Assignment 4:

Version Control Git

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).
* Create a new branch named after your username (e.g., Tutedude).
* Add your Flask project files to this branch.
* Commit the changes and merge the branch into the main branch.

Answer:

**1. Repository Setup**

**Steps:**

1. Create a **new GitHub repository** (e.g., flask-todo-app).
2. Clone repository using SSH:
3. git clone [git@github.com:ChauhanDivyanshu/flask-todo-app.git](mailto:git@github.com:ChauhanDivyanshu/flask-todo-app.git)
4. cd flask-todo-app
5. Create a branch with your username:
6. git checkout -b < ChauhanDivyanshu >
7. Add Flask project files:
8. git add .
9. git commit -m "flask-todo-app"
10. git push origin < ChauhanDivyanshu >
11. Merge branch into main:
12. git checkout main
13. git merge < ChauhanDivyanshu >
14. git push origin main

2. Create a new branch named <your\_name>\_new (e.g., Tutedude\_new).

* Update the content of the JSON file used for the /api route in this branch.
* Merge the <your\_name>\_new branch into the main branch.
* If there are conflicts during the merge, resolve them by accepting the changes from the <your\_name>\_new branch.
* Add the resolved changes to the staging area, commit them, and push the updates to the remote repository.

Answer:

**Steps:**

1. Create new branch:
2. git checkout -b <ChauhanDivyanshu>\_new
3. Update JSON file used in /api route.
4. Commit & push:
5. git add data.json
6. git commit -m "Updated JSON file for /api route"
7. git push origin < ChauhanDivyanshu >\_new
8. Merge into main:
9. git checkout main
10. git merge < ChauhanDivyanshu >\_new
    * If conflicts:
    * git status # check conflicts
    * # Open conflicted file, keep < ChauhanDivyanshu>\_new version
    * git add <conflicted\_file>
    * git commit -m "Resolved merge conflicts by accepting < ChauhanDivyanshu>\_new"
    * git push origin main

3. **Branch Creation**:

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

Answer:

**Steps:**

1. Create branches:
2. git checkout main
3. git checkout -b master\_1
4. git checkout main
5. git checkout -b master\_2

**In master\_1:**

* Create a **To-Do Page** in frontend with form fields:
  + Item Name
  + Item Description

git add frontend/todo.html

git commit -m "Added To-Do Page with Item Name and Item Description fields"

git push origin master\_1

**In master\_2:**

* Create backend route /submittodoitem (Flask):

@app.route('/submittodoitem', methods=['POST'])

def submit\_todo\_item():

data = request.get\_json()

itemName = data.get('itemName')

itemDescription = data.get('itemDescription')

db.todos.insert\_one({"name": itemName, "description": itemDescription})

return jsonify({"message": "Item added successfully"}), 201

git add app.py

git commit -m "Created backend /submittodoitem route with MongoDB"

git push origin master\_2

**Merge both branches into main:**

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.
* **Merging to** main:
* Merge the master\_1 branch into the main branch.
* **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.
* Re-commit this state to the main branch.
* Merge this updated state to the main branch.
* **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.

**Answer:**

**4. Enhancing the To-Do Form**

**In master\_1:**

1. Add **Item ID field** → Commit:
2. git add frontend/todo.html
3. git commit -m "Added Item ID field"
4. Add **Item UUID field** → Commit:
5. git add frontend/todo.html
6. git commit -m "Added Item UUID field"
7. Add **Item Hash field** → Commit:
8. git add frontend/todo.html
9. git commit -m "Added Item Hash field"
10. git push origin master\_1

**Merge into main:**

git checkout main

git merge master\_1

git push origin main

**5. Git Reset & Re-commit**

**Roll back main branch to first commit (Item ID only):**

git log # copy commit hash of "Added Item ID field"

git reset --soft <commit\_hash>

git commit -m "Re-commit only Item ID field"

git push origin main --force

**6. Rebasing**

**Rebase updated main branch into master\_1:**

git checkout master\_1

git rebase main

git push origin master\_1 --force

* Preserve commits individually (no squash).
* Resolve conflicts (if any) with:
* git status
* git add <conflicted\_file>
* git rebase --continue