LAB LESSON PLAN FOR WEEK 2

Lab Topics:

1. How to work using local/ basic Git commands?
2. How to create project at local and upload in private and public repository
3. What is Collaborative coding?
4. Take url download from Github (PULL). If conflict arise when two people doing changes to same code how admin will come to know. And how is it resolved.
5. What is patch
6. How do you create patch

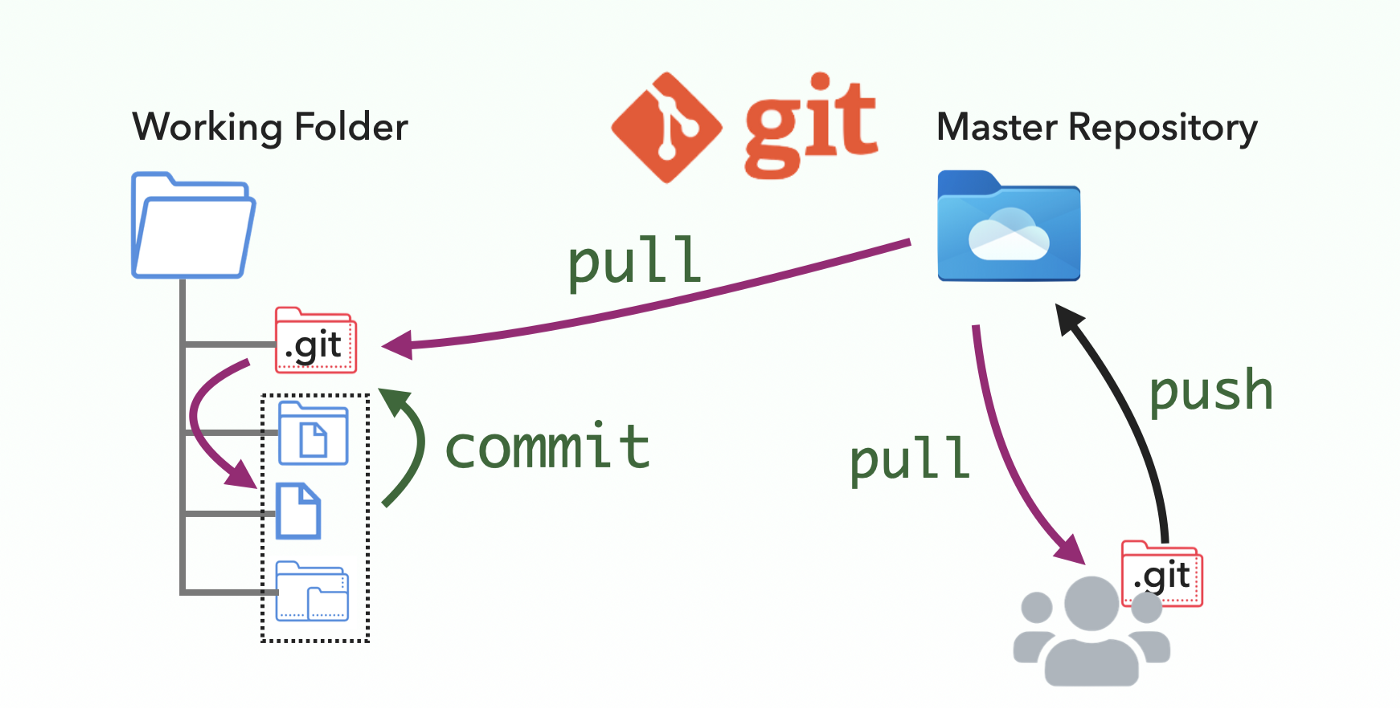
**Introduction:**

Git is a popular version control system. It was created by Linus Torvalds in 2005. It is designed to make it easier to have multiple versions of a code base, sometimes across multiple developers or teams

**Functions of Git**

* Manage projects with Repositories
* Clone a project to work on a local copy
* Control and track changes with Staging and Committing
* Branch and Merge to allow for work on different parts and versions of a project
* Pull the latest version of the project to a local copy
* Push local updates to the main project

**GIT VERSION CONTROL SYSTEM**



**WHAT IS GITHUB**

Github.com is a website that hosts git repositories on a remote server

Hosting repositories on Github facilitates the sharing of codebases among teams by providing a GUI to easily fork or clone repos to a local machine

1. How to work using local/ basic Git commands
2. **Git Commands:**

* git version : The command git version is used to check the version of git.

**git --version**

* git config: Configures Git settings. Commonly used to set up user information

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

**git config --global user.email "youremail@example.com"**

* **git config --list**: Displays all the Git configurations for the current user.

**2. Repository Management**

* **git init**: Initializes a new Git repository in the current directory

git init

* **git clone**: Creates a copy of an existing Git repository from a remote source (e.g., GitHub) to your local machine

git clone https://github.com/username/repository.git

**3. Staging and Committing**

* **git status**: Shows the status of changes in your working directory and staging area. It tells you which files are untracked, modified, or ready to be committed.

**git status**

* **git add**: Adds changes in the working directory to the staging area.

**git add filename.txt # Adds a specific file**

**git add . # Adds all changes in the directory**

* **git commit**: Commits the staged changes to the repository with a descriptive message. The -m option allows you to include a commit message.

**git commit -m "Commit message describing changes"**

**4. Branching and Merging**

* **git branch**: Lists all branches or creates a new branch

**git branch # Lists all branches**

**git branch branch-name # Creates a new branch**

**git branch -d <branch\_name> # Deletes a branch**

* **git checkout**: Switches to a different branch

**git checkout branch-name # Switches to an existing branch**

**git checkout -b new-branch # Creates and switches to a new branch**

* **git merge**: Merges the specified branch into the current branch. This command integrates the changes from the feature branch into the main branch.

**git checkout main # Switch to the main branch**

**git merge branch-name # Merge branch-name into main**

* 1. **Undoing Changes**
* git reset :Removes the specified file from the staging area but leaves the working directory unchanged. git reset --hard can also reset the working directory and staging area to the last commit.

**git reset <file>:**

* git revert**:** Creates a new commit that undoes the changes from a specified commit, leaving the history intact**.**

**git revert <commit>:**

* 1. **Viewing History**
* git log**:** Shows a history of commits in the repository, including commit hashes, messages, and timestamps. Use git log --oneline for a more concise view.

**git log:**

* git diff**:** Displays differences between various commits, the working directory, and the staging area. git diff without arguments shows changes not yet staged.

**git diff:**

* git show**:** Shows the details of a specific commit, including the changes made and the commit message.

**git show <commit>**

1. **How to create project at local and upload in private and public repository**

**AIM:** To experience the real-time scenario of collaborative coding. When multiple developers work on the same project all the team members will be provided with reading/write access. This facilitates tracking of the modification made to the code by everyone and maintains different versions of the developed modules.

**Procedure:**

* Create a directory in the local system for working on the project
* Change to that directory
* Create an empty git repository by running - the **git init**  command.



* Create any static page for simple login using HTML.

**HTML code for creating the above form.**

<!DOCTYPE html>

<html>

<head>

</head>

<body>

<center> <h1> Student Login Form </h1> </center>

</br>

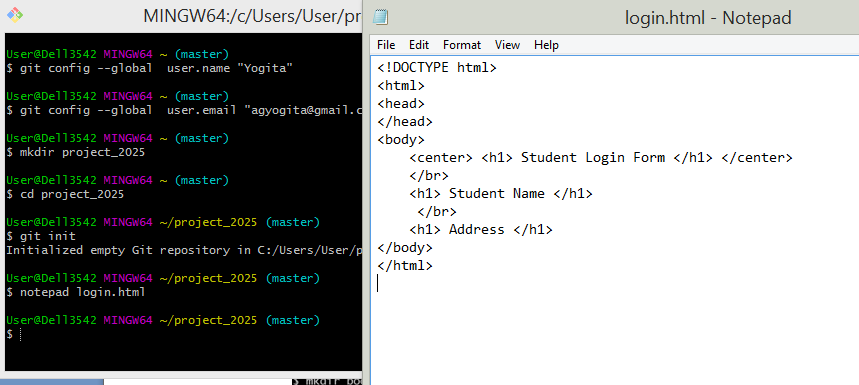
<h1> Student Name </h1>

</br>

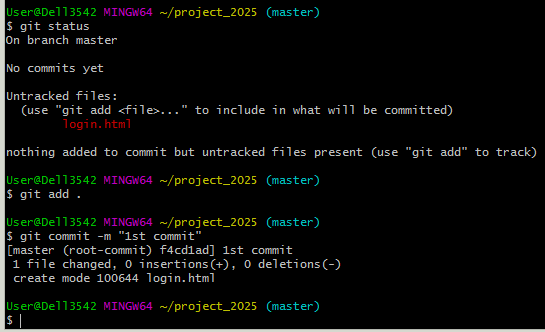
<h1> Address </h1>

</body>

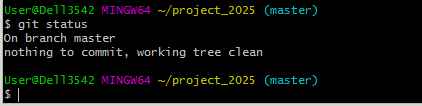
</html>



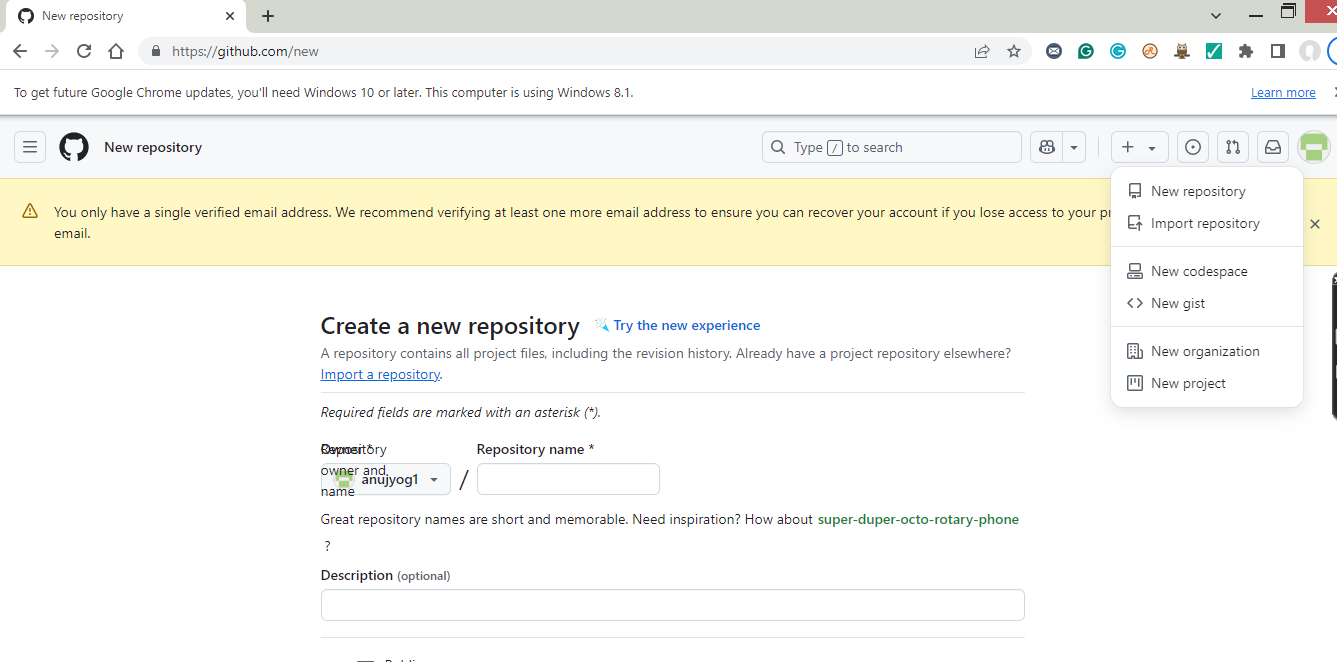
* Add file to staging area using command “git add .” in local repository.
* Commit the new file. You can also view the status and observe the changes before and after commit.



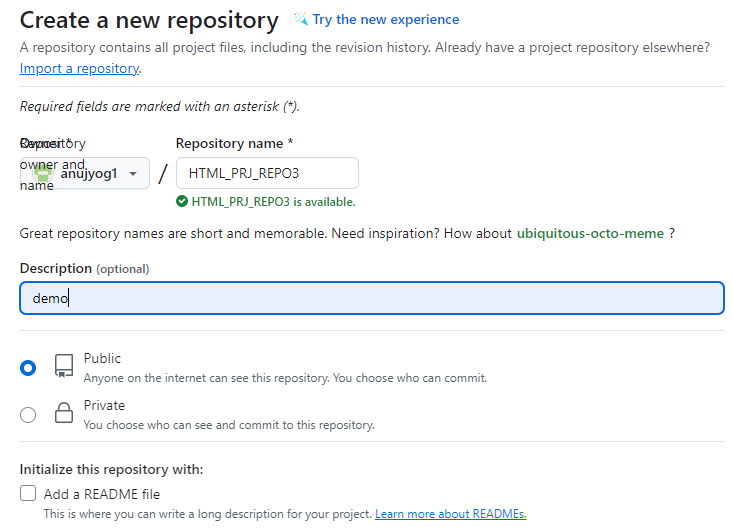
Note git status shows working tree clean as below

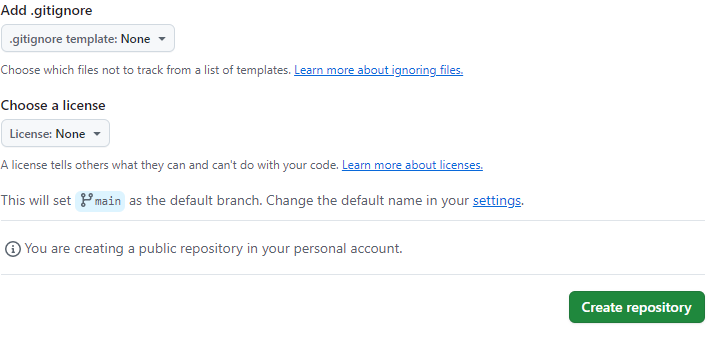


* Create a new repository on GitHub:
* Click on + then New repository

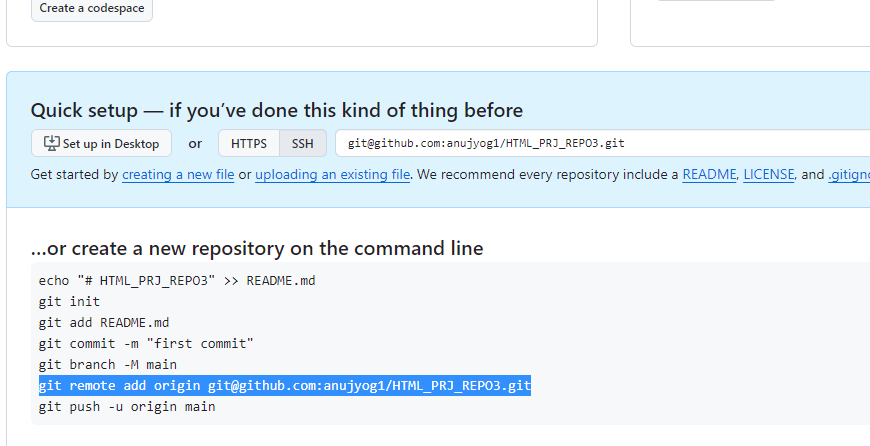


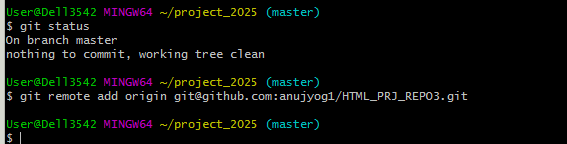
* Click on Public and Create Repository



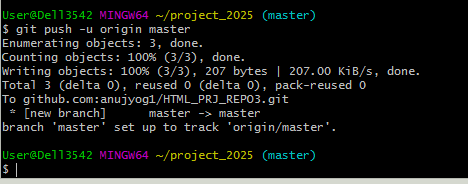


* Link the remote repository and the local repository using the git remote command.
* Copy this command from git hub and paste in git bash
* **git remote add origin git@github.com:anujyog1/HTML\_PRJ\_REPO3.git**

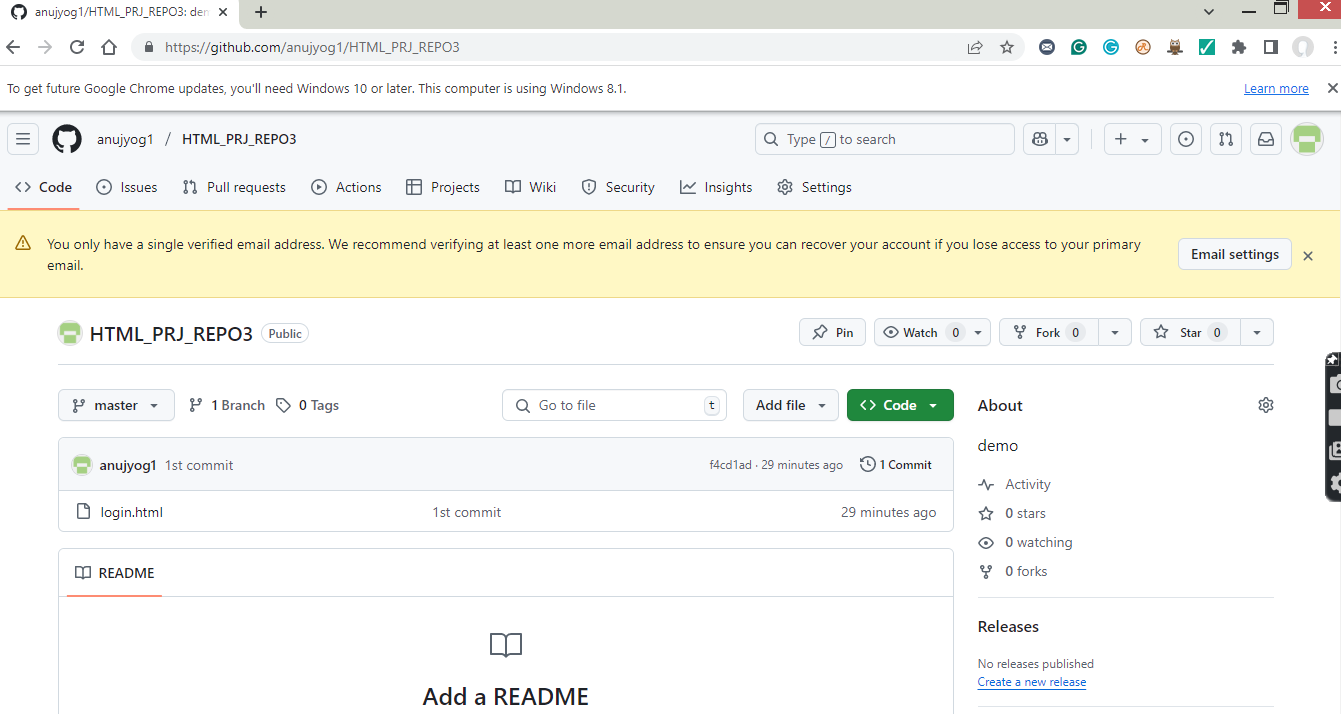




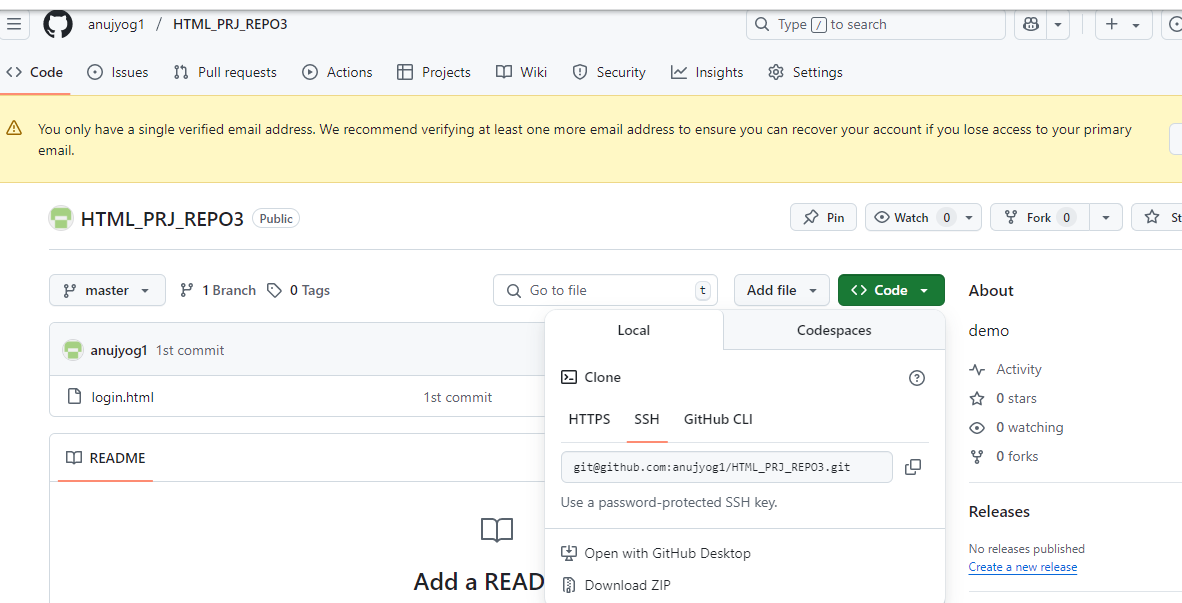
* Push the local repository contents on the remote repository using command
  + **git push -u origin master**

****

* Refresh GitHub to check the uploaded content



* The co-developers can clone this to their local repository by coping HTTP or SSH link, and view, edit, and contribute their inputs.



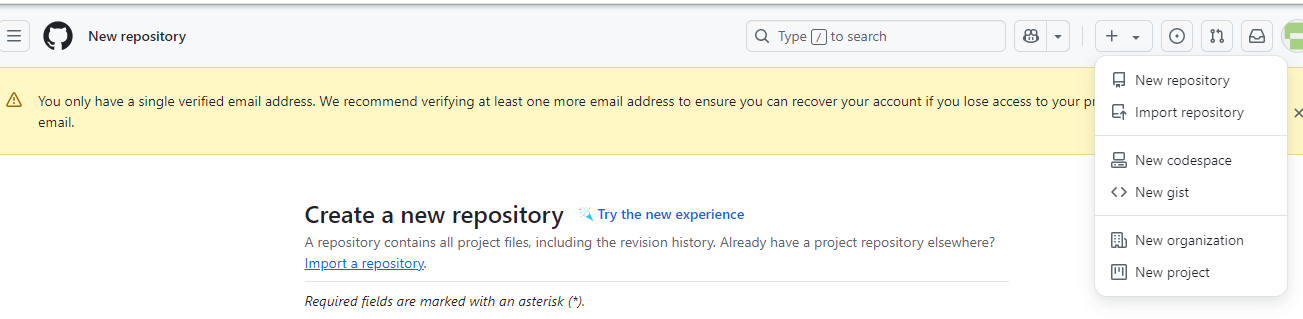
1. What is Collaborative coding.

What is code collaboration? 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.

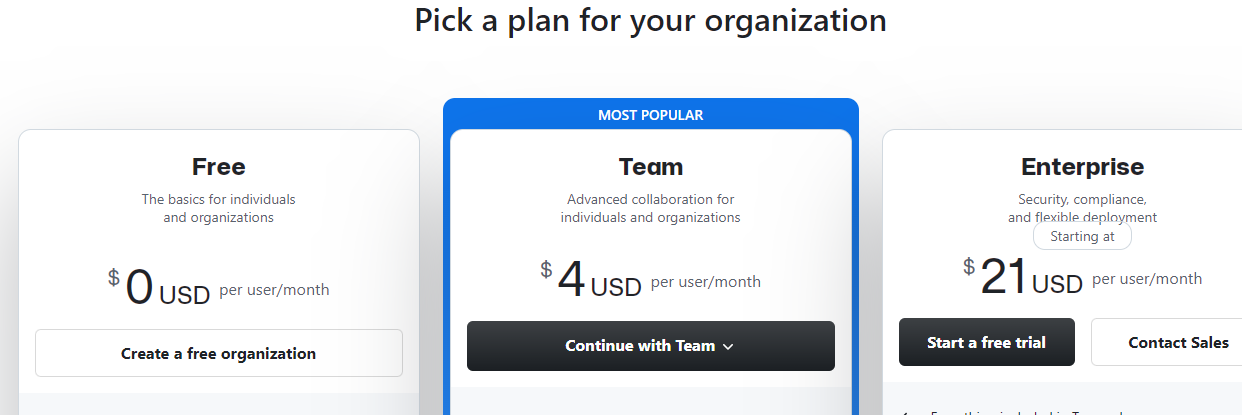
**Facilitating Collaborative Work**

**Step 1:** Create a new 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.

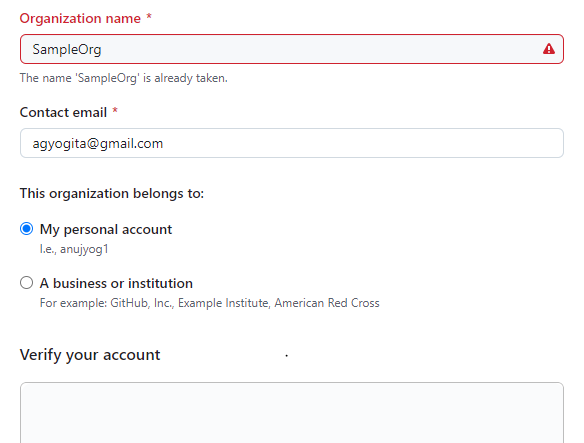
Click on New organization in Github

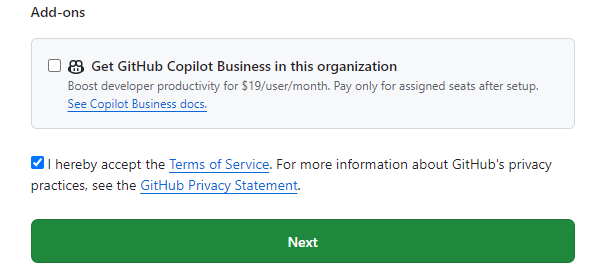


Click on free organization



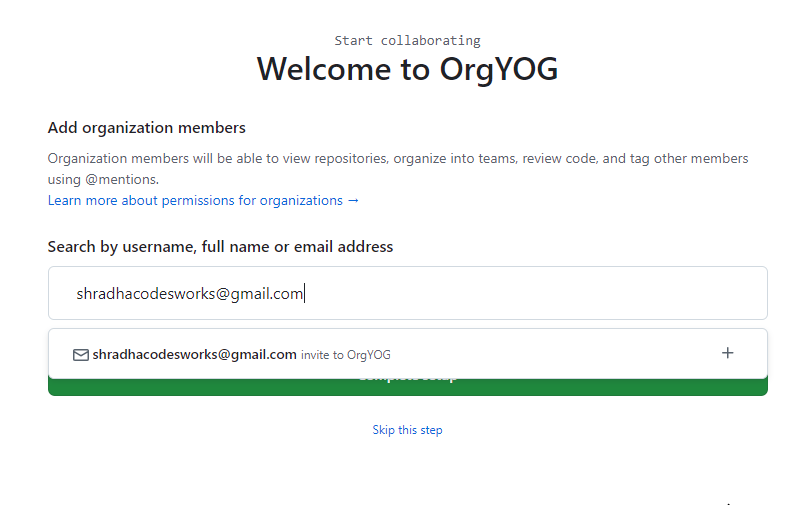
**Step 2:** Set up your organization by entering the details like name, associated email, account verification, inviting or adding members to the organization, etc. I gave here organization name as OrgYOG not SampleOrg



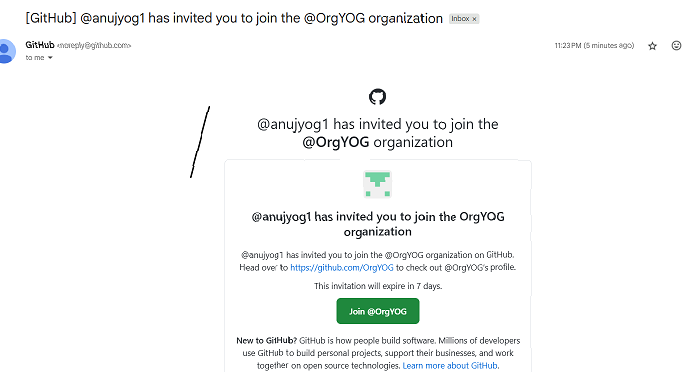


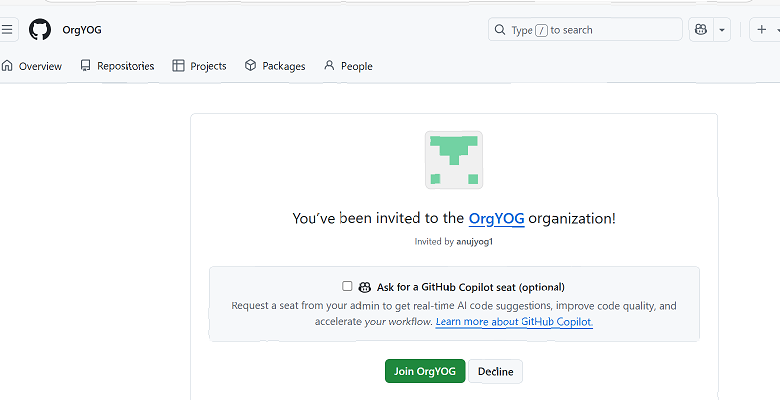
Click on Next

**Step 3**: Complete the setup by adding members to the group



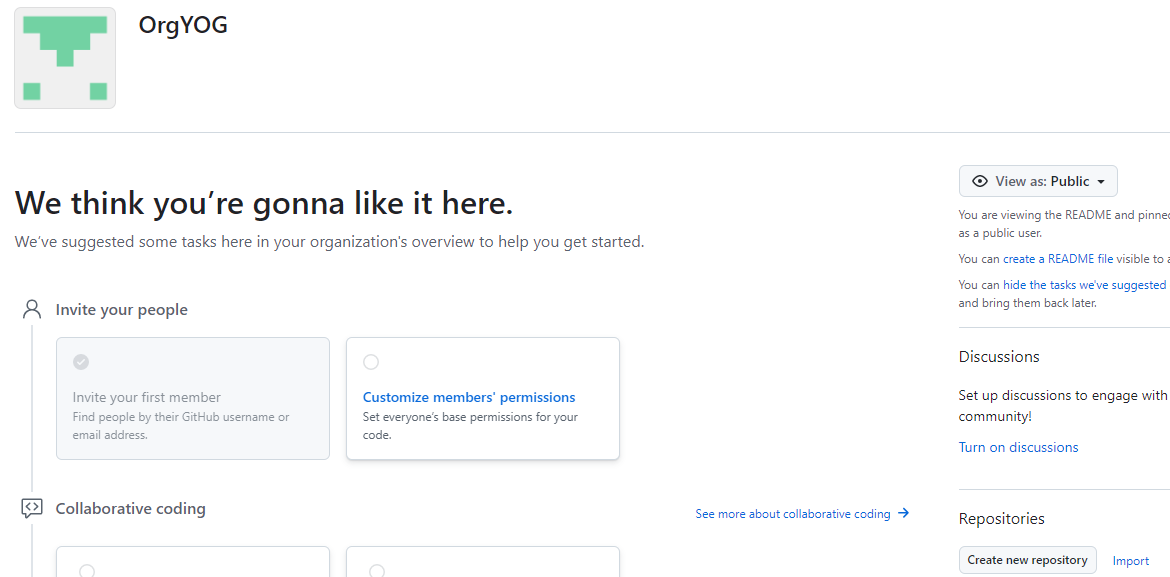
Now the invitation go to other collaborator as below and they must accept.

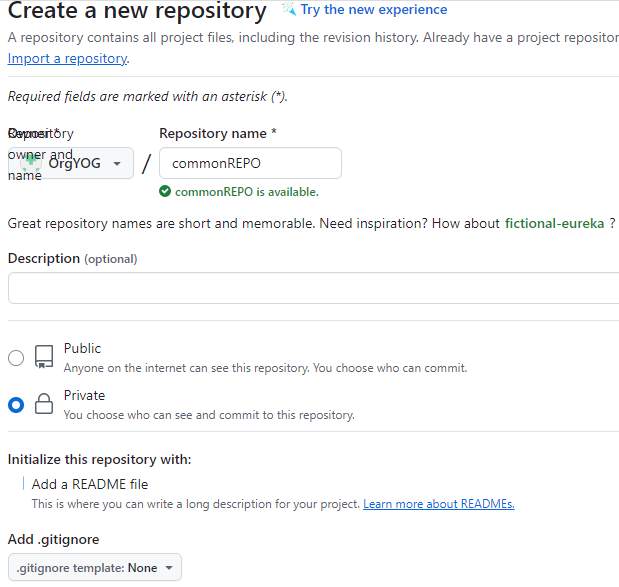


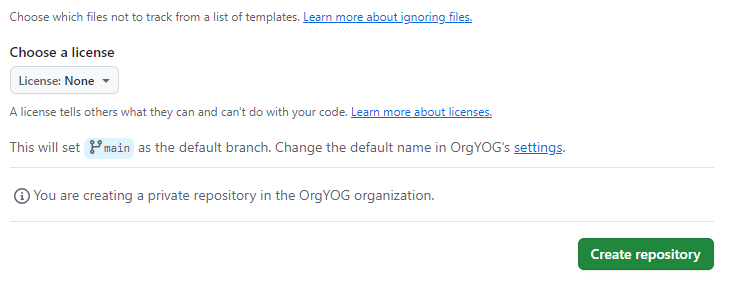


**Next click on complete setup**

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







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. Take url download from Github (PULL). If conflict arise when two people doing changes to same code how admin will come to know. And how is it resolved.

Exercise: Collaborating using GitHub involves sharing code and coordinating with others through a shared repository Collaborating using GitHub involves sharing code and coordinating with others through a shared repository. Here’s a step-by-step guide on how to collaborate effectively on GitHub:

**✅ 1. Set Up Git and GitHub**

In git bash

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

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

**✅ 2. Create or Fork a Repository**

* If you're starting a new project:
  + Go to GitHub > New Repository
* If you're contributing to an existing project:
  + Visit the repository page and click Fork (if you don’t have write access)

**✅ 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. Open a Pull Request (PR)**

* Go to the GitHub repo in your browser
* Click Compare & pull request
* Add a meaningful title and description
* Submit the pull request for review

**✅ 8. Review and Discuss Changes**

* Collaborators can comment, request changes, or approve the PR(Pull Request)
* Make edits if needed and push again; they will update the PR automatically

**✅ 9. Merge the Pull Request**

Once approved:

* The repo owner can click Merge pull request
* Or you can squash and merge, depending on the team’s workflow

**✅ 10. Keep Your Branch Updated**

Before making new changes:

In git bash

git checkout main

git pull origin main

git checkout feature/your-new-feature

git merge main

Exercise: 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.

Here’s a **step-by-step guide** to resolving Git conflicts:

**🔥 When Do Conflicts Happen?**

Typically during:

* git merge
* git pull
* git rebase

**✅ Steps to Resolve a Conflict**

**1. See Which Files Are in Conflict**

In git bash

git status

Conflicted files will be marked like:

In git bash

both modified: main.py

**2. Open the File and Look for Conflict Markers**

Git will insert conflict markers in the file like this:

def greet():

<<<<<<< HEAD

print("Hello from Alice!")

=======

print("Hello from Bob!")

>>>>>>> feature/bob-greeting

This shows the two conflicting changes:

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

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

Choose one version or combine them:

def greet():

print("Hello from Alice and Bob!")

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

**4. Mark the Conflict as Resolved**

Once you've edited and saved the file:

git add main.py

**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

**Tips to Avoid Conflicts**

* Pull the latest changes before starting work:

In git bash

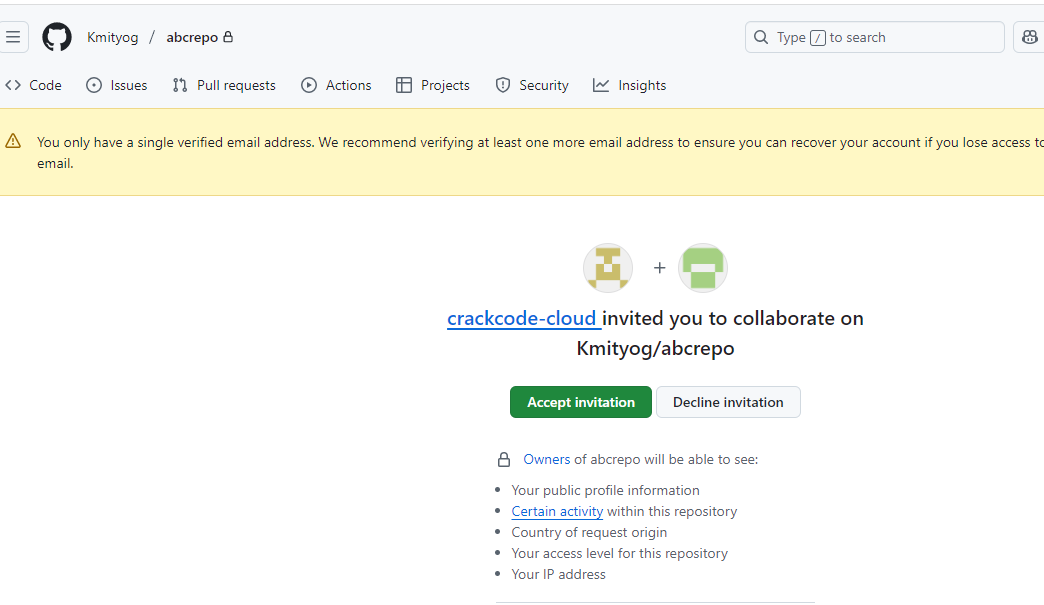
git pull origin main

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

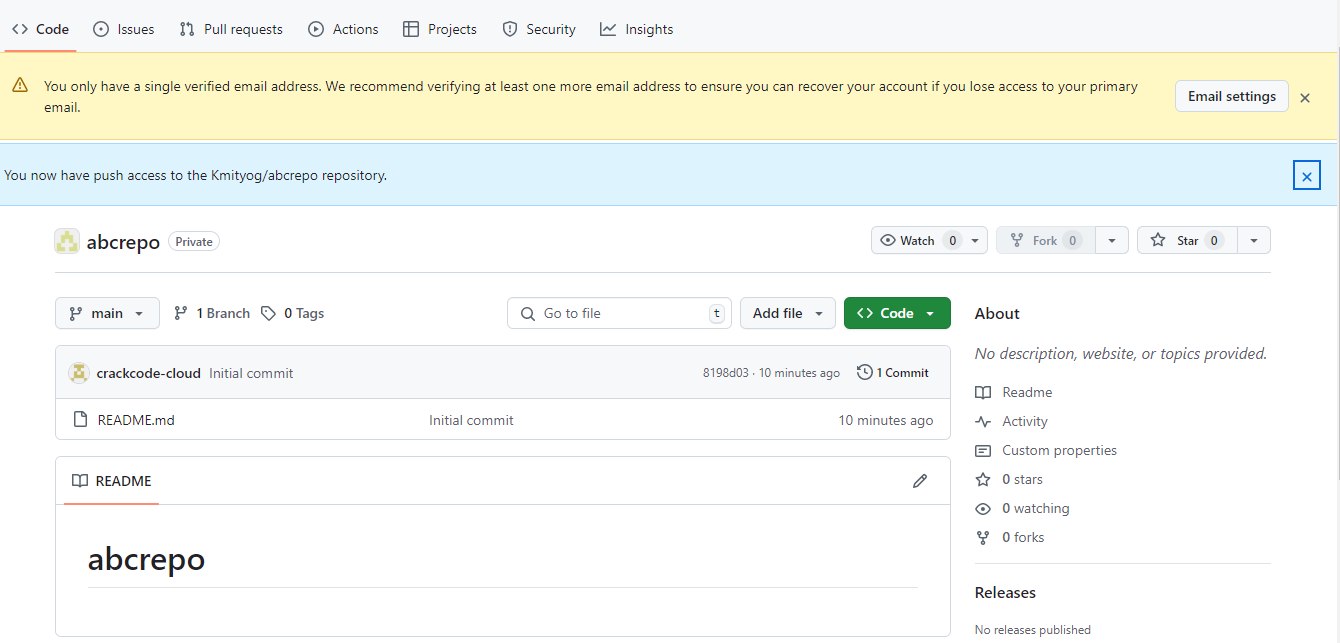
**Merge Conflict – Changes made to a file by two different users leads to merge conflict as below.**

**I am collaborator 2, now**

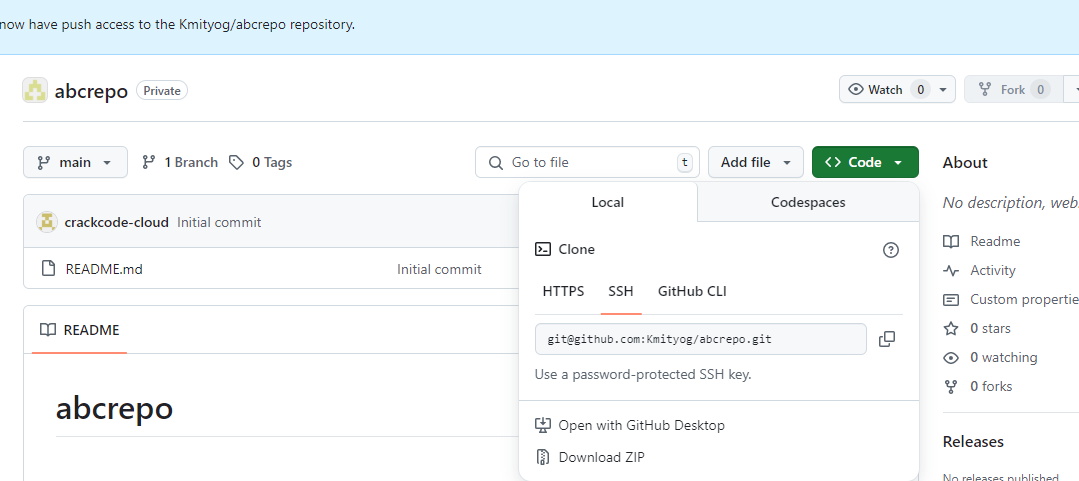
**Accept invitation from collaborator 1**

****

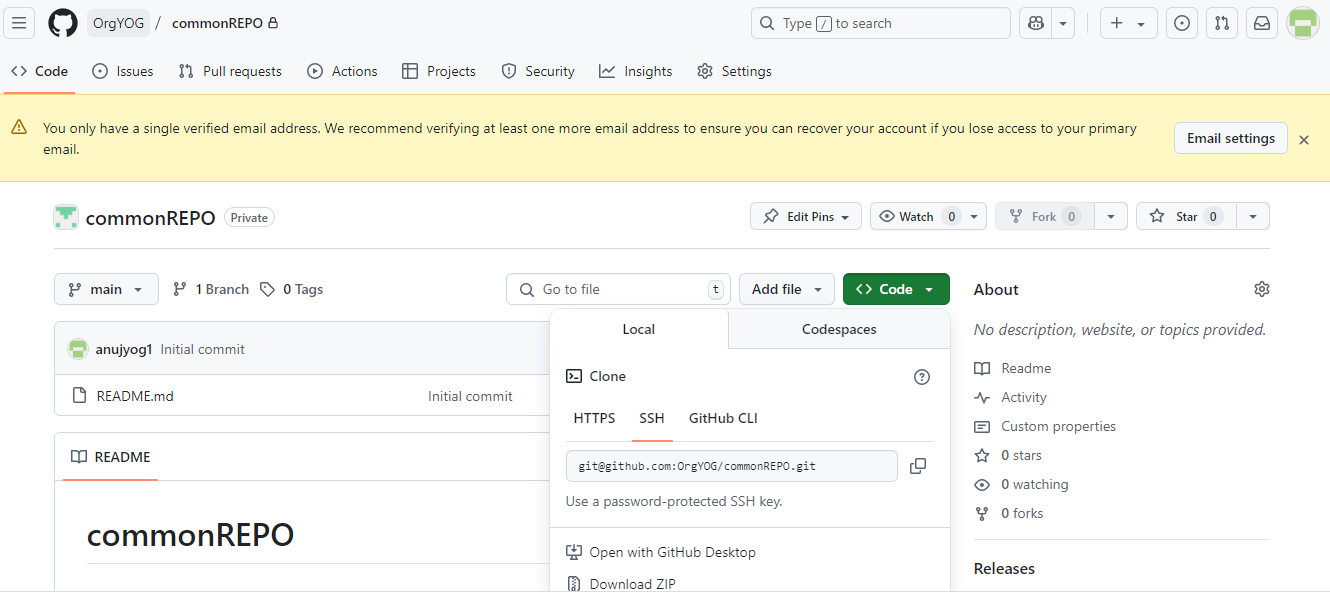
**Next goes to abcrepo**

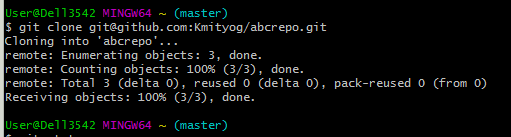
****

**Now open Git bash and clone below repo**

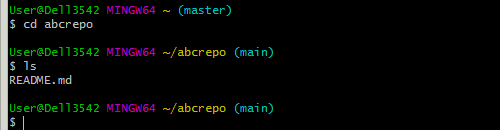
****

**Step 1: first connect to this commonREPO to local repository**

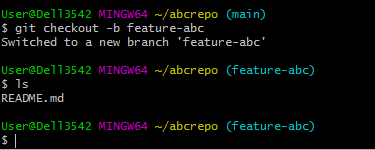
****

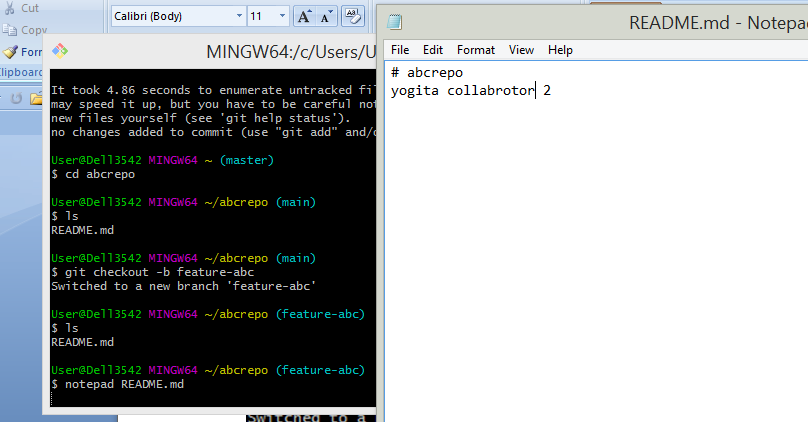
****

**Now change directory to abcrepo**

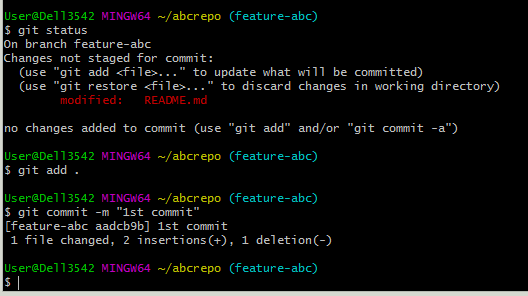
****

**Switch to branch feature-abc**

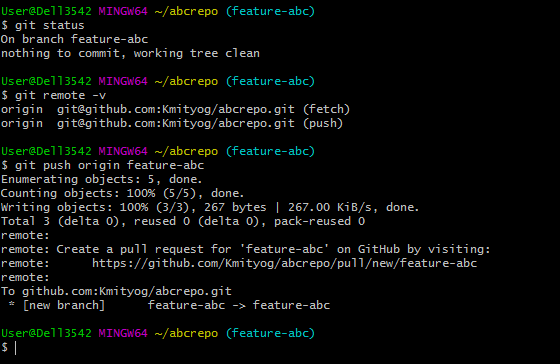
****

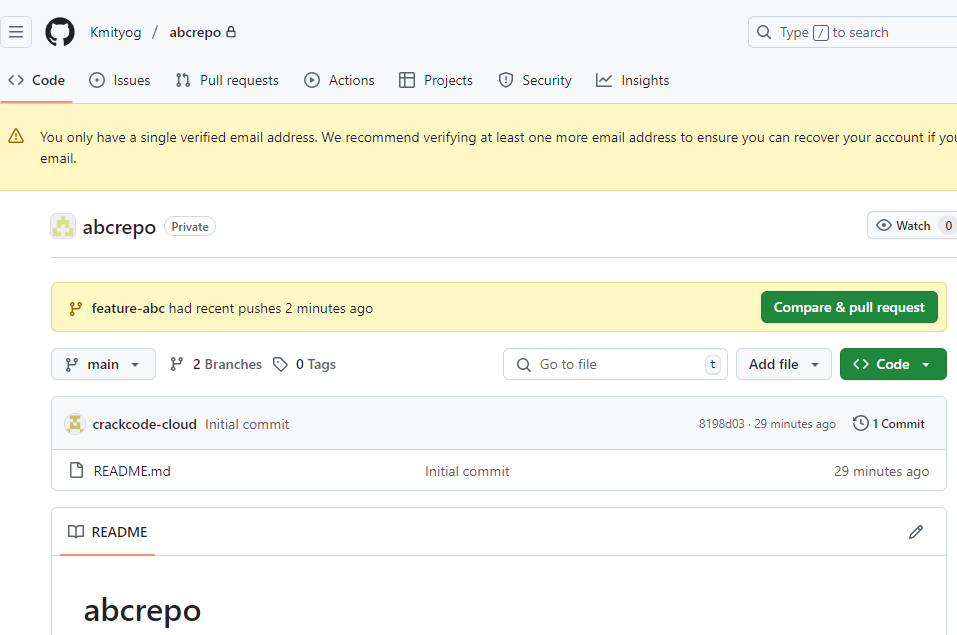
**Now add line in README.md file**

**Now git add, commit**

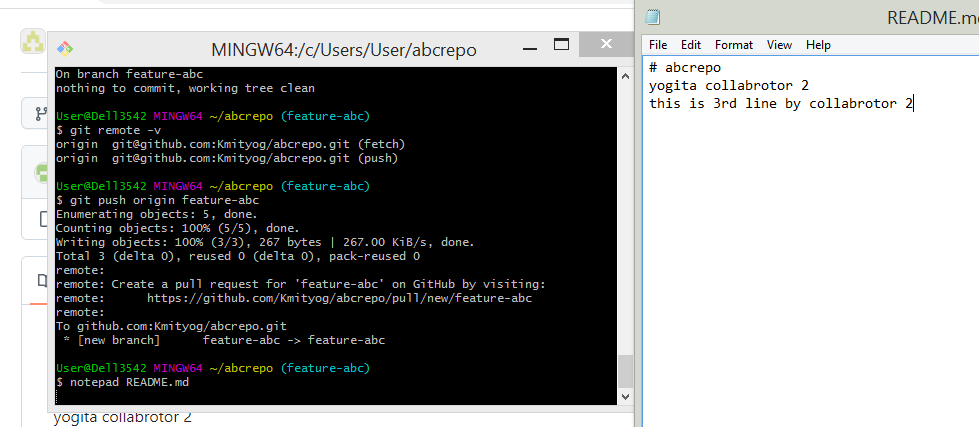
****

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

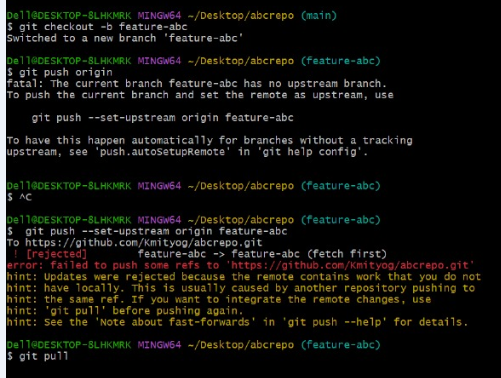
****

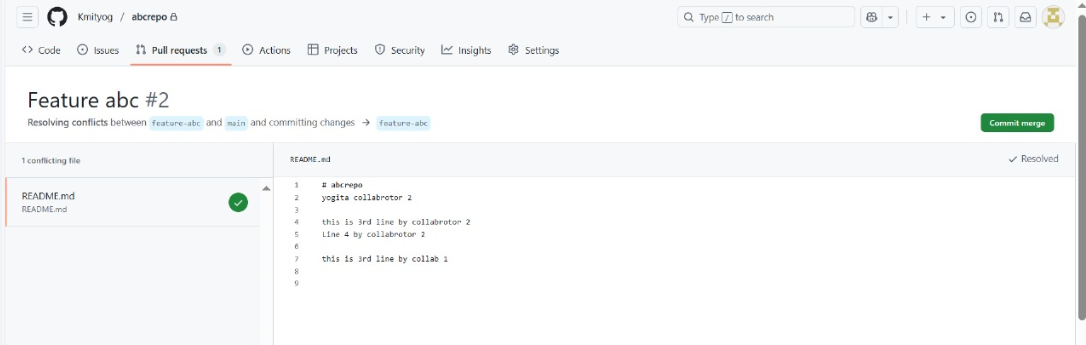
**Now go to Github and refresh or click on abcrepo, you can see **

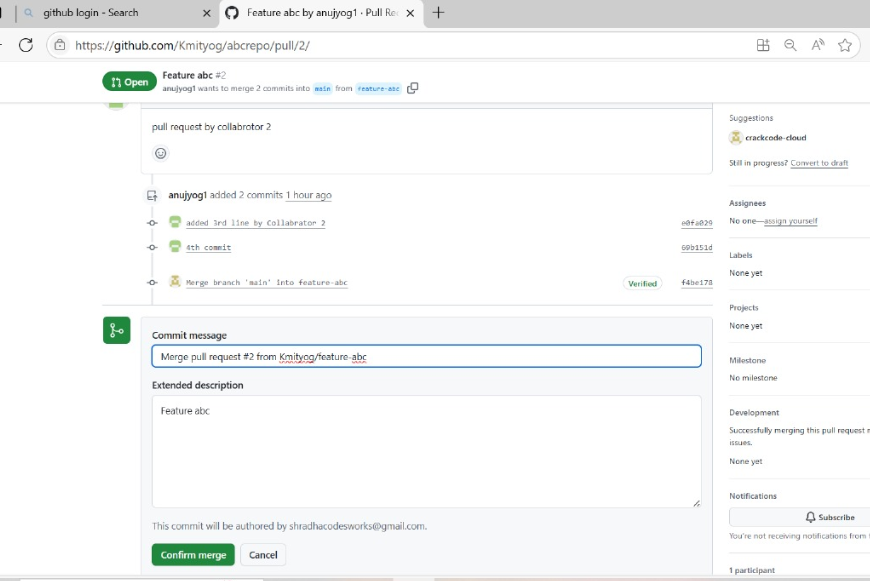
**Now collorator 2 adding line**

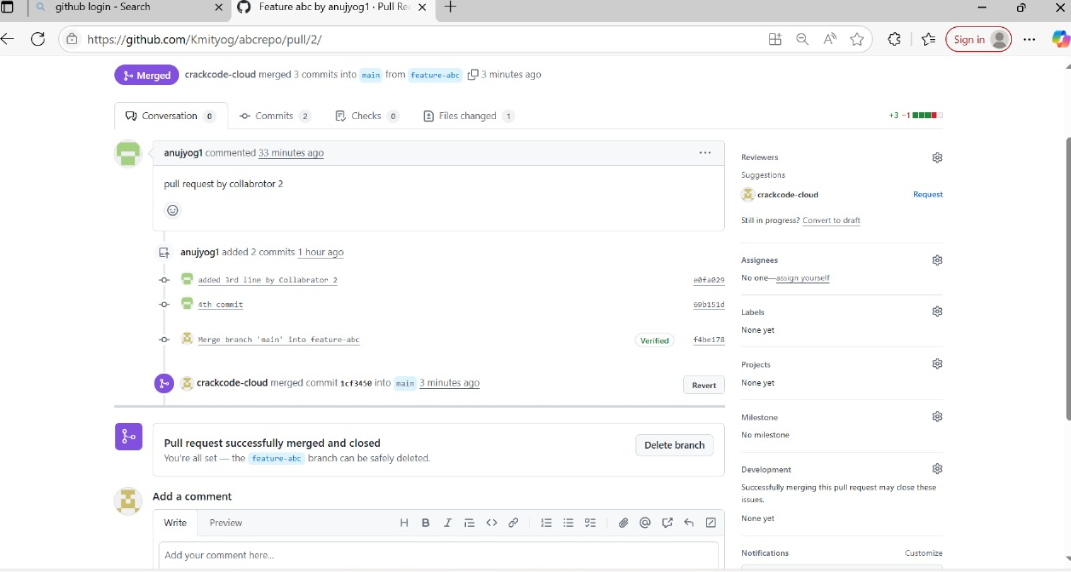
****

**Also collobrator 1 adding line**

****

****

****

****

1. **What is patch**

differences (or changes) between two versions of code, typically in the form of a **diff**. These changes could be additions, deletions, or modifications of lines of code. A patch is usually used to update a project or to propose changes to a project, often through a **pull request**.

**Common Git Patch Usage:**

1. **Creating a Patch**:
   * A patch can be generated by using the git diff command. This shows the differences between the working directory and the index or betsween the index and the last commit.
   * You can create a patch file with this command:

git diff > changes.patch

* + This creates a .patch file with the differences that can be shared with others.

1. **Applying a Patch**:
   * If someone provides you with a .patch file (which contains code changes), you can apply it to your local repository using:

git apply changes.patch

* + This will apply the changes from the patch to your current working directory.

1. **Pull Requests**:
   * On GitHub, contributors typically submit patches to a project by creating **pull requests** (PRs). A PR is a request to merge changes from one branch (often a feature branch) into another (often the main branch).
   * GitHub automatically generates the patch and shows the changes visually, allowing reviewers to approve, suggest changes, or decline the patch.
2. **Patch from Commit**:
   * If you want to create a patch from a specific commit, you can use the following command:

git format-patch -1 <commit-hash>

* + This generates a patch file that can be applied to another repository.

1. How do you create patch

To **create a patch in Git**, you can do it in a few different ways depending on what you want to include in the patch (unstaged changes, specific commits, etc.).

## 1. ****Create a Patch from Uncommitted Changes****

If you’ve made changes to your working directory and want to save them as a patch file:

git diff > my-changes.patch

* This will save the unstaged changes to a file named my-changes.patch.
* If you want to include staged changes:

git diff --cached > my-staged-changes.patch

## 2. ****Create a Patch from One or More Commits****

If you’ve already committed your changes, you can use git format-patch:

git format-patch -1 <commit-hash>

* -1 means "create a patch from 1 commit"
* Example:

bash

CopyEdit

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

## 3. ****Create a Patch Between Two Branches****

To create a patch file with all the differences between two branches:

git diff main feature-branch > feature.patch

Or between commits:

git diff commit1 commit2 > changes.patch

## ✅ Bonus: Apply a Patch

To apply a .patch file to your repo:

git apply my-changes.patch

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

git am 0001-Your-commit-message.patch