Hello `git`

Have you ever worked on a class project with a partner where the two of you constantly emailed files to each other, and as time went on, it got harder and harder to keep track of which version of your file was most current, what changes had been made when, etc.? *Version Control Systems* are designed precisely to address this problem. In this class, you'll be using `git` and GitHub to collaborate effectively with your partner.

1. The Pro Git book offers an excellent introduction to version control and `git` in particular. It is available for free at the following URL:

   `http://git-scm.com/book/en/v2`

   I recommend starting here and reading chapters 1, 2, 3, and 6, to get a basic idea of what `git` does and how it works.

2. GitHub is a hosting service for `git` repositories. First, sign up for an account at:

   `http://education.github.com`.

   Note that as students, you can get a micro-account (normally $7/month) for free. Having a micro-account allows you to create private repositories — ones that are only visible to you (and any collaborators), but not to the public at large. On the following page, you'll find some starter guides that walk you through the basics of using GitHub:

   `https://guides.github.com/`

   I recommend completing the Hello World, Understanding the GitHub Flow, and Getting Your Project On GitHub tutorials.

3. Finally, here's a handy cheat-sheet with all the `git` commands in one place (if you choose to use it from the command-line — note that GitHub offers a GUI if you prefer that instead).

   `https://education.github.com/git-cheat-sheet-education.pdf`

Exercise

Either you or your partner should create a new repository on GitHub and add a couple of files to it (they can just be text or Word documents, to keep things simple). Invite the other person to join your repository. Now, the second person should create a branch of the master and start making edits to one of the files. Simultaneously, the first person should edit a file on the master branch. When you're both done, merge the two branches to arrive at a version of the files that incorporates the changes both of you made. Finally, revert one of the files to a previously committed version.

Recommended Best Practices

1. Commit early and often.

2. Write meaningful commit messages — note things like what features you added since the previous commit, what remains to be done, any known bugs etc. Poor documentation severely cripples the effectiveness of tools like `git`, as you'll otherwise find yourself manually examining different versions of your code to determine what changed between commits.

3. Your repository doesn't need to exclusively contain source code; it is a great idea to use `git` to also share your write-up drafts (LaTeX source files). But do not include things that can be regenerated from other files — for example, include your source code, but not the compiled binaries.