Coding Lab 2 `Git` Instructions: `Git` Branch and Code Reviews

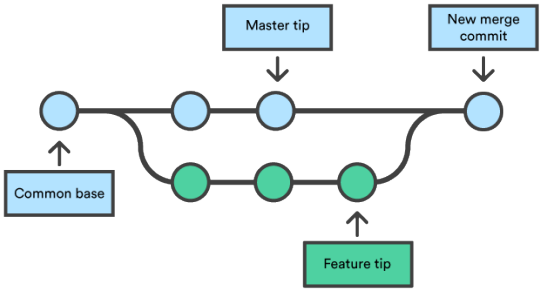
1 Pre-instructions

The first thing to do is to download the Coding Lab 2 materials from myCourses. Unzip the files and put them into a new folder to separate them from last week's coding lab. Do a git commit of all the files so that your repo is keeping track of them. (Remember how to do this? Feel free to look back at the Week 1 instructions if you need to!) Then do a git push so that the copy of your repo on the Github site contains Coding Lab 2. Check that you did this properly by navigating to your Github repo in your web browser.

2 Branching

Welcome back! Today we're going to start using `git` in a collaborative way. So far, we've always worked directly on the `main` branch. This is ok if you're doing stuff on your own, but it becomes unworkable as you start collaborating with more and more people. Suppose you're trying to add some new functionality to a data analysis pipeline that you and your collaborators are using. Adding new functionality is great, but during code development, you might have to (temporarily) break some old features. This isn't great for your collaborators, if they can't run any of their analyses while you're working on your newest feature.

This is where it becomes immensely useful to be able to define multiple branches. Doing so allows you to maintain multiple versions of your code in parallel. Diagrammatically, here is what it might look like:



Whereas our commits (symbolized by the circles) used to form a single linear history (light blue circles), now we have a separate branch (green circles) where I can do code development without worrying about breaking anything. Only when everything is tested, debugged, and documented do I merge it back into `main` branch. In fact, many large scientific collaborations have a rule where nobody is allowed to commit directly to `main`. In this class we'll encourage this workflow, but we will not insist on it (except for today's coding lab, just so that you can practice the mechanics).

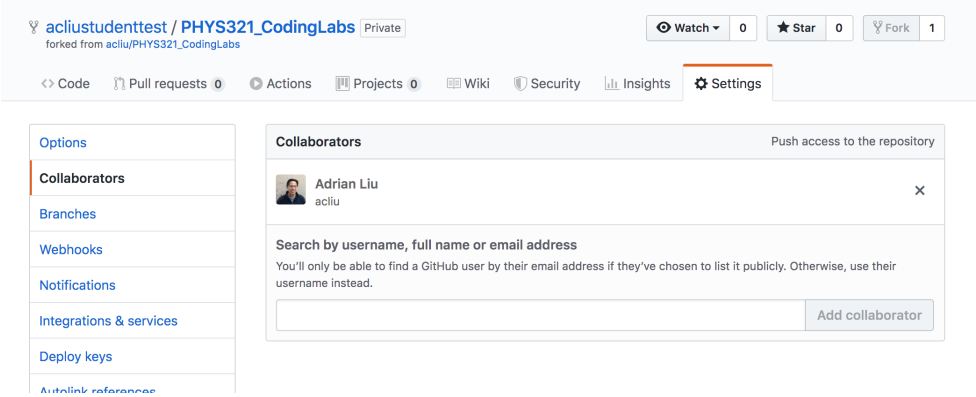
Let's try this out. Type "`git checkout -b CodingLab02_draft`". You'll be familiar with the `git checkout` command from last week, but here we're using it in a slightly different way. The "`-b`" flag tells `git` to create a new branch for us called "CodingLab02_draft". You can now do whatever you want here to your code, making `git` commits as you normally would, and there will be no effect on the `main` branch. To switch back to the `main` branch, simply type "`git checkout main`". Try it. If you're on main and want to go to the new branch that we've created, type "`git checkout CodingLab02_draft`" (without the "`-b`", which is needed only when you're creating the branch). Try this as well. If you ever forget which branches have been defined, all you have to do is "`git branch`".

Now go off and complete the Jupyter notebook for today's Coding Lab! **Work in the new branch that we just defined.**

3 Pull requests

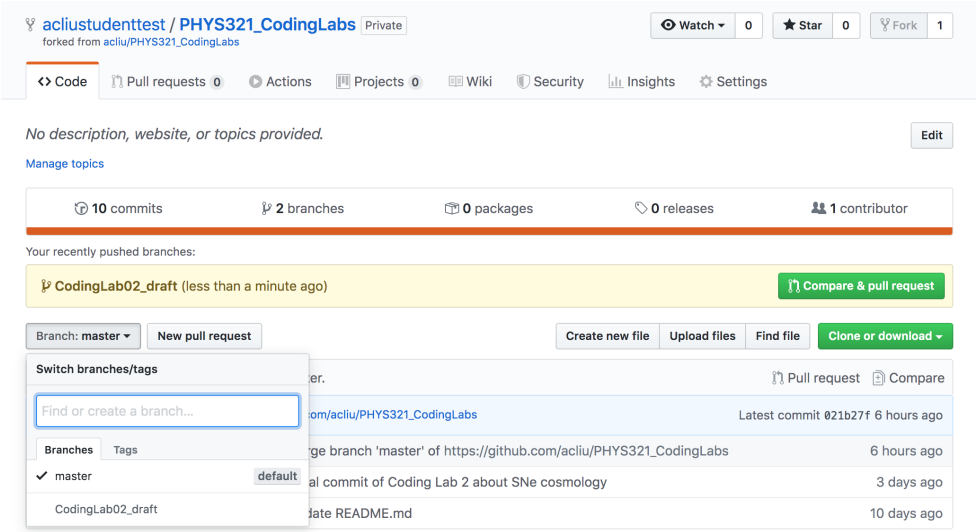
Having completed this week's Coding Lab, go ahead and commit your work (on the `CodingLab02_draft` branch). What we're now going to do is practice folding this into the `main` branch. There are several ways of doing this. To begin with, we'll simulate what we might do if this were a big team project with lots of contributors.

First, we'll need to add some contributors to your repo. Head to your repo on the `github` site, then go to Settings and Collaborators:



Search for your lab partner's `github` username and add them as a collaborator to your repo.

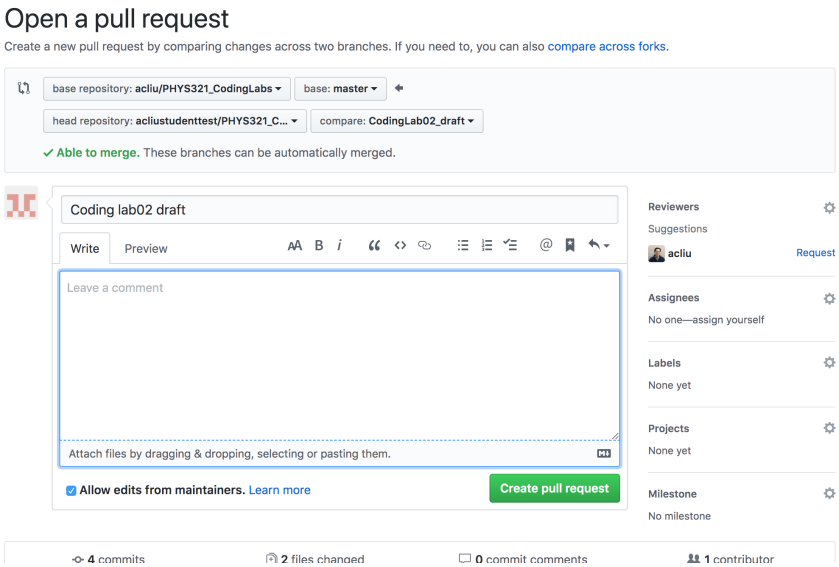
Now do a `git push` of the new branch to your forked repo on `github`. Remember how to do this? All you have to do is say "`git push origin CodingLab02_draft`". Recall that `origin` is the copy of your repo on `github`, and here the branch that we want to push up is `CodingLab02_draft`. Once you've done the push, the `github` site should look something like this:



Notice how in the "Branch" dropdown menu, there's now a new branch. You can select different branches and explore what the files look like.

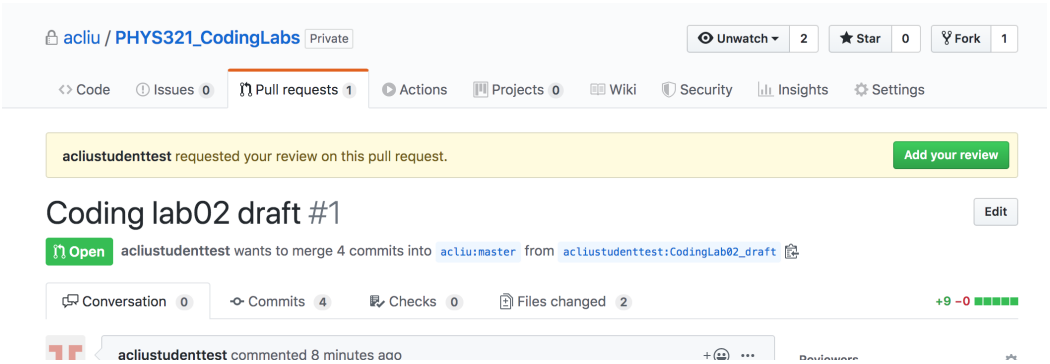
What we need to do now is to make a *pull request*. The basic idea is that we've written some great new code that we think deserves to be part of the main branch. We've done lots of our own testing of our new code and we think it's bug-free, but it's always a good idea to have someone else take a look. A *pull request* is a formal way of asking one of our collaborators to take a look at our work and to give their formal approval before the merge to the `main` branch takes place.

Click "Compare & pull request" to start the process.[1] You'll be taken to a screen like this:



The top bit lets you select which branch you want to pull in. (Make sure the "base repository" that you are trying to merge into is the `main` branch). The text box below allows you to write a short description of the new code you've written. You might want to explain, for example, why the new functionality that you've developed is necessary. On the right side, you can select various collaborators and request that they review your pull request. Go ahead and request your lab partner's review, and click "Create pull request" to start the pull request.

If your lab partner is reached the same point in this Coding Lab, you should now have a pull request (PR) to review! Head over to their repo, where you will be able to access the PR. It'll look something like this:



Click "Add your review" to start the review process. Comment on your lab partner's code. ***This is a graded part of your Coding Lab, in that we will assess the quality of your comments***. Some things you might want to consider:

- Are there mistakes? Perhaps there are some physics mistakes, or perhaps you see a bug that your partner missed?
- Is the code written efficiently? Is your lab partner using a lot of unnecessary loops, for example?

[1] If you wait long enough such that the push is no longer recent enough for `github` to highlight, you can still initiate a pull request by clicking on the "Pull requests" tab.

- Is the code written in a readable way? Is your lab partner writing code that is so clever and compact that nobody else can understand why it works?
- Is the code well-documented?
- Is the code written in a sufficiently general way, or are lots of things hard-coded?
- Where appropriate, is the code sufficiently modularized? (E.g., defining functions that are called rather than writing one monolithic code).

Normally, you would iterate with your partner and not approve the PR until you feel that your partner has addressed all your great feedback. But for the purposes of this exercise, just approve the PR. Optionally, you may revise your code based on your lab partner's feedback.

Going back to your own repo, once your lab partner has approved your PR, you can go ahead and merge. After a merge, it's generally good practice to tidy up by deleting branches that have been merged. On the `github` site, click the "Pull requests" tab, and bring up the (now closed) PR. Click on that PR. You should see a box labelled "Pull request successfully merged and closed" and a "Delete branch" button. Click the button.

4 Pulling changes down to your local copy

Go back to your command line. In the *local* copy of your repo, check out the main branch. You should see that the the new changes haven't been folded in. That's ok! The copy of your repo on `github` is now ahead of your local. But we can bring your local copy up to date.

First, verify that locally you are indeed back on the main branch. Type "`git branch`" and make sure there is an asterisk (*) next to `main`. Now type "`git pull origin main`" to *pull* down the changes from `github`. Now, the tidying up that we did on `github` ought to be done locally as well. Type "`git branch -d CodingLab02_draft`" (the "`-d`" stands for "delete", as you can image) to delete the development branch, leaving just `main`.

You've now completed a full cycle of code development. Congratulations!

5 Merging branches locally

The workflow that we just went through is one that makes a lot of sense when multiple people are working on the same code and sharing things via `github`. But if you're just developing your own code on your own machine, it's overkill to do the merging on `github`. As one final little exercise, let's practice merging a branch locally.

Create a new branch called "`merge_practice`". On that branch, create a file called "`merge.txt`". Commit this change. Now go back to the `main` branch. Type "`git merge merge_practice`" to merge the branch "`merge_practice`" to the branch that you are currently on (i.e., `main`). Delete the `merge_practice` branch. That's it! You're done!