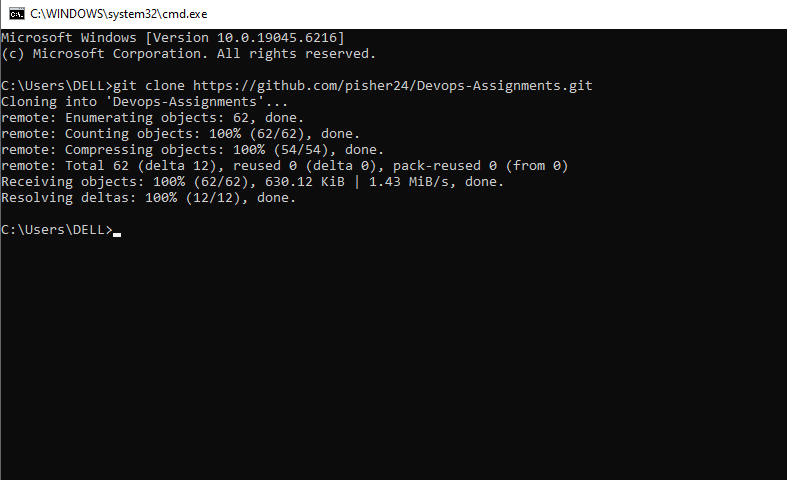
1. Create a new GitHub repository.

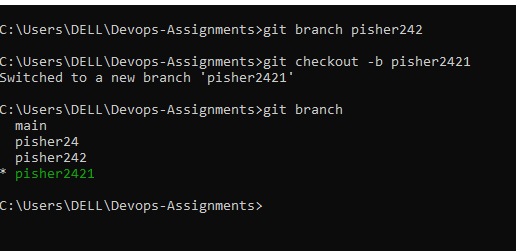
* Clone the repository to your local machine using SSH (generate an SSH key if needed, add the public key to your GitHub account).

We can use git clone command to clone the git repos to our local machine



* Create a new branch named after your username (e.g., Tutedude).

To create a new branch we can use git branch command or git checkout with –b to create a new branch and check out to it

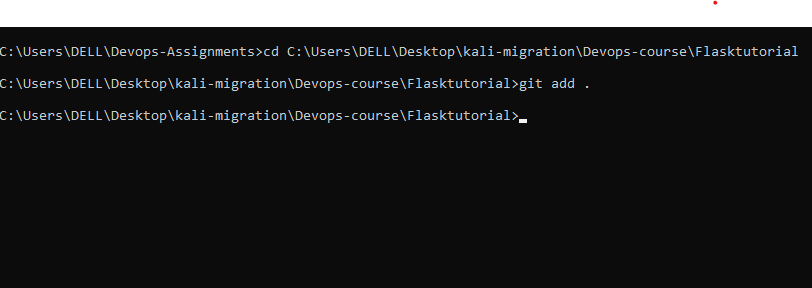


* Add your Flask project files to this branch.

In order to add project files in a specific branch we can use the git add command. I did cd to another folder because my project files are in that folder and to actually move all the files inside that folder we used the “.” Along with git add

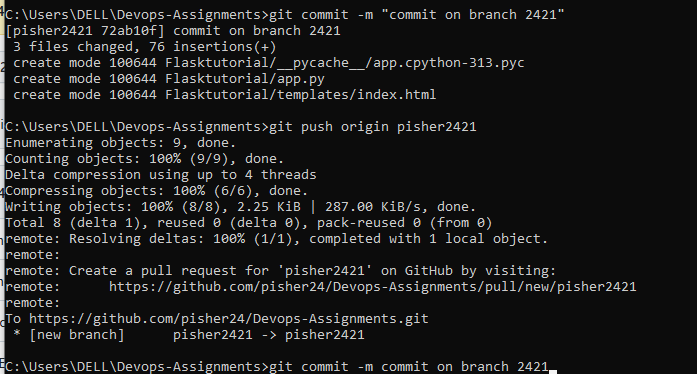
Git checkout pisher2421

Cd path to directory and then finally used git add



* Commit the changes and merge the branch into the main branch

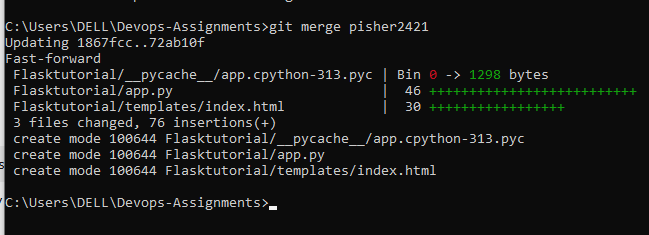
To commit the changes, we can use the git comit command commit only saves the changes to files in the repository . to actually push the changes to the repo we use git push command



To merge the pisher2421 branch with main branch this what we do

Git checkout main

Git merge pisoher2421. Here pisher2421 is the branch that I want to merge with main and finally push the updated main branch to github



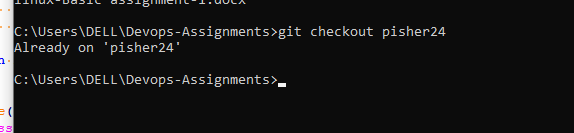
.

Create a new branch with my user name

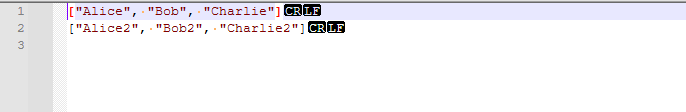
* Update the content of the JSON file used for the /api route in this branch.

To update the specific file in a branch this is what u have to follow

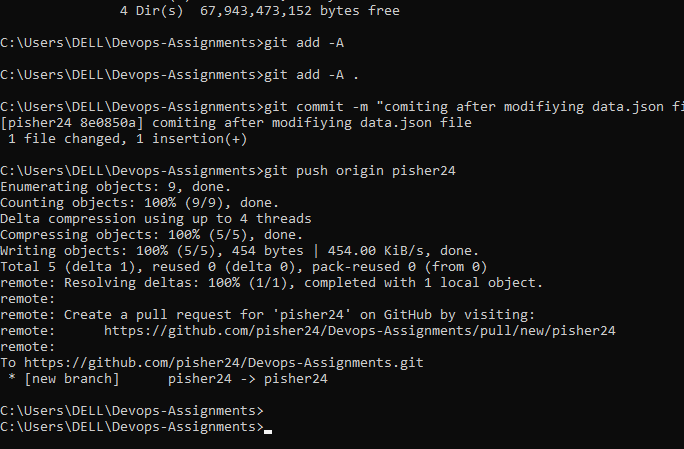
First switch to that branch**: git checkout pisher24**

****

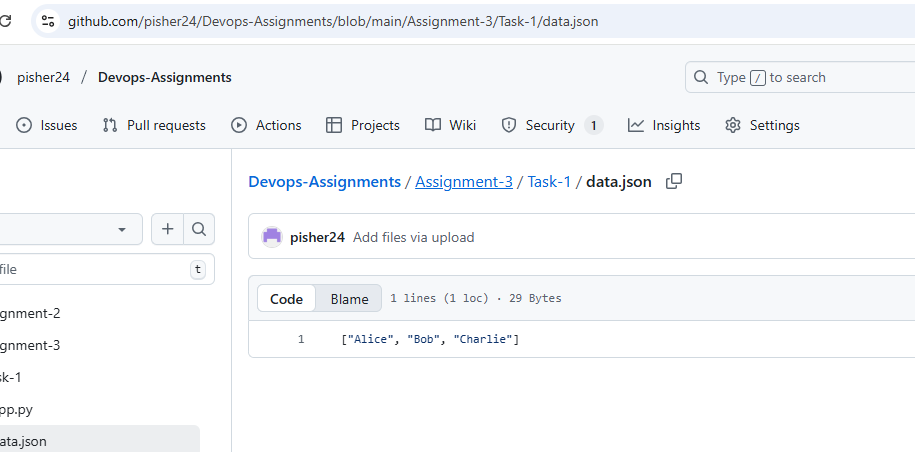
Now open the file u want edit in vscode and then make the changes and commit it

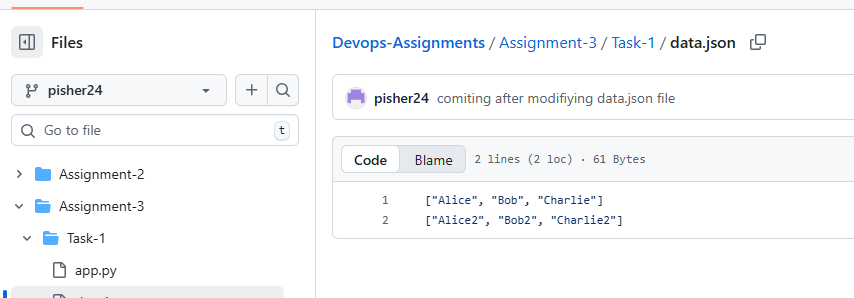


Now we can use git add command with –A flag to stage all the changes and the finally push the changes to github.



For verification u can looj at the json file in both branches





* Merge the <your\_name>\_new branch into the main branch.

In order to merge the repos use git merge command but before that checkout to main branch first

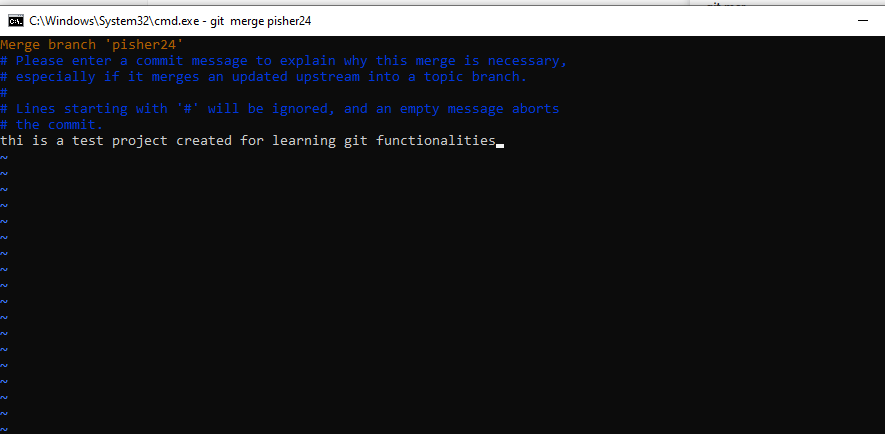
Git checkout main

Git merge pisher24

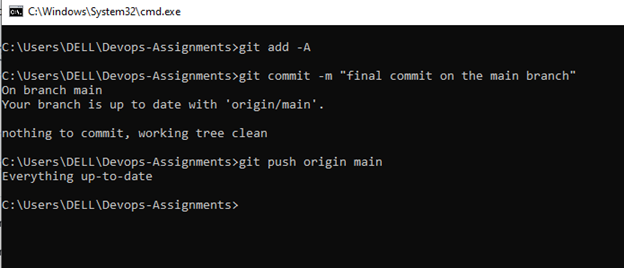
Git push origin main

* If there are conflicts during the merge, resolve them by accepting the changes from the <your\_name>\_new branch.

while doing the previous task a conflict arose saying that a merge comment is required so resolved it by adding the comment



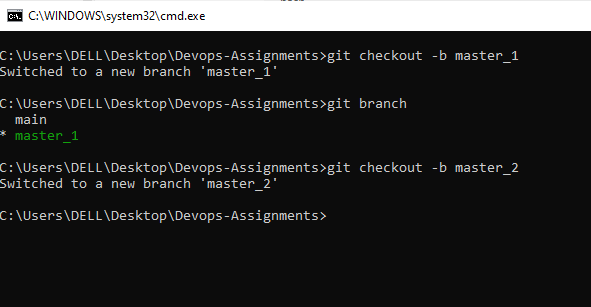
* Add the resolved changes to the staging area, commit them, and push the updates to the remote repository. Add the resolved changes to the staging area, commit them, and push the updates to the remote repository.

v

**3 Branch creation**

* Create two branches: master\_1 and master\_2 from the main branch.

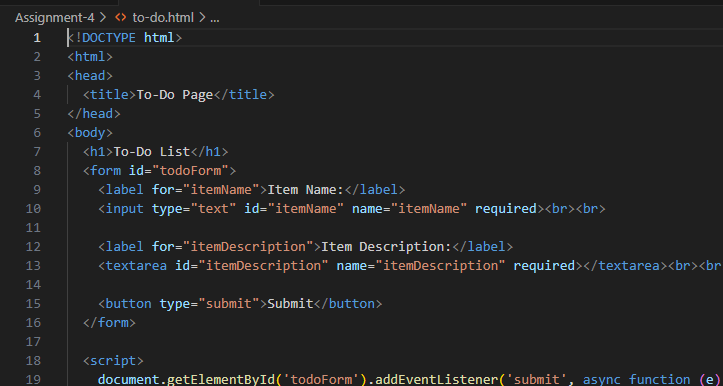
Create two branches master\_1 and master\_2 like this

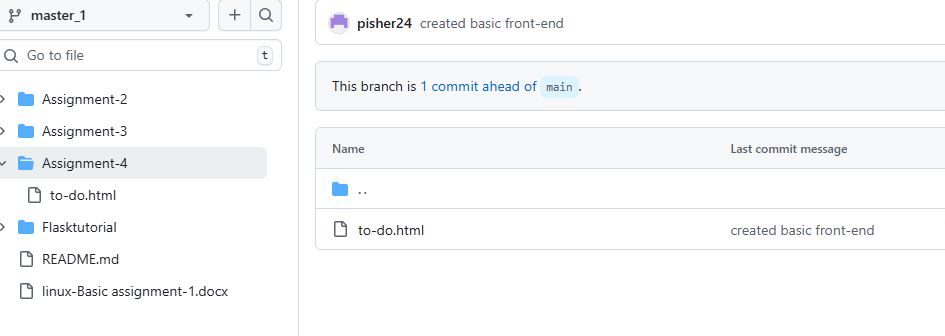


* **Feature Development in** master\_1:
* In the master\_1 branch, create a **To-Do Page** in the frontend.
  + The page should contain a form with the following fields:
    - **Item Name**
    - **Item Description**

Let’s start creating the front end of to do list.

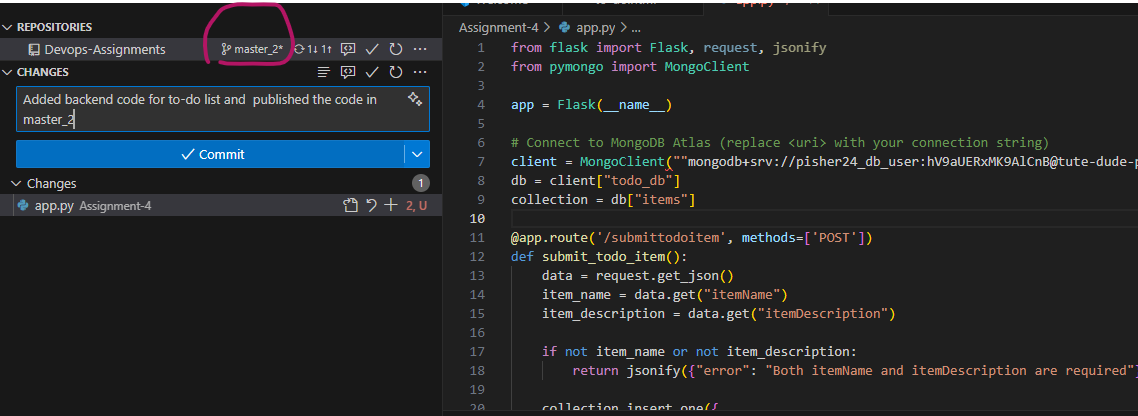
Create a to do html file and upload it to master\_1 branch





* **Backend API in** master\_2:
* In the master\_2 branch, create a backend route named /submittodoitem.
* This route will:
  + Accept itemName and itemDescription via a POST request.
  + :Store these details in a MongoDB database.

Solution:: Now lets focus on creating the backend code of the corresponding front end. Now lets move to master\_2 branch and create a app.py file and starting coding on



* **Merging Changes**:
* Merge the changes from both master\_1 and master\_2 into the main branch

We can merge master\_1 and 2 into the main branch like this

Git checkout main

Git merge master\_1

Git merge master\_2

Git push origin main

4. **Enhancing the To-Do Form in** master\_1:

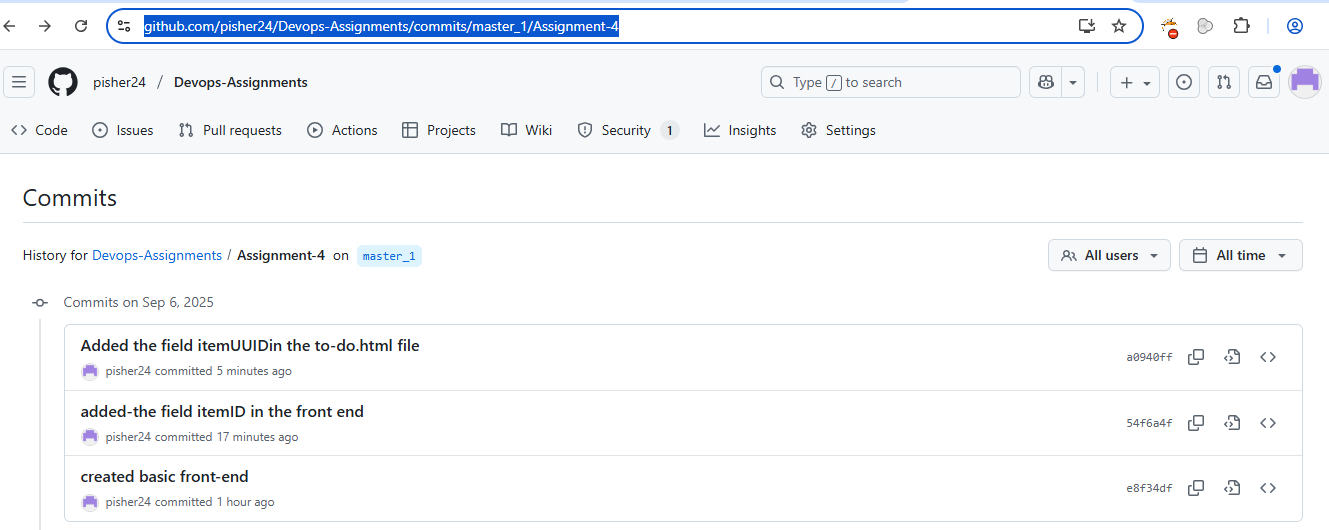
* In the master\_1 branch, add the following fields to the To-Do form:
  + **Item ID**
  + **Item UUID**
  + **Item Hash**
* **Committing in Sequence**:
* Add and commit each field separately in the following order:
  + **First commit**: Add **Item ID** field.
  + **Second commit**: Add **Item UUID** field.
  + **Third commit**: Add **Item Hash** field.

**solution:** For the modification of code in master\_1 branch first checkout the corresponding branch and then modify the to-do.html file and the commit then type the below commands

git add todo.html

git commit -m "Added Item ID field to to-do.html" **once that is done create two other** fields on the front-end and follow same process for for committing the changes. Once commiting is done u can look at git hub commit history of the file in master\_1 branch. There u can see all the commits created for the file

<https://github.com/pisher24/Devops-Assignments/commits/master_1/Assignment-4>



**Merging to** main::

* Merge the master\_1 branch into the main branch.

Inorder to merge master\_1 branch to the main branch first checkout to main branch and type the following command to merge

**Git merge master\_1**

**Git push origin main**

**Git Reset and Commit Deletion**:

* In the main branch, use **Git Reset** to roll back to the commit where only the **Item ID** field was added..
* Use git reset --soft to ensure changes remain staged.

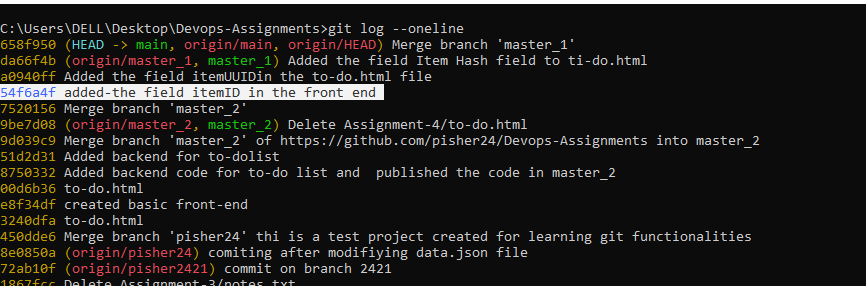
To rollback to a purticular commit in the main branch first navigate to main branch by typing

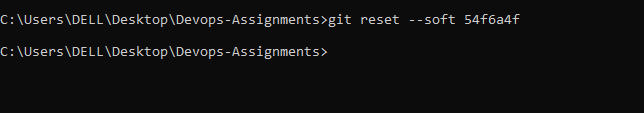
**Git checkout main and then use**

**git log –oneline** to pull commit history from that history we have to identify the commit id of the specific commit that we want out main branch to roll backto . Once we identify the commit id we can then use the reset option like the below to roll back

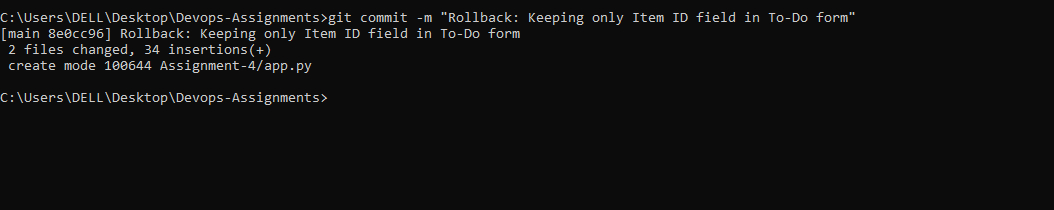
**git reset --soft <commit id>**

--soft keeps UUID + Hash changes staged but not committed.



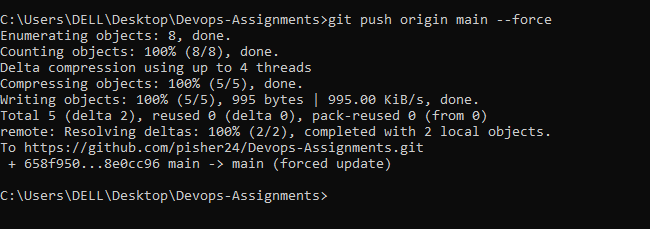


* Re-commit this state to the main branch.

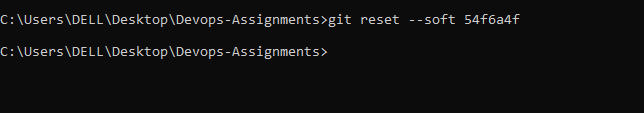


* Merge this updated state to the main branch.

git push origin main –force



**Rebasing Changes**:

* Rebase the updated changes in the main branch to the master\_1 branch.  
  **Clarification**:
  + During rebasing, **preserve individual commits** to maintain the commit history for each change (i.e., do not squash commits).
  + Use git rebase main master\_1 to integrate changes from the main branch back into the master\_1 branch.
  + 

After this we can just checkout to the master\_1 branch and just do

Git rebase main to get the desired changes

**Submission Guidelines -:** Attach Screenshots or command along with explanation and submit in doc(google doc or microsoft doc) format , also share link of your github repo