WEEK 3 Work flow for lab Conduction

Objectives:

1. Demonstrating Collaborative coding.
2. To resolve conflicts when collaborating on same part of code.
3. Demonstrating how to create and apply patch
4. Scenario Based Questions
5. **Collaborative coding.**

Code collaboration refers to the process of multiple people working together on the same codebase. This typically involves sharing code changes and reviewing each other's work in order to improve the quality and reliability of the code.

There are **two ways** of collaborative coding:

1. Creating Organization
2. Coordinating with others through a shared repository
3. **Facilitating Collaborative Work by creating Organization**

Organization - Organizations are shared accounts where businesses and open-source projects can collaborate across many projects at once. Owners and administrators can manage member access to the organization's data and projects with sophisticated security and administrative features. Steps to facilitate collaborative work by creating organization are:

**Step 1:** Click on New organization in Git hub

Click on Create a free organization.

S**tep 2:** Set up your organization by entering the details like name, associated email, account verification, etc.

Click on Next

**Step 3**: Complete the setup by

* adding members to the group.
* Now, the invitation goes to other collaborator and they must accept.
* Next click on complete setup
* Goto Github and click on

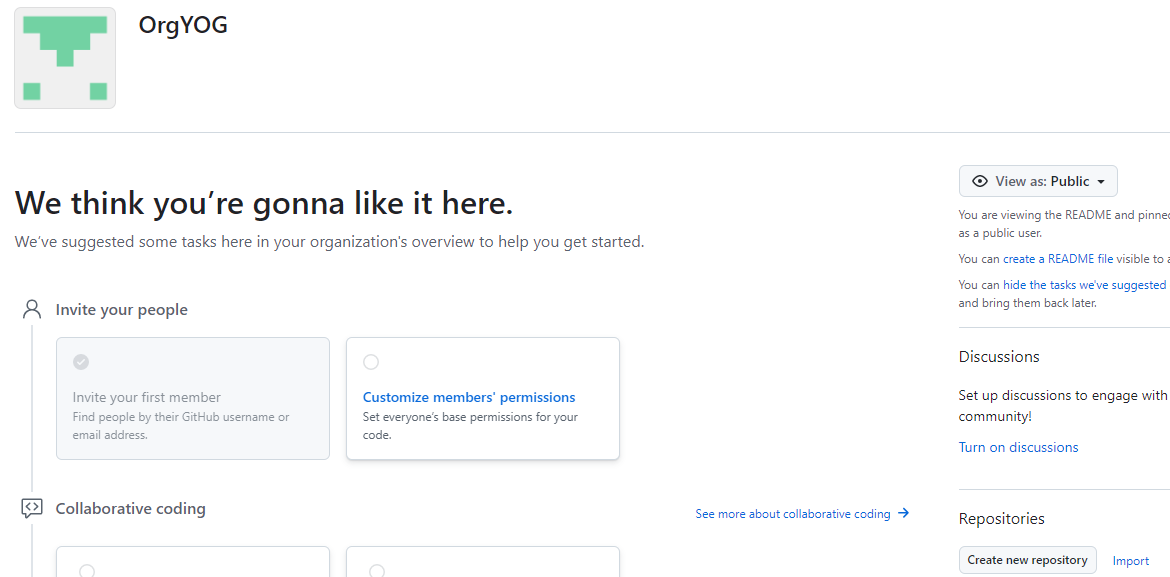
### Settings 🡪 Member privileges 🡪 Base permissions 🡪 select write 🡪 change base permission to “Write”

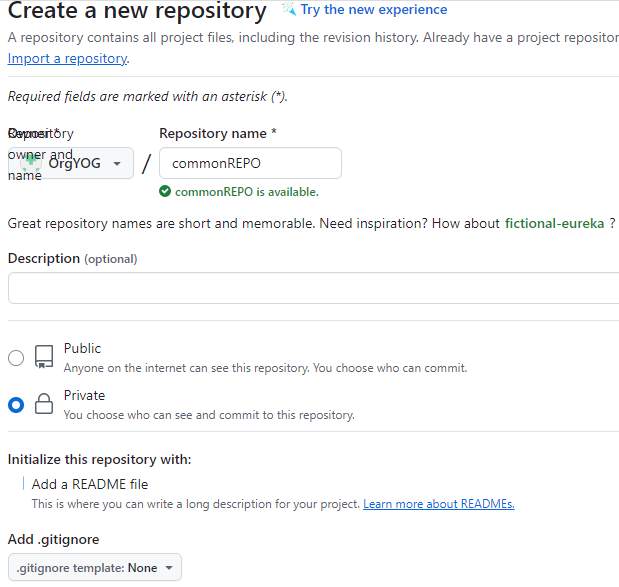
### Next under,

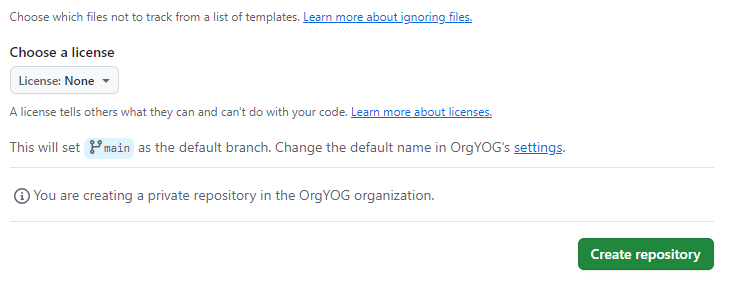
### Projects base permissions 🡪select Write 🡪 change base permission to “Write”

### 

**Step 4**: Create the remote repository for storing the project related files in Organization. This repository is accessible to every member of the team as per the permissions given.







Note: The repository can be made private so that it is accessible only to the group members rather than being in a public domain.

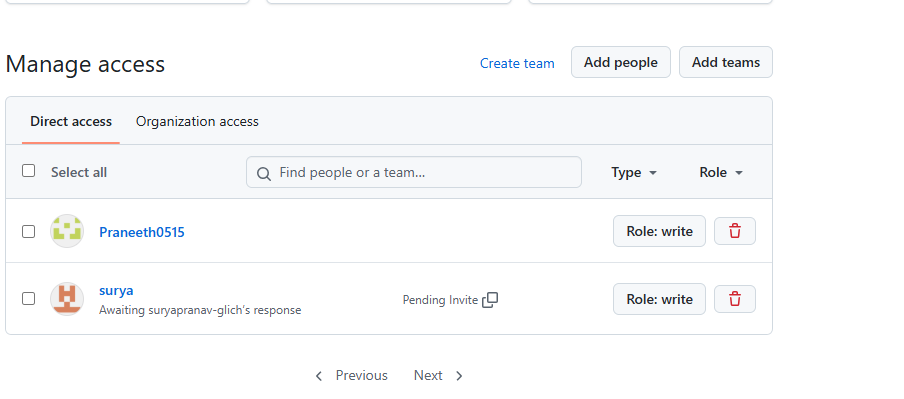
Now all the members of the team can contribute to the development of the project and the different files with all the versions and modification notices will be available in the respective repositories and is accessible to all the members

1. **To collaborate effectively on GitHub shared repository by sharing code and coordinating with others through a shared repository.**

🡪**Steps for Collaborator 1 are**: (collaborator-1 has repository that he wants to share with

collaborator-2)

* Go to Settings > collaborators
* Now in Manage Access click on Add people
* Now you will be able to see ONE collaborator added
* After this invitation goes to collaborator 2 and they need to accept it.



🡪**Steps for Collaborator 2 are:**

**✅ 1. Set Up Git and GitHub**

In git bash

git config --global user.name "Your Name"

git config --global user.email [you@example.com](mailto:you@example.com)

Note : run $ git remote -v

If origin git@github.com:OrgYOG/commonREPO.git (fetch)

origin git@github.com:OrgYOG/commonREPO.git (push)

then

$ git remote –v // Note now, not connected to any remote

**✅ 2. Goto Git hub and copy url of shared repository by collaborator 1**

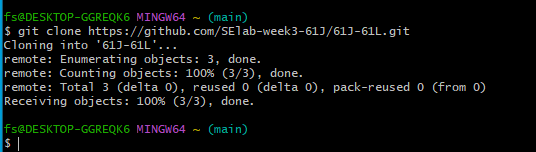
* Copy url (option 1)

**✅ 3. Clone the Repository**

Get a local copy of the repo:

In git bash

git clone <https://github.com/username/repository-name.git>



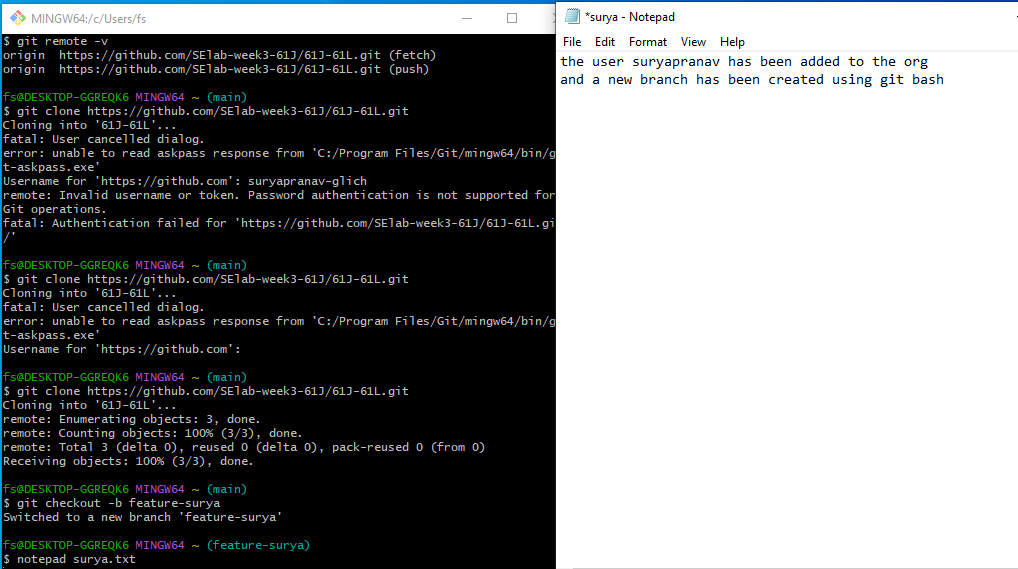
cd repository-name

**✅ 4. Create a Branch for Your Work**

Always work on a separate branch:

In git bash

git checkout -b feature/your-feature-name



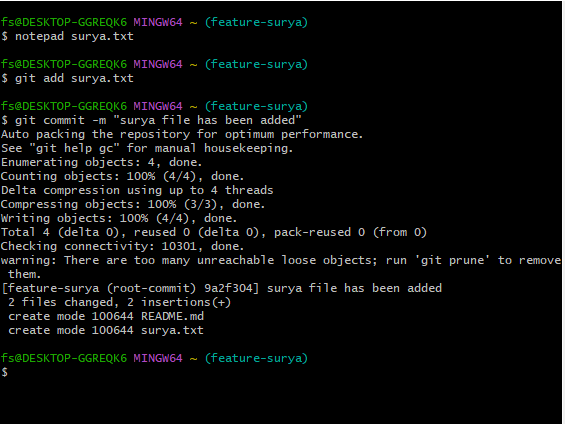
**✅ 5. Make Changes and Commit**

Edit files, then stage and commit your changes:

In git bash to existing file / new file

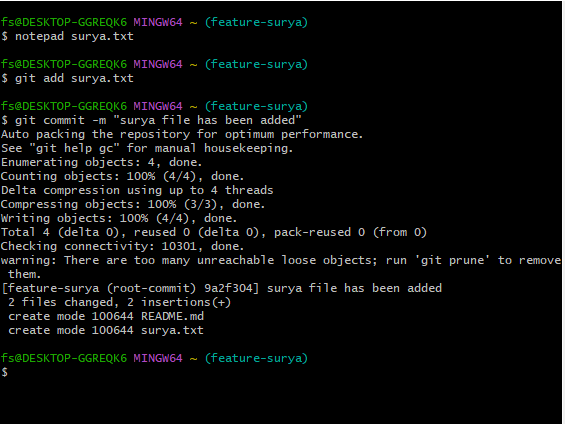
git add .

git commit -m "Add a clear and descriptive commit message"



**✅ 6. Push Your Branch to GitHub**

* In git bash
* git push origin feature/your-feature-name



**✅ 7. Keep Your Branch Updated (has to be done by owner collaborator 1)**

Before making new changes:

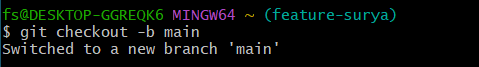
In git bash

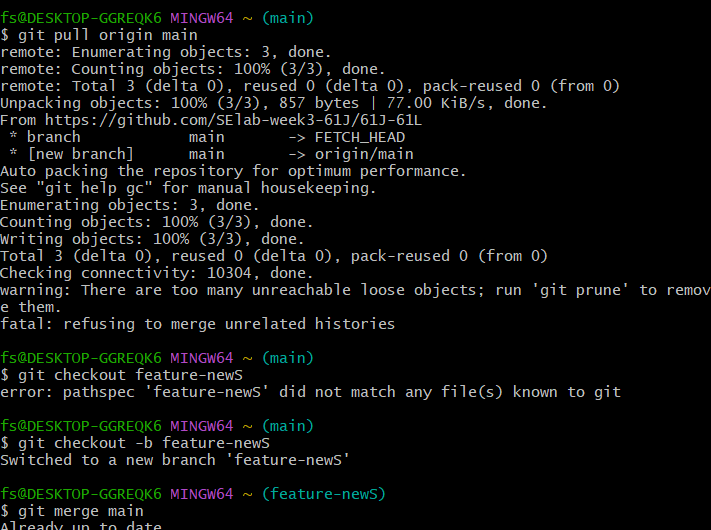
git checkout main

git pull origin main

git checkout feature/your-new-feature

git merge main



****

**Exercise on Fork**

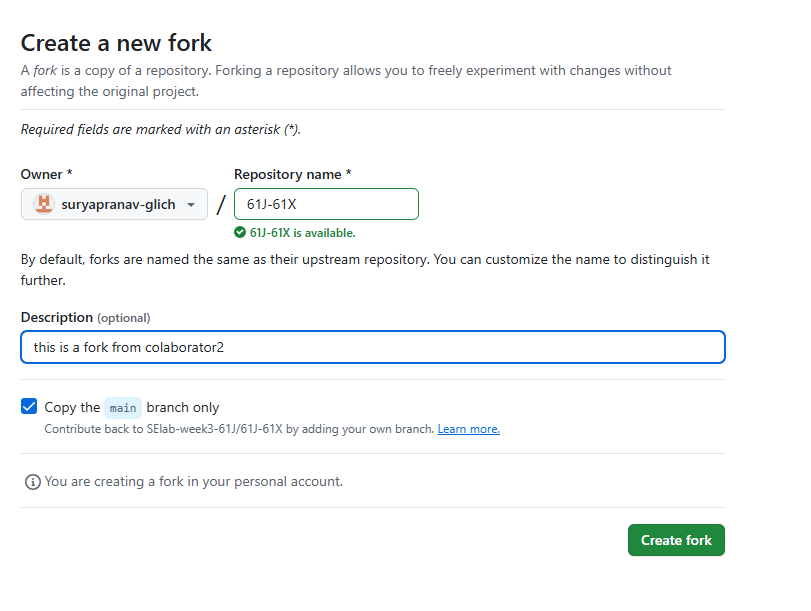
* If you're contributing to an existing project: (in public repository say of Person1)
* Note: Person 1means other public repository already exit that is pushed by person 1.

Person 2 is you who want to contribute.

Note: Following steps needed if you forked from any public repository to contribute to Person 1

1. Go to: `https://github.com/Person1/awesome-project` (of Person 1)

2. Click the “Fork” button at the top-right of the pag e.

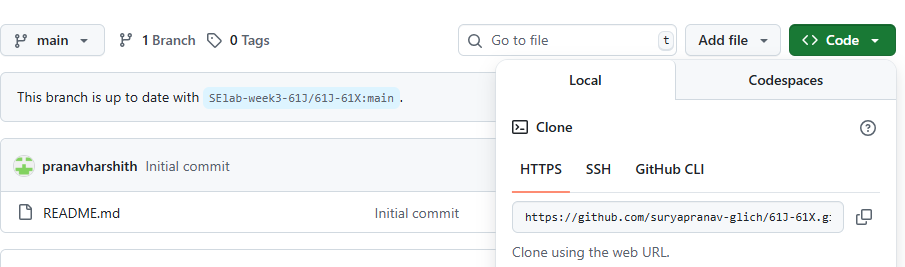


3. GitHub will create a copy of that repository in your account (say Person2):

<https://github.com/your-username/awesome-project>

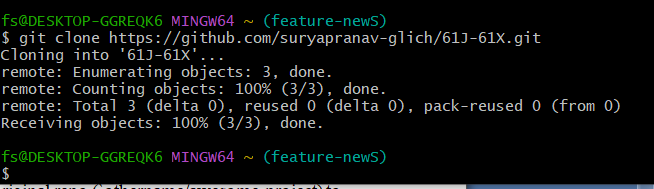
as

<https://github.com/Person2/awesome-project>

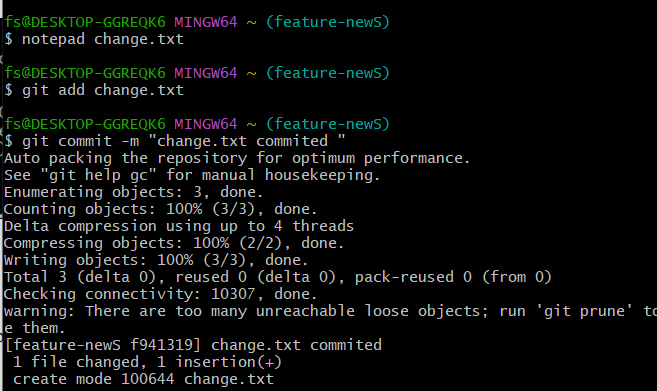


4. Clone your forked repo: in bash (by Person 2)

* git clone <https://github.com/your-username/awesome-project>



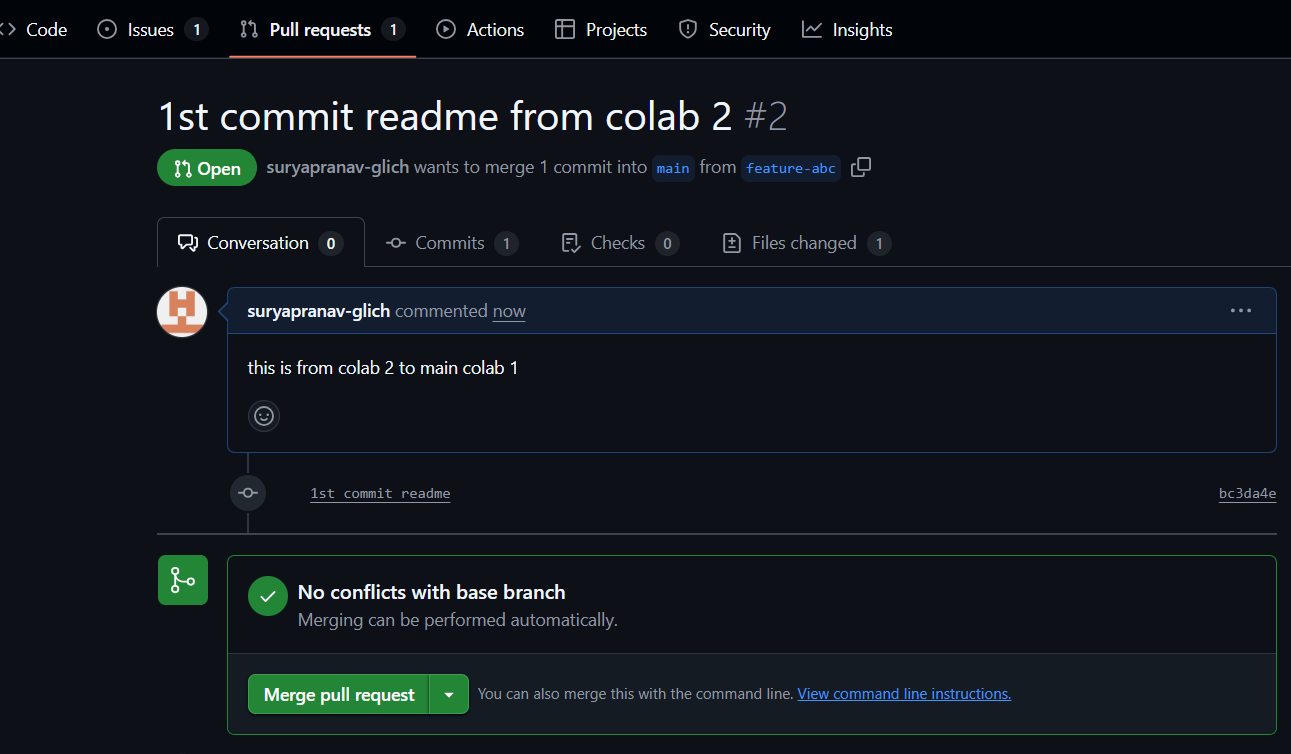
* Make changes.
* git add .
* git Commit



* git push origin main

1. Create or a “pull request” back to the original repo (`othername/awesome-project) to suggest your changes. (by Person 2)

* Click at top “Pull request” or click on at top or click on repo “awesome-project” then click on Contribute at top then click on “open pull request”
* Add title and Add a description if you want. Below it shows what changes you made to file
* Click on Create pull request



Now you see on page message “NO conflicts with base branch”

✅ 6. By Person 1

* Pull requests 1 appears at top in git hub account of Person 1
* Clicking on “Pull requests” shows:

Commit by Person 2. Click on this commit message.

It redirect to repository (say awesome-project)

* Click on “File changed 1” to see changes made by Person 2
* Click on “Review Changes”, add review and click on “submit review”
* Click on Merge pull request
* Confirm merge

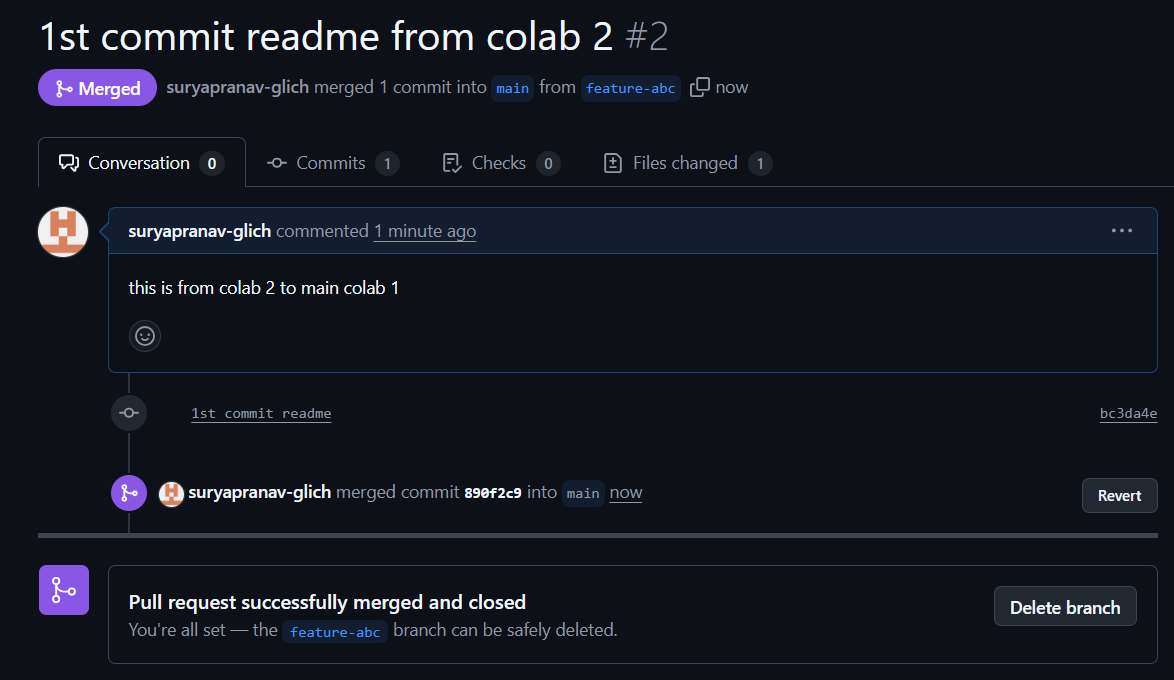
Note: It displays “Pull request Successfully merged and closed” to Person 1

* Now go to Code and see changes have been merged to the file.
* In commit it shows: Commit by Person 2

Merge commit by Person 1

✅ 7. Merge the Pull Request (has to be done by both collaborators)

Once approved:

* The repo owner can click Merge pull request  
  

✅ 8. Keep Your Branch Updated (has to be done by Person 1 and Person 2)

Before making new changes:

In git bash

git checkout main

git pull origin main

git checkout feature/your-new-feature

git merge main

cd repository-name

1. **To resolve conflicts when collaborating on same part of code.**

Resolving conflicts when collaborating with GitHub (or Git in general) is a normal part of team development. Conflicts happen when two people make changes to the same part of the code, and Git doesn't know which version to keep.

**✅ Steps to Resolve a Conflict**

**1. See Which Files Are in Conflict** (Say f1.txt)

In git bash

git status

Conflicted files will be marked like:

In git bash

both modified: (Say f1.txt)

1. **Open the File and Look for Conflict Markers by clicking on**

* Click on **Resolve Conflict**

Git will insert conflict markers in the file like this:

def greet():

<<<<<<< HEAD

print("Hello from First collaborator ")

=======

print("Hello from second collaborator ")

>>>>>>> feature/second collaborator branch

This shows the two conflicting changes:

* HEAD is your version
* The section below ======= is the other branch’s version or second collaborator branch version

**3. Edit the File to Fix the Conflict**

* Choose one version or combine them:

def greet():

print("Hello from First collaborator ")

print("Hello from second collaborator ")

* Then **delete** all the **conflict markers** (<<<<<<<, =======, >>>>>>>).

**4. Mark the Conflict as Resolved**

Once you've edited and saved the file:

* git add f1.txt

**5. Complete the Merge or Pull**

* git commit # Only needed if you're merging

**Or** if you're rebasing:

* git rebase --continue

**⚠️ If You Want to Abort the Merge**

If it gets messy and you want to cancel:

git merge --abort

Here below screen shot shows conflict raised when collaborator 1 wanted to merge changes to same line that already collaborator 2 added the line of code at that line in same file but in their own branch.

**Tips to Avoid Conflicts**

* Pull the latest changes before starting work:

In git bash

git pull origin main

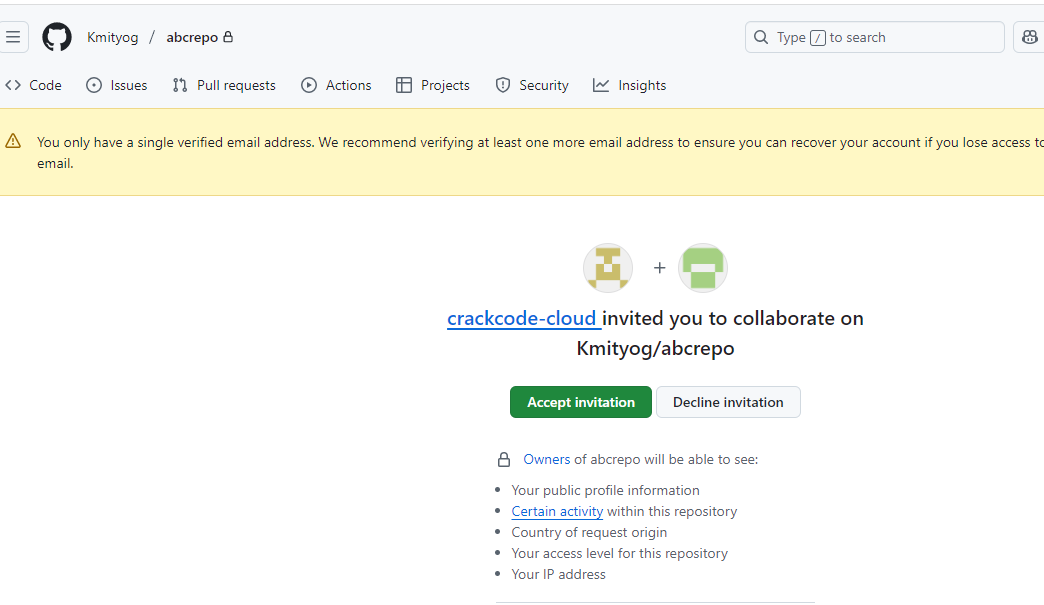
* Communicate with your team
* Work in small branches with short-lived changes

**Example :**

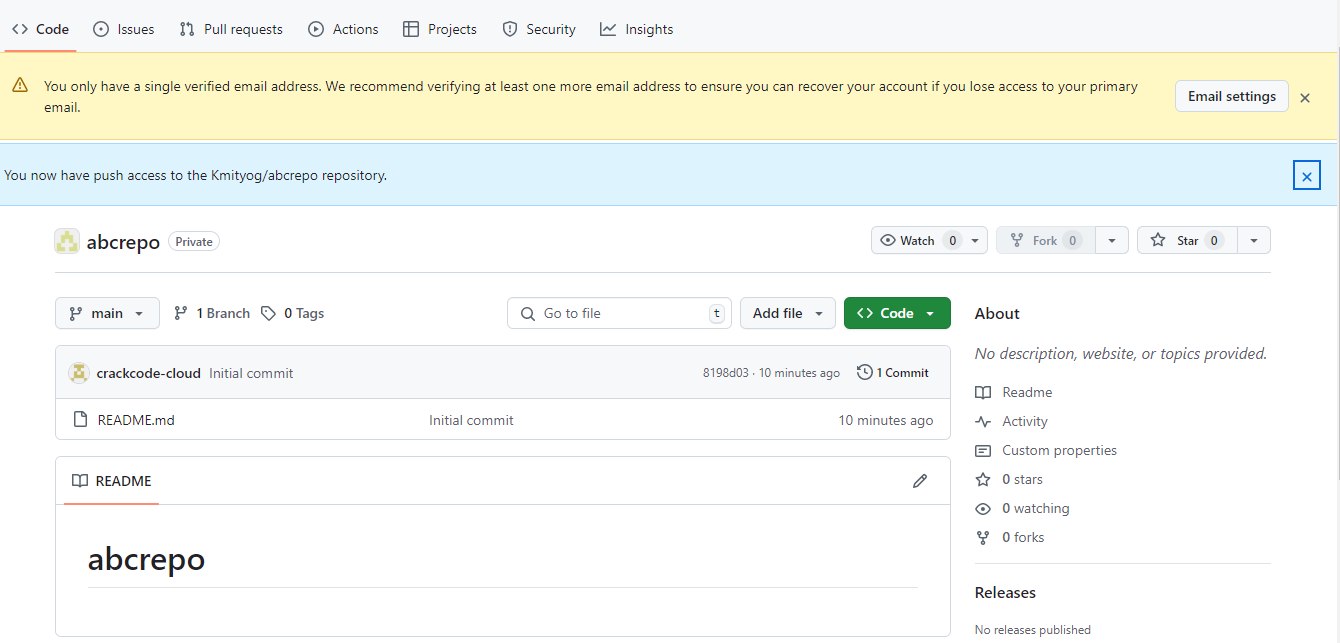
**Merge Conflict – Changes made to a file by two different users leads to merge conflict as below.**

**I am collaborator 2, now**

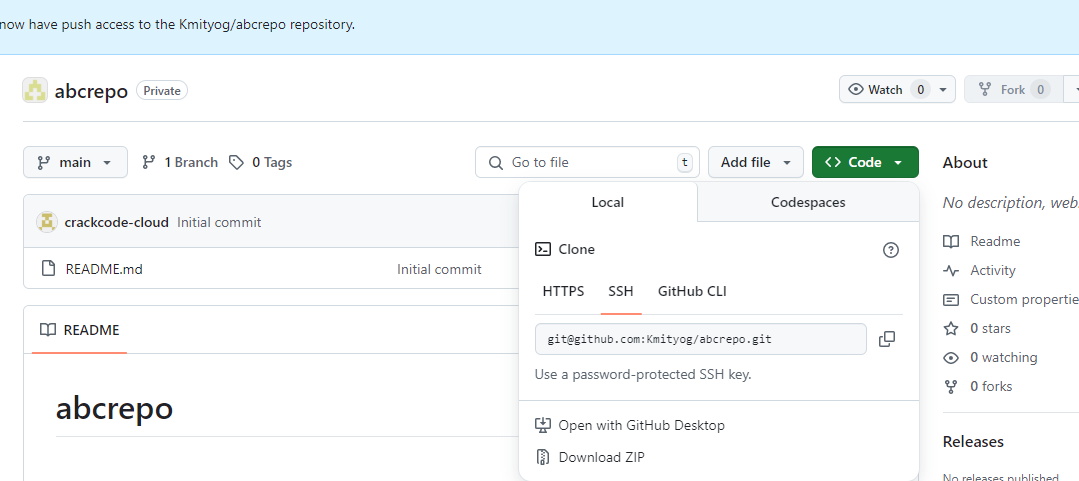
* **Accept invitation from collaborator 1**

****

* **Next goes to abcrepo**

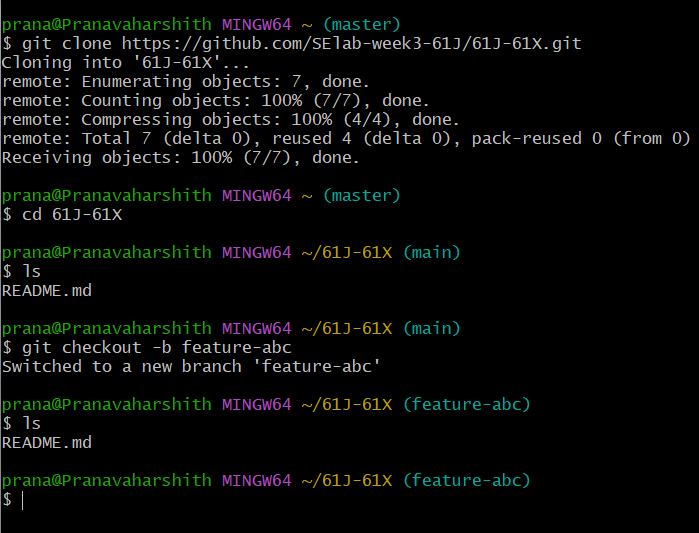
****

* **Now open Git bash and clone below repo by cpoing its url**

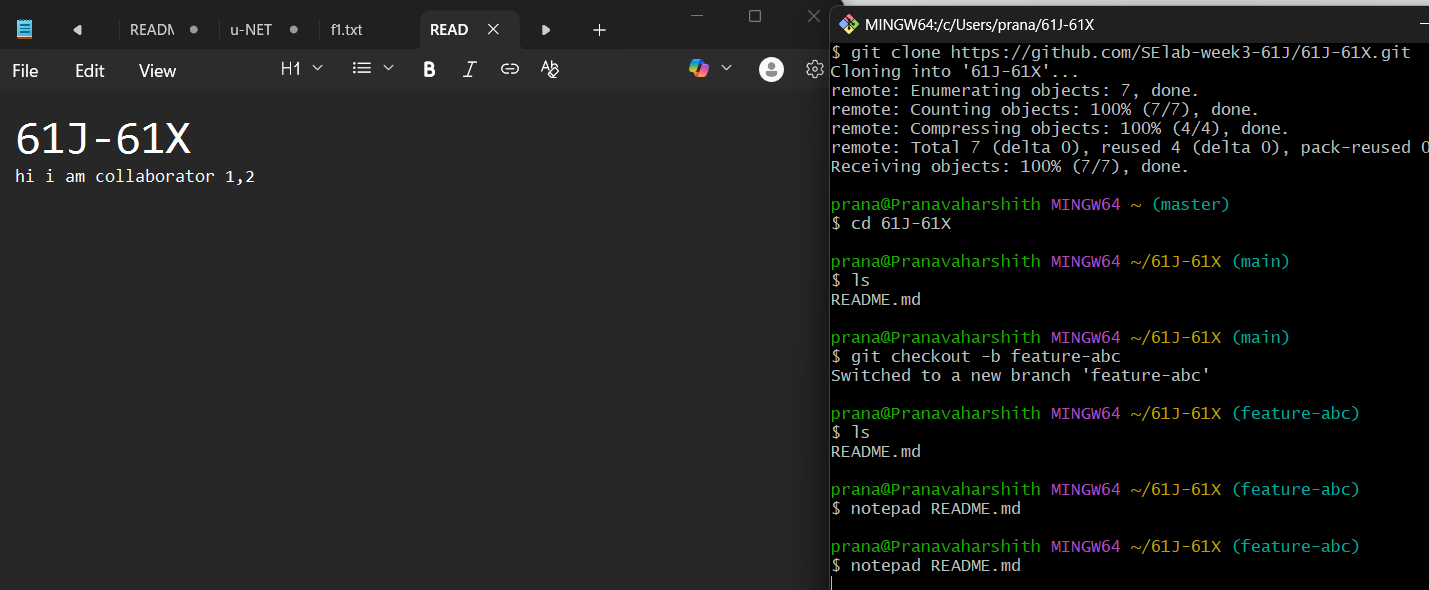
****

**Step 1: first connect to this abcrepo to local repository by clone**

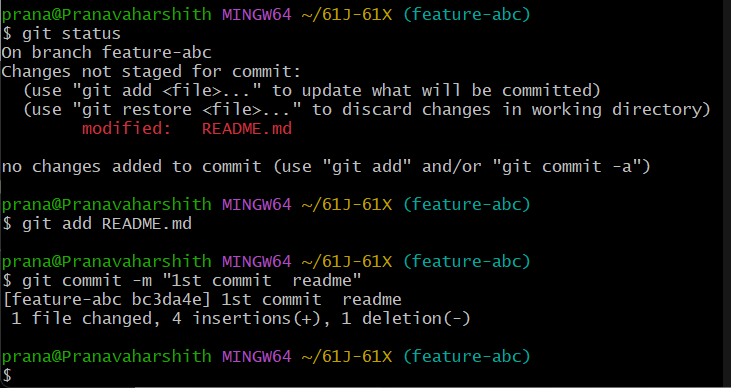
* **Now change directory to abcrepo**
* **Switch to branch feature-abc**



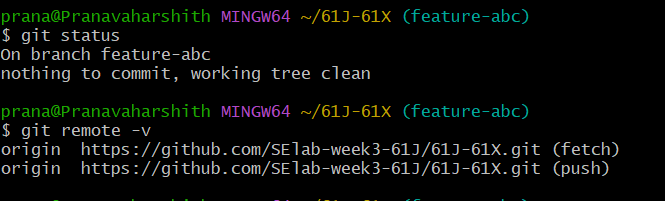
* **Now add line in README.md file**

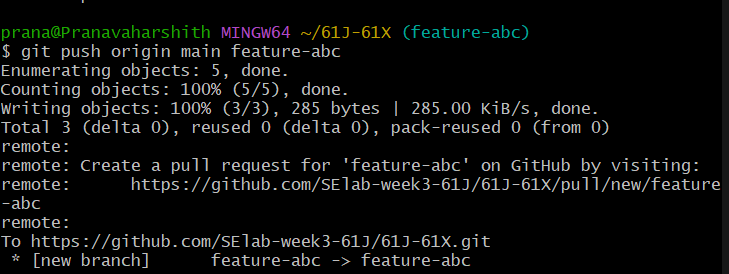


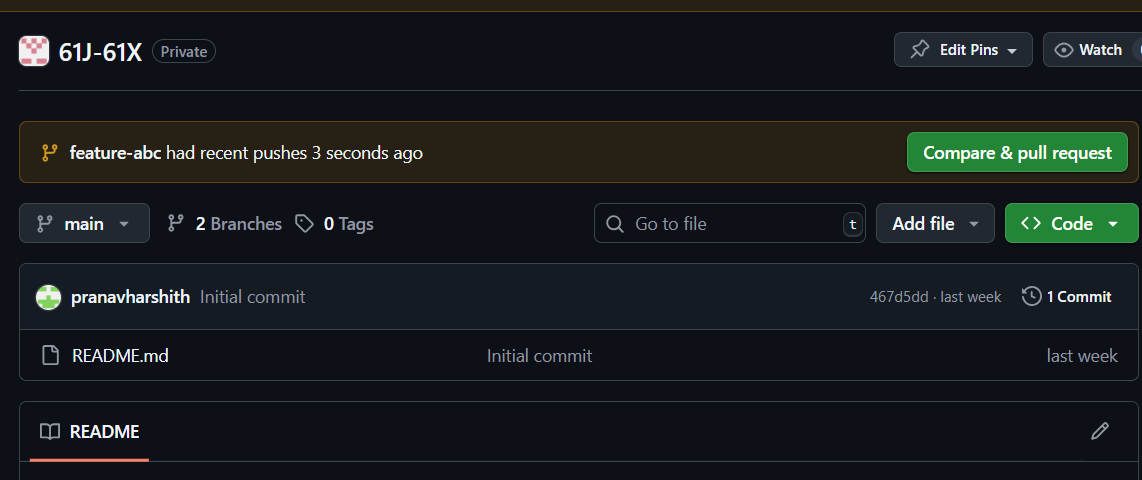
* **Now git add, commit**



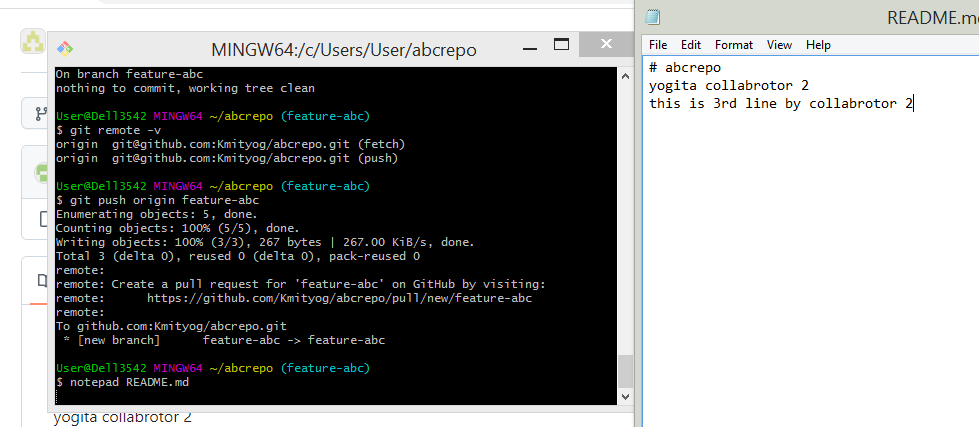
* **Now see status and push to remote branch feature-abc**

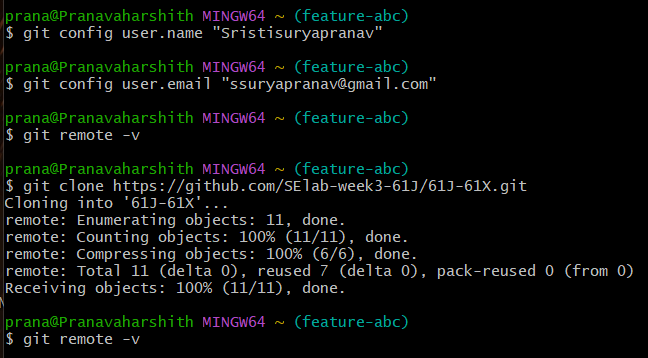


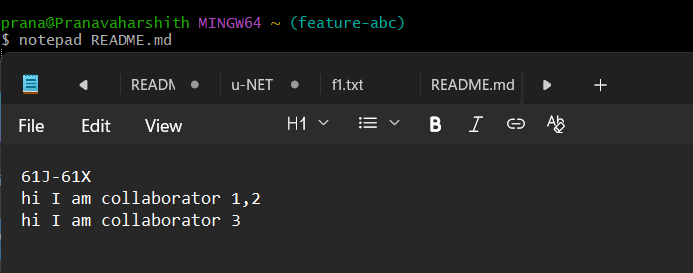


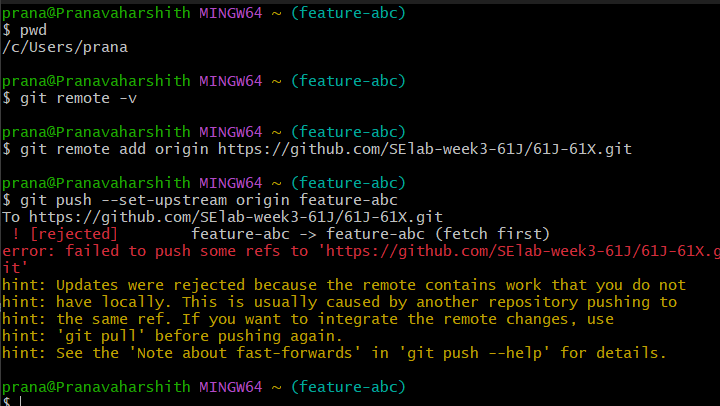
* **Now go to Github and refresh or click on abcrepo, you can see** 

**Step 2:** Now collaborator 2 adding line in README.md, save, git add, commit, push. **No conflicts.**

****





**Step 3:** Also collaborator 1 adding line in README.md at same line collaborator 2 added, then git add, commit, push gives **merge conflict.**

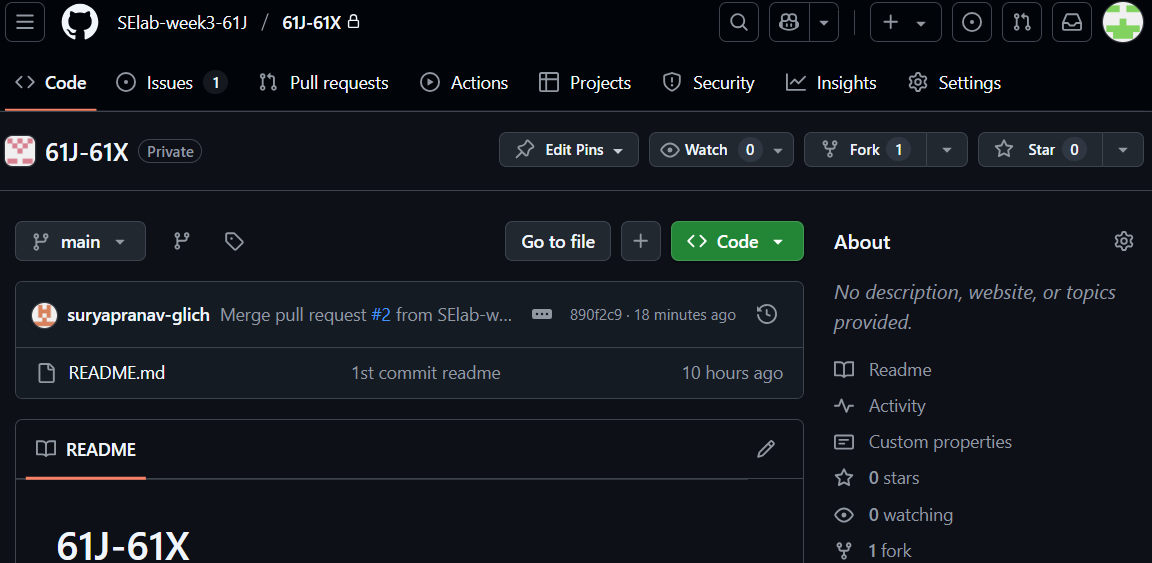
Now open file, Edit the File to Fix the Conflict by keeping either or both collaborator changes and remove conflict markers. Add file, commit, and push.

* 1. **What is patch**

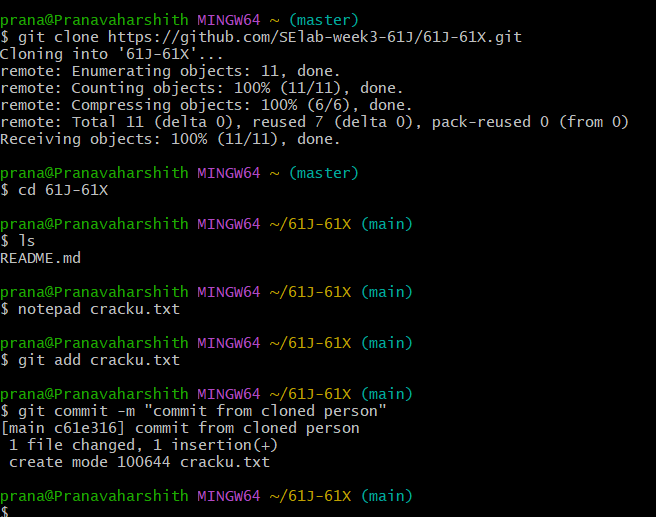
A patch in Git is a text file that represents changes between two sets of files, or commits. It's essentially the output of the git diff command, packaged in a format that can be applied to another set of files.

Steps:

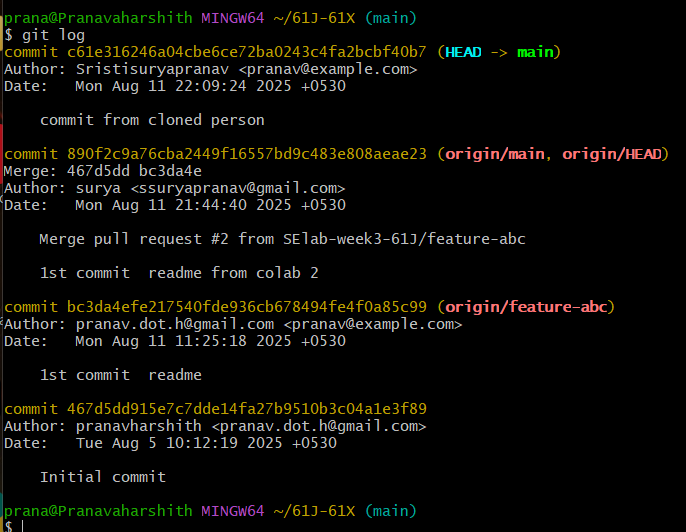
1. Goto github and take public project url



1. Clone into git bash
2. Now open file and edit it, add, commit

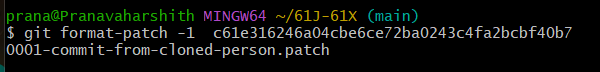


1. Run git log



1. Create patch with commit hash code as:

git format-patch -1 commit-hash-code



This creates 0001-commit-from-cloned-person.patch file.

Note: git format-patch -1 <commit-hash>

* -1 means "create a patch from 1 commit"
* This creates a .patch file for a single commit.
* Example:

git format-patch -1 abc1234

* This creates a file like 0001-Your-commit-message.patch

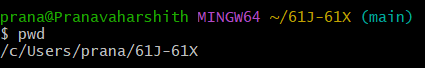
To create patches for the **last 3 commits**: git format-patch -3

or

#### ****Create patches from multiple commits****

git format-patch <base\_commit>..HEAD

1. Get the path of above patch file and email or what’s up to the remote owner of file.



1. Once the remote owner gets the patch file 0001-commit-from-cloned-person.patch next the owner need to apply patch file in his git bash using git command below:

Syntax:

git apply my-changes.patch

Or, if it's a patch made by git format-patch, use:

git am 0001-Your-commit-message.patch

* git apply 0001-Your-commit-message.patch