IGME-230 WEBSITE DESIGN & DEVELOPMENT

GIT AND GITHUB EXERCISE

In today's exercise, you'll work with two different repositories on GitHub to perform tasks that are commonly done by web developers. Version control with Git can be a hard concept to wrap your head around at first, so exercise will be much easier if you've done the readings for today. The exercise assumes that you're using VS Code as your editor. If you're using Brackets, you can use chapters 5-6 of the Practical Series Git & GitHub book as a guide.

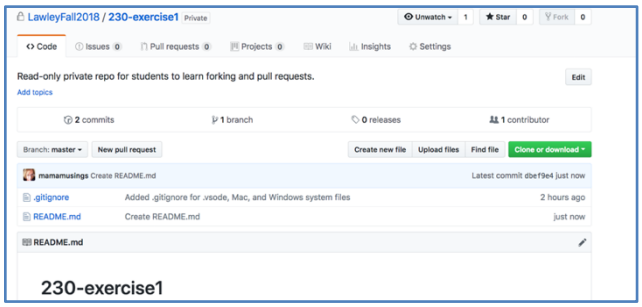
Because you'll be working with private repositories that I've created, you need to be added to the 230-students team for the repo--that's why we asked you to provide your github username last week. If you haven't done so, you'll need to talk to Liz or one of the TAs to get access before you start the exercise. You'll also need the ability to create private repositories, which is why I had you apply for the student developer pack last week.

In the first part of the exercise, you'll make a fork (your own copy) of a repo that you don't have write (push) permissions on, add a file in your copy of the repo, and then request that your new file be added into the original repo. This is called a "pull request," and it's typically how you will add files to a repo that someone else controls (like open source software, or an industry project with a large team contributing to it). It gives the owner of the repo a chance to review and approve your changes before merging them into the codebase.

In the second part of the exercise, you will have write (push) access to a shared repo--which is more likely to happen if you're in a small workgroup without a single person approving all updates to the code--like a group project here at RIT. You'll be editing the same file that the rest of the class is editing, which means there's the potential for conflicts if you don't follow the instructions carefully!

Part 1: Contributing to a Repository When You Don't Have Write (Push) Access

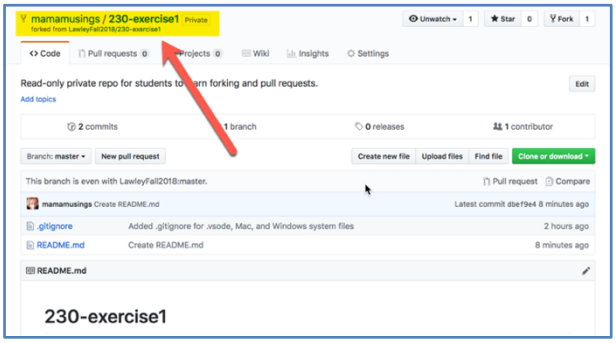
1. Go to https://github.com/LawleyFall2018/230-exercise1 .
   You should see at least two files: a .gitignore file that's used to tell your client what types of files not to sync to the server, and a README.md file that explains the purpose of the repo. You'll also see text files from anyone who's already completed this exercise.



2. You are going to to create a "fork" of this repo on GitHub. A fork is a new repo in your own GitHub account that is a copy of an existing repo. A common reason to make a fork is to take a project that someone else owns in a new direction. For instance, a faculty member at another university might want to use my course repo on GitHub as a starting point for their own class. They could fork the class repo so that they have the basic structure and content, and then edit the files to reflect their own class details. The process of forking leaves a pointer back to the original, which is a helpful way to let people see where content originates, and how it has changed.

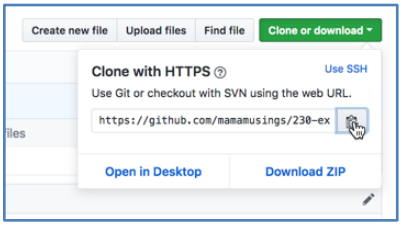
   A fork can also be used if someone who doesn't have access to modify a repo wants to make changes—for instance, a programmer who wants to help fix a bug on a big open source project. In that case they would make a fork of the code, fix a bug or add a feature, and then send a request to the original owner to pull in their changes. This is referred to as a pull request.

   In the top right corner of the page, click the button labeled "Fork". This will create an exact copy of the repo in your account. If your GitHub Education Pack was approved, your repo should also be set to private. (If it isn't, please talk to an instructor ASAP.)

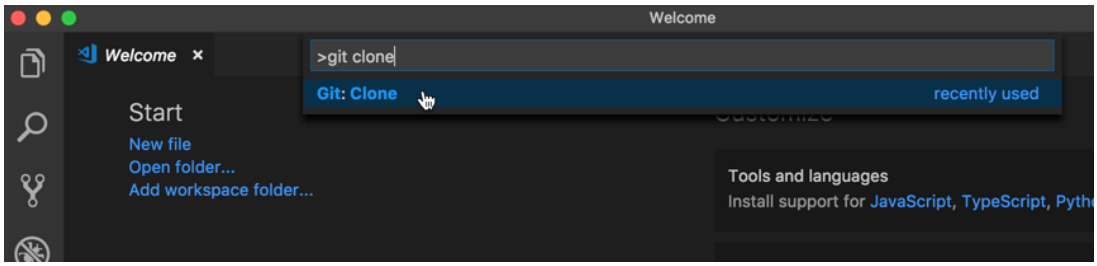


   You should be taken to the new repo page. While the content is the same you can see that the path to the repo now has your user name rather than LawleyFall2018.

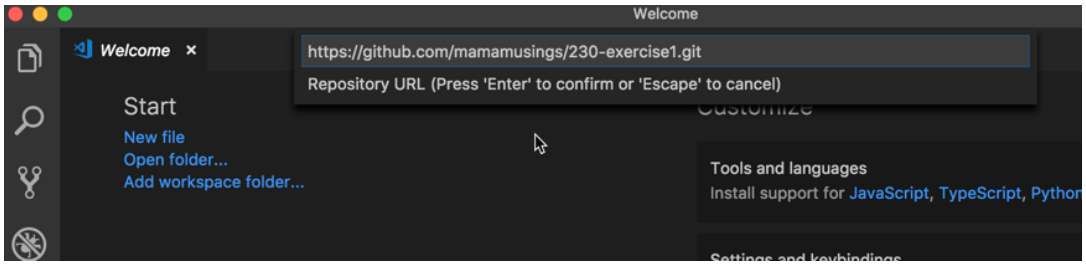
3. Now you're going to clone your copy of the repo to your computer, so you can edit the files locally. Copy the Clone URL from your new repo (the new one you created in your account, not the one in LawleyFall2018), using the "Clone or Download" button. If you hit the little clipboard button next to the URL, it will copy the entire URL to your clipboard for pasting.



4. Launch Visual Studio Code (or, if it's already open, choose "New Window" from the File menu). Press F1 for the command menu, and type Git Clone. You should see the Git: Clone command—press enter.



You'll be prompted a valid Git repo URL; paste the GitHub clone URL here.

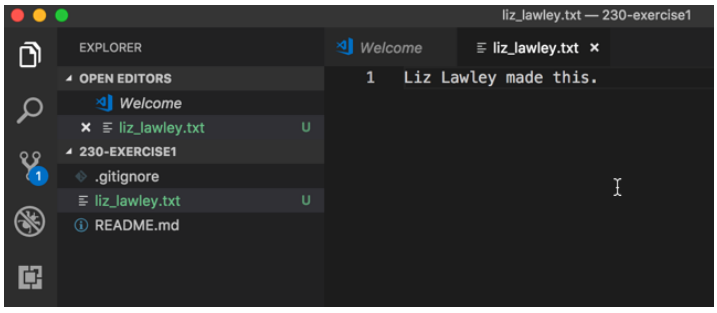


Press enter once you've pasted in the URL, and you'll be prompted to select a location for the repository. VS Code will create a new folder for the repo in the location you choose. (If you're not sure how to specify a directory path, it's easiest to just take the default, and then move the folder later if necessary.)

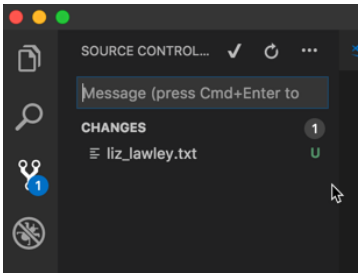
Since this is a private repo, VS Code should prompt you for your GitHub username and password so it can access your files. (If you've enabled 2-factor authentication on GitHub, this step may not work properly, and you might have to authenticate using the command line, as explained in the assigned GitHub tutorial. Ask me for help with this.)

After it's done retrieving the files, VS Code will ask if you want to open the new repo—say yes. The files from the repo will be displayed in the explorer bar on the left side of the window.

5. Create a new file called firstlast.txt, where first=your first name, and last=your last name. Add some text to the file, and save it. You should now see a notification on the Git icon in the left sidebar, telling you that there's a been a change to a file.



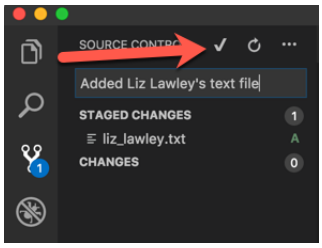
6. Click on the icon in the sidebar to open the source control view. The file you just created should be listed under "Changes", with a U (for "Untracked") next to the file. The file has been saved on your computer, but it has not yet been recorded in your local Git repo, which tracking the history and changes of the files in the folder. You need to explicitly tell it that you want it to keep track of the new file.



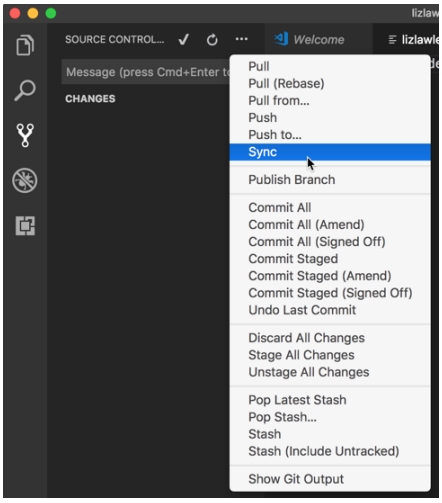
   The first step is to *stage* this file, which you can do by clicking on the + sign to the right of the file. When you do this, the file will move from "Changes" to a new category of "Staged Changes." The "U" for untracked should change to "A" to reflect that the file will be added to the repo.

   Staged changes have not been added to your local repo yet—that won't happen until you commit the changes. Why the extra step? You might be changing a lot of files, but are only ready to commit a few of them to the repository. Only the files that you've staged will be included when you commit them.

7. Since this is the only file you're adding, it's time to commit it to the repo. Type your commit message in the box above "Staged Changes." The message should be both brief *and* useful. (I used "Added Liz Lawley's test file". Commit messages should allow other repo users to easily see at a glance what you changed. Once you've added a commit message, click the check mark to commit the files to your local copy of the repo.



8. Your changed files are now part of the local copy of the repo on your computer, but they are not yet synced to the GitHub version of the repo. That's what we'll do next.

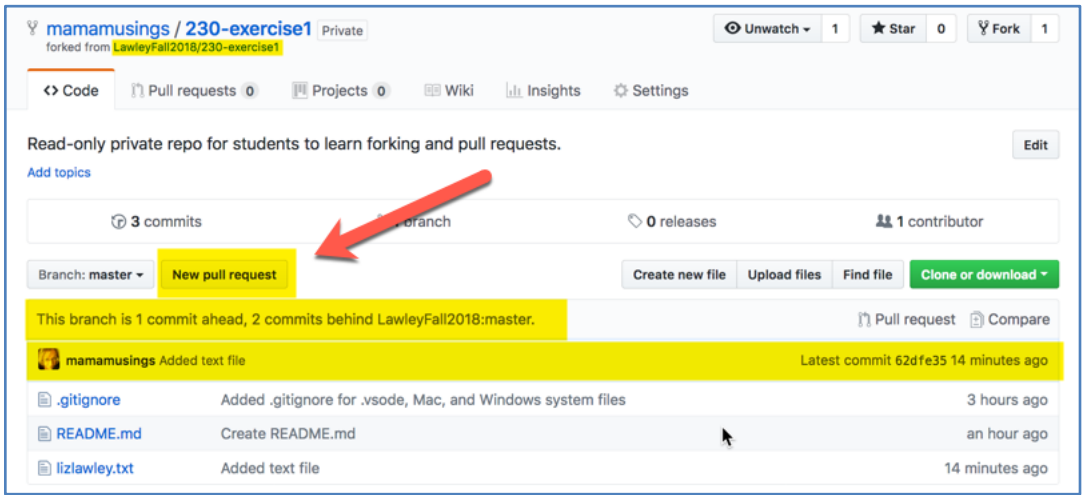


   You can see the various options for interacting with the GitHub version of your repo by clicking on the … in the top right corner of the source control sidebar. There are options to "Pull" (downloads the most recent copy of the GitHub files and updates your local repo), "Push" (uploads your commits to the GitHub repo), and a number of others. There's also a "Sync" option, which combines a Pull, a merge, and a Push. I suggest using Sync until you've got a good handle on how version control works. (If you're using Brackets, you'll have to do a pull followed by a push, since it doesn't have a Sync option.)

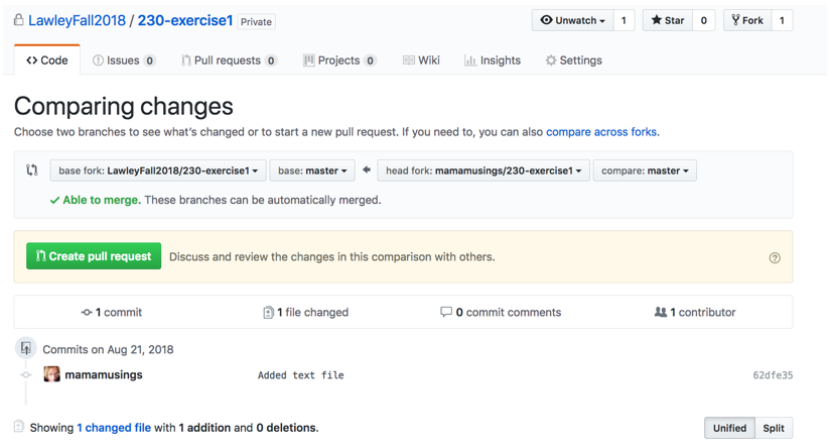
9. Go back to your copy of the repo on GitHub (you may need to refresh the page). At the top of the list of files you should see your most recent commit message, and that message will also show up next to the file you uploaded. There will also be a message above the latest commit saying “This branch is 1 commit ahead of LawleyFall2018:master”, because GitHub keeps track of the differences between the original repo (the one in LawleyFall2018) and your forked copy. If anyone has made a change to the original repository since you forked it, you’ll also see a message saying that the branch is “x commits behind” that repo.

10. Now go back to the LawleyFall2018 version of the repo (you can use the link at the top of the repo saying “Forked from…”) Notice that your file isn’t there. That’s because your “fork” of the repo is entirely separate from the original. Since you don’t have permission to write to this repo, you can’t push your changes to it directly. But you can submit a “pull request” to me, asking me to pull your file into the shared repo.

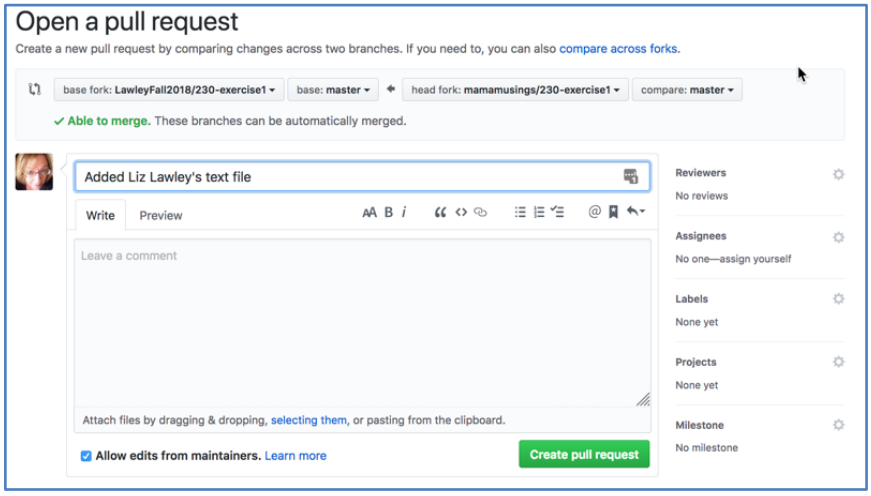
11. Go back to your personal copy of the repo. Notice that above the line saying “This branch is one commit ahead of LawleyFall2017-master,” there’s a button that says “Create pull request”.



12. Click that, and you'll be taken back to the original repo, with a screen showing whether your proposed changes conflict with any changes made by others. Since you're adding a new file, rather than editing an existing file, there shouldn't have been any conflicts. Now click the green button to create a pull request.



13. On the screen that follows, make sure the message has your name in it. Then click the green button at the bottom to send the request.



14. Your pull request has now been submitted for approval by the repo maintainer(s). Your file will not appear in the repo until it's been reviewed and approved. (You can check on the status of your pull requests by selecting "Pull requests" from the menu at the top of the GitHub window.) Once you've submitted the pull request, you can move on to the second part of this exercise—you don't need to wait for it to be approved.

Using Git Part II: Modifying Files in a Shared Repo

1. For this part of the exercise, you'll be using the https://github.com/LawleyFall2018/230-exercise2 repo. This is also a shared repo, but it unlike the exercise1 repo, *everyone in the the class has the ability to directly modify the files*.

2. Instead of making a fork of the repo in your own GitHub account, you're going to clone the shared LawleyFall2018 repo to your computer. Using the process described above in step 3 of Part I, copy the clone URL for the shared repo, and clone it to your local computer in VS Code.

3. Open the classlist.html file in VS Code, find your name, and make it a link to your igme230 directory on people.rit.edu. (e.g. `<a href="people.rit.edu/ellics/igme230">Liz Lawley</a>`)

4. Once you've added the link, stage and commit the classlist.html (and *only* that file!) to your local repo. Make sure your commit message includes your name (e.g. "Added link to Liz Lawley's page")

5. Use the "Sync" command to pull the most recent copy of the file (which may have other student's links already added), and then push your changes. If you've only modified the line with your name, there shouldn't be any conflicts. If you do get a conflict, ask for help before moving on!

6. Once you've successfully updated the document, take a look at the repo on GitHub, and view the classlist.html file. You should see your change reflected in the file.

Grading

There are four points for this exercise:

- 1 point for successfully submitting a pull request to the exercise1 repo
- 1 point for the pull request including the correct content
- 1 point for successfully committing changes to the sharedclasslist.html file in the exercise2 repo
- 1 point for properly including a link from your name to your igme230 page on people.rit.edu in sharedclasslist.html