**Source Code Management**

**Task 1.2**

(CS181)

Submitted by

Name-Aadithya Roll No.- 2110990002



**Department of Computer Science & Engineering**

Chitkara University Institute of Engineering and Technology, Punjab

Jan- June   
(2021-22)

| Institute/School Name | **Chitkara University Institute of Engineering and Technology** | | |
| --- | --- | --- | --- |
| Department Name | **Department of Computer Science & Engineering** | | |
| Programme Name | **Bachelor of Engineering (B.E.), Computer Science & Engineering** | | |
| Course Name | **Source Code Management** | Session | **2021-22** |
| Course Code | **CS181** | Semester/Batch | **2nd/2021** |
| Vertical Name | **Beta** | Group No | G01 |
| Course Coordinator | **Dr. Neeraj Singla** | | |
| Faculty Name | **Mr. Monit Kapoor** | | |

**INDEX**

| **S. No.** | **Title** |
| --- | --- |
| 1 | Add collaborators on GitHub Repo |
| 2 | Fork and Commit |
| 3 | Merge and Resolve conflicts created due to own activity and collaborators activity. |
| 4 | Reset and Revert |

***ADD COLLABORATORS ON GITHUB REPO***

In GitHub, we can invite other GitHub users to become collaborators to our private repositories (which expires after 7 days if not accepted, restoring any unclaimed licenses). Being a collaborator of a personal repository you can pull (read) the contents of the repository and push (write) changes to the repository. You can add unlimited collaborators on public and private repositories.

Collaborators can perform a number of actions into someone else’s personal repositories, they have gained access to. Some of them are,

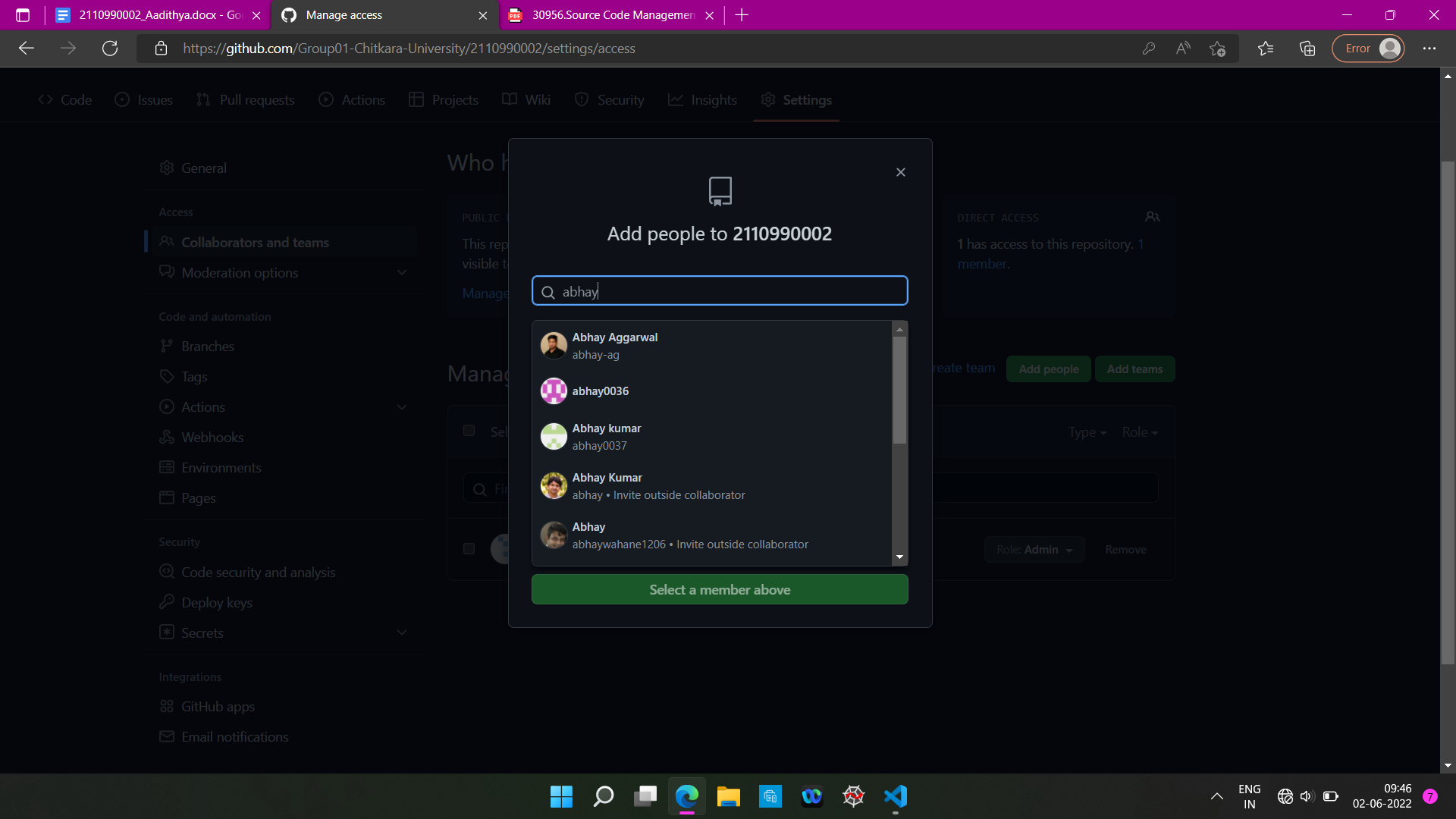
1. Create, merge, and close pull requests in the repository
2. Publish, view, install the packages
3. Fork the repositories
4. Make the changes on the repositories as suggested by the Pull requests.
5. Mark issues or pull requests as duplicate
6. Create, edit, and delete any comments on commits, pull requests, and issues in the repository
7. Removing themselves as collaborators on the repositories.
8. Manage releases in the repositories.

STEPS TO ADD COLLABORATORS:

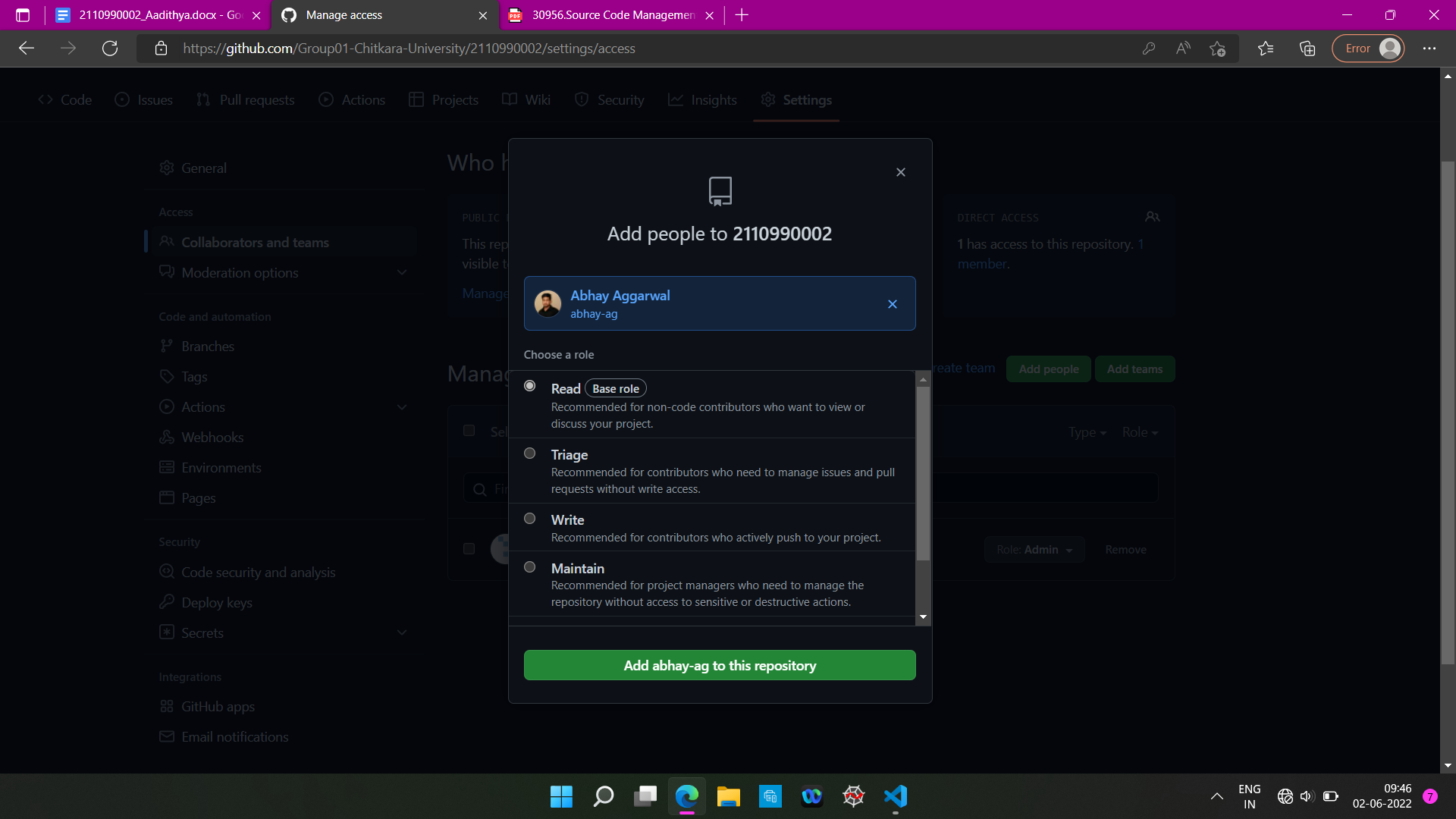
1. Navigate to the repository on Github you wish to share with your collaborator.
2. Click on the "Settings" tab on the right side of the menu at the top of the screen.

3.On the new page, click the "Collaborators" menu item on the left side of the page.

1. Start typing the new collaborator's GitHub username into the text box.



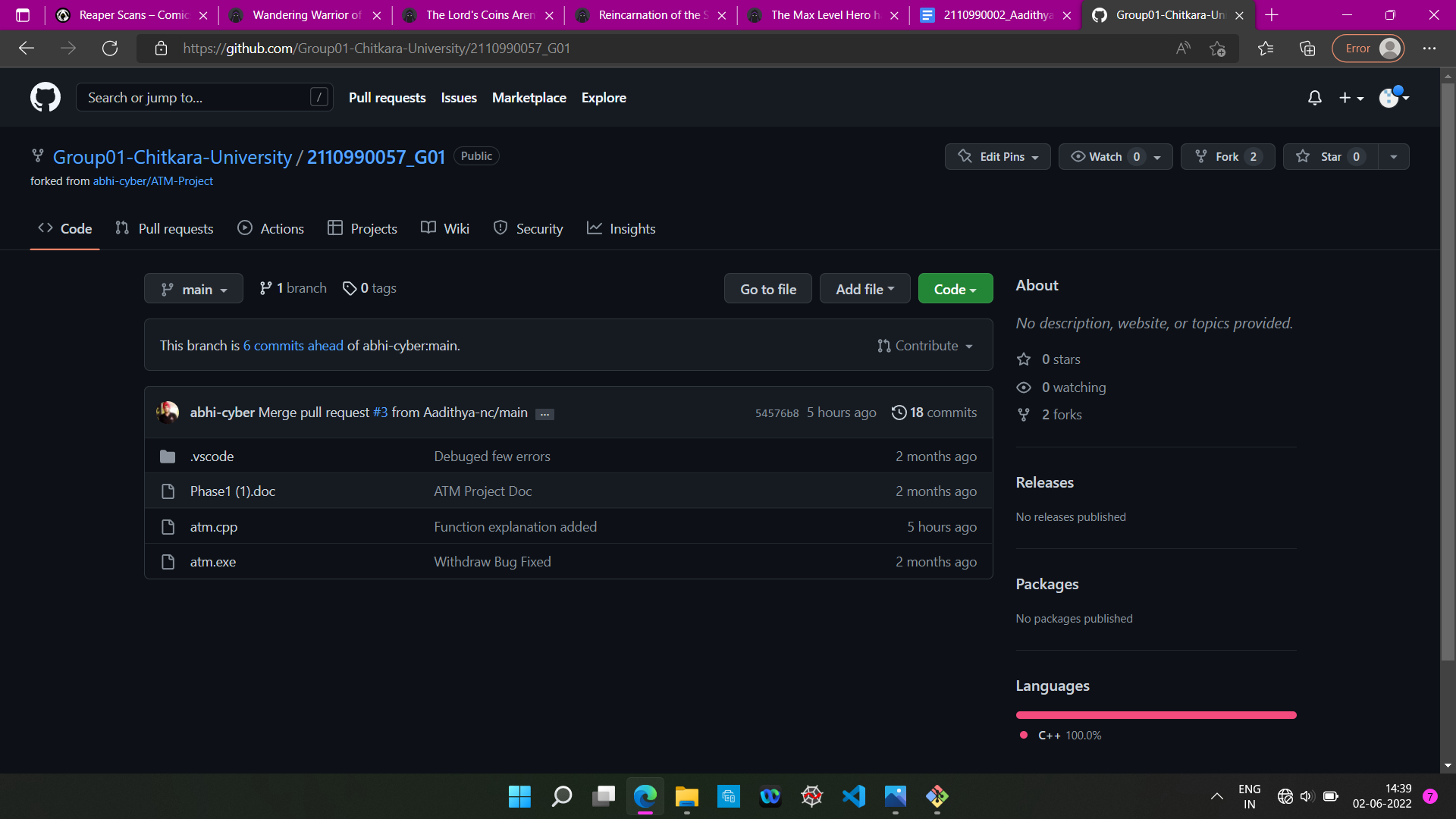
1. Select the GitHub user from the list that appears below the text box.
2. Click the "Add" button.



***FORK AND COMMIT***

A fork is a copy of a repository. Forking a repository allows you to freely experiment with changes without affecting the original project. Most commonly, forks are used to either propose changes to someone else's project to which you do not have write access, or to use someone else's project as a starting point for your own idea.

**STEPS TO FORK A REPO-**



1. Go to the repository that you wish to fork.

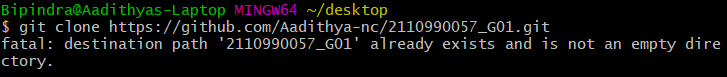
2.Click on the option ‘Fork’ in the top right corner.

3.You now have a forked repository.

**CLONING THE REPO INTO YOUR DEVICE**

When you create a repository on GitHub.com, it exists as a remote repository. You can clone your repository to create a local copy on your computer and sync between the two locations.

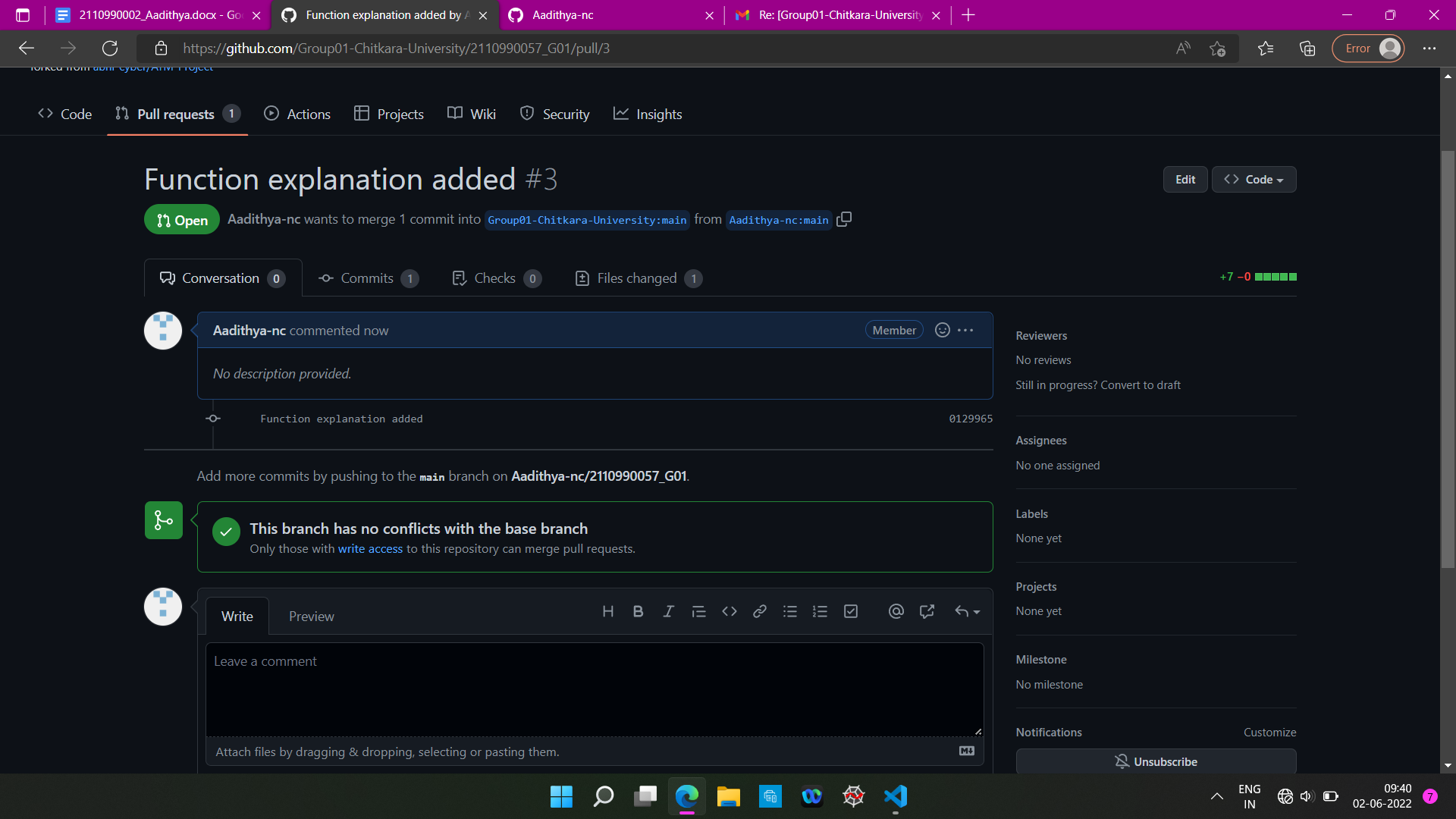
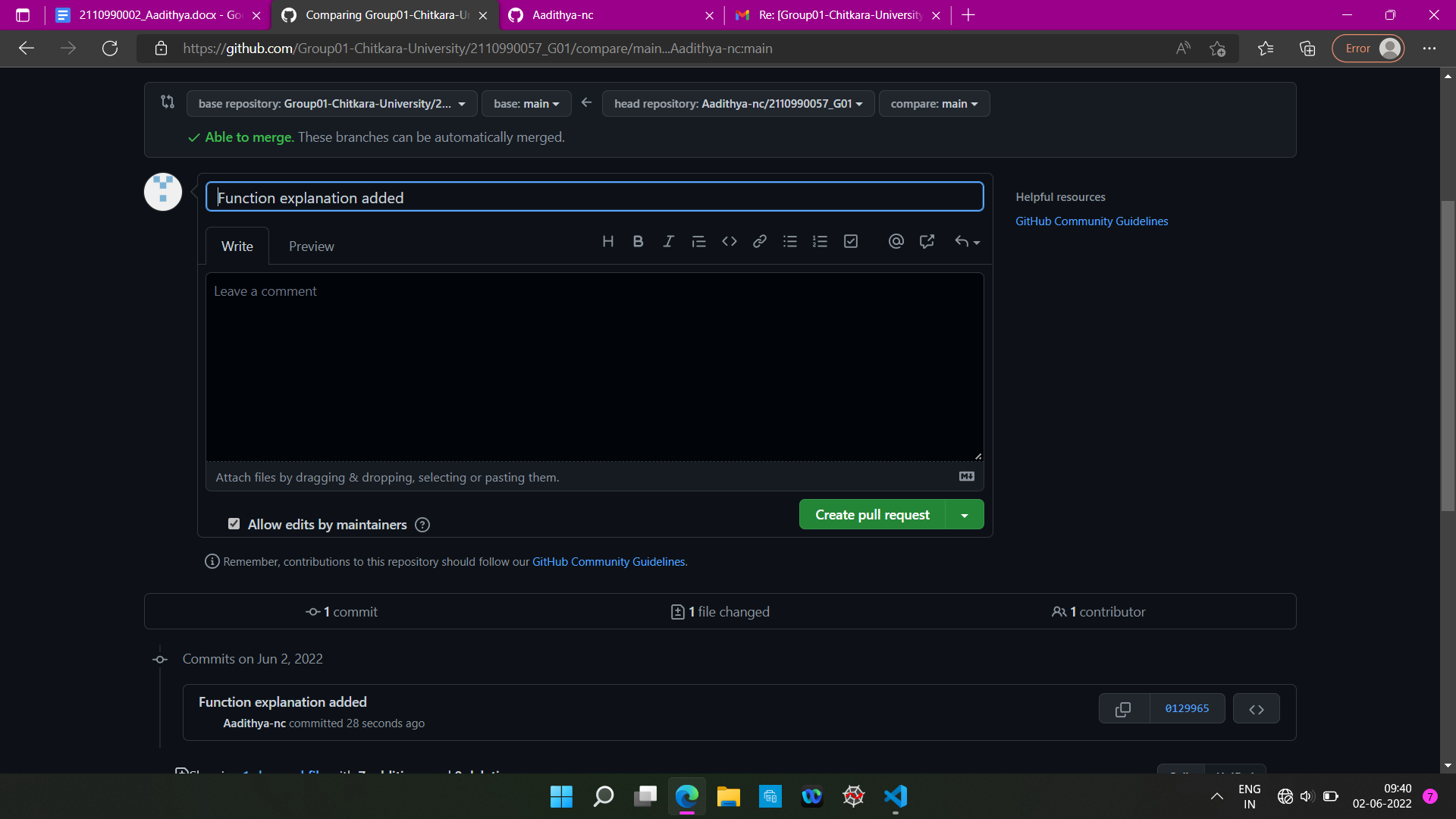
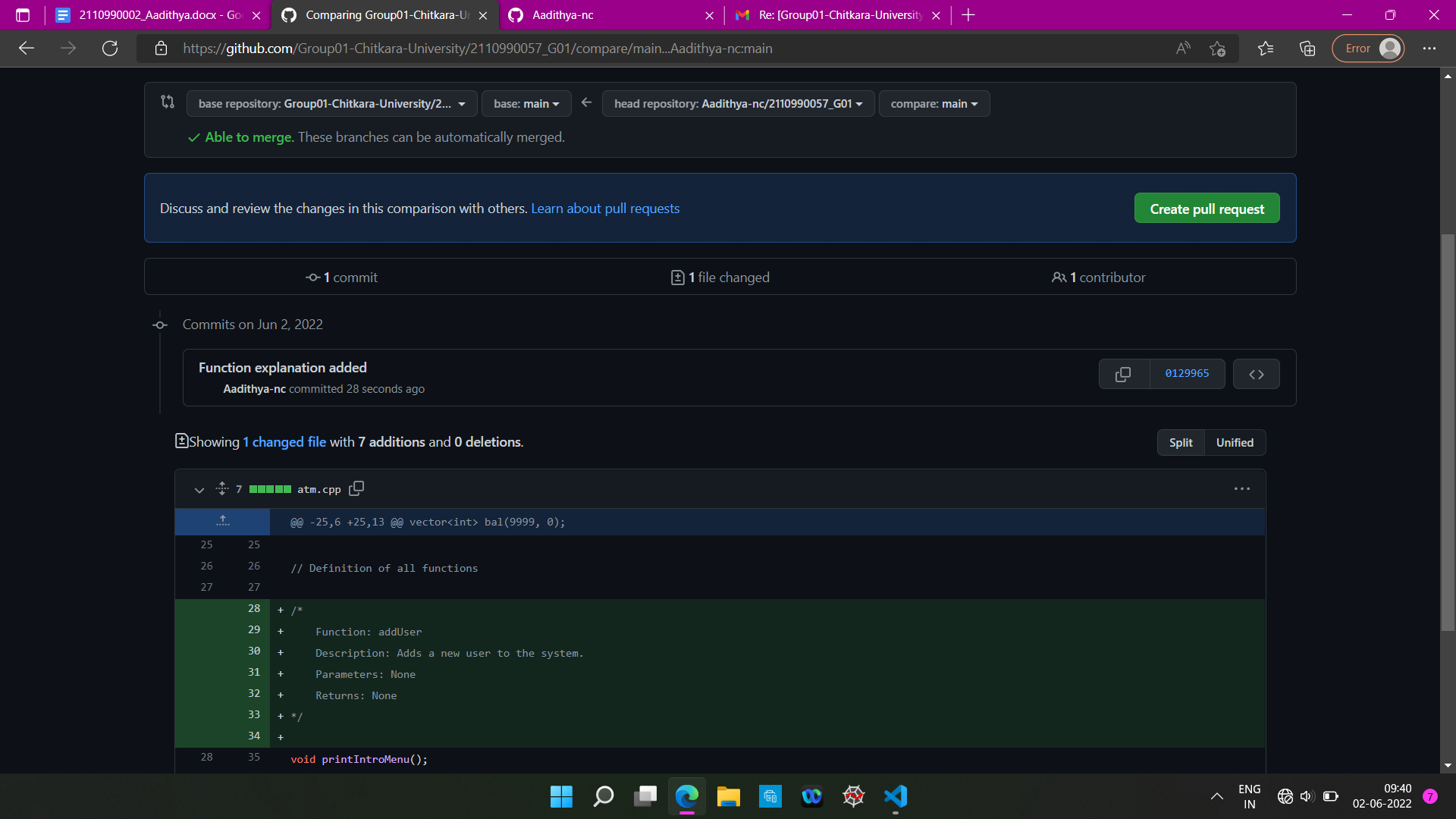
1. Once you have forked the repository, you can clone it into your computer using directly the option given on github or through running git clone command in git bash.
2. Copy the URL of the forked repository
3. Open git bash and type the command “ git clone <url of the forked repository>”



(here it says destination path already exists because i had cloned the repo few days before i took the screenshot)

**COMMITING CHANGES TO THE FORKED REPOSITORY**

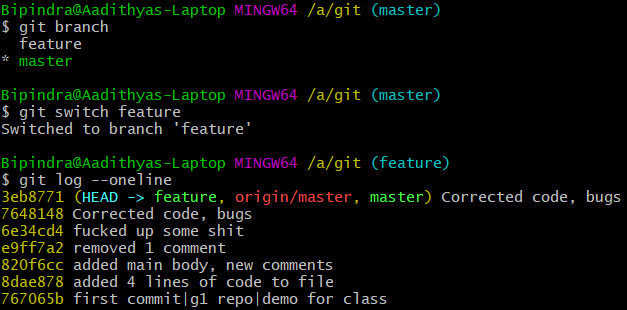
1. Once you have cloned the repository you can introduce changes to it as per your wish.
2. After changing it you have to stage the file and then commit it.
3. After committing changes push it to your remote repository.



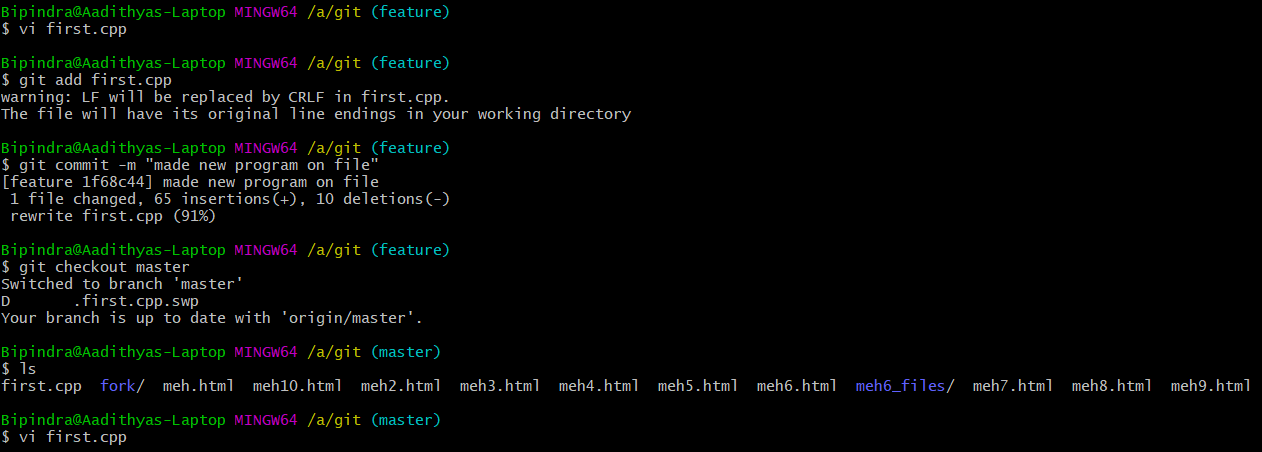
***MERGE AND RESOLVE CONFLICTS CREATED DUE TO OWN ACTIVITY AND COLLABORATORS ACTIVITY***

Merging and conflicts are a common part of the Git experience. Conflicts generally arise when two people have changed the same lines in a file, or if one developer deleted a file while another developer was modifying it. In these cases, Git cannot automatically determine what is correct. Conflicts only affect the developer conducting the merge, the rest of the team is unaware of the conflict. Git will mark the file as being conflicted and halt the merging process. It is then the developers' responsibility to resolve the conflict.

1.To understand the merging concept of branches, create a branch named “feature” in your repository.



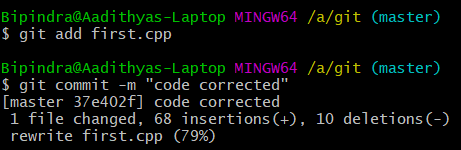
2.Here, there is a file called ‘day2.cpp’. Make changes to it, add and commit them.

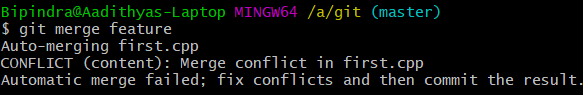


3.Similarly, change the same lines of day2.cpp file in the master branch.

4.If you are not already on the branch that you want the other one to merged in (in this example master branch), then switch to it.

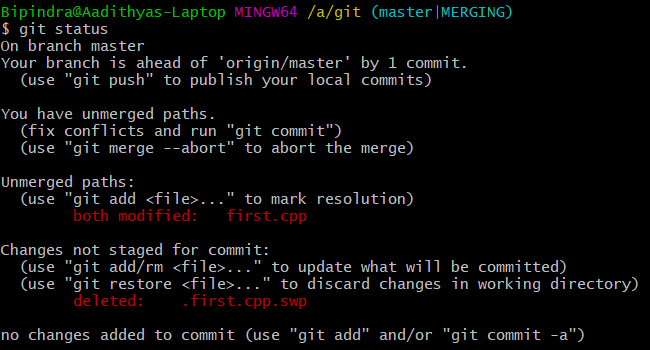
5.Using the command try merging feature branch into master branch using the “git merge <branch name>”





6.Auto merging fails and conflict arises. In order to resolve it we make use of the mergetool by running the command “git mergetool”. The mergetool editor will open.

7.Make changes as per requirement in order to resolve the conflicts and exit the editor.



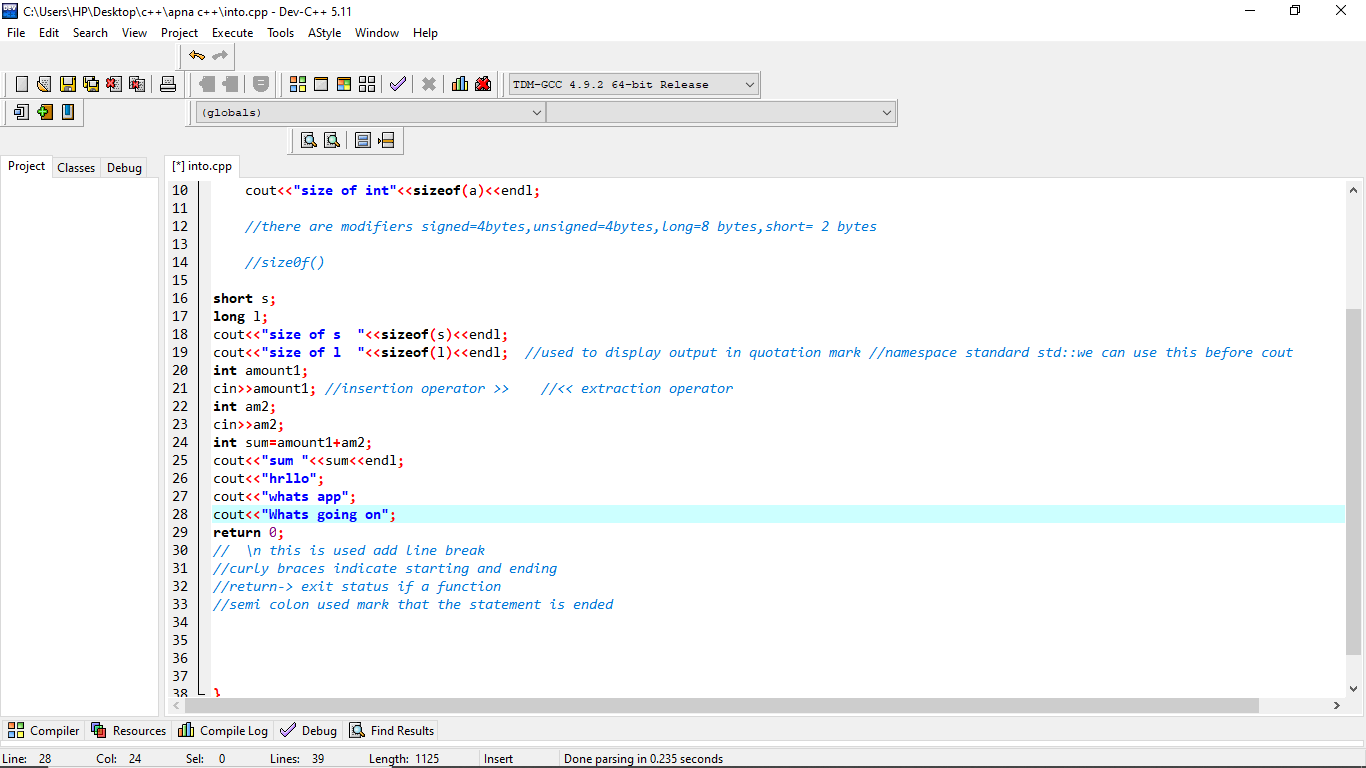
***RESET AND REVERT***

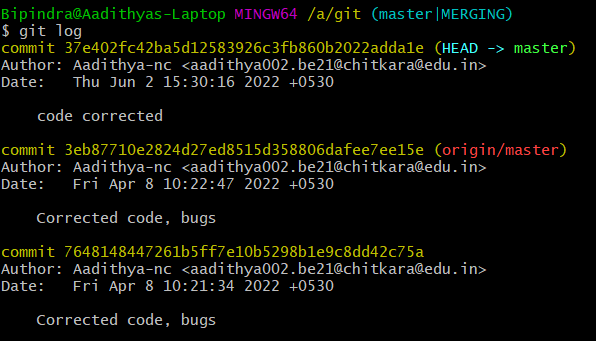
While Working with Git in certain situations we want to undo changes in the working area or index area, sometimes remove commits locally or remotely and we need to reverse those changes. We can do it by using the git reset, git revert, git checkout commands.

**RESET**-

git reset is used when we want to unstage a file and bring our changes back to the working directory. Git reset can also be used to remove commits from the local repository.

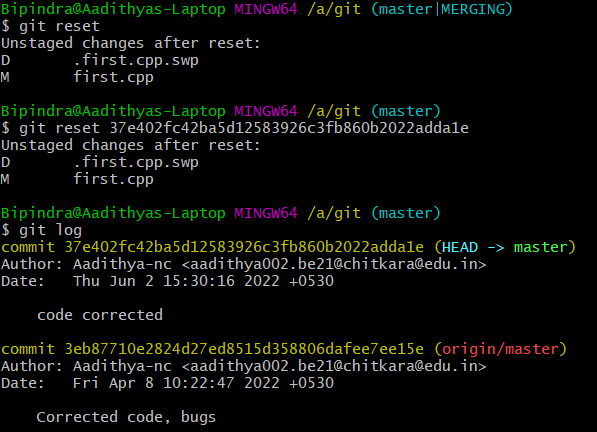
Suppose we make edits to a file, stage it and commit it



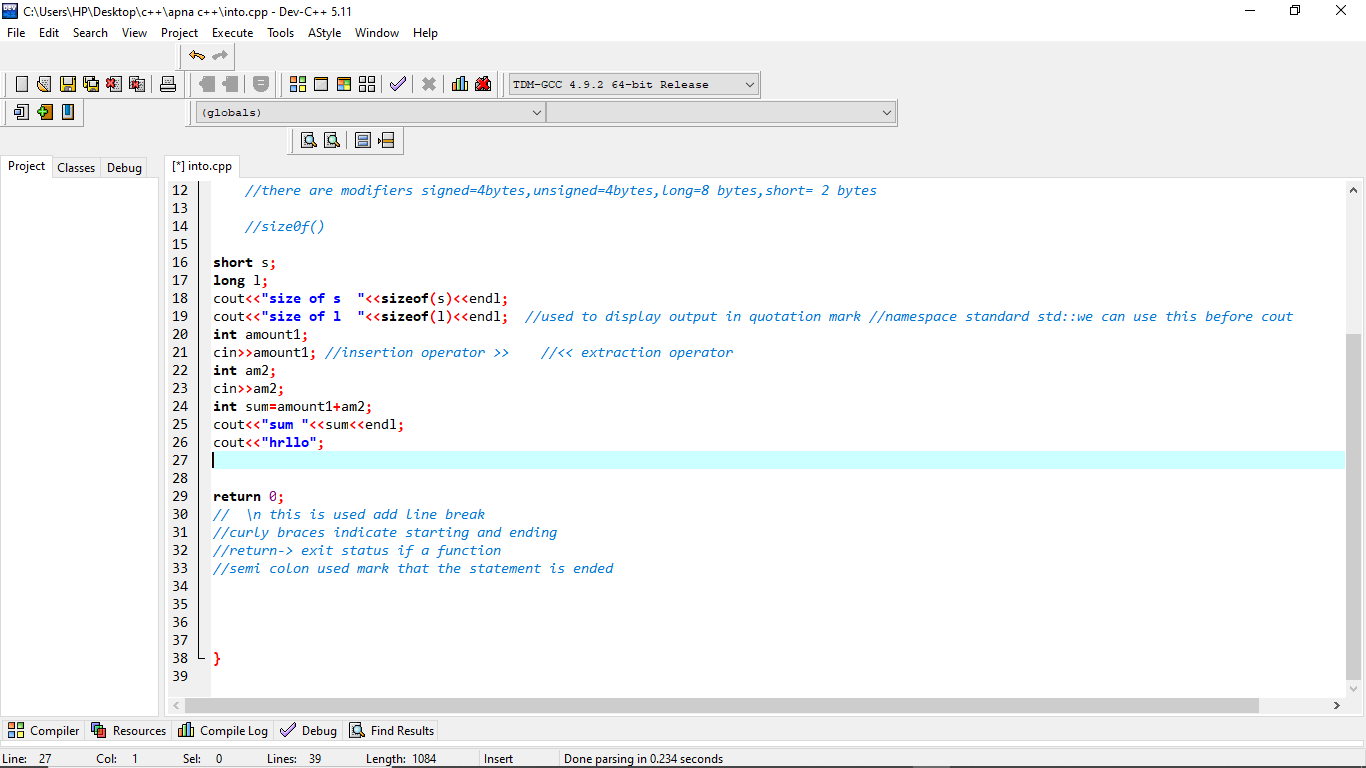


In order to reset the changes made in the recent commit, run the “git reset --hard HEAD~1” command.

Or a command git “reset commit no.”



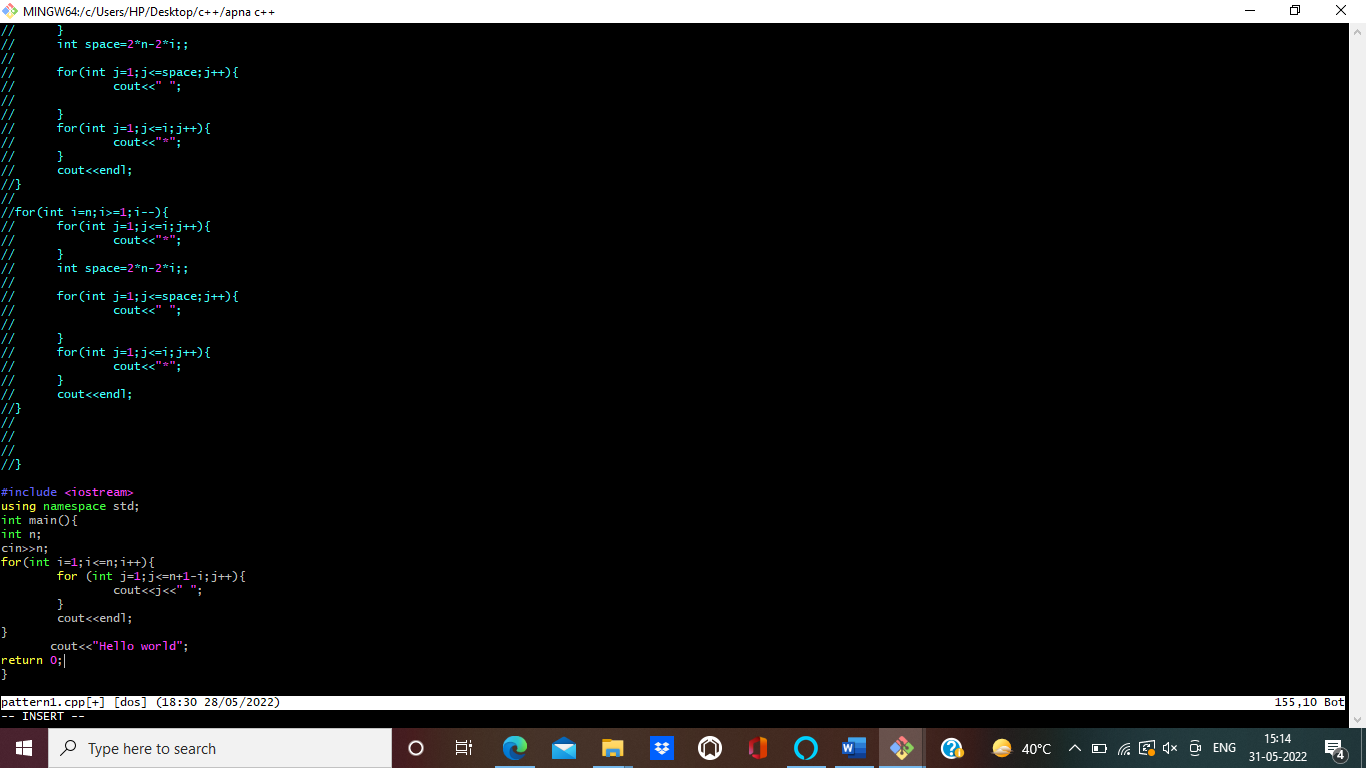
The HEAD returns to the previous commit and the changes made are reset.





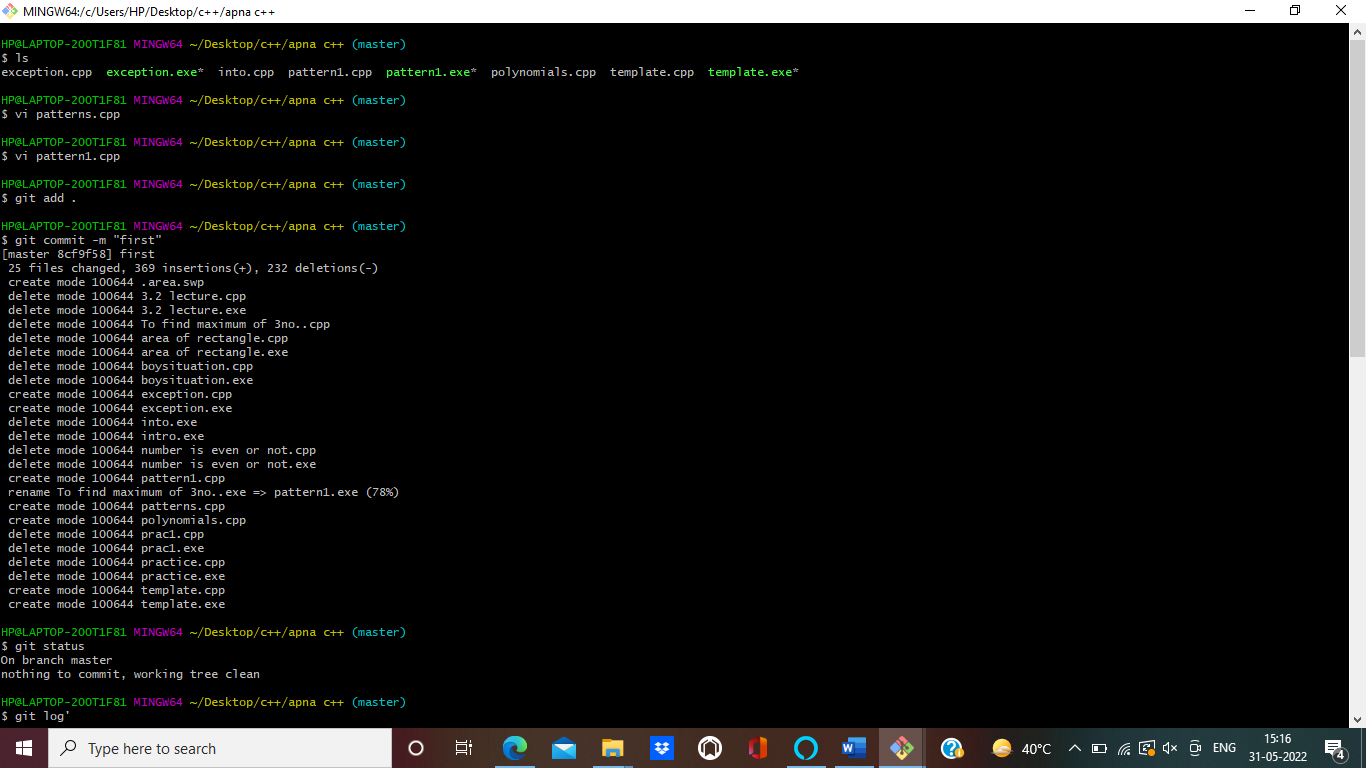
**REVERT**-

git revert is used to remove the commits from the remote repository. git revert removes the commit that we have done but adds one more commit which tells us that the revert has been done.

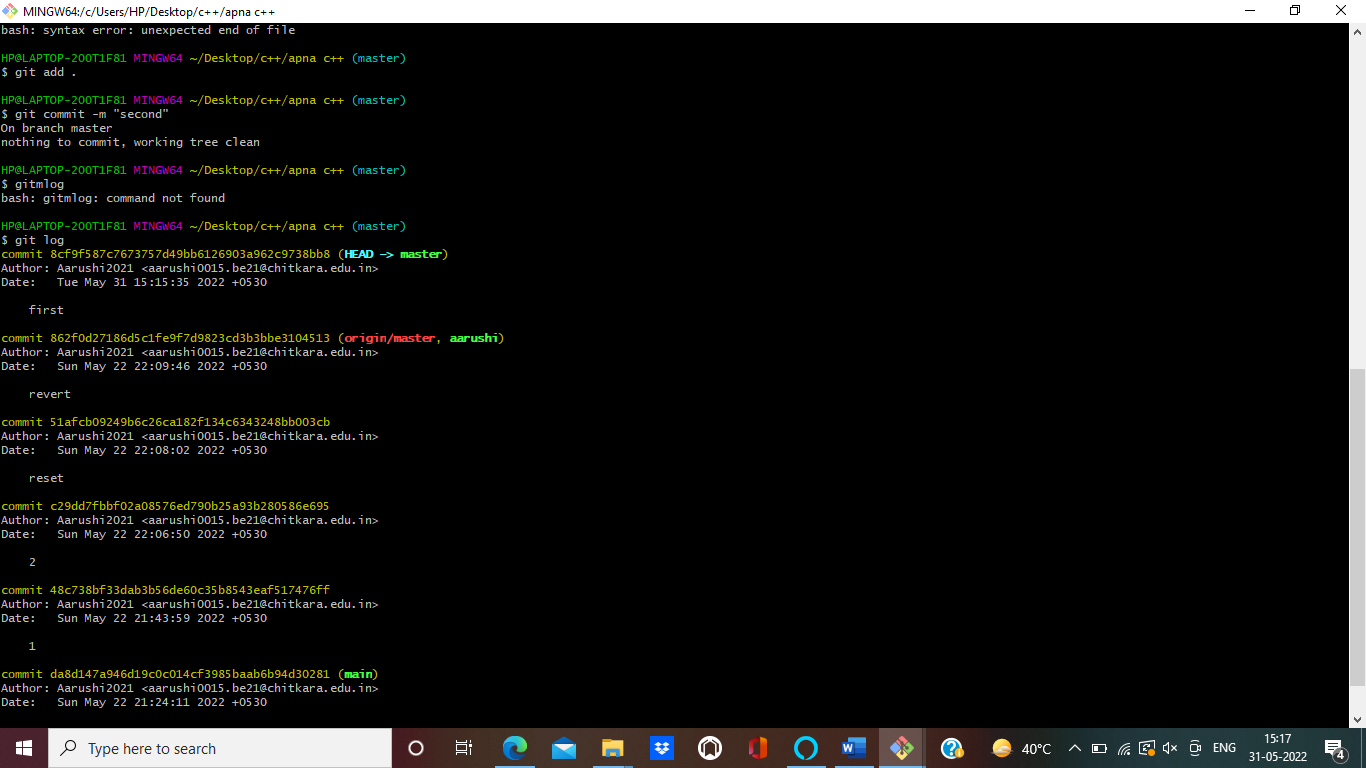


In order to understand it add changes to a file, stage and commit it.

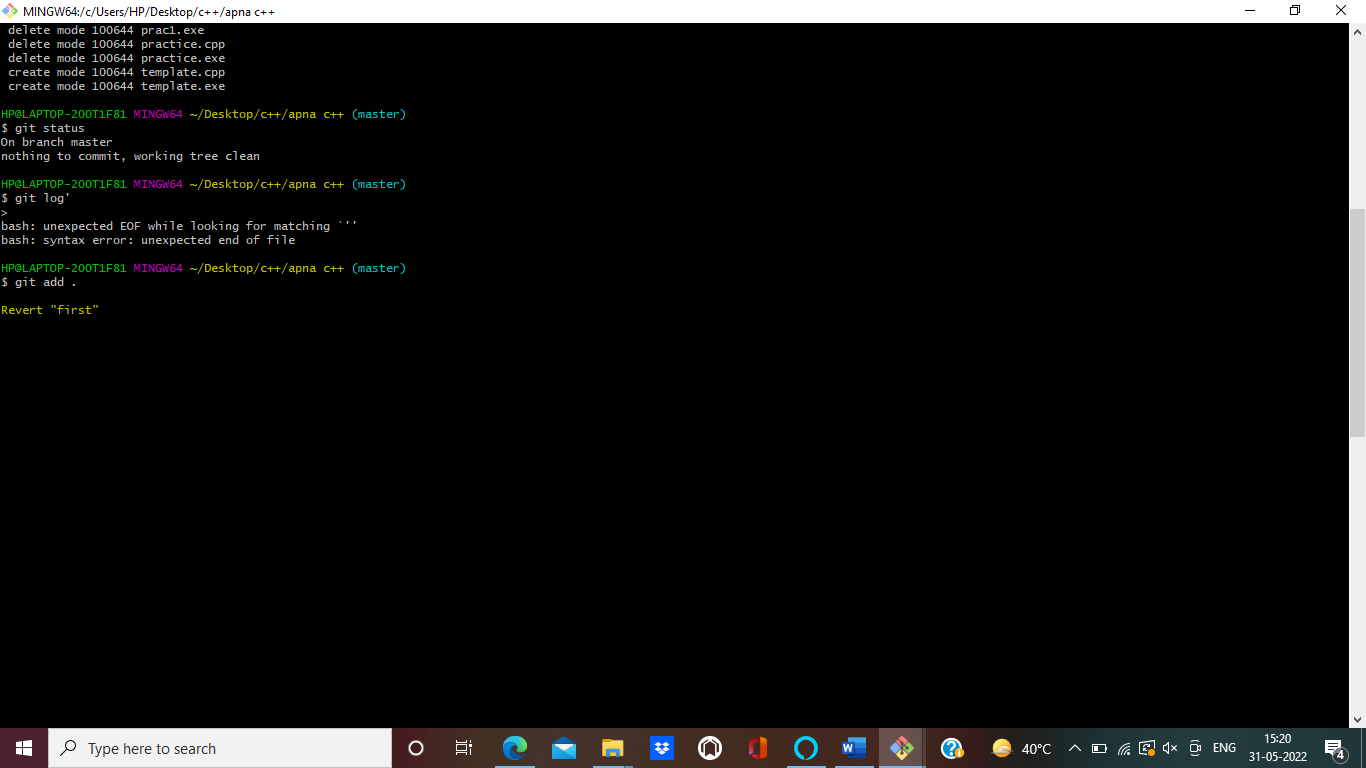


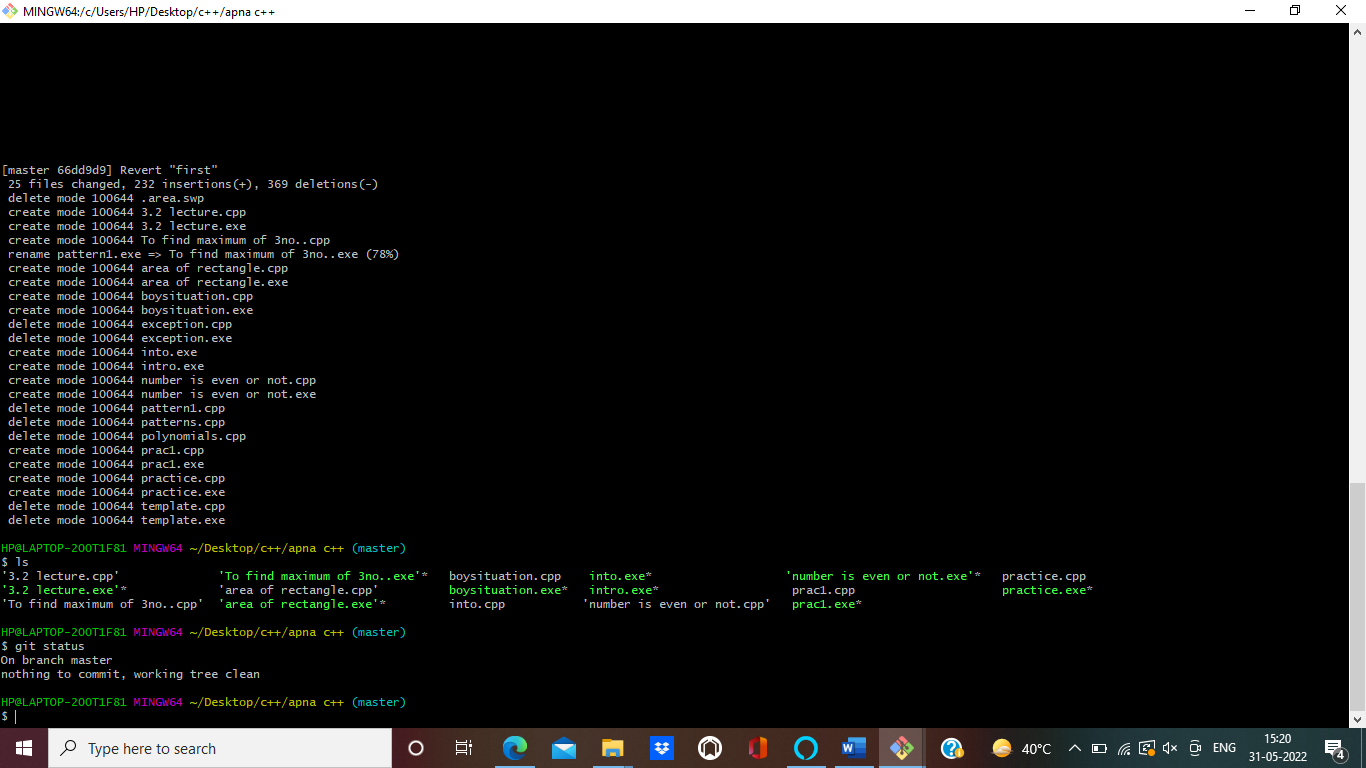






Now to revert the changes made in the commit run the “git revert <commit id>” command.





You can see that a new commit as ‘revert “changes made”’ is there and the file has returned to its previous state.