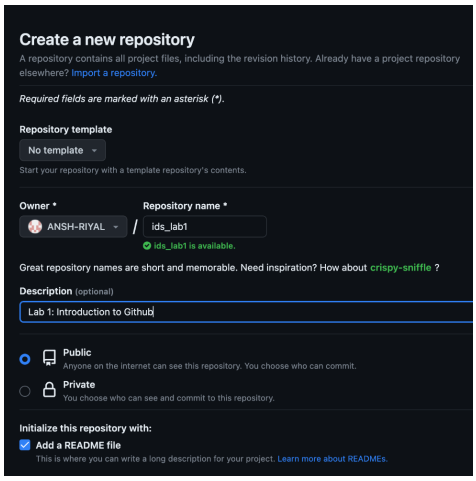
Introduction to Data Science DSGA 1001
Lab 1: Github

1. Make an Account:

Go to github.com and sign up to create your account.
Signup: https://github.com/signup

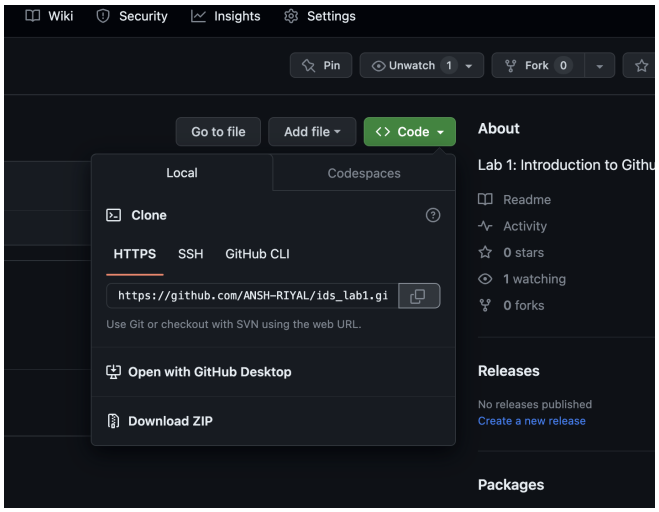
2. Create a Repository

Go to add a new repository, add title, description and click on add a readme file.



Open the repository and edit the README.md file using either your personal text editor or directly from the website.

Copy the repository link



Open terminal and check git is installed properly
Navigate to the directory, clone the repository and navigate to the repository.

```
Last login: Tue Sep  5 23:47:48 on ttys001
anshriyal@10-19-177-233 ~ % git --version
git version 2.39.1
anshriyal@10-19-177-233 ~ % cd Downloads
anshriyal@10-19-177-233 Downloads % mkdir labs_ids_2023
anshriyal@10-19-177-233 Downloads % cd labs_ids_2023
anshriyal@10-19-177-233 labs_ids_2023 % git clone https://github.com/ANSH-RIYAL/ids_lab1.git
Cloning into 'ids_lab1'...
remote: Enumerating objects: 3, done.
remote: Counting objects: 100% (3/3), done.
remote: Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
Receiving objects: 100% (3/3), done.
anshriyal@10-19-177-233 labs_ids_2023 % ls
ids_lab1
anshriyal@10-19-177-233 labs_ids_2023 % cd ids_lab1
anshriyal@10-19-177-233 ids_lab1 % ls
README.md
anshriyal@10-19-177-233 ids_lab1 %
```

Terminal commands:
cd <path>
git clone <repository-url>
cd <repository-name>

3. Git add, commit, push (on main)

Create/modify/delete one or more files on your main branch

```
anshriyal@10-19-177-233 ids_lab1 % echo 'This is the first file on the main branch. echo command is an easy way to create a text file via terminal. So is doing vim main_file1' > main_file1.txt
anshriyal@10-19-177-233 ids_lab1 % ls
README.md       main_file1.txt
anshriyal@10-19-177-233 ids_lab1 % git status
On branch main
Your branch is up to date with 'origin/main'.

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        main_file1.txt

nothing added to commit but untracked files present (use "git add" to track)
```

Tell git to add the file to be tracked

```
anshriyal@10-19-177-233 ids_lab1 % git add main_file1.txt
anshriyal@10-19-177-233 ids_lab1 % git status
On branch main
Your branch is up to date with 'origin/main'.

Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        new file:   main_file1.txt
```

Commit the files along with a message

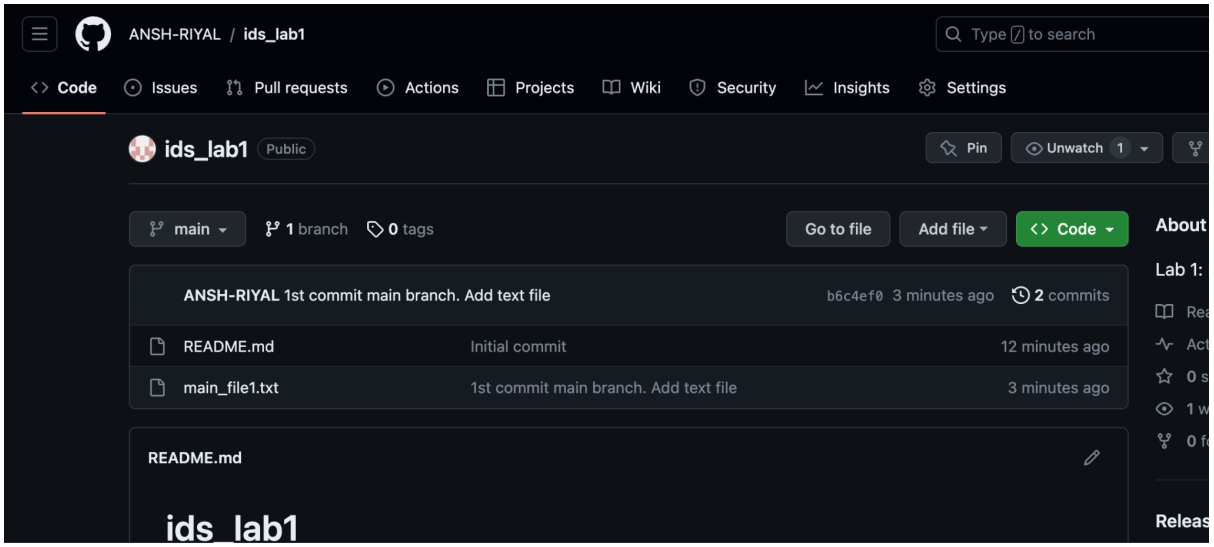
```
anshriyal@10-19-177-233 ids_lab1 % git commit -m'1st commit main branch. Add text file'
[main b6c4ef0] 1st commit main branch. Add text file
 Committer: ANSH-RIYAL <anshriyal@10-19-177-233.dynapool.wireless.nyu.edu>
Your name and email address were configured automatically based
on your username and hostname. Please check that they are accurate.
You can suppress this message by setting them explicitly:

    git config --global user.name "Your Name"
    git config --global user.email you@example.com

After doing this, you may fix the identity used for this commit with:

    git commit --amend --reset-author

 1 file changed, 1 insertion(+)
 create mode 100644 main_file1.txt
anshriyal@10-19-177-233 ids_lab1 %
```

Push the latest commit(s) to the main branch on the repository.

```
[anshriyal@10-19-177-233 ids_lab1 % git push origin main
 Enumerating objects: 4, done.
 Counting objects: 100% (4/4), done.
 Delta compression using up to 8 threads
 Compressing objects: 100% (3/3), done.
 Writing objects: 100% (3/3), 426 bytes | 426.00 KiB/s, done.
 Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
 To https://github.com/ANSH-RIYAL/ids_lab1.git
    6ae6fd5..b6c4ef0  main -> main
```

Terminal commands:
echo ‘text file for main branch commit’ > main_file1.txt
git add main_file1.txt
git commit -m’1st commit main branch. Added text file’
git push origin main

Now check out your repository on the github page of your repository.



4. Create Branch and switch branch

Create a branch using git branch command. Then switch to the new branch using git checkout

```
[anshriyal@10-19-177-233 ids_lab1 % git status
 On branch main
 Your branch is up to date with 'origin/main'.

 nothing to commit, working tree clean
[anshriyal@10-19-177-233 ids_lab1 % git branch test_branch
[anshriyal@10-19-177-233 ids_lab1 % git checkout test_branch
 Switched to branch 'test_branch'
[anshriyal@10-19-177-233 ids_lab1 % git status
 On branch test_branch
 nothing to commit, working tree clean
 anshriyal@10-19-177-233 ids_lab1 %
```

*(use git pull to make sure your branch is up to date with the work already done before starting new stuff to avoid any merge conflicts)

Add/modify/delete file(s) in the new branch and commit those changes in the branch.

```
anshriyal@10-19-177-233 ids_lab1 % echo 'text file for test_branch' > branch_file1.txt
anshriyal@10-19-177-233 ids_lab1 % git add branch_file1.txt
anshriyal@10-19-177-233 ids_lab1 % git commit -m'This is the first commit in the test branch'
[test_branch 91fc30a] This is the first commit in the test branch
 Committer: ANSH-RIYAL <anshriyal@10-19-177-233.dynapool.wireless.nyu.edu>
Your name and email address were configured automatically based
on your username and hostname. Please check that they are accurate.
You can suppress this message by setting them explicitly:

    git config --global user.name "Your Name"
    git config --global user.email you@example.com

After doing this, you may fix the identity used for this commit with:

    git commit --amend --reset-author

 1 file changed, 1 insertion(+)
 create mode 100644 branch_file1.txt
anshriyal@10-19-177-233 ids_lab1 % echo 'text file number 2 for test_branch' > branch_file2.txt
anshriyal@10-19-177-233 ids_lab1 % git add branch_file2.txt
anshriyal@10-19-177-233 ids_lab1 % git commit -m'This is the second commit in the test branch'
[test_branch 6f4dfbe] This is the second commit in the test branch
 Committer: ANSH-RIYAL <anshriyal@10-19-177-233.dynapool.wireless.nyu.edu>
Your name and email address were configured automatically based
on your username and hostname. Please check that they are accurate.
You can suppress this message by setting them explicitly:

    git config --global user.name "Your Name"
    git config --global user.email you@example.com

After doing this, you may fix the identity used for this commit with:

    git commit --amend --reset-author

 1 file changed, 1 insertion(+)
 create mode 100644 branch_file2.txt
anshriyal@10-19-177-233 ids_lab1 % git status
On branch test_branch
```

Now push the changes you have made on the test branch using:
git push origin test_branch

```
anshriyal@10-19-177-233 ids_lab1 % git push origin test_branch
Enumerating objects: 7, done.
Counting objects: 100% (7/7), done.
Delta compression using up to 8 threads
Compressing objects: 100% (4/4), done.
Writing objects: 100% (6/6), 683 bytes | 683.00 KiB/s, done.
Total 6 (delta 1), reused 0 (delta 0), pack-reused 0
remote: Resolving deltas: 100% (1/1), done.
remote:
remote: Create a pull request for 'test_branch' on GitHub by visiting:
remote:      https://github.com/ANSH-RIYAL/ids_lab1/pull/new/test_branch
remote:
To https://github.com/ANSH-RIYAL/ids_lab1.git
 * [new branch]      test_branch -> test_branch
anshriyal@10-19-177-233 ids_lab1 %
```

Terminal commands:

git branch test_branch
git checkout test_branch
echo ‘test file for test branch’ > branch_file1.txt
git add branch_file1.txt
git commit -m’This is the first commit in the test_branch’

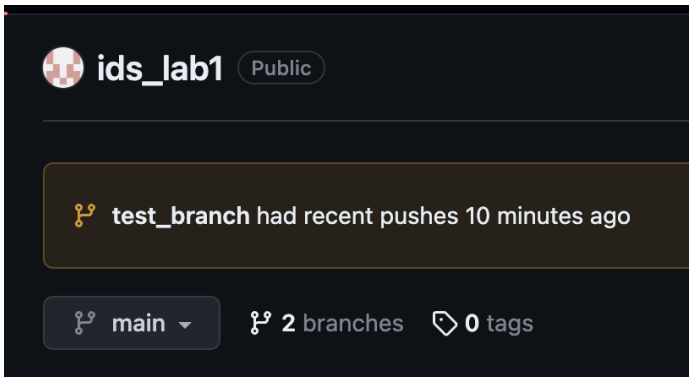
echo ‘text file number 2 for test_branch’ > branch_file2.txt
git add branch_file2.txt
git commit -m’This is the second commit in the test_branch’

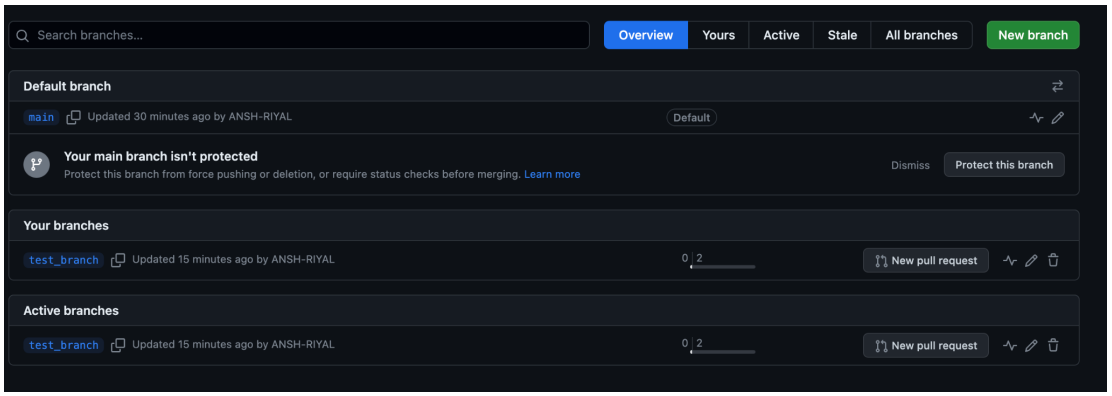
git push origin test_branch

*: there are cases where multiple people are working on the same files. It is always a good idea to keep your changes up to date with the repository's current state. Using git pull basically pulls all the changes from the github repository online to your local device.

5. Merging branches

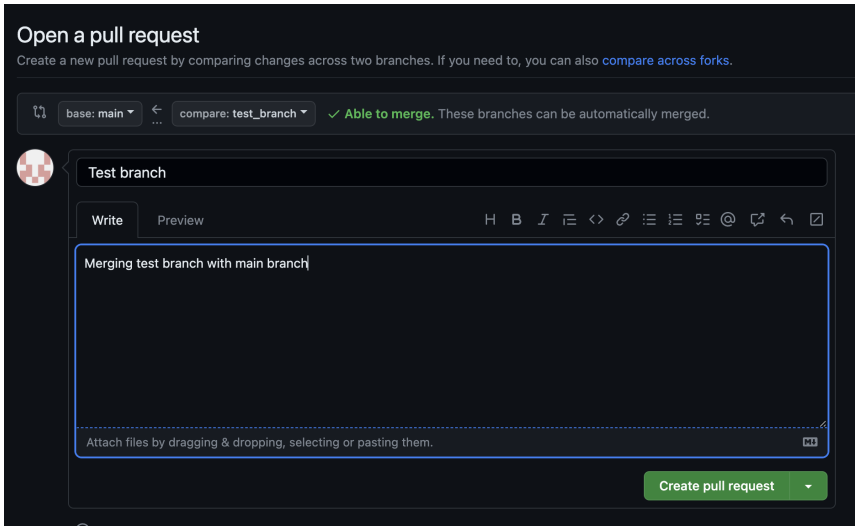
Open your repository on github.com and open the branches



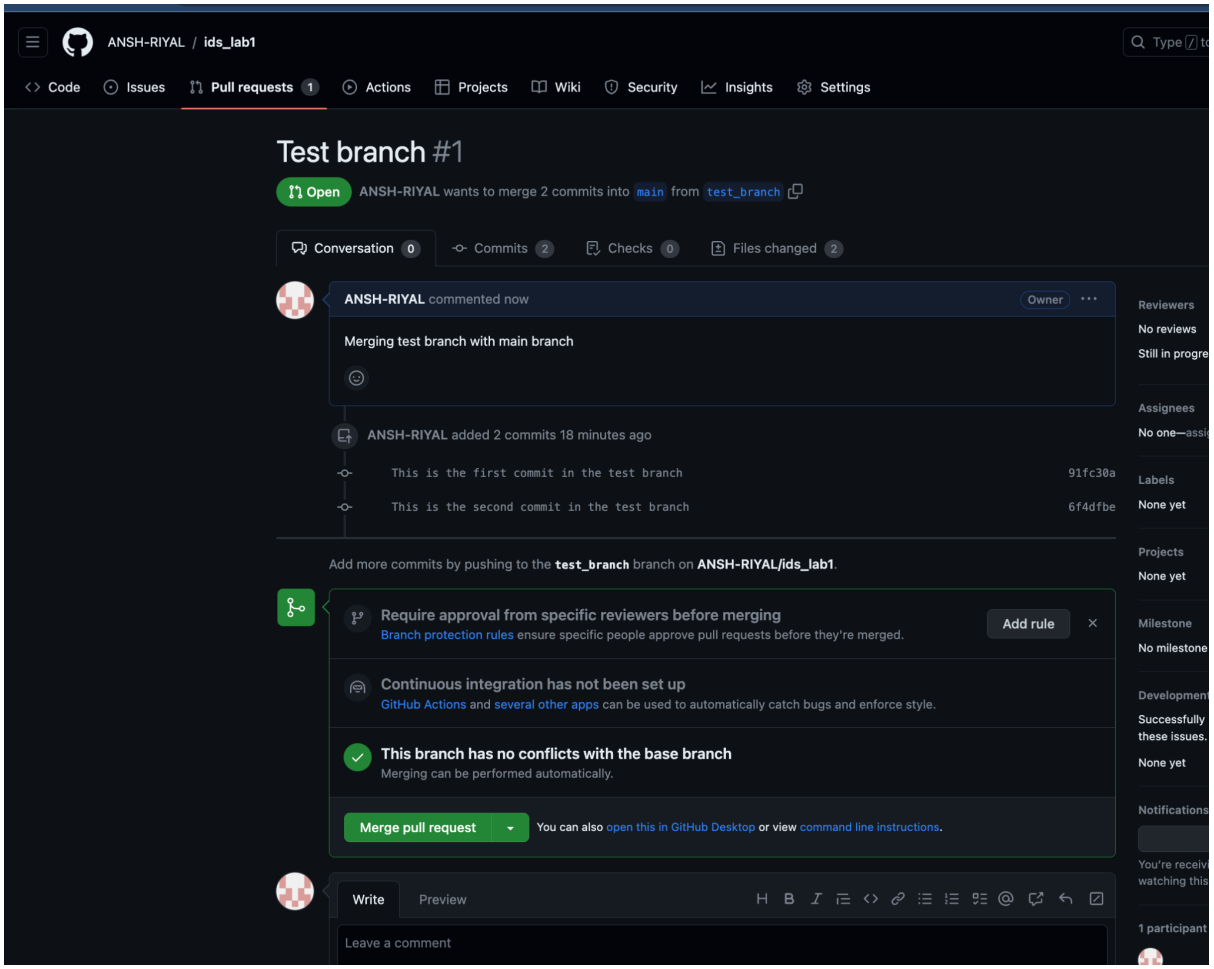


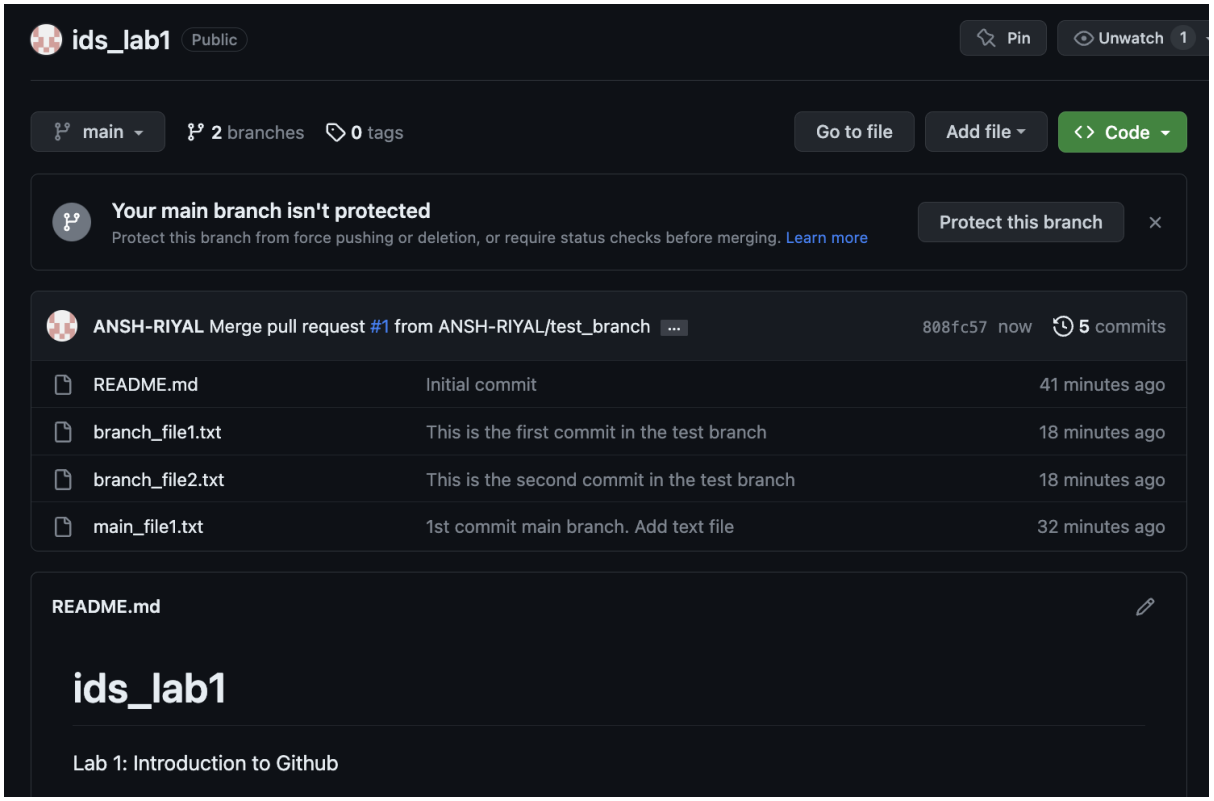
(observe the branches being ahead of main by some commits or behind by some commits)

Review your changes and commits and then do a pull request.
(Github does an automatic check for merge conflicts)



Your colleagues who have access will review your pull request and accept the changes and combine the branch with your main branch if there are no merge conflicts.



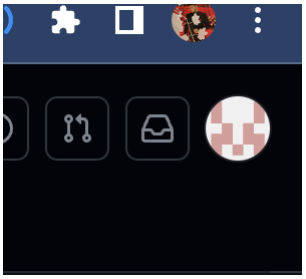


After Merging, we have incorporated all the changes from the test_branch into the main branch. When 2 branches have conflicting versions of the same file and the changes are not completely independent of each other, then git doesn't understand how to merge the branches or which version of which function is ideal, so we get a merge error

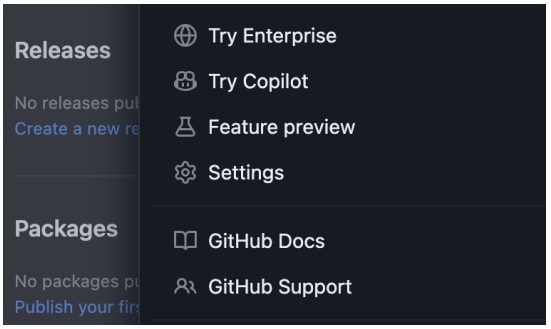
Some noteworthy tips:

1. Create Personal Access Tokens for https authentication:

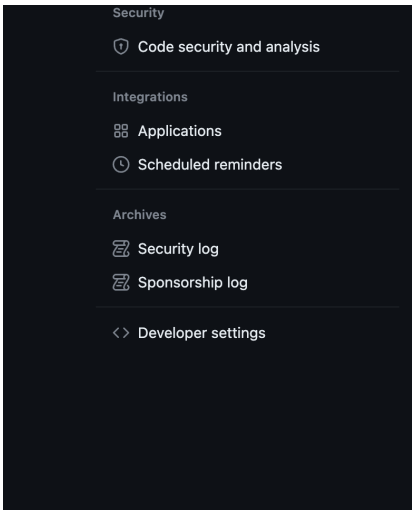
   Click on top right



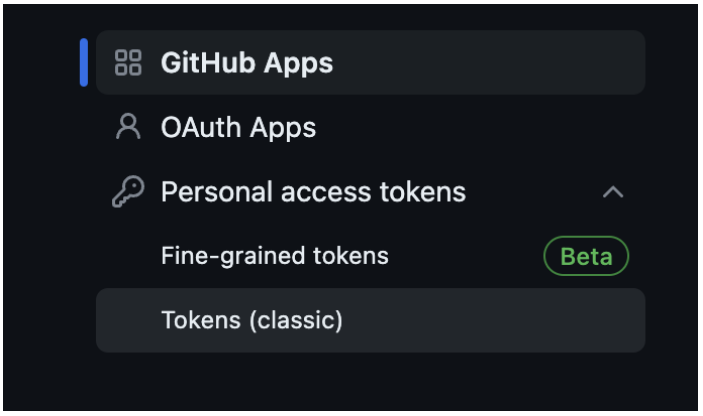
   Go to settings



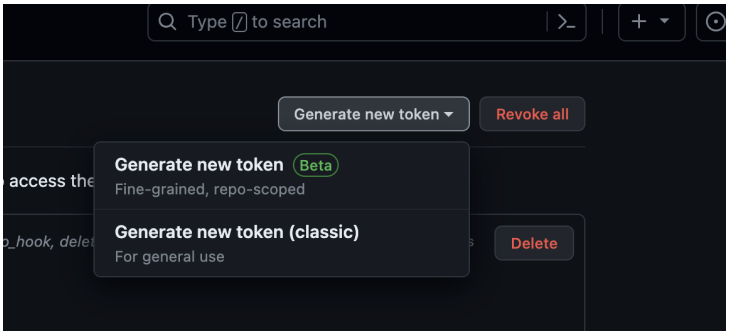
   Developer settings



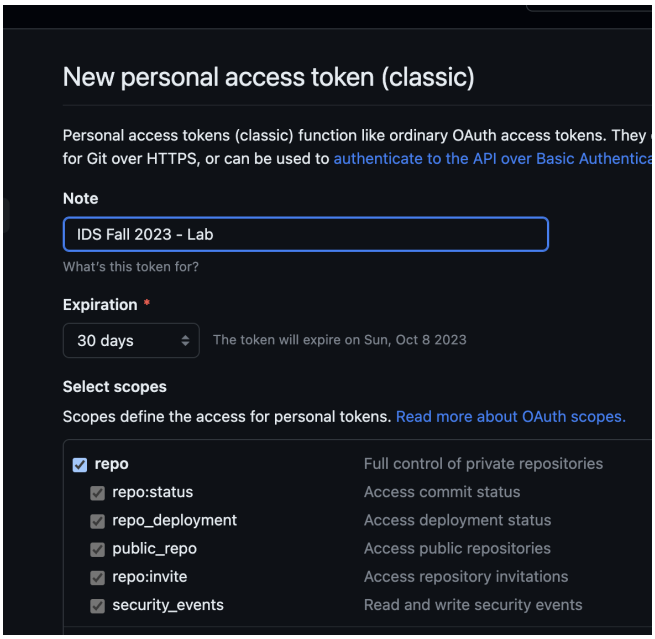
   Personal Access tokens -> tokens (classic)



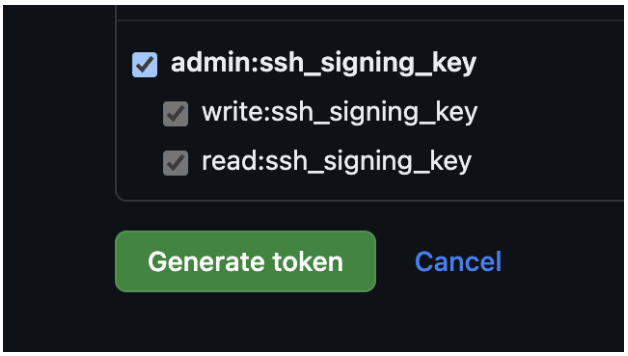
   Generate New token -> Generate New Token(classic):



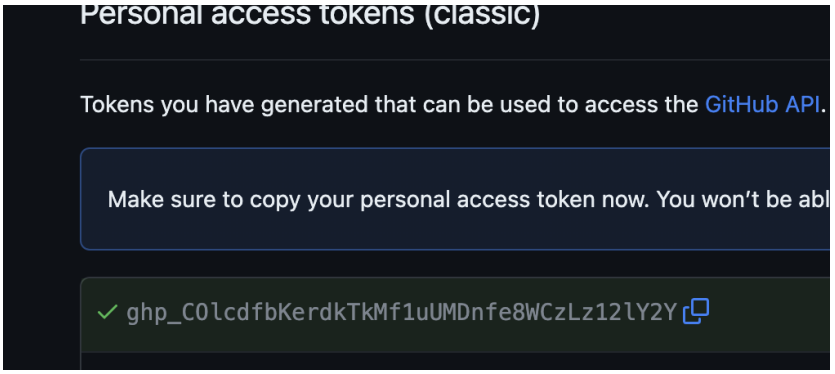
Give it a Note, an expiration date and add scope (permissions for the token)



Generate token



Copy the generated token (this key is for demo purposes, please don't copy my key…)



Save your token somewhere, this will act as your password.

2. Saving credentials in terminal: (using the https authentication)

If you also get annoyed when writing your user id and password everytime

1. Install git-credential-manager from homebrew(mac)
2. Run the following command by replacing the username and the password

printf "protocol=https\nhost=github.com\nusername=<REPLACE THIS WITH YOUR USERNAME>\npassword=<REPLACE THIS WITH YOUR TOKEN>" | git credential-manager store

Eg. if your username is ‘Samaritan-Good’ and your Token is ‘9fn4rlkdfhv0’ then run the following on terminal(with homebrew installed):

brew install git-credential-manager

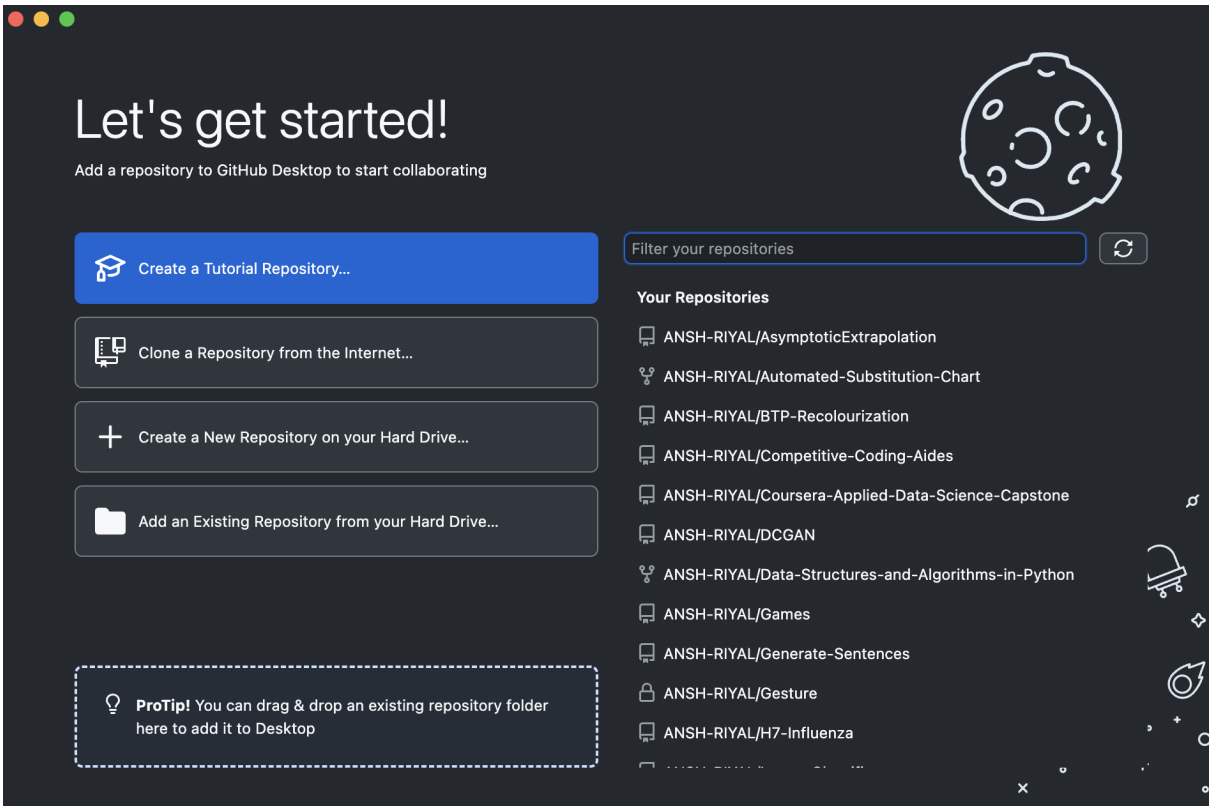
printf "protocol=https\nhost=github.com\nusername=Samaritan-Good\npassword=9fn4rlkdfhv0" | git credential-manager store

After this, when you use the git commands which needed username and password will be handled by the credential manager.

3. Github Desktop:

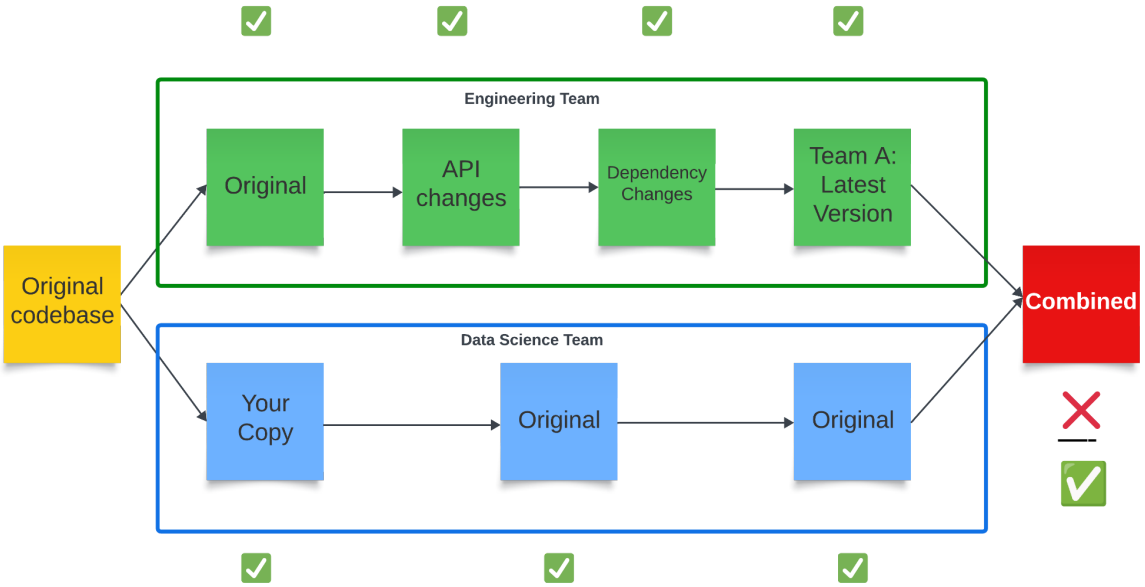
Github desktop is an amazing tool with a clean UI and it simplifies all the git related actions you can take through the github website and the terminal/command prompt (cmd)

Website: https://desktop.github.com/



4. Using ssh authentication instead of https: https://docs.github.com/en/authentication/connecting-to-github-with-ssh/generating-a-new-ssh-key-and-adding-it-to-the-ssh-agent

5. Branching (Visualisation):



In the slides, the representation of git branches are a little unclear, so I am adding this example as a way to visualise the timeline of creating branches, modifying and finally merging them.

Scenario: You work as a data scientist in a big tech company. You have to work on the same project as the engineering team, but you wish to utilize the branching capabilities of github and then finally when both teams are done with their tasks, merge everything together.

Over here, the original codebase/repository was branched using the **git branch** command. The new branch is for the Data Science Team, while the main branch is being used by the Engineering team/ Software development team.
They have their own set of commits and pushes on their respective branches (following the order of commands: **git add, git commit, git push**).
Finally, when we use the **git merge** command on the data science team, Github checks the 2 branches to see if there are any conflicts.
If there are no conflicts, you can go ahead (✅) and start your pull request to finish merging everything. However if there are conflicts(❌), Github will not be able to automatically merge the branches and then you need to review the branches' commit history in order to restore the proper versions of the files under conflict.
Advice: Do not blindly automatically merge branches when there is a conflict, even though the branches are technically merged together, but there are sometimes unintended changes in your files which often go undetected.