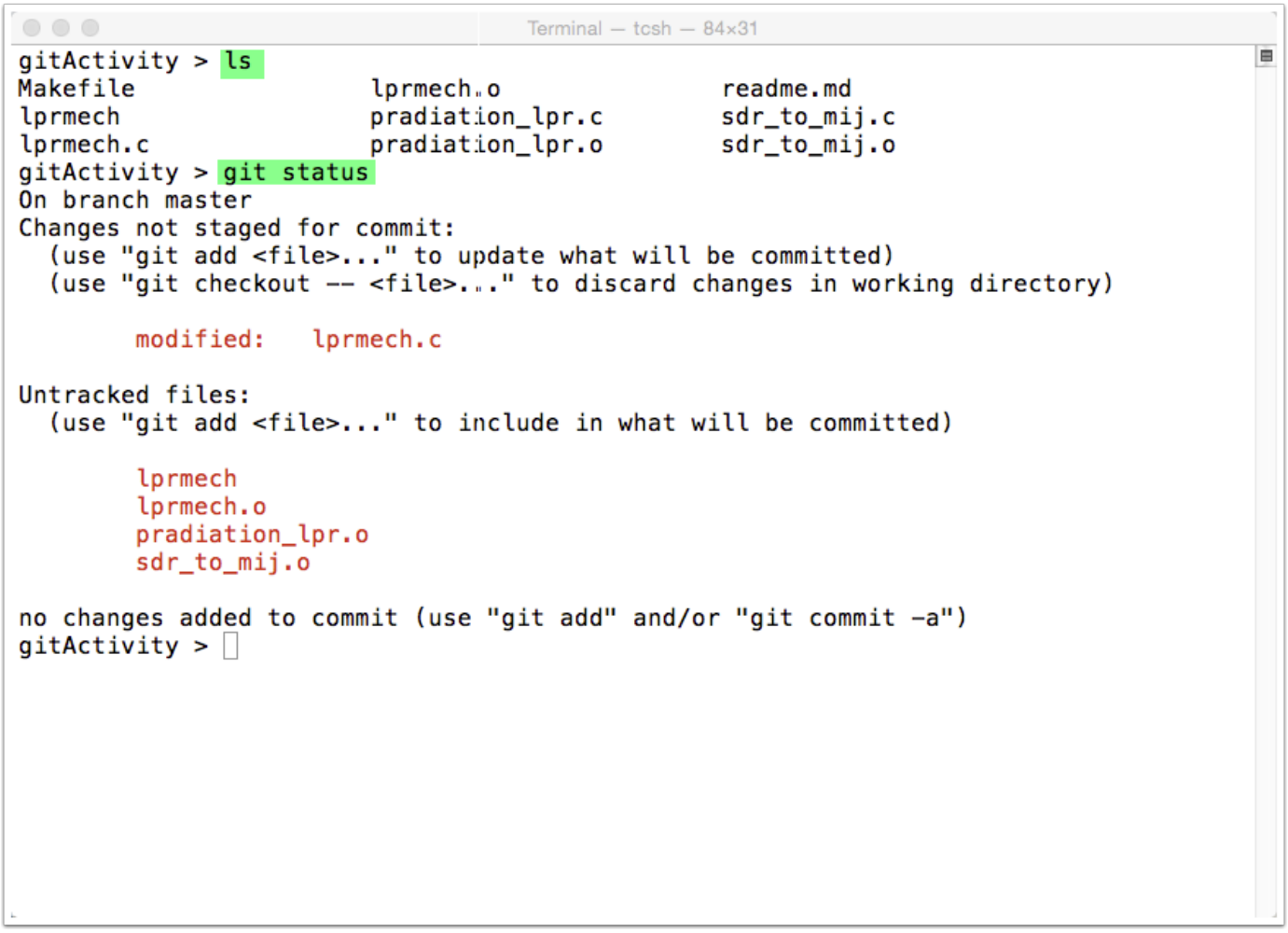
An Introduction to Version Control with Git

Single-click the lprmech.c file so that we can change the output statement to include "GCMT Convention" in place or "Harvard Convention". The change should be made on line 41. Select the word Harvard and replace it with GCMT.



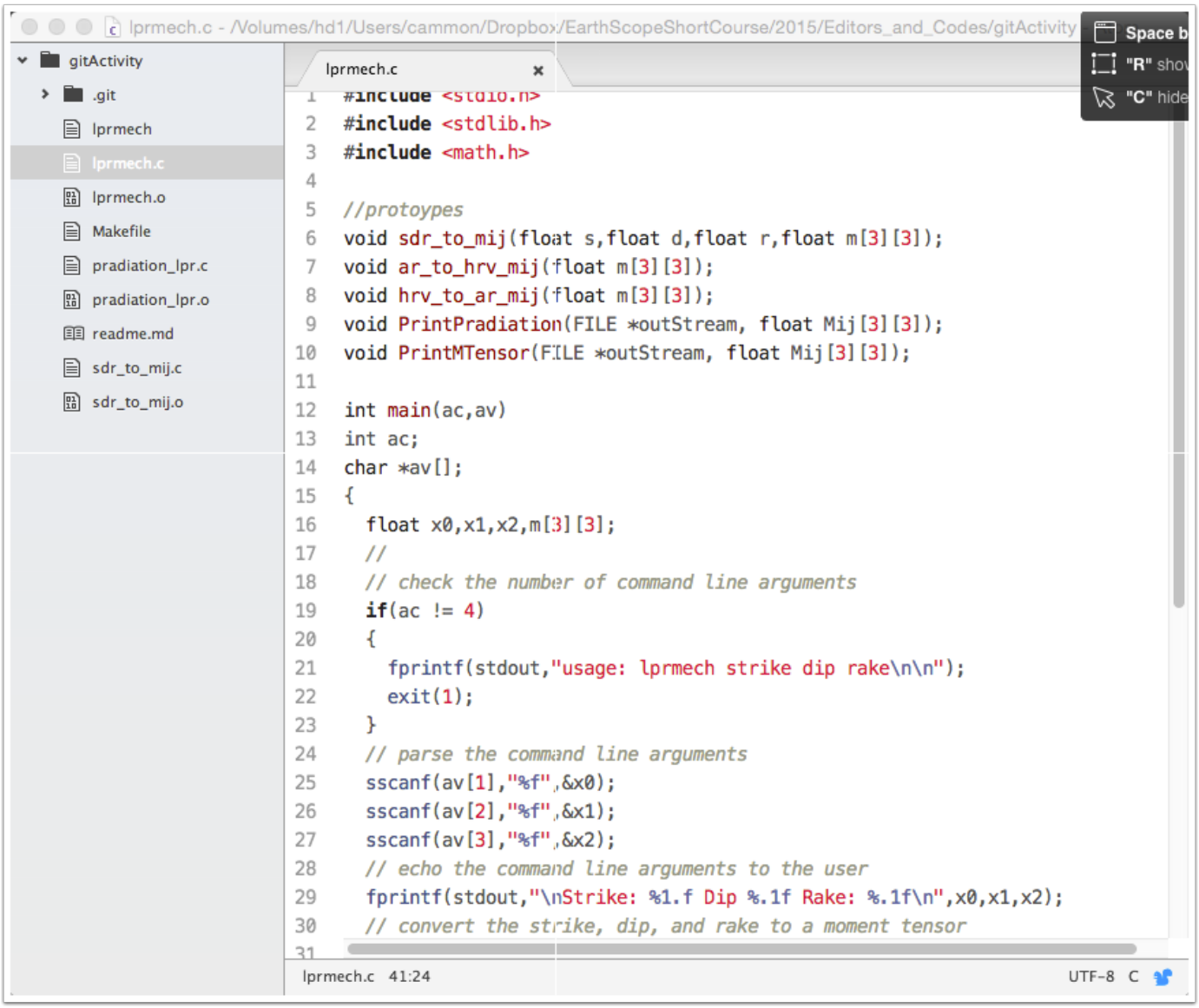
Save the change to lprmech.c and go back to the terminal and execute a ***git status*** command.

You will see that git has recognized a change to the tracked file ***lprmech.c,*** as shown below.



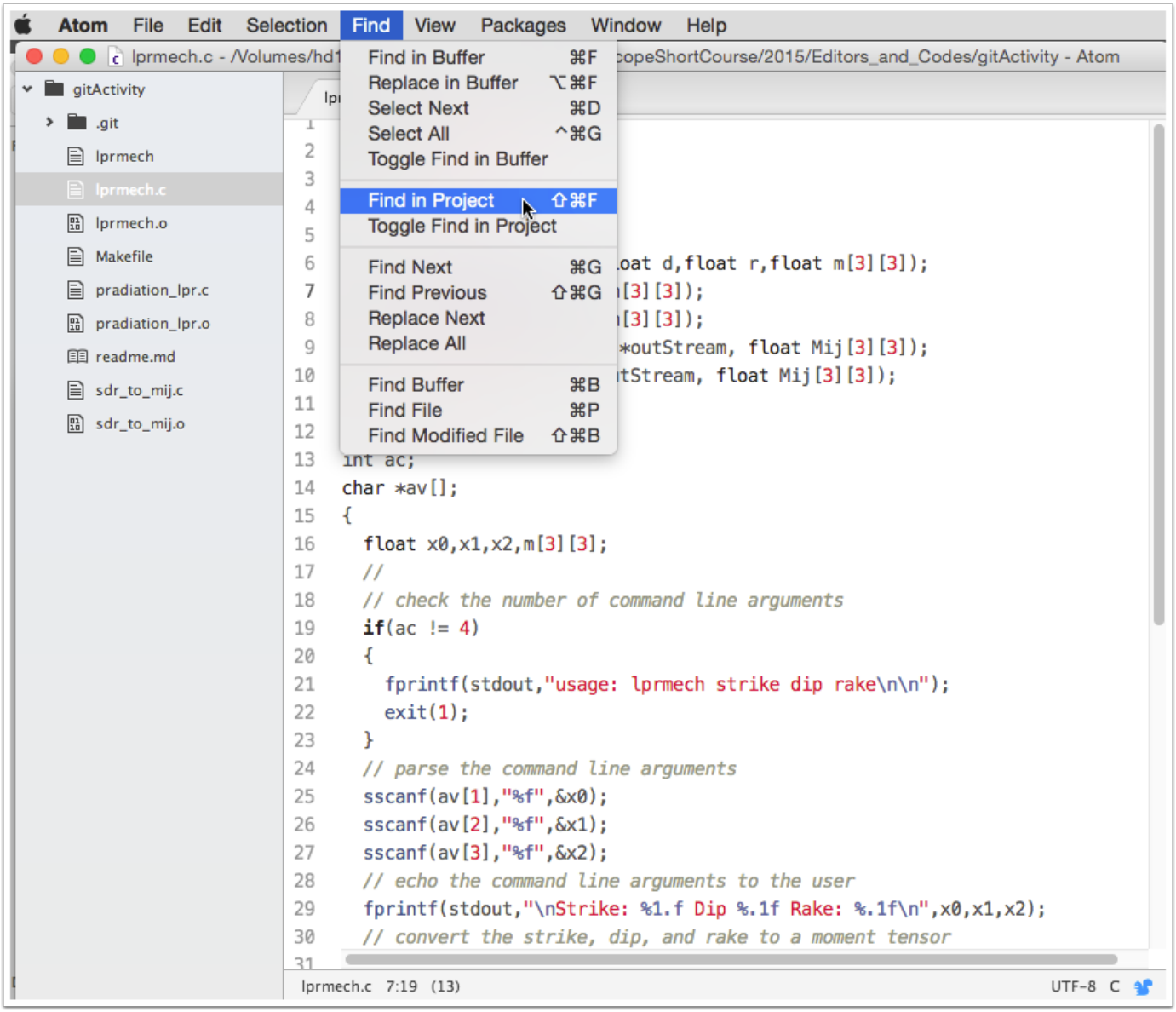
Recognize the change is all git has done so far. If we want to commit this change, we need to stage the modified file with ***git add***, and then commit the changes with ***git commit***. We are not done yet.

Two other references to the Harvard CMT remain in the code. Look at lines 7 and 8, which contain function names that include the abbreviation hrv. We might as well change those as well.

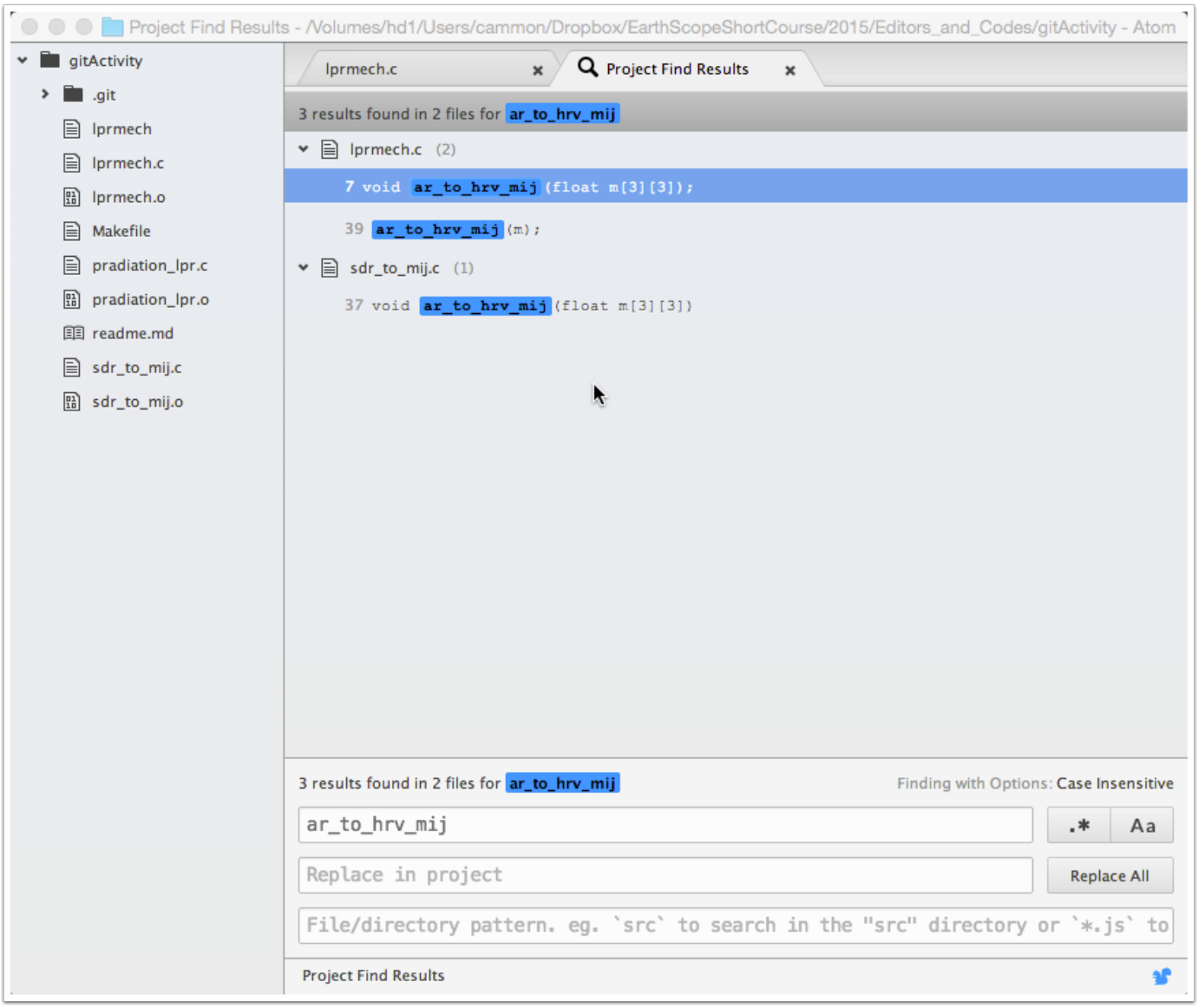


An Introduction to Version Control with Git

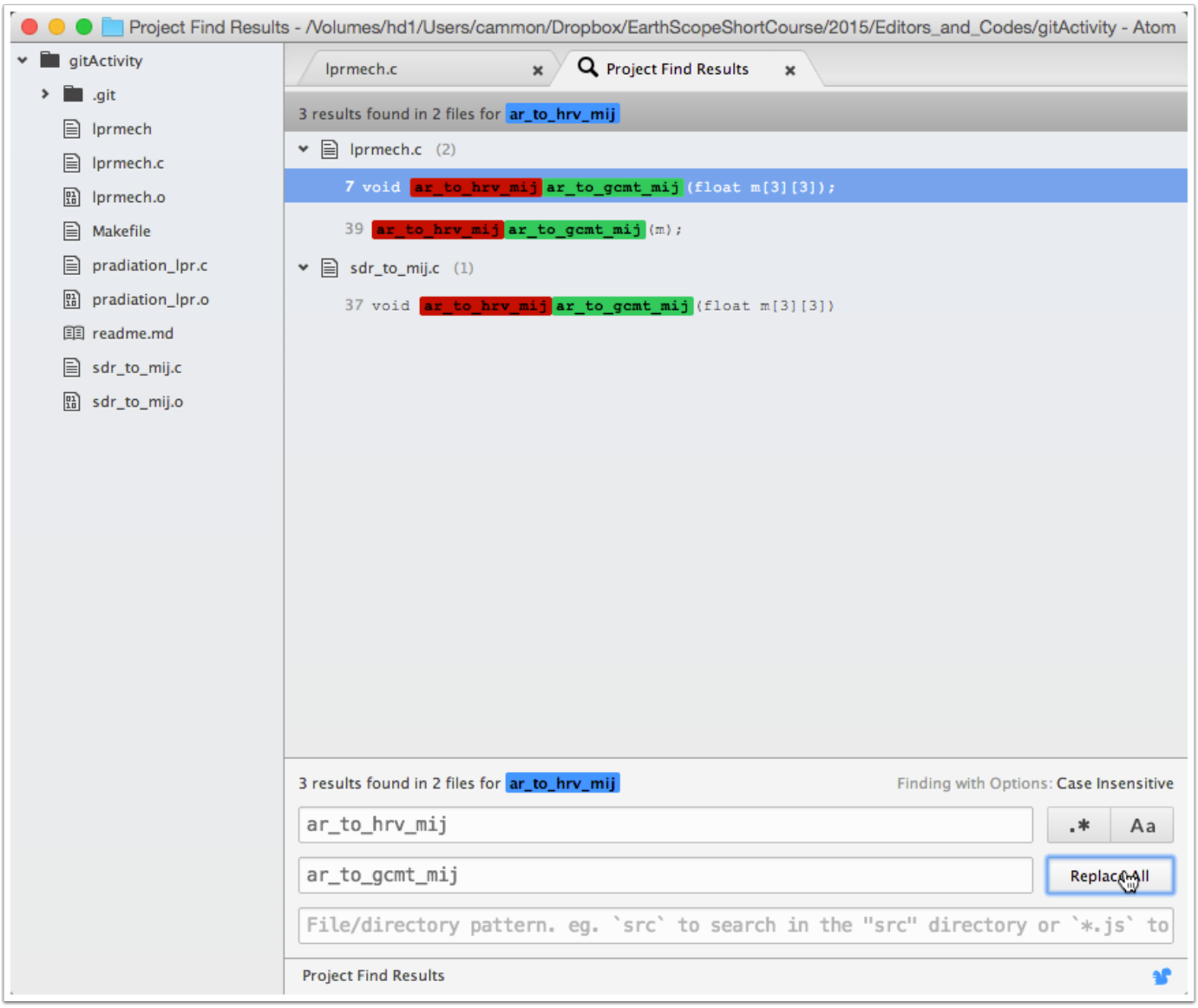
However, these function names appear in several files in our project. We'll have to change them in multiple files. To find all the instances of ***ar_to_hrv_mij*** in our project, select ***Find in Project*** from the ***Find Menu***.



Here's the results of the search. The function name occurs in three places over two files.



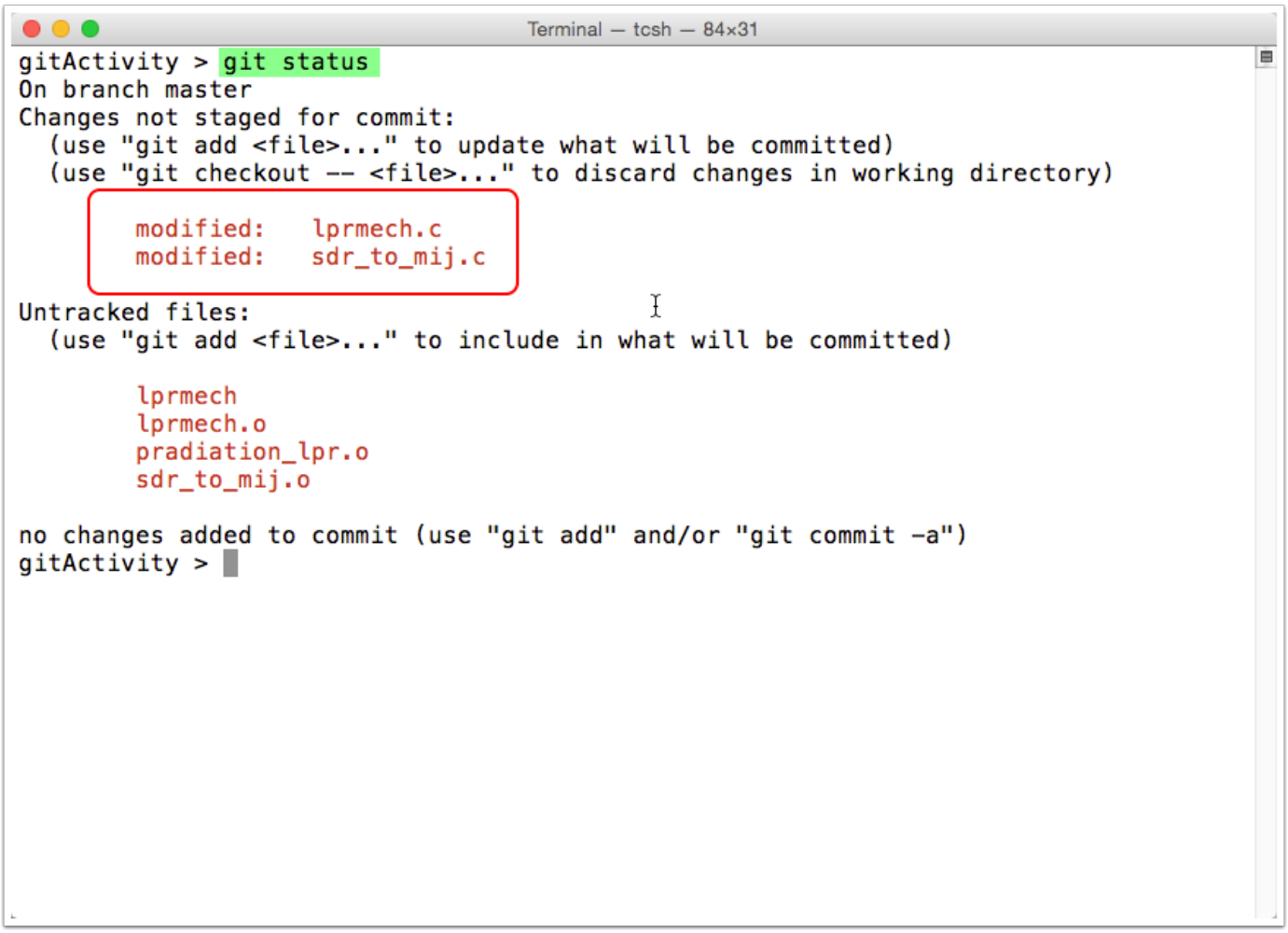
To change all occurrences of the function name, use the form at the bottom of the ***Project Find Results*** tab.



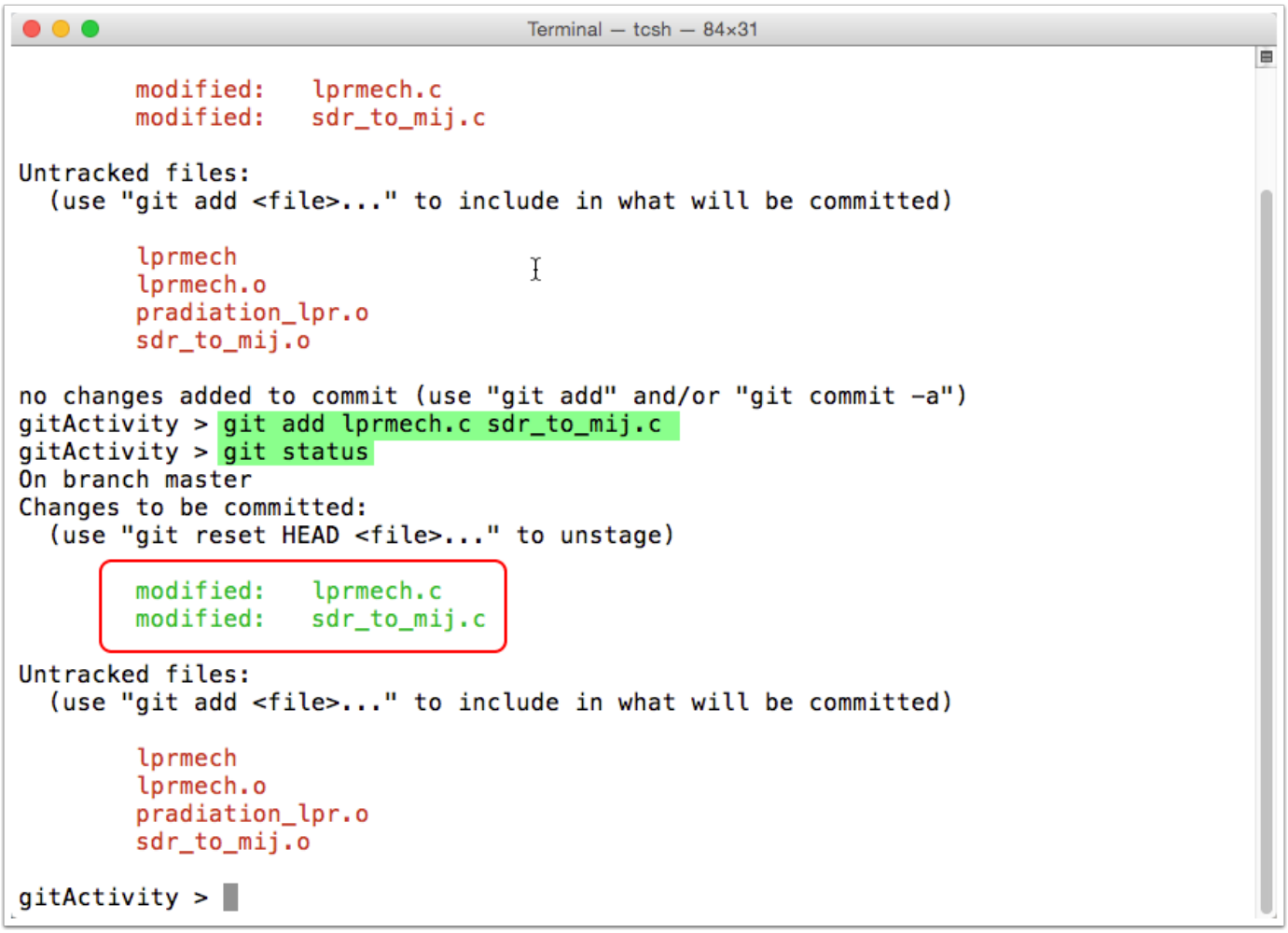
Repeat the procedure to change ***hrv_to_ar_mij*** to ***gcmt_to_ar_mij.*** Dismiss the find subwindow using the T***oggle Find in Project*** menu item in the ***Find Menu.***

An Introduction to Version Control with Git

The git status should now show two modified files.

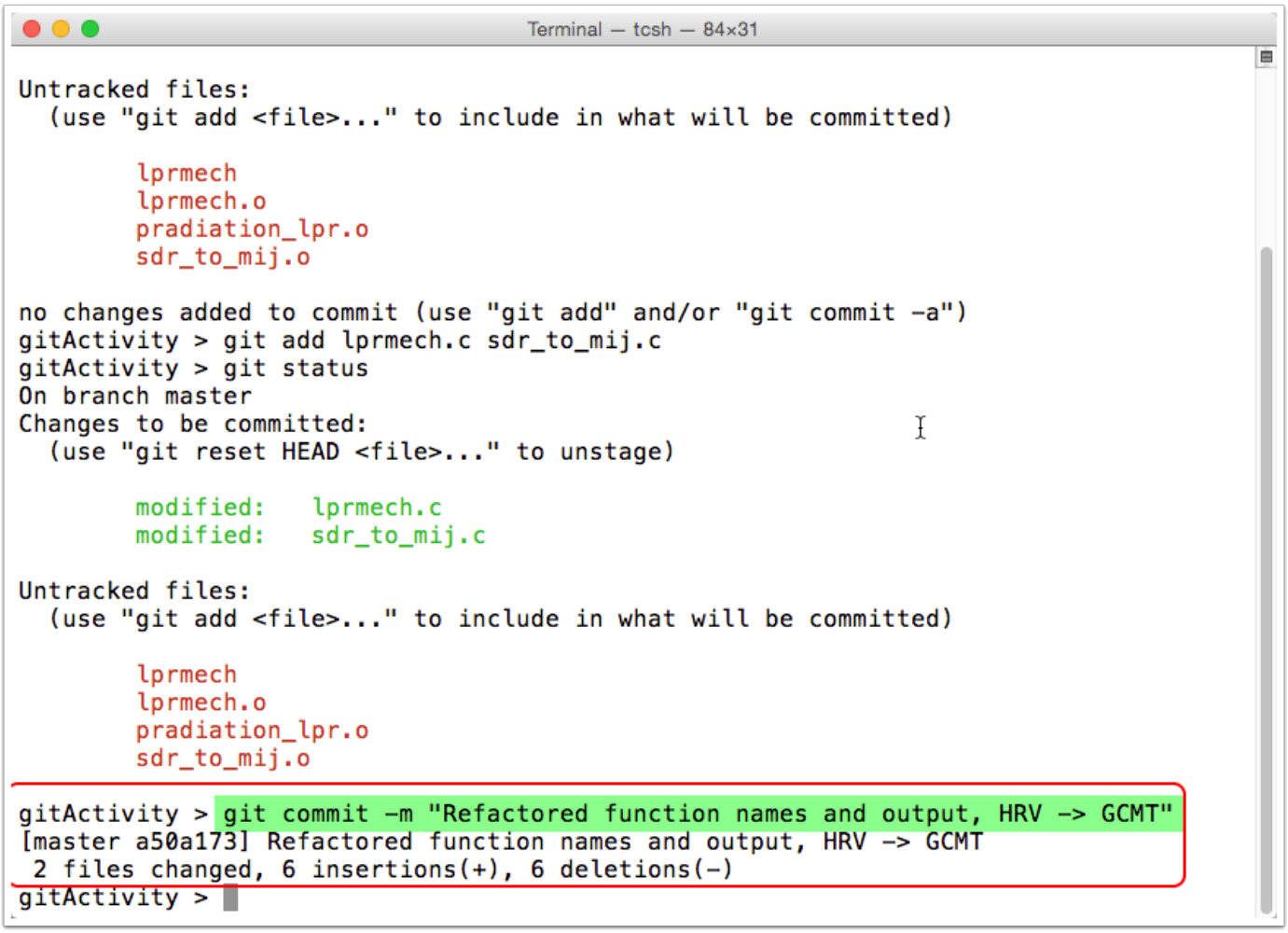


Before staging and committing the changes, ***make*** the executable and test it. You should see a correct output, including the label "GCMT" in place of Harvard. Once you are sure that the codes work, stage the files for the commit using ***git add***.



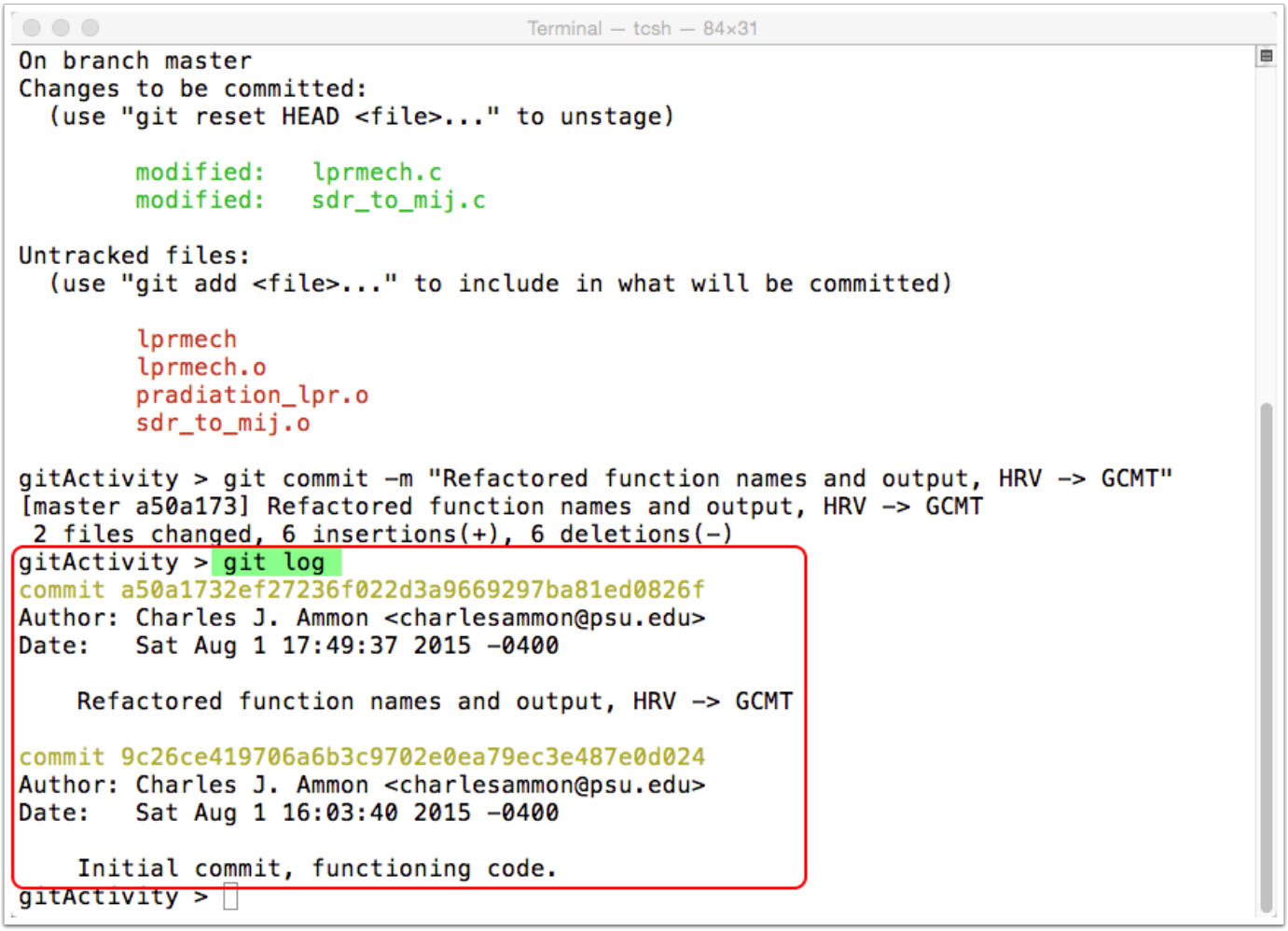
An Introduction to Version Control with Git

Then commit the changes using ***git commit -m "description..."***.



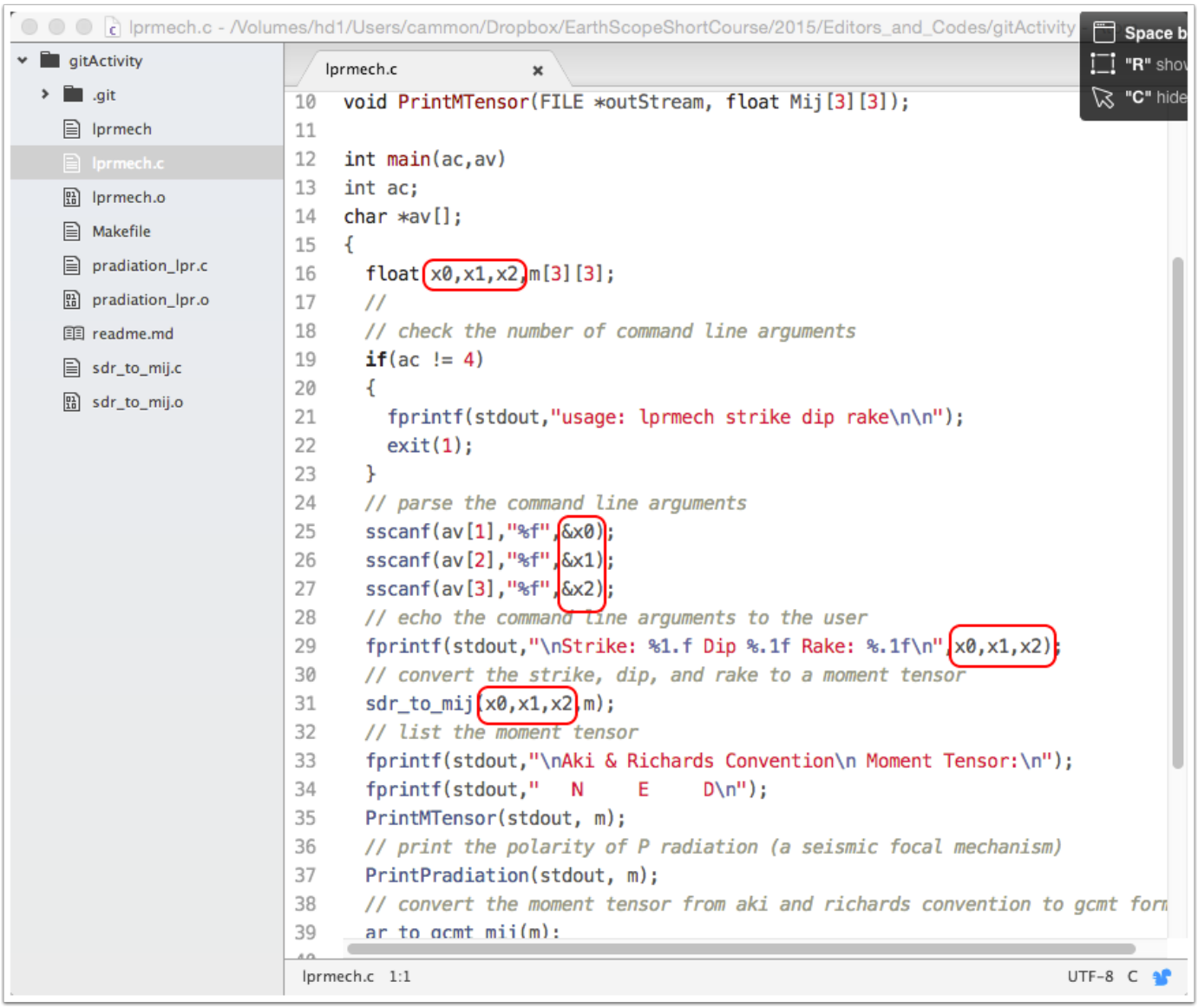
An Introduction to Version Control with Git

Now we have two commits (or versions of our codes stored within the git repository). you can see them with ***git log***.



Multiple Cursors in Atom

Let's make one more change to the codes. If you look closely, the variables that are used to represent strike, dip, and rake are the uninformative x0, x1, and x2. This is not a good programming practice.

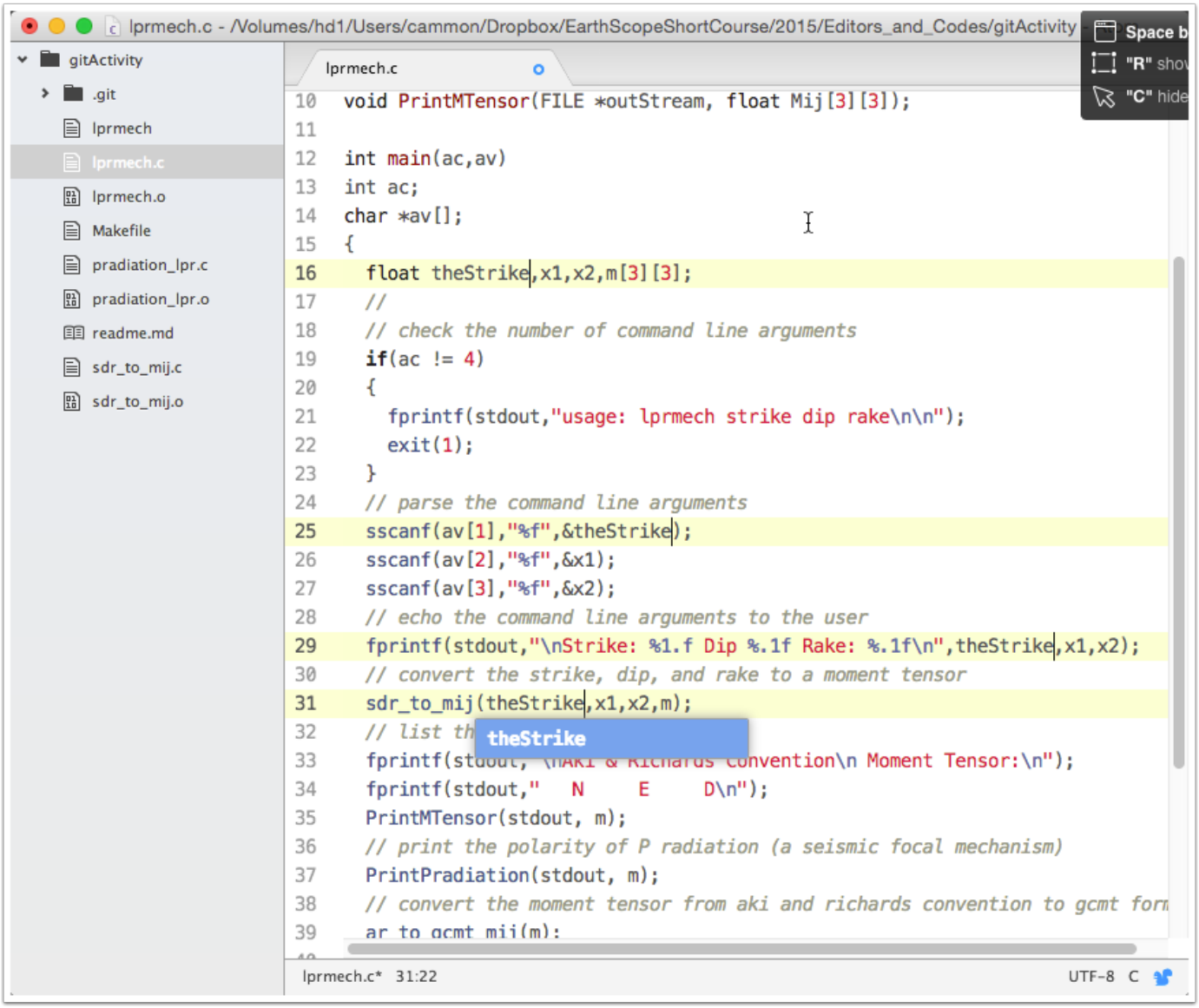


We'll use the mulitple cursor feature in Atom to change these variable names to something more informative, like theStrike, theDip, theRake.

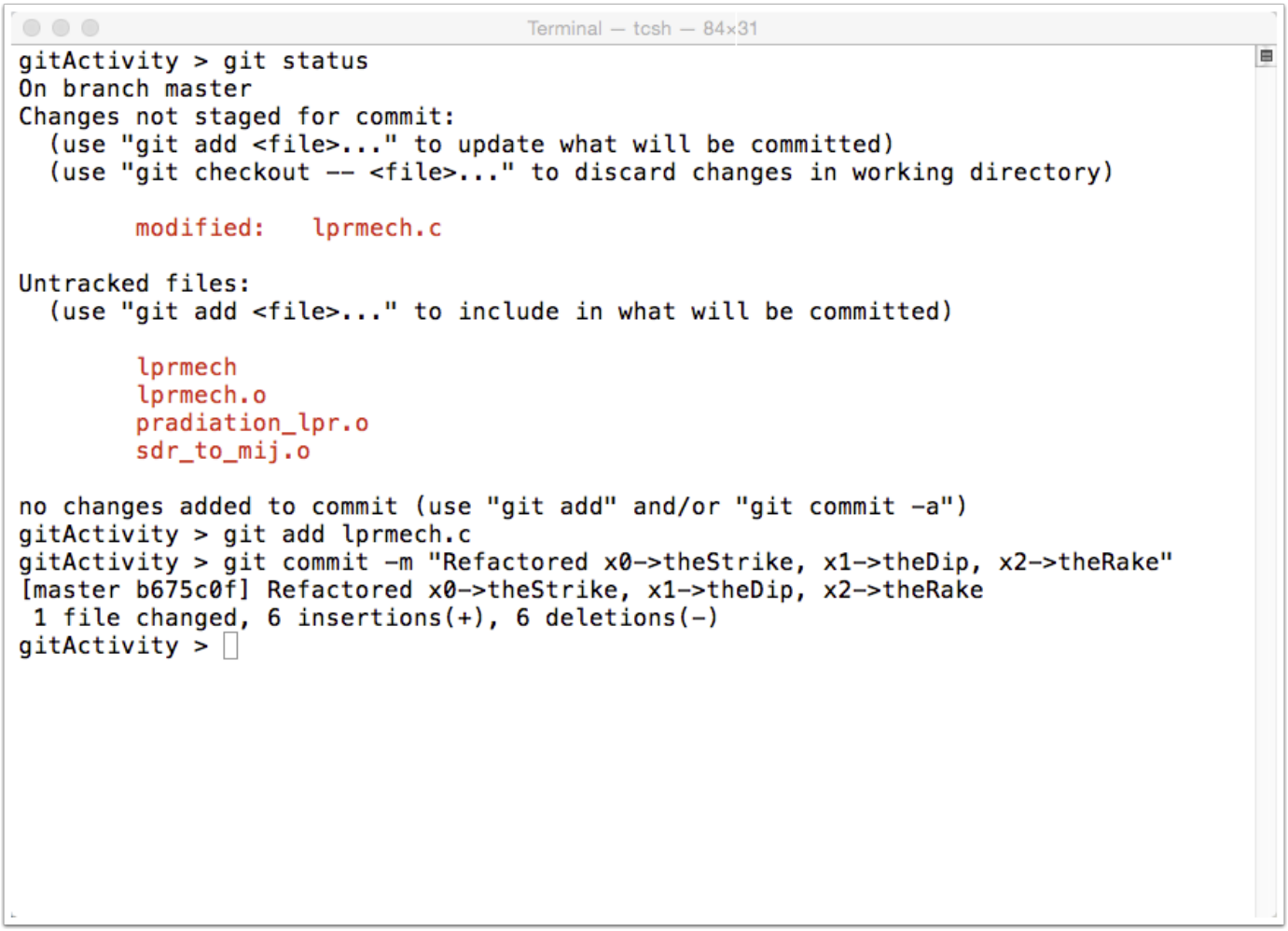
In the editor, select ***x0*** in line 16. Then enter the key comination ***ctrl-cmd-g*** (hold the control and command keys down and then press g). This will select all instances of ***x0*** in the file.

When all the ***x0's*** are selected, enter ***theStrike***, the replace all instances of ***x0*** with ***theStrike***.

An Introduction to Version Control with Git



Repeat the procedure for the x1 and x2 variables and save the results. The stage the file, and commit the change into the git repository.



Now we have three commits in the git repository.

Undoing

Just about anything in the git repository can be undone and you can reocver earlier versions or the code, but you have to be careful when doing so. You can revert any file that has not yet been committed using

git checkout <filename>

If you've already committed changes that you want to undo, then you use

git reset <last good commit hash>

Be careful with that, it will revert to the referenced commit. Of course if you have to recover from a reset, you can, but things can get confusing. You can use ***git reflog*** and ***git reset*** or ***git checkout***. For a complete description of the process, see

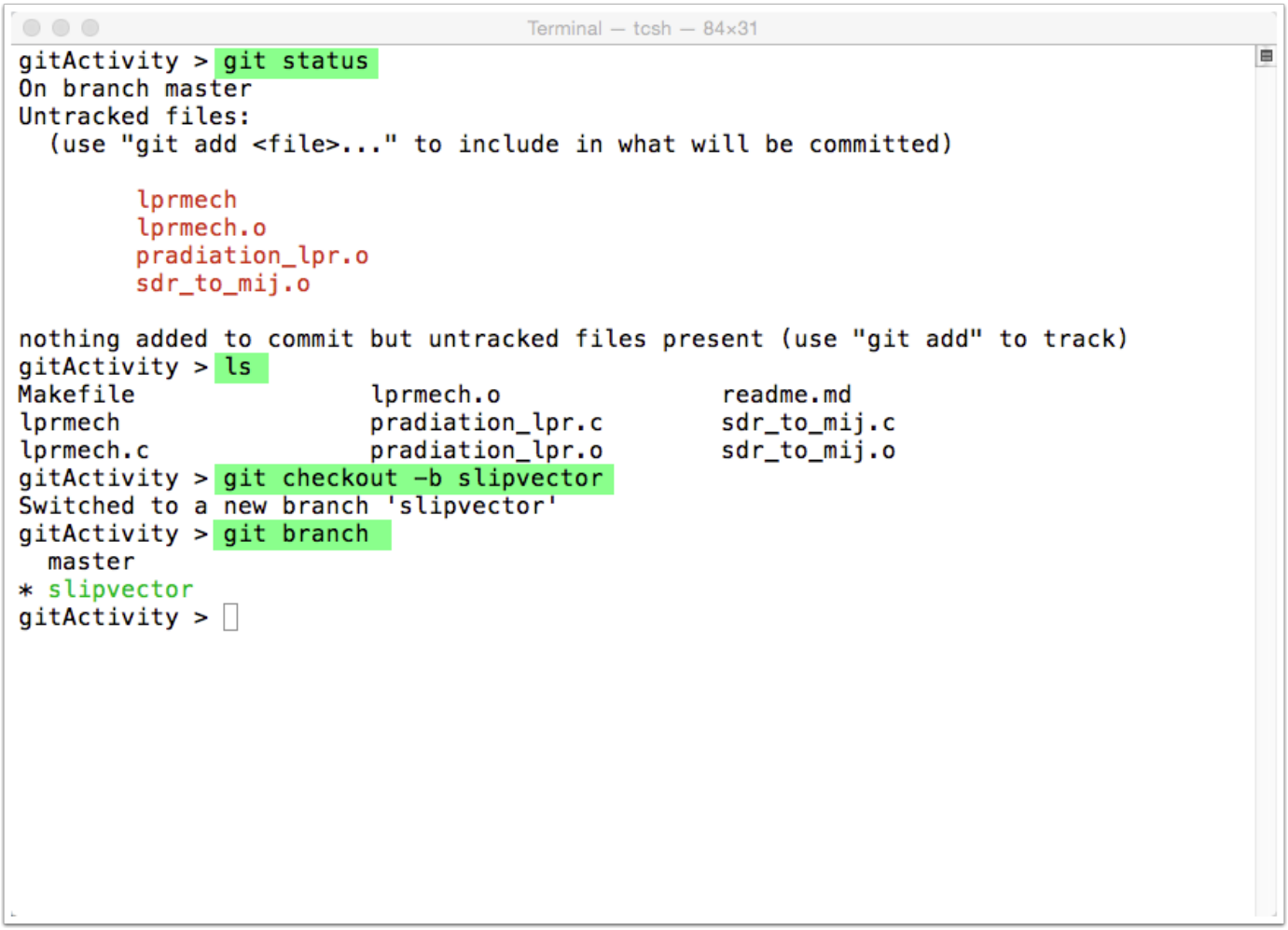
> https://github.com/blog/2019-how-to-undo-almost-anything-with-git.

Right now, our changes are probably worth keeping, so we won't revert anything. But suppose that we wanted to extend the program, say by adding a function to compute the stress axes and slip vector for the specified strike, dip, and rake.

Branches

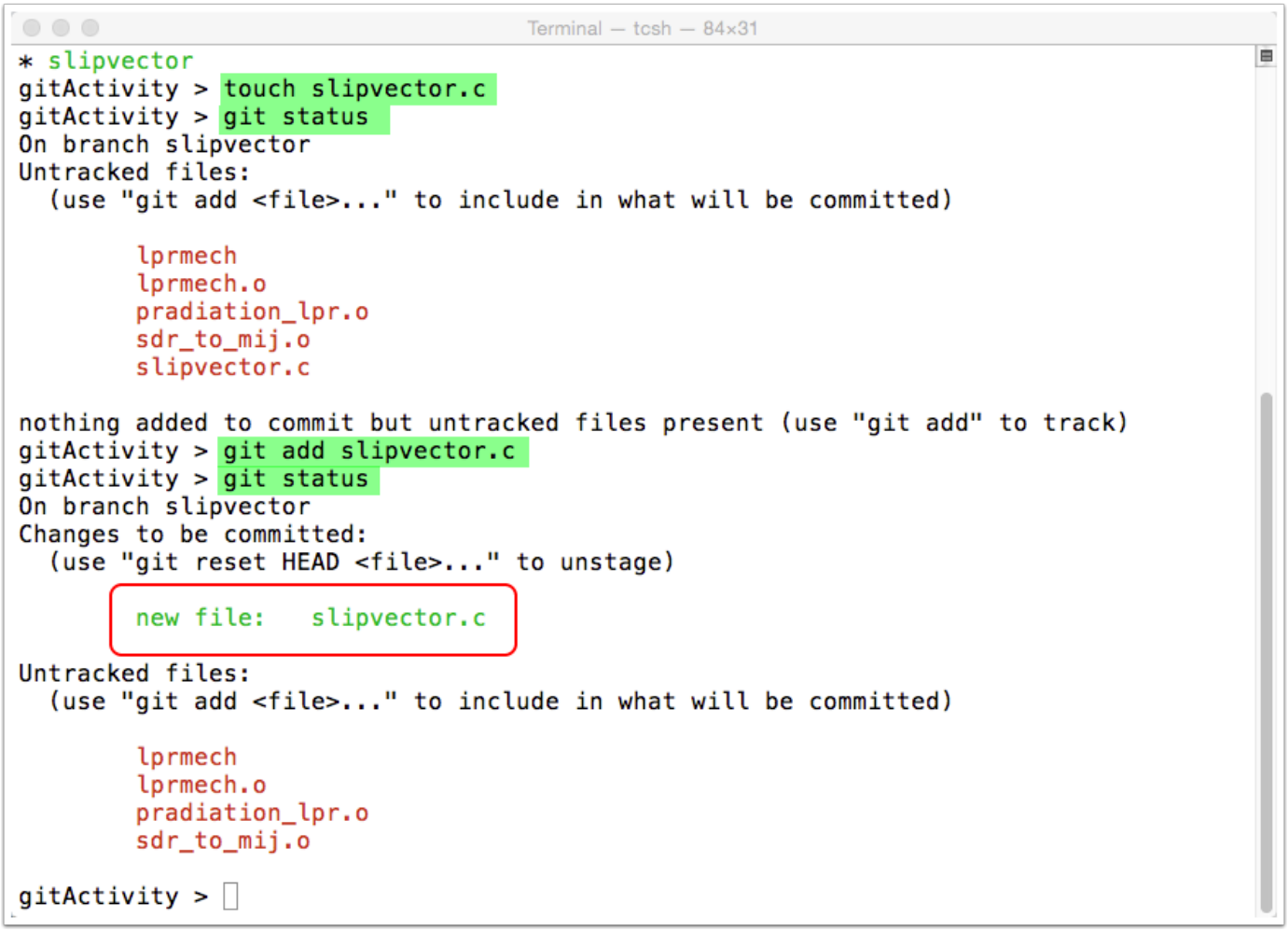
Git provides a superb facility for experimenting with the code. We can create a new branch of the code, make changes, test them, decide whether to keep them, and then later merge the experimental branch with the "master" branch that we just created.

We start by "checking out" the existing code into a new branch, which we'll call ***slipvector***. Specifically, we create the new branch with the ***git checkout*** command, and then list the branches with the ***git branch*** command. The asterisk indicates that we are now working with the ***slipvector*** branch.



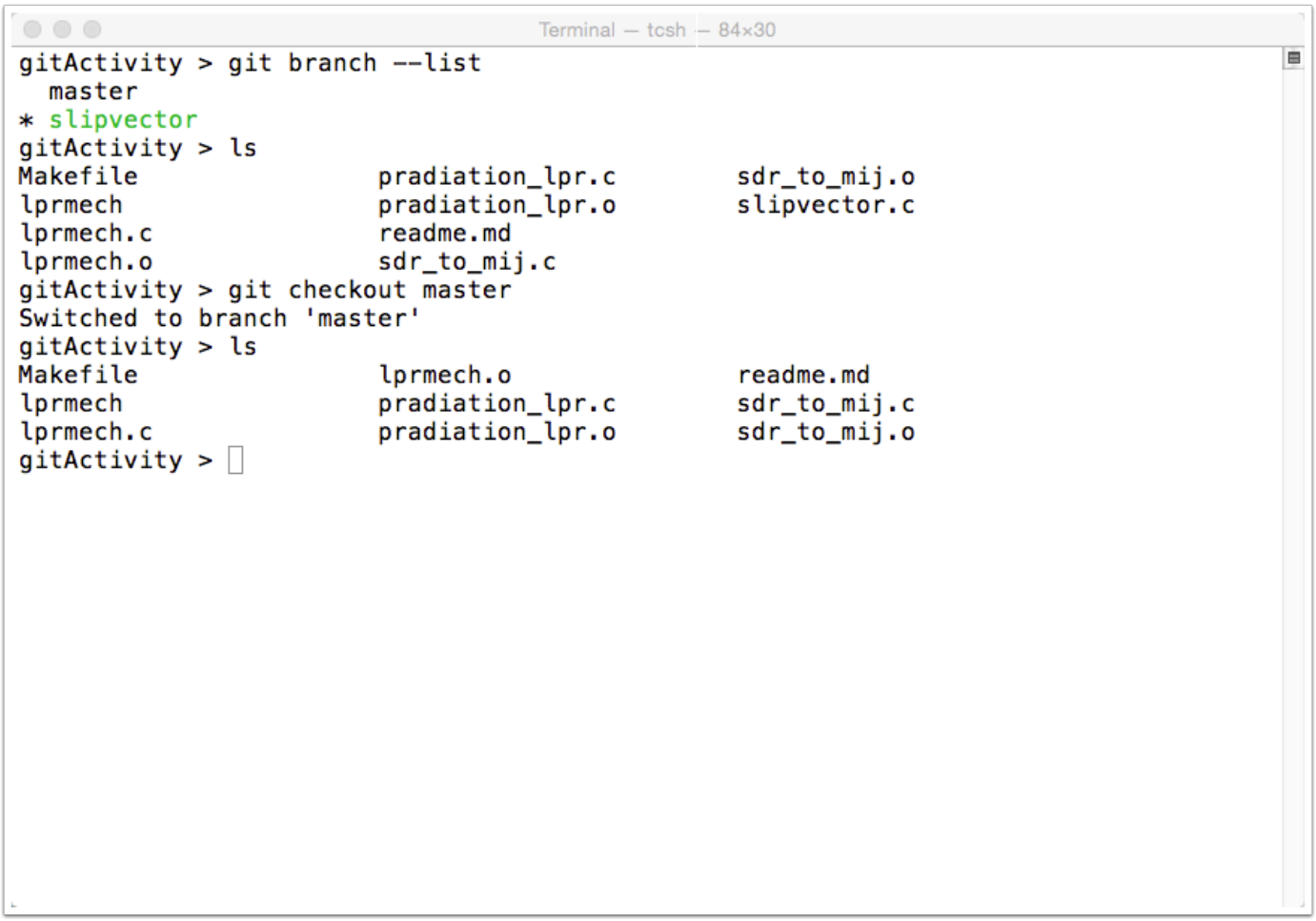
An Introduction to Version Control with Git

We can create a file called ***slipvector.c*** and add it to the slipvector branch of the code using ***git add***. Use ***git status*** to check the state of the repository.



We can then commit the change (the addition of the slipvector.c file to the slipvector branch).

We can switch back and forth between the branches using ***git checkout*** to checkout a particular branch. Note that when I return to the master branch, the file ***slipvector.c*** is not in the working directory.



Once we complete development of the slipvector.c code, test it, and are satisfied that it works, we can ***git merge*** the slipvector branch with the master branch. We can always checkout the master branch if someone needs the code before we complete the addition of the slipvector feature (or we mess everything up, we can delete the experimental slipvector branch and start over).

Exploring git and http://github.com

We have barely scratched the surface of git's capabilities and applications. You can use it with latex documents (it's not the most efficent situation, but it will work), or any other text files that you maintain.

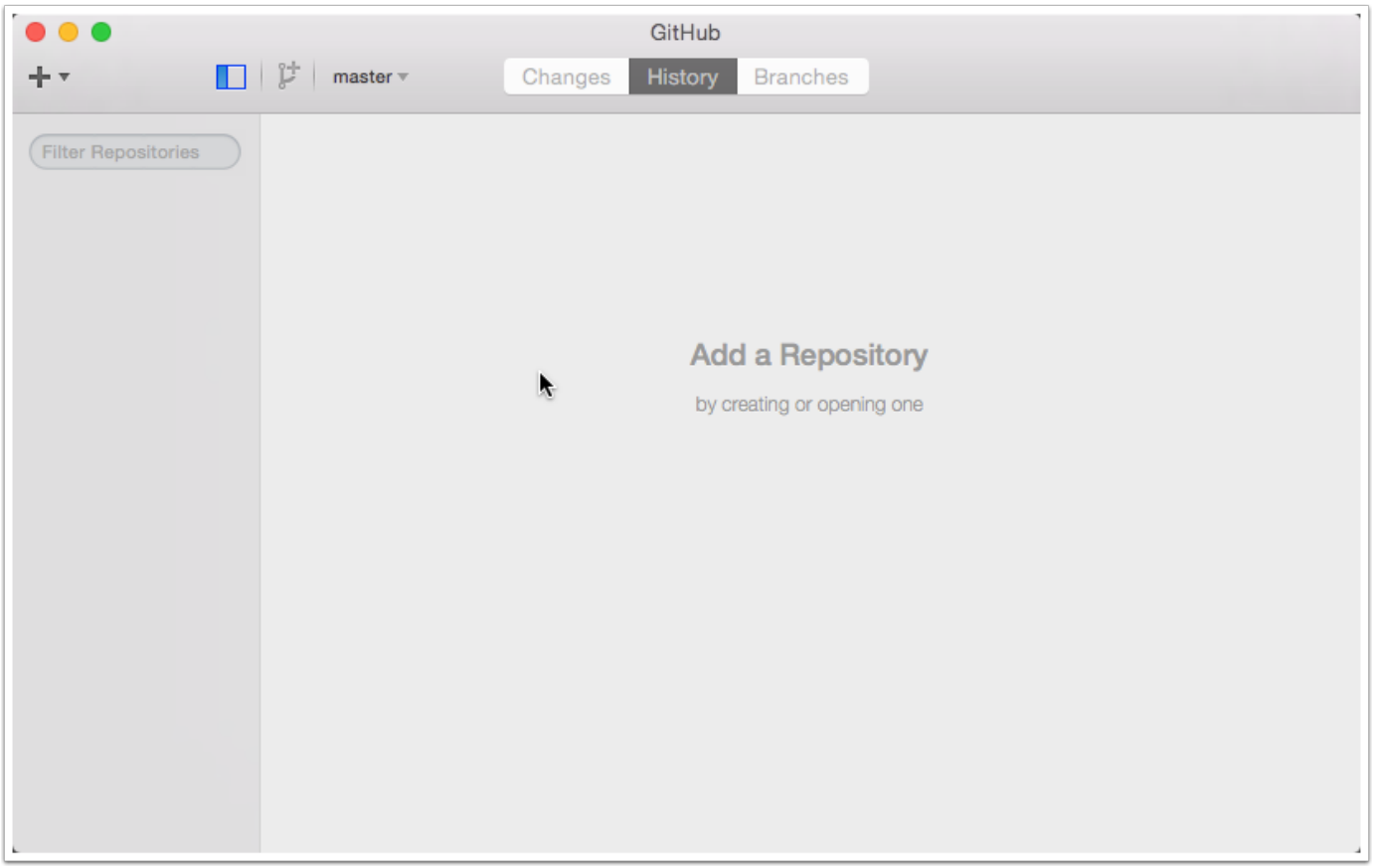
You should have no problem finding tutorials and information on git. Explore them because git will help you stay organized, track your software development and help you create more reproducible results.

We have discussed local repositories that are located on the computer that we are using. The broad computing community has largely adopted cloud-based repositories that form a valuable resource, and help make sure the codes are backed up, and easy to install on new systems. Browse or search http://github.com for an incredible number of projects. If you haven't visited that site, you may be surprised to see how much code is being shared with git respositories.

The GitHub App

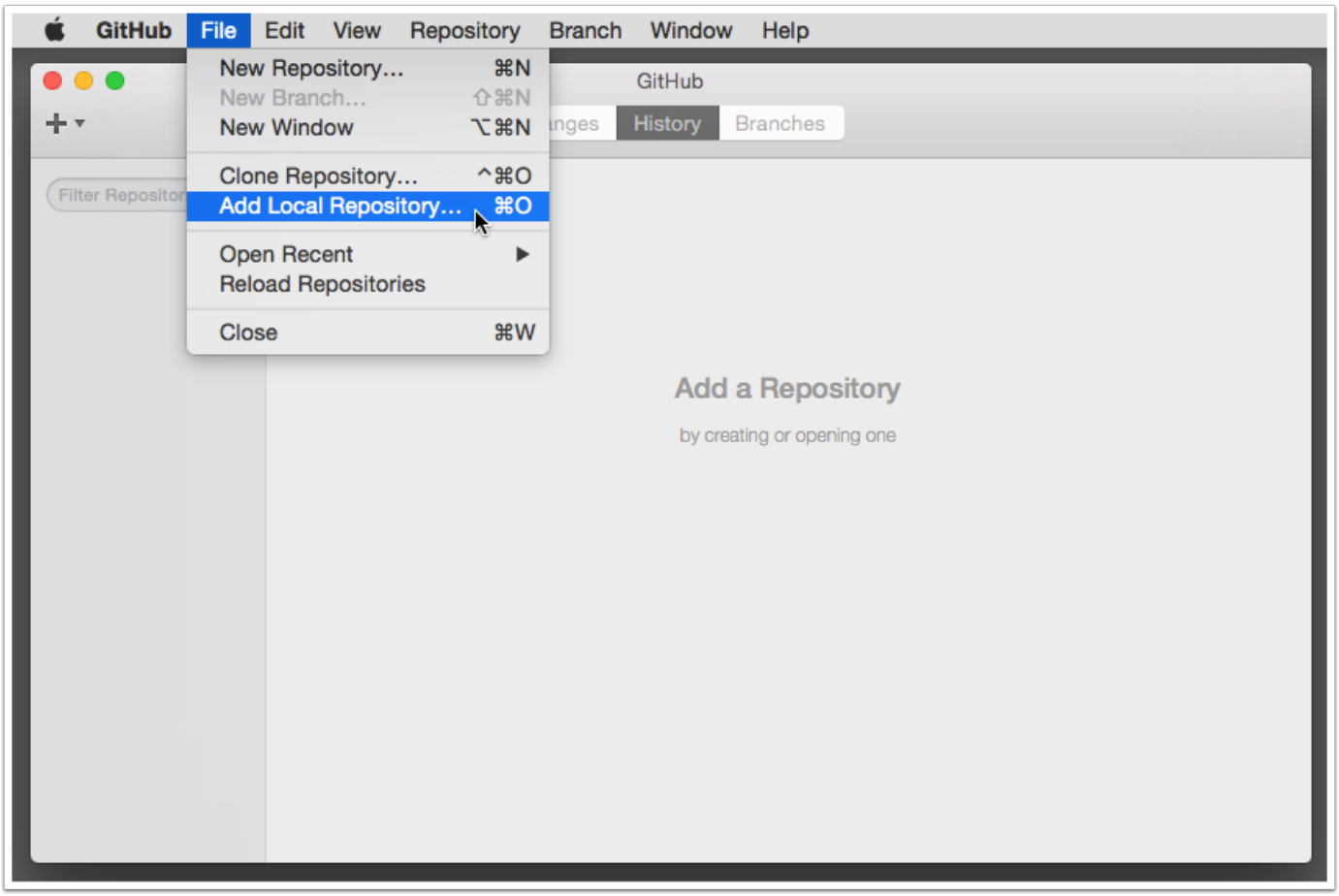
Using the command line is a good way to start using git, and may be the only way to use it when you are remotely logged into a machine. However, there are GUI tools that can allow to use git more conveniently.

Connect to http://mac.gihub.com and download GitHub for the mac. Then launch the application.

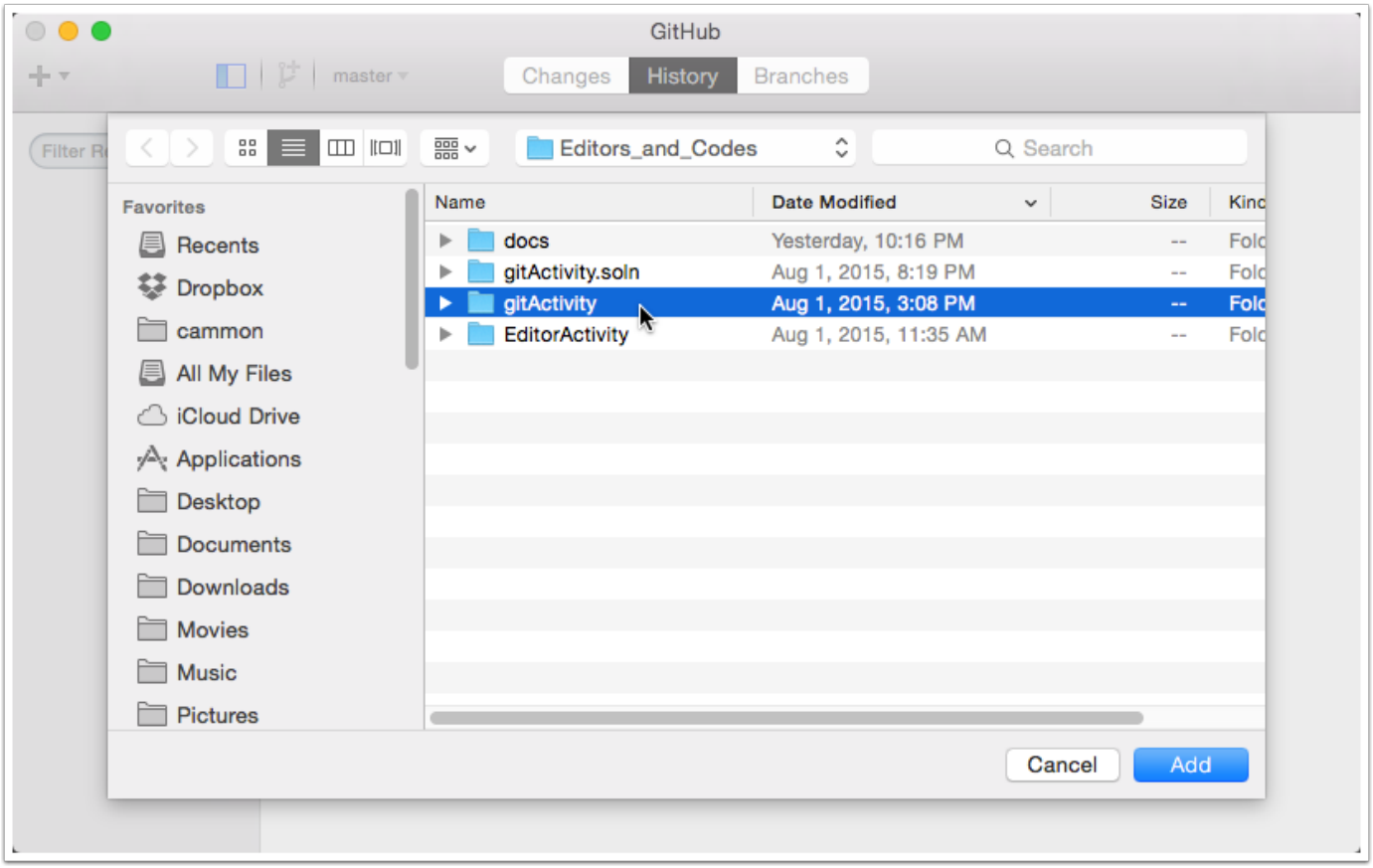


Open the lprmech git repository that you just created.

Select Add Local Repository from the File Menu. The navigate to the folder containing your recently created lprmech repository. I was working in the folder gitActivity, so I selected that folder.

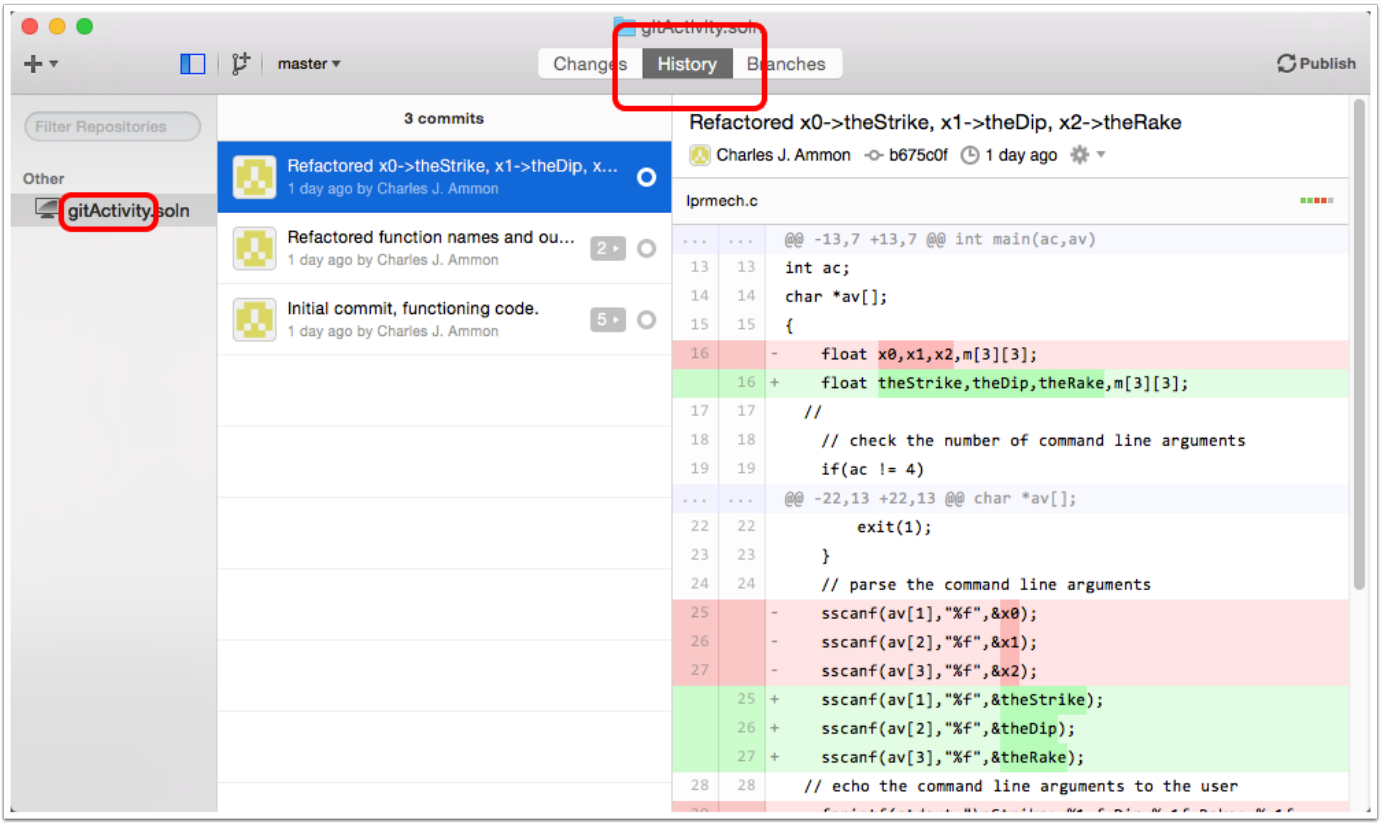


An Introduction to Version Control with Git



You'll see a list of repositories on the left, the list of commits in the selected repository in the middle, and then a comparison of the changes included in the selected commit on the right. You can browse the history of the file quite conveniently with this tool.

Select some of the commits to review your actviity.



The GitHub app also lets you publish your repository to [http://github.com,](http://github.com) which allows you to share your work with others and back up your work to the cloud. github memberships are free, and although most repositories are public, they allow a few private repositories for educational work, and more if you are willing to pay a small annual fee.

Penn State runs their own online git service based on the http://git.psu.edu software that members of Penn State community can use to create and store repositories. I am sure that your institution likely has a similar resource.

We don't have time to explore the many features and uses of github, but I recommend you make some time when you can. To get started, go to github.com and search for ***user:USGS***, you'll see many software items being distributed by one part of our community. For general exploration, click the ***explore*** link on the github home page.