This document comprises the GIT demonstration steps covered in class on 8/29.

PART 1: CREATING A NEW REMOTE REPOSITORY ON GITHUB.COM AND CLONING THAT IN THE WORKSPACE, GOING BACK AND FORTH

1. *Git help*: to get help with any command. E.g., *git help init* – opens a browser about the init command
2. Configure your name and email ID to be associated with your GIT account:
   a. *Git config –global user.name "<your name>"*
   b. *Git config –global user.email "<your email>"*
3. Create a new repo on github.com. This is where you will collaborate with your team.
4. *Git clone https://github.com/Nimisha-Roy/<the remote repository>* to download remote repository to workspace with a new folder name using HTTPS protocol
5. *Cd myproject*
6. *Echo created a newfile > newfile*: create a new file called newfile
7. *Git add newfile*: stage it
8. *Git commit -m "added new file" :* commit it to local repository
9. *Git push*: push it to the remote repository
10. If someone else made changes to remote and you wanted working directory to reflect those changes → Git pull

The typical user scenario for this will be that each user will have their local copy, work on some local file, commit them and push them to a remote repository where others can get changes, do further changes, push them, and so on and so forth.

PART 2: CREATING AND MERGING BRANCHES

Branching means making a copy of the current project so that we can work on that copy independently from the other copies, be it other branches or the main branch. Then we can decide whether we want to keep both branches or merge them at some point. This is particularly useful because if you think about how we generally develop software, we work with artifacts. For example, we might need to create a separate copy of your work space to do some experiments. You want to change something in the code; you are not sure it will work out, and you do not want to touch your main copy (main branch). So that is the perfect application for branching. If you are happy with the changes, you will merge that branch with the original one; If you are not happy with the changes, you will delete that branch.

Git branch: to see which branches are present. (Until this point of the demonstration, we only had one main branch)

Git branch newBranch : to create a new branch

Git branch: We have two branches now, with the current branch (main) as star marked

Git checkout newBranch: To switch to *newBranch*

Git checkout -b testing: To create a new branch and switch to it

Create a new file called *testfile* in *testing* branch, and push it to the remote repository

Echo this is a testfile > testFile

Git add testFile – staged state

Git commit -m "testfile added" – committed state

Move to the new branch and merge *testing* branch with *main* branch since we are happy with the changes made in the *testing* branch

Git checkout main

Git merge testing: merge testing branch with main

Let us delete the testing branch because it is no longer of any use

Git branch -d testing

Git push

PART 3: BRANCH CONFLICTS

So, something that might happen when you merge a branch is that you might have conflicts, such as changing the same file in two different branches. Let's see an example of that.

Change *newfile* in main branch

Git branch : It shows we have two branches, *main* and *newBranch*

Notepad newfile : Change newfile

Git commit -a -m "new file changed in main branch"

Move to the *newBranch* branch and change *newfile* there

Git checkout newBranch

Notepad newfile : Change newfile again

Git commit -a -m "new file changed in newBranch"

Now, newfile is modified independently in *newBranch* and *main* branch

Move to the *main* branch and merge newBranch

Git checkout main

Git merge newBranch

Conflict message displayed since both branches have independent copies of *newfile*

How to Resolve:

Open *newfile*

You will see annotations showing the different versions in both branches. You can edit that file, decide which version to keep and which to delete, delete the annotations and save the file.

Git commit -a -m "merged version of newfile" – Git already has merged the branches.

Git branch -d newBranch: We have now resolved the conflict and can delete the newBranch

CODE REVIEWS

Pull requests and code reviews play a crucial role in the collaborative development process. A pull request is a mechanism for suggesting changes to a codebase, allowing contributors to propose modifications, bug fixes, or new features. Once a pull request is submitted, it undergoes a code review, where other team members carefully examine the proposed changes. Code reviews help maintain code quality, improve consistency, and catch potential bugs or issues early on. They foster collaboration, knowledge sharing, and ensure that the codebase aligns with established standards. Through pull requests and code reviews, teams can work together effectively, producing high-quality software with fewer errors and better overall maintainability.

Setup your branch protection Rule

Always have a branch protection rule enforced in your main GitHub repository branch.

Settings → Branches → Branch Protection Rule → Require a pull request before merging

Now when you push something to the main repository, you will be prompted to create a PR (image 1 below). You can assign someone or yourself to review the PR (image 2 below). If you enable "Require approvals" in your settings, you will not be able to merge your PR yourself. This is the ideal setting in workplace.

Git checkout newbranch

Echo this is a new feature > feature1

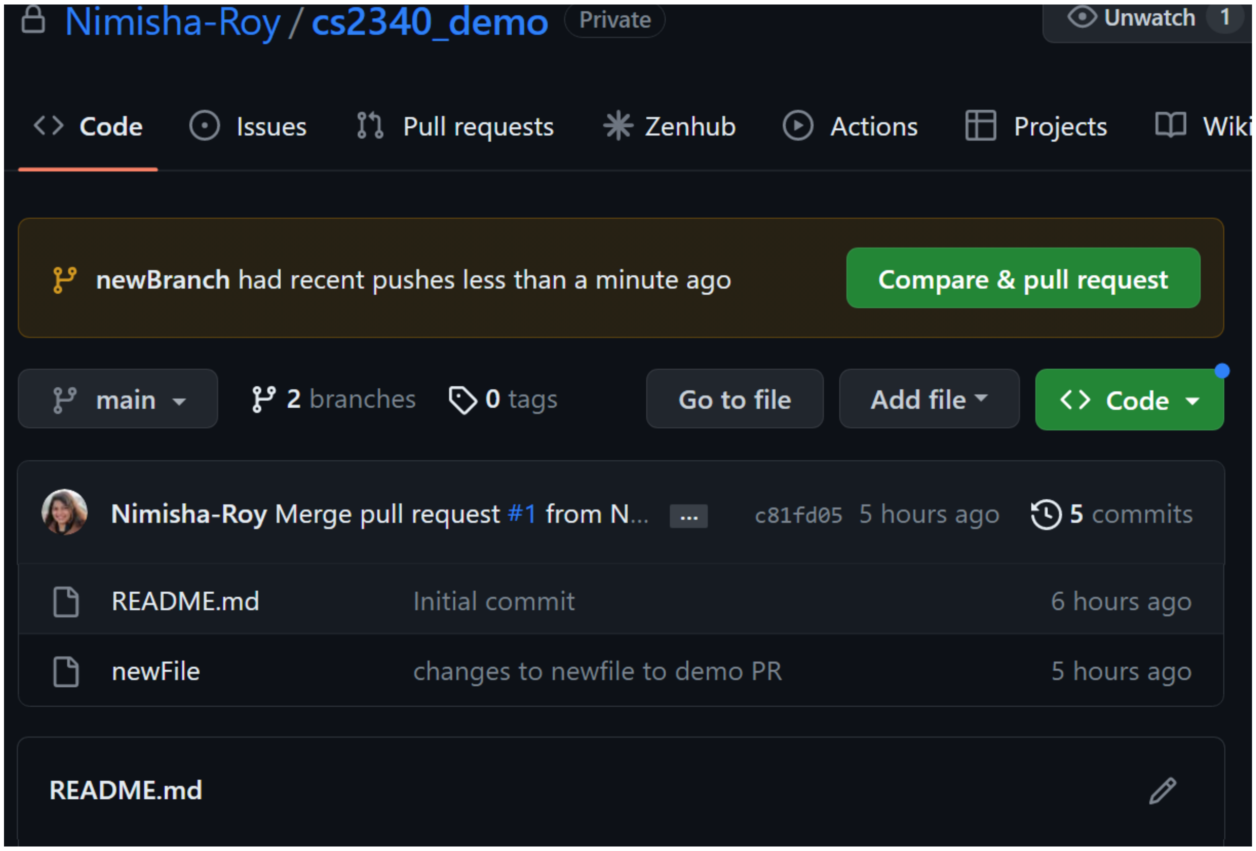
Git add feature1 – staged state

Git commit -m "new feature code added in feature1 branch" – committed state

Git push

Now pull request can be created on github.com

While reviewing a PR, you can compare changes in files (image 3 below), add comments within lines of code (+ sign in image 3 below), and make a final review, either by commenting, approving changes or requesting changes (image 4 below).



Nimisha-Roy / cs2340_demo Private

Unwatch 1

Code Issues Pull requests Zenhub Actions Projects Wiki

newBranch had recent pushes less than a minute ago

Compare & pull request

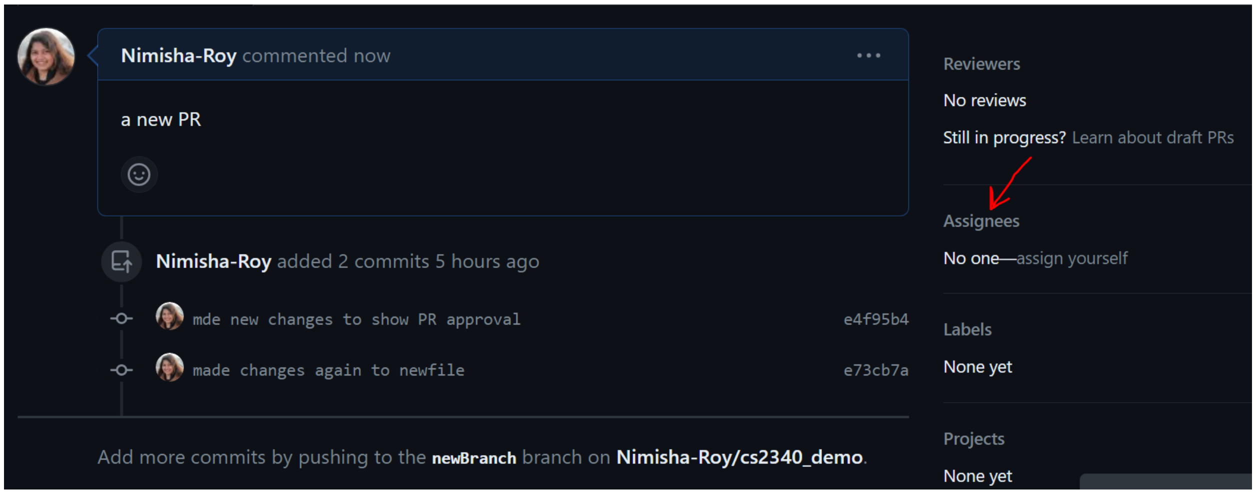
main 2 branches 0 tags Go to file Add file Code

Nimisha-Roy Merge pull request #1 from N... ... c81fd05 5 hours ago 5 commits

README.md	Initial commit	6 hours ago
newFile	changes to newfile to demo PR	5 hours ago

README.md

Image 1



Nimisha-Roy commented now

a new PR

Nimisha-Roy added 2 commits 5 hours ago

```
mde new changes to show PR approval          e4f95b4
made changes again to newfile                e73cb7a
```

Add more commits by pushing to the **newBranch** branch on Nimisha-Roy/cs2340_demo.

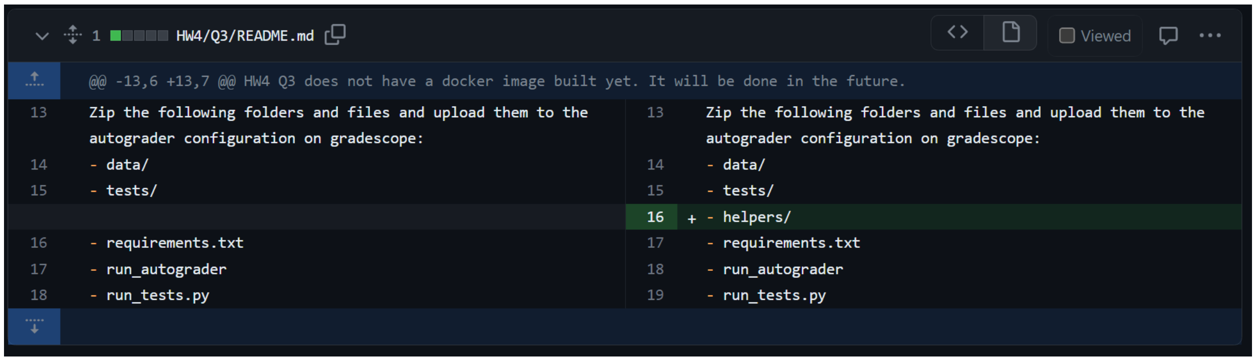
Reviewers
No reviews
Still in progress? Learn about draft PRs

Assignees
No one—assign yourself

Labels
None yet

Projects
None yet

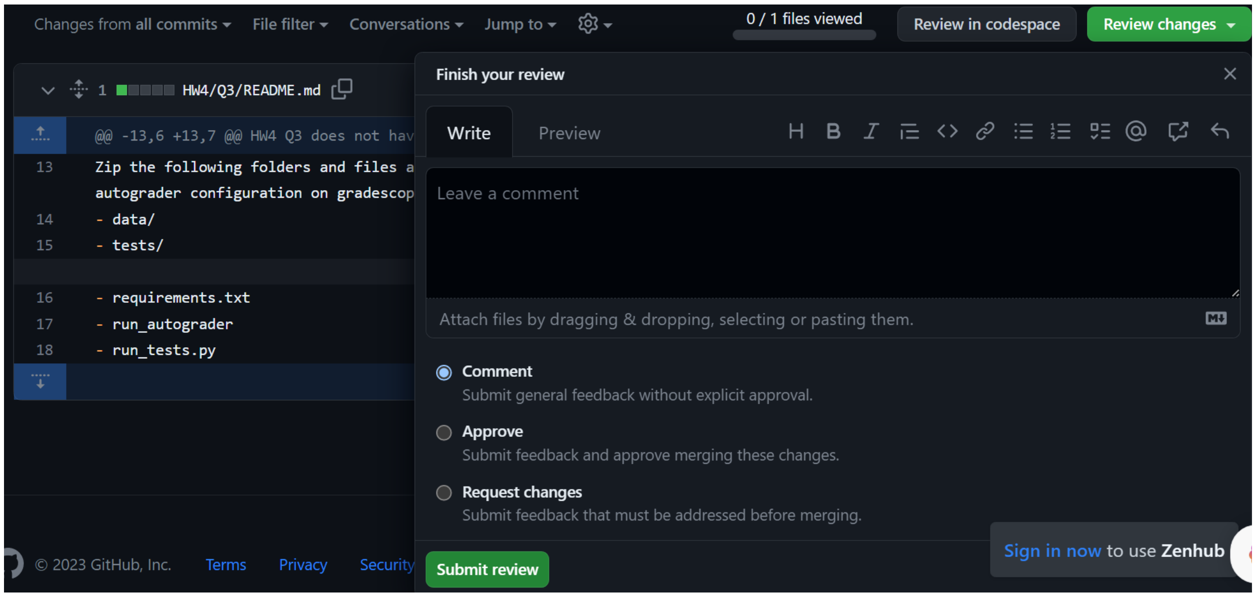
Image 2


HW4/Q3/README.md

Viewed

```
@@ -13,6 +13,7 @@ HW4 Q3 does not have a docker image built yet. It will be done in the future.
13  Zip the following folders and files and upload them to the    13  Zip the following folders and files and upload them to the
    autograder configuration on gradescope:                             autograder configuration on gradescope:
14  - data/                                                          14  - data/
15  - tests/                                                         15  - tests/
                                                                     16 + - helpers/
16  - requirements.txt                                               17  - requirements.txt
17  - run_autograder                                                 18  - run_autograder
18  - run_tests.py                                                   19  - run_tests.py
```

Image 3



HW4/Q3/README.md

```
@@ -13,6 +13,7 @@ HW4 Q3 does not hav
13  Zip the following folders and files a
    autograder configuration on gradescop
14  - data/
15  - tests/
16  - requirements.txt
17  - run_autograder
18  - run_tests.py
```

Finish your review

Leave a comment

Attach files by dragging & dropping, selecting or pasting them.

- [✓ **Comment**] Submit general feedback without explicit approval.
- **Approve** Submit feedback and approve merging these changes.
- **Request changes** Submit feedback that must be addressed before merging.

Image 4