**Branch protection in GitHub.**

Branch protection in GitHub is a feature that helps prevent **unauthorized or accidental changes** to important branches (such as **main** or **develop**). It enforces rules that contributors must follow before merging or pushing changes.

Branch protection is essential to:

1. **Prevent Accidental Deletions or Force Pushes**
   * Stops developers from accidentally deleting or force-pushing changes to critical branches.
2. **Enforce Code Reviews and Quality Standards**
   * Ensures that pull requests **(PRs)** are reviewed before merging.
3. **Require Passing CI/CD Checks**
   * Prevents merging unless tests (GitHub Actions, Jenkins, etc.) pass.
4. **Ensure Commit Signing and Verification**
   * Enforces rules where commits must be signed and verified for security.
5. **Restrict Who Can Push to Certain Branches**
   * Only allows maintainers or authorized users to make direct changes.

**Common Branch Protection Rules**

| **Rule** | **Purpose** |
| --- | --- |
| Require pull request reviews | Ensures code is reviewed before merging |
| Require status checks to pass | Prevents broken code from being merged |
| Restrict who can push | Limits access to maintainers or admins |
| Prevent force pushes | Avoids accidental overwrites of commit history |
| Prevent branch deletion | Protects critical branches from being deleted |

Let’s us assume that you are a project manager of “bloodyyoung” project for this project a team is created, you are the project manager for it.

A company administrator created a repository in GitHub with the name of “**bloodyyoung”** and you and your team members should work collaboratively with this repository by inviting/adding people.

As a project manager and company administrator have full control/ access to this repository.

But for the project team members have limited access over the repository of “**bloodyyoung”.**

Mainly the team members does not have access/control on the main/master branch of bloodyyoung repository.

Some of the branching rules are assigned to the team members by the project manager or company administrator.

Let us perform, assigning of branching rules for the team members by inviting/adding team members to the repository in practical.

**Step1:** Create a repository with name (bloodyyoung) and add peoples to it.

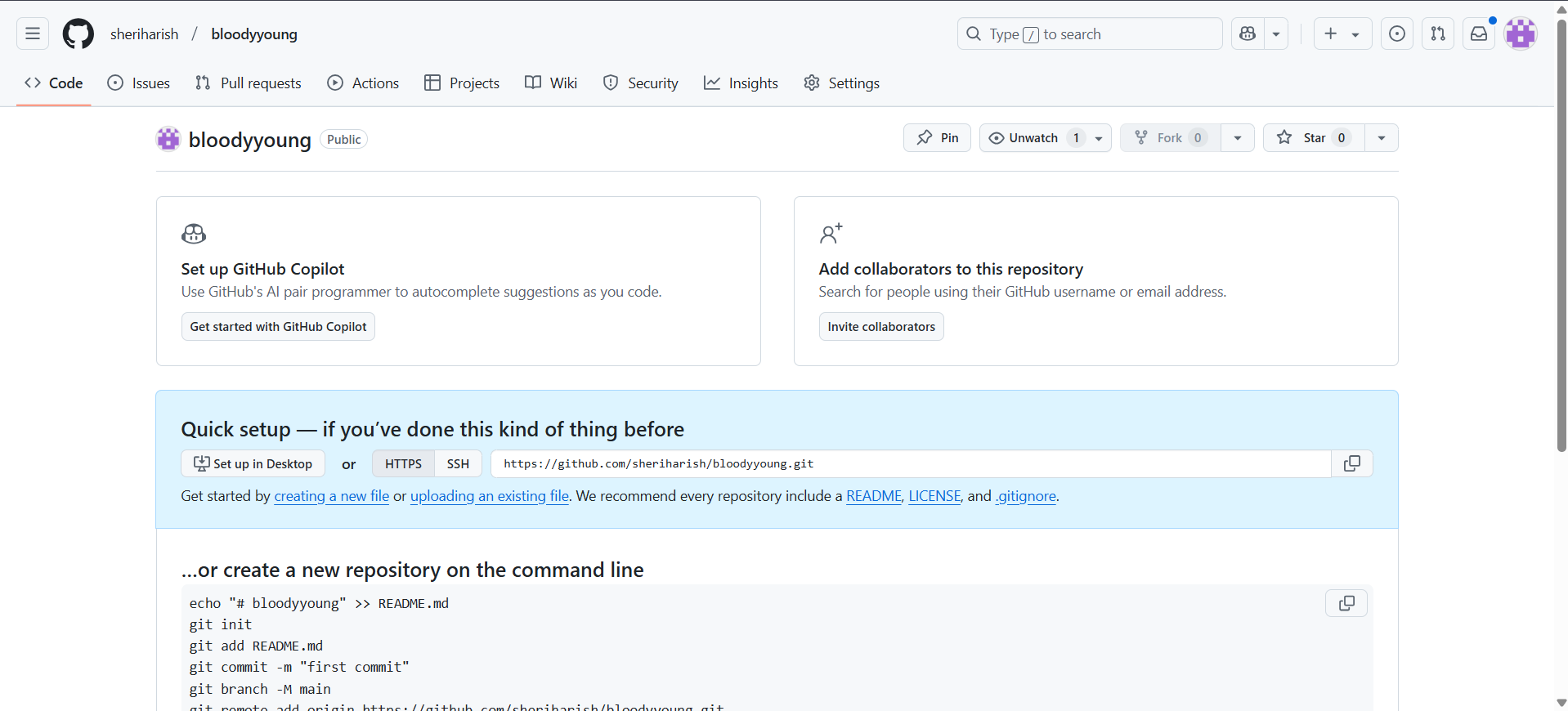


Fig: Repository “bloodyyoung”.

To add people to the repository: Go to 🡪repository setting 🡪collaborators 🡪Add people.

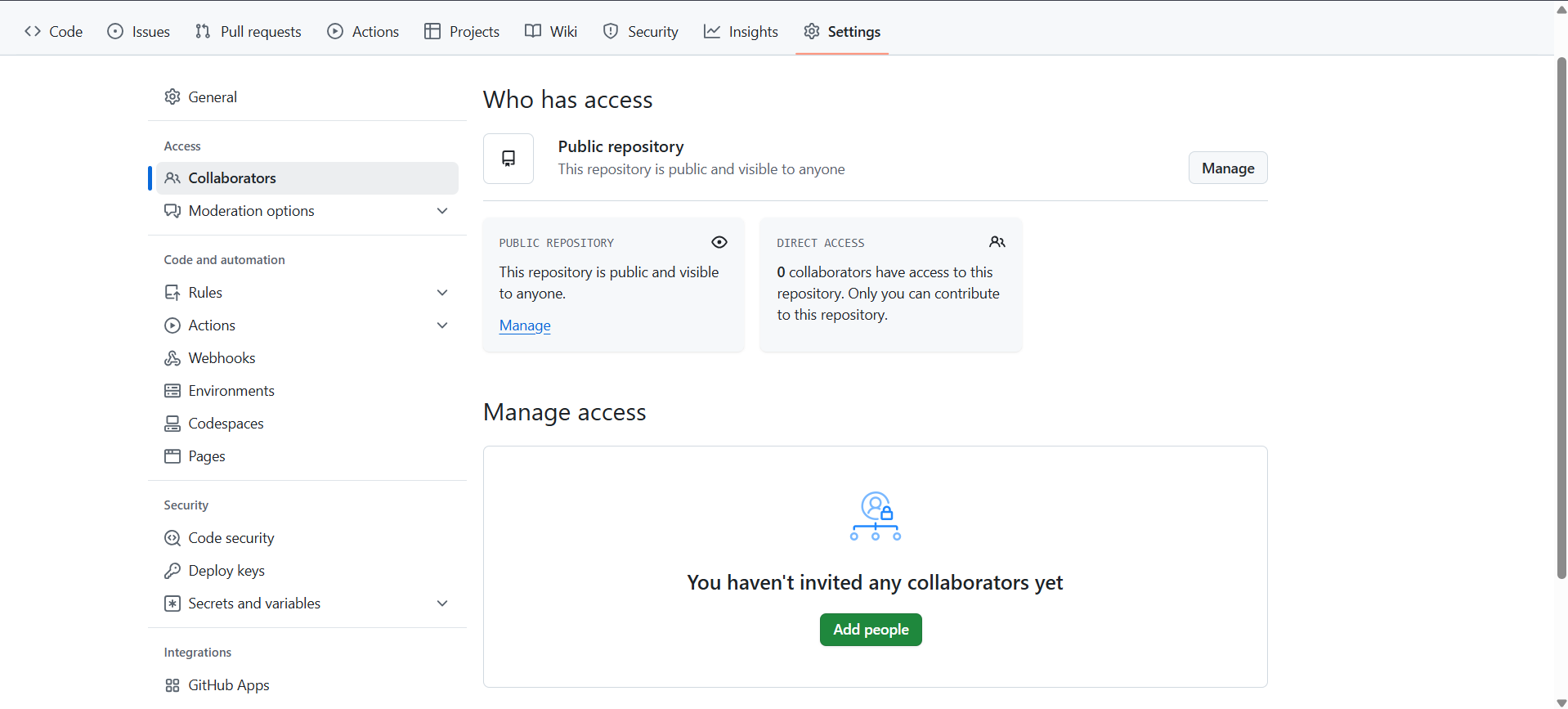
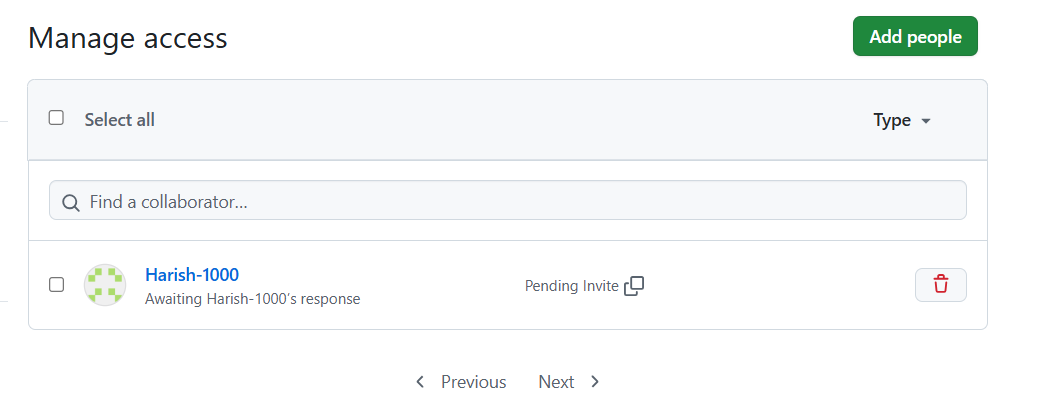
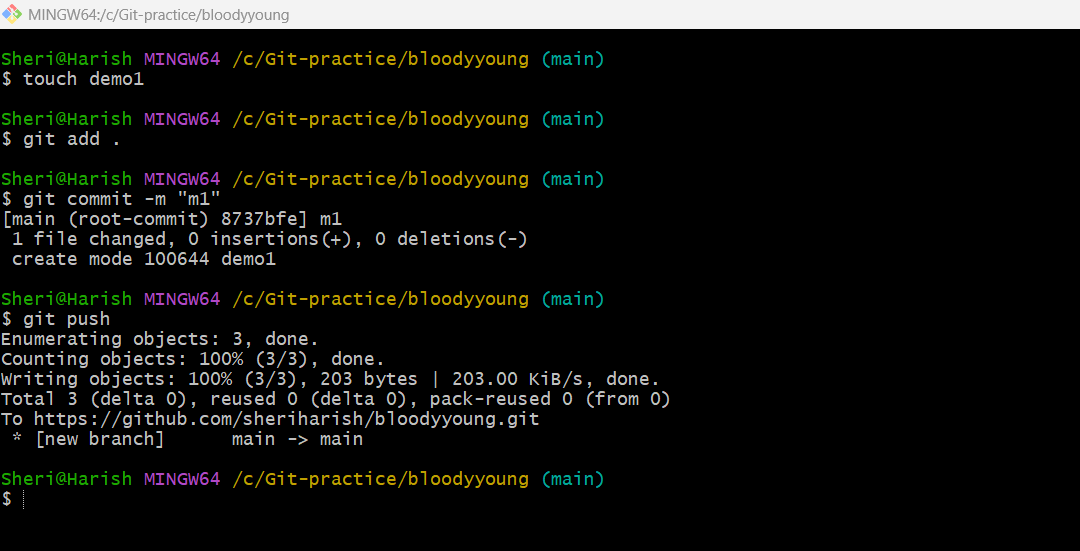


Fig: Harish-1000 is added to the repository bloodyyoung.

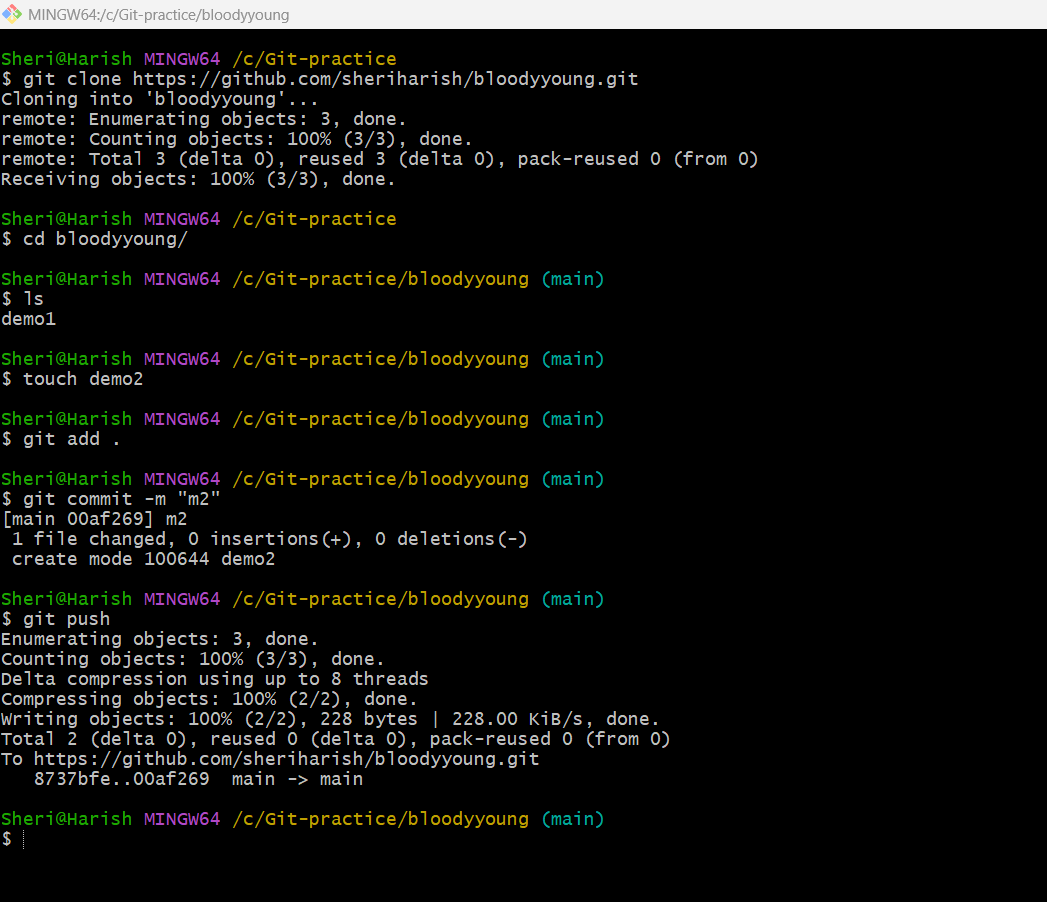
Now Harish-1000 can work collaboratively with the repository bloodyyoung.

**Note:** The invitation link is send to the Harish-1000 email as well as to repository notification. If he accept then only he can work collaboratively with this repository.

**Step2:** Now push any file to the main branch. (done by project manager).

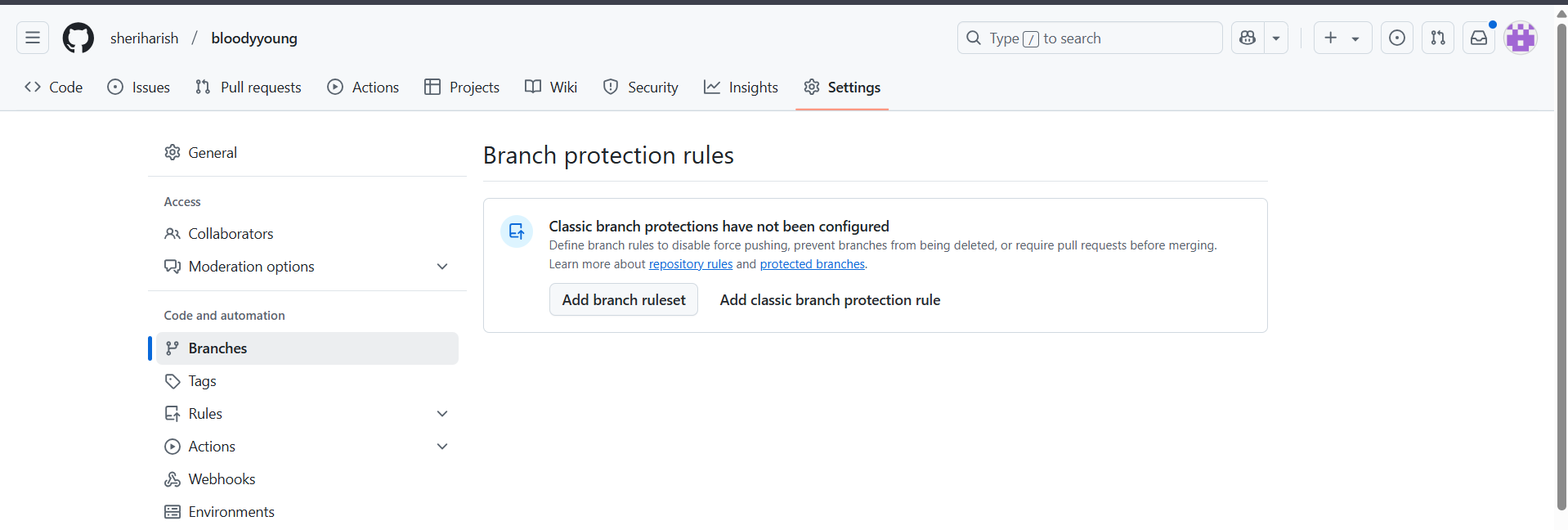


**Step3:** Again push any file to the main branch before assigning branch rule (Done by team member).



Before assigning the branch rules the team member Harish pushed the file “demo2” successfully.

Step4: Now assign the branch rules. (Done by project manager).



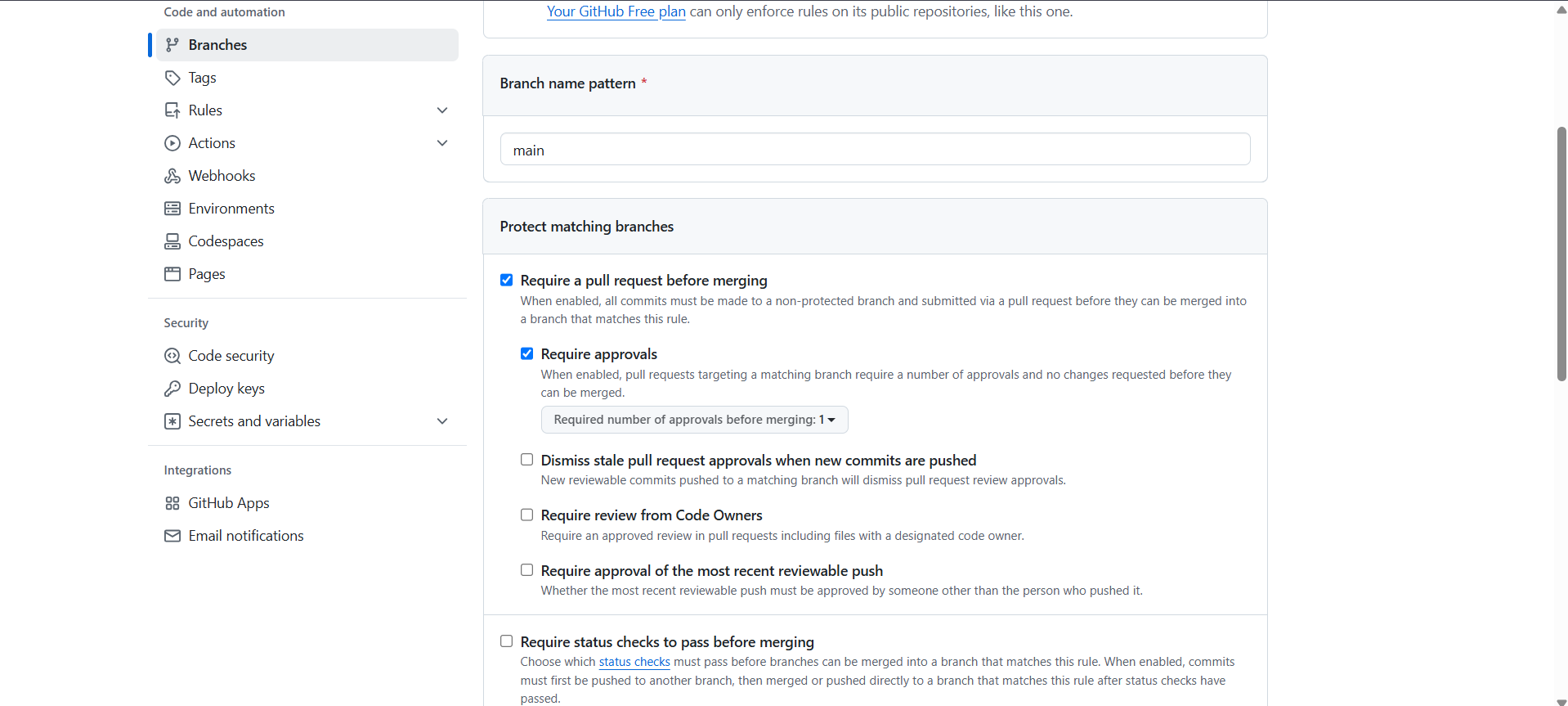
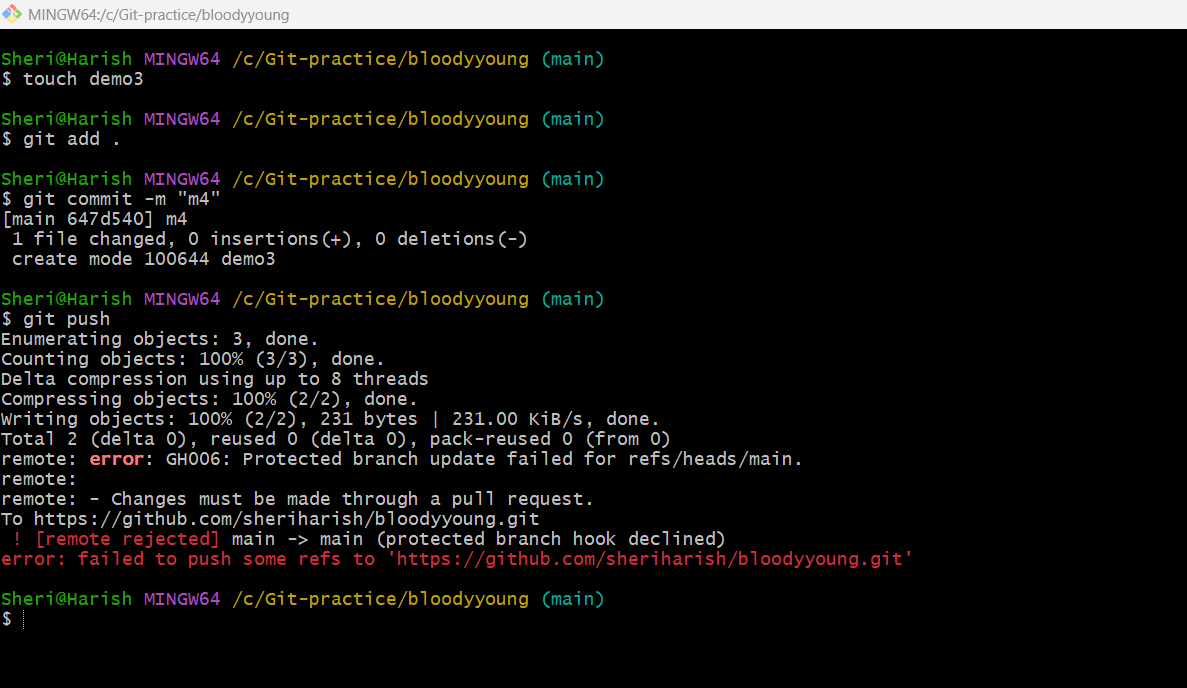


Fig: Branch rules are added.

**Step5:** Now again try to push ant file after adding the branching rules to the repository bloodyyoung (Done by team member Harish-1000).



Then it show some error message that failed to push. That means after adding braching rules Harish-1000 is unable to push the file “demo3” to the main branch of bloodyyoung repository.

**Note:** Branching rules are mainly used to protect the main code or source code which is present in the main/master branch from the team members. But the team members can work on by creating the child/ feature branches by copying the source code from main branch.

Step6: let us create a feature branch and merge it with main branch.

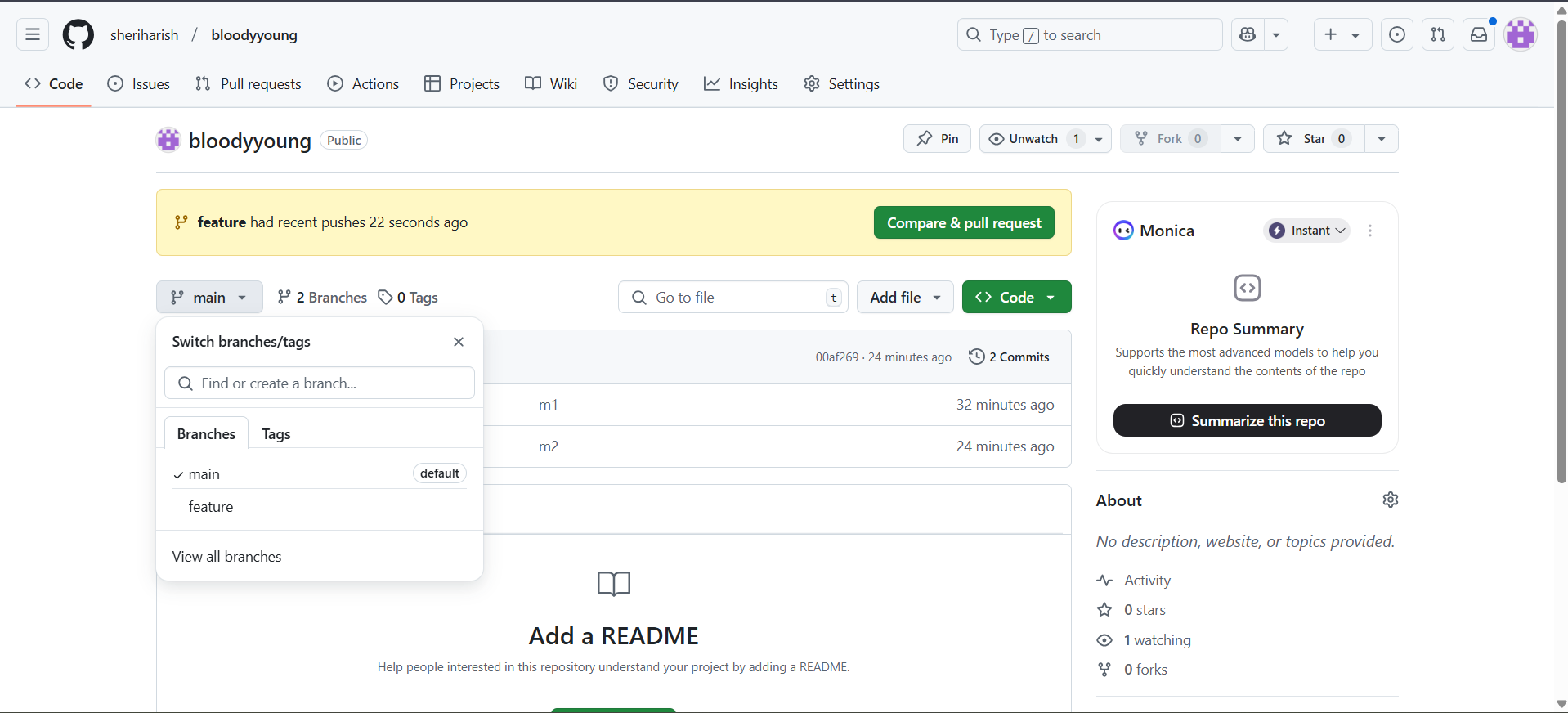
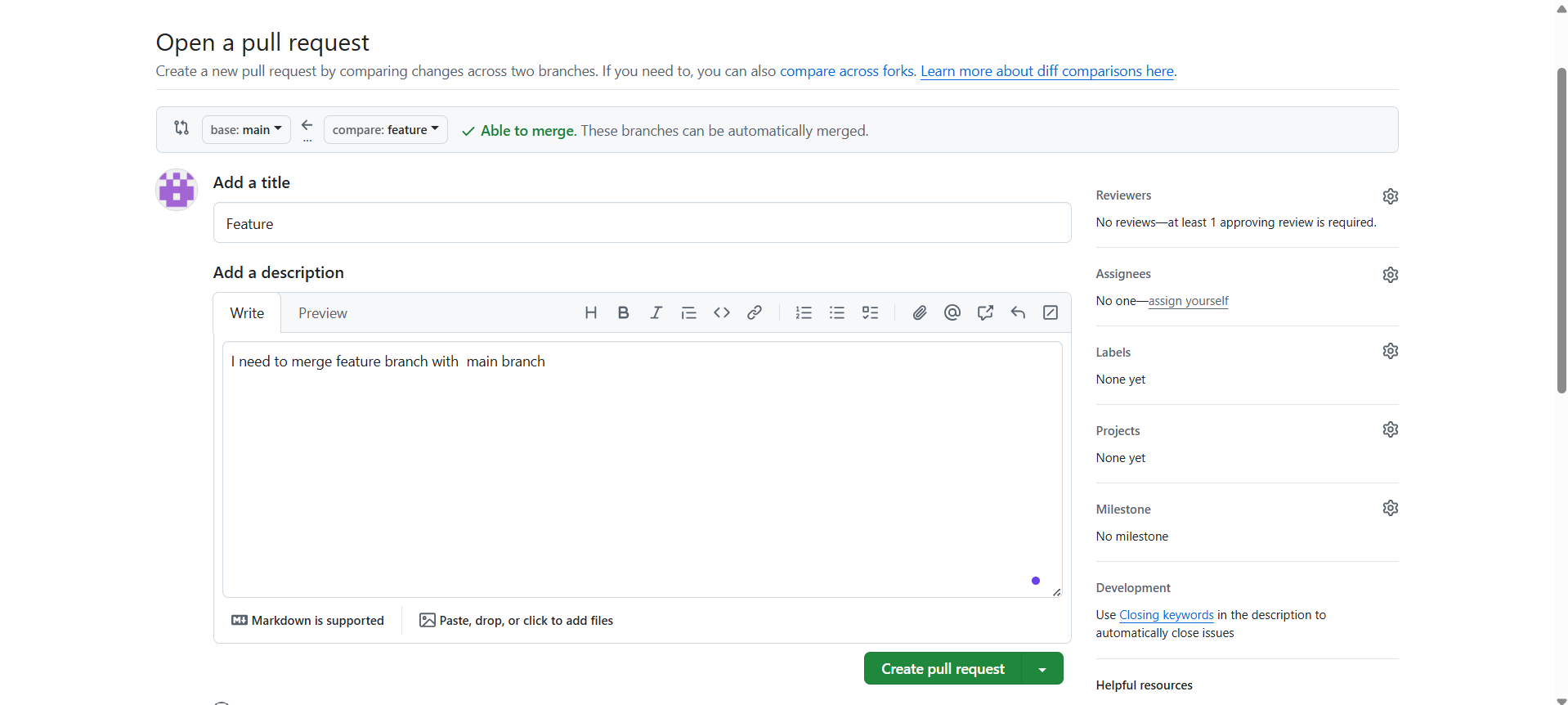


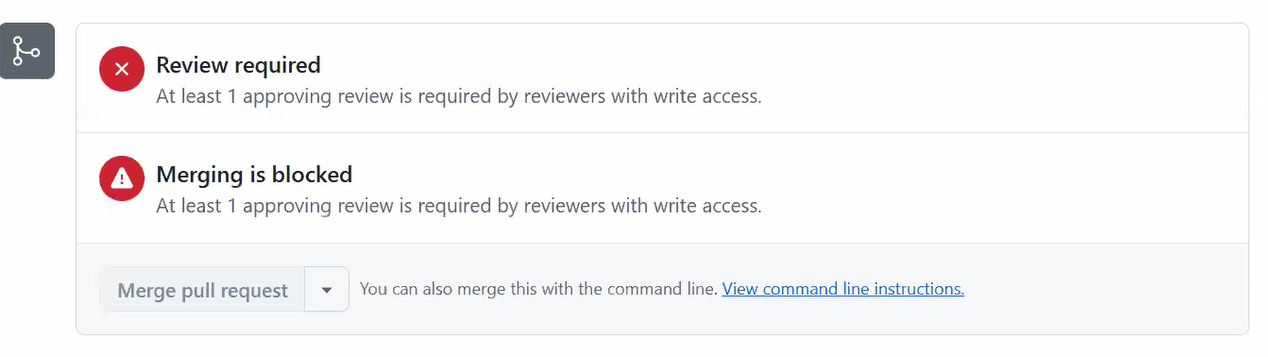
Fig: Feature branch is created successfully.

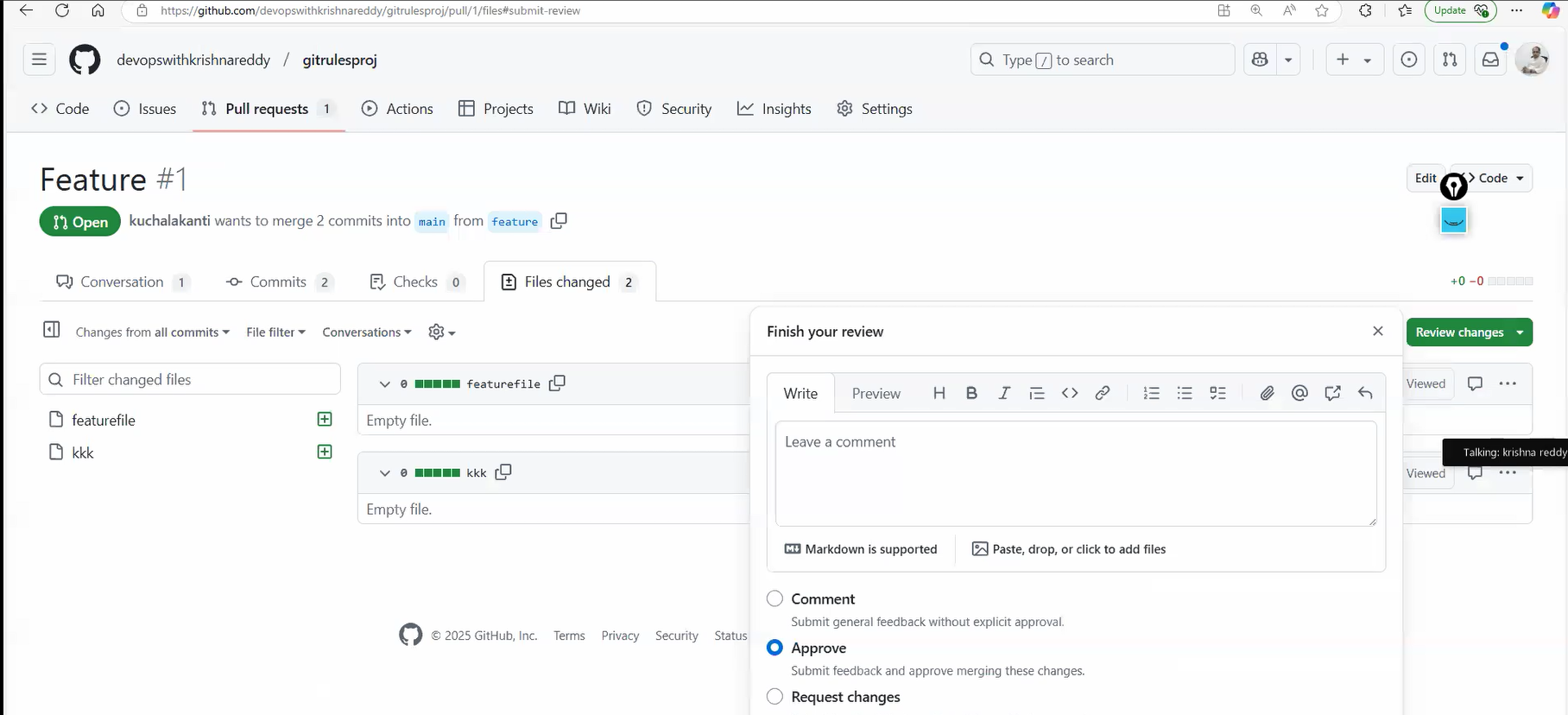
Step7: Merge the feature branch with main branch.

**Note**: In order to merge the feature branch with main branch after applying branching rules we require the pull request, if the owner of the repository (project manager/administrator) accepts then only we can merge the feature branch with main branch.



After sending the pull request it wait for the approval from the owner of the repository (/manager/administrator) as show below figure.





**Fig:** Pull request is approved.

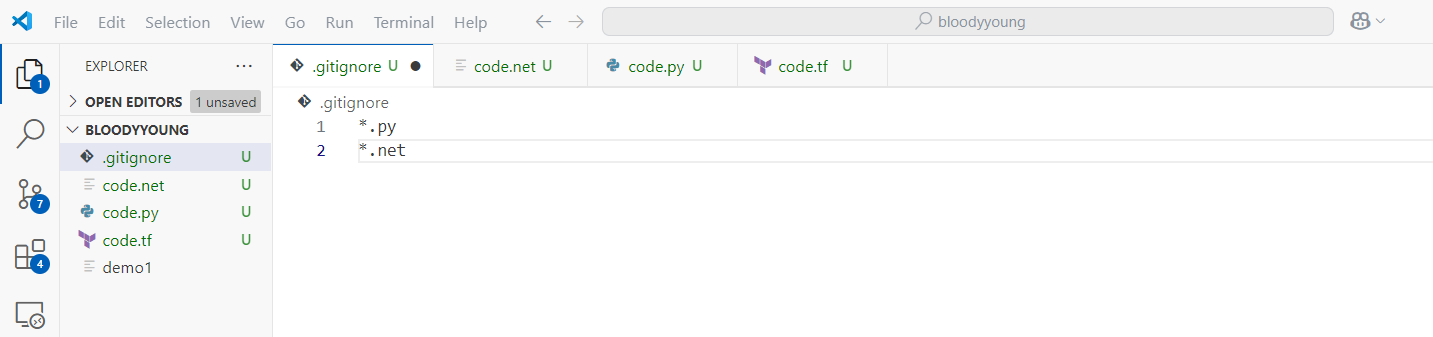
Then now the feature branch is merged with the main branch successfully.

**.gitignorefile.**

The **.gitignore** file is used to **specify files and directories that Git should ignore**. It prevents unnecessary or sensitive files from being tracked in a Git repository.

### ****Why is “****.gitignore” ****Important?****

1. **Avoid Committing Unnecessary Files**
   * Example: Logs, temporary files, dependencies (node\_modules, venv).
2. **Protect Sensitive Information**
   * Example: API keys, database credentials (.env, config.json).
3. **Improve Performance**
   * Avoid tracking large files that shouldn't be in the repository.
4. **Prevent OS-Specific or Editor Files**
   * Example: .DS\_Store (macOS), Thumbs.db (Windows), \*.swp (Vim).



The files or directories which are present in .gitignore file are not be pushed into the repository, “.py” & “\*.net” files are ignored while pushing in to the repository.