Section #1: Introduction

Version Control Systems (VCSs) allow developers to:

- Track changes and history since each modification, addition, or deletion is recorded as a commit.
- Collaborate effectively with branching, merging and conflict resolution mechanisms.

With no version control:

- Manually managing different versions of files can lead to confusion, lost changes, and accidental overwrites.
- Coordinating changes becomes cumbersome, leading to conflicts and delays.
- Code quality suffers due to lack of a systematic tracking and collaboration.

Examples of version control systems:

- Subversion (SVN).
- Mercurial.
- Team Foundation Version Control (TFVC).
- Git.

Git (pronounced like get) is a distributed VCS designed to manage source code and track changes in software projects in a decentralized manner. Each developer has a complete copy of the entire repository, including its history.

In Git, a commit represents a snapshot of the project at a specific point in time.

Developers create branches to work on specific features or fixes independently. Branches allow parallel development without affecting the main codebase. Merging then combines changes (commits) from one branch into another.

When a new Git repository (repo) is created, it comes with a first branch (main or master). This branch is called the default branch.

Pull requests are used to propose changes to a codebase.

Source: Ayodele – GitHub Foundations Certification Guide (1ED-2025)

Section #2: GitHub account and GitHub Desktop setup

1. Using a web browser go to https://github.com/. Then click on the Sign Up button.
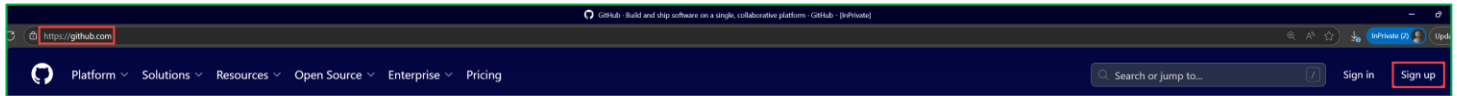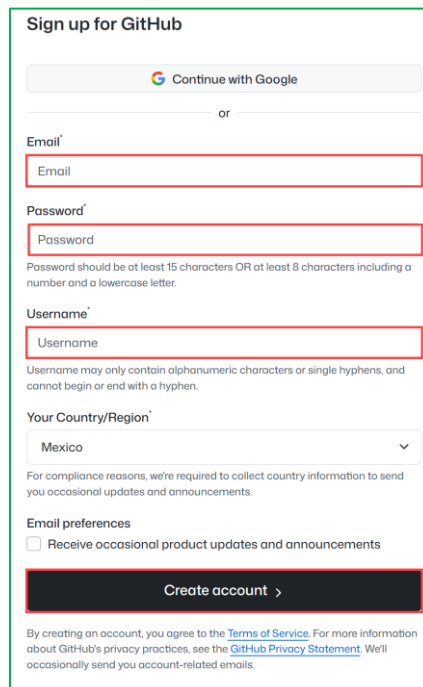

Figure 2.1. GitHub welcome page.

2. Fill out the required information and click on the Create account > button. After the account is verified, enter the code sent to the registered e-mail.

Figure 2.2. Filling out information required to create the GitHub account.

3. After entering the code, the browser will be redirected into the GitHub Sign in page. Enter your credentials and then click on the Sign in button.

Figure 2.3. Filling out information required to create the GitHub account.

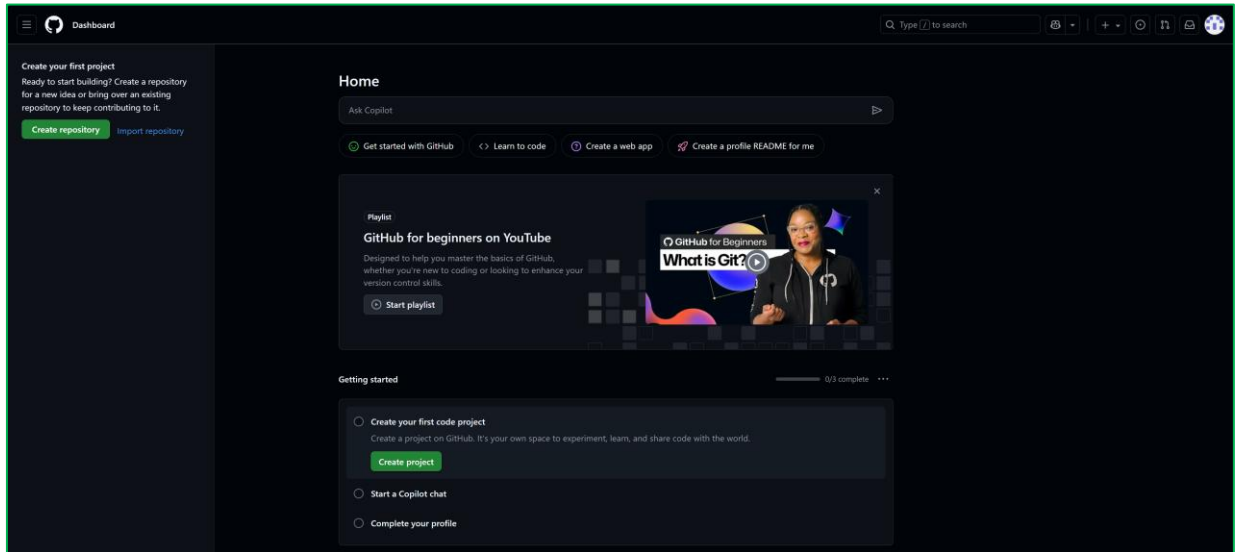4. Now the welcome screen from GitHub should be visible.

Figure 2.4. GitHub welcome screen.

5. Open the following link: https://desktop.github.com/download/. Then click on the Download for Windows (64bit) button (name depends on the actual operating system).
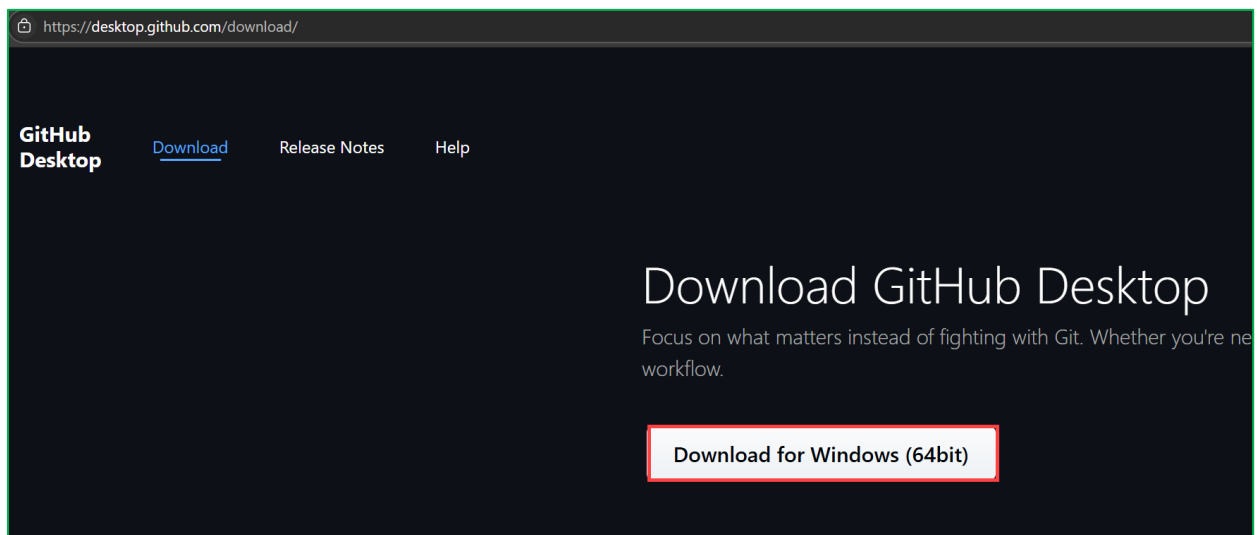

Figure 2.5. GitHub Desktop download site.

6. After the download is over, right-click on the application and select Run as Administrator. Wait until the installer finishes the setup.
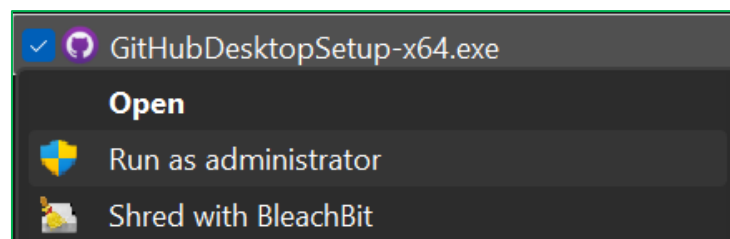

Figure 2.6. Running GitHub Desktop installer with Administrator privileges.

Section #3: GitHub basics tutorial

1. Log into your GitHub account from a web browser and click on the Repositories tab.
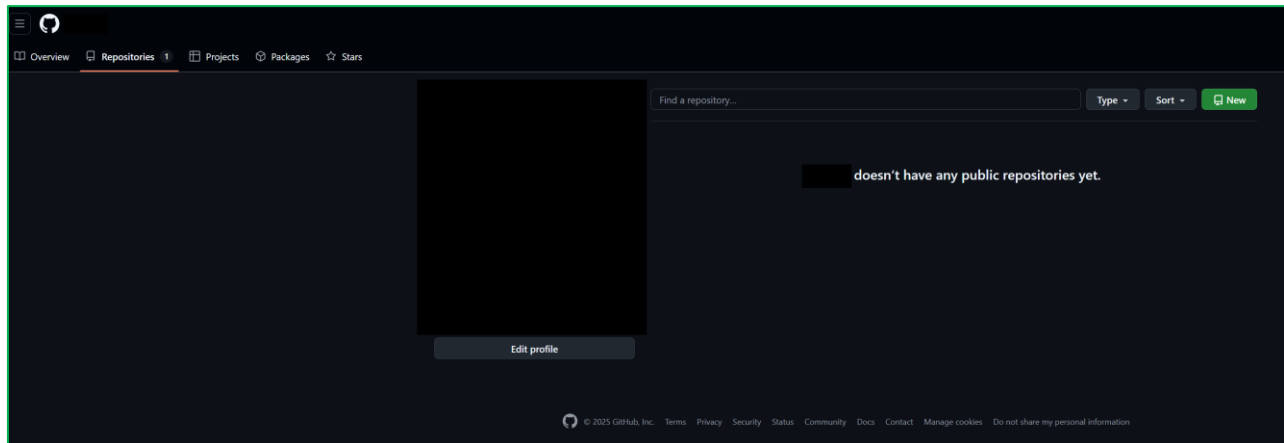

Figure 3.1. Repository tab on GitHub web interface.

2. Open GitHub Desktop and click on the Create a New Repository on your local drive… button.
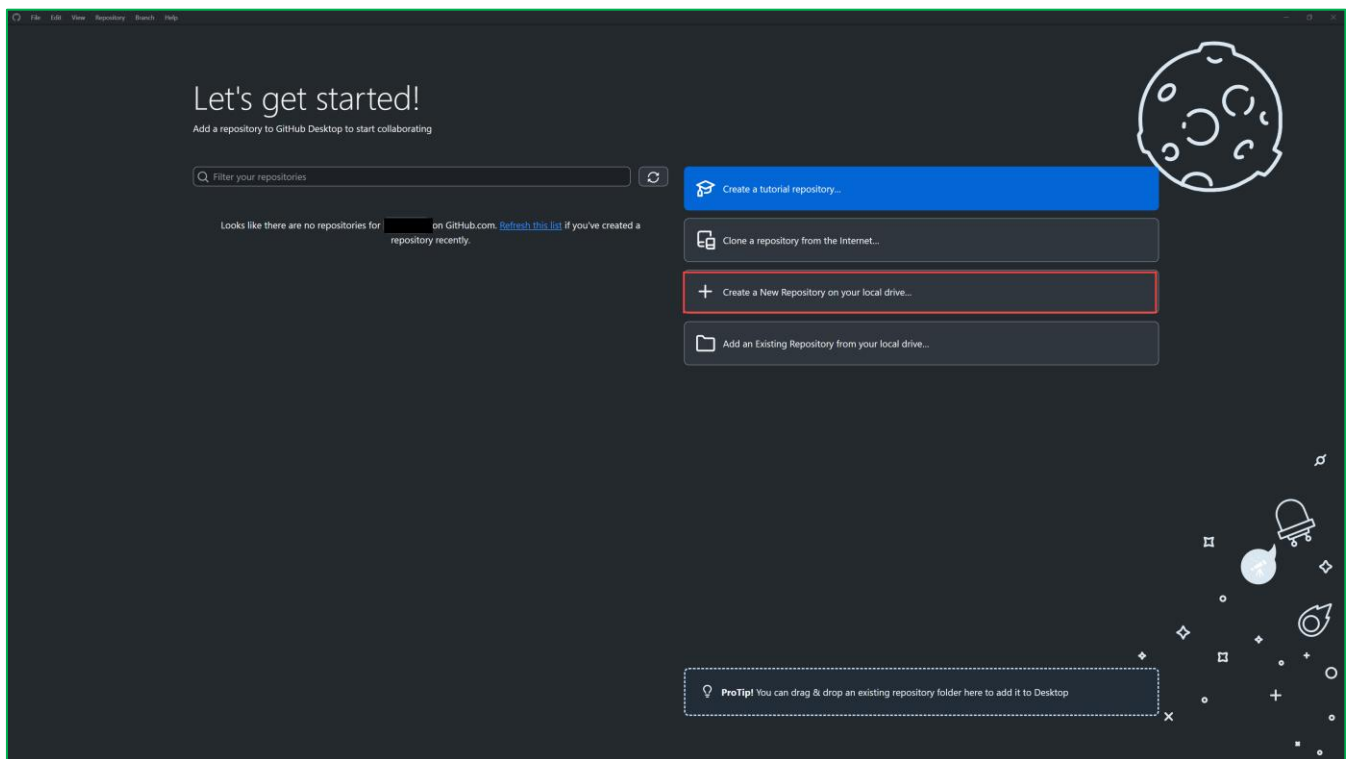

Figure 3.2. GitHub Desktop welcome window.

3. On the <u>Name</u> field enter the name of your repository and on the <u>Description</u> field a small description of it. Then click on the <u>Create repository</u> button.
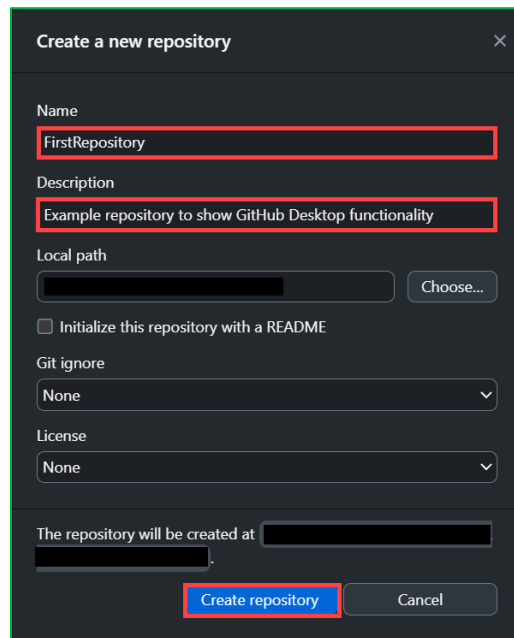

Figure 3.3. Creation of a new repository on GitHub Desktop.

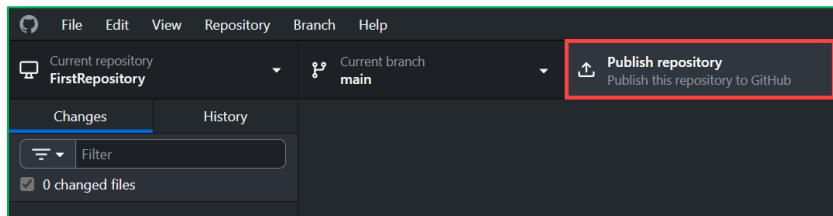4. Click on the <u>Publish repository</u> button.

Figure 3.4. Publish Repository button location on GitHub Desktop.

5. A prompt requiring access to GitHub appears. Click on the <u>Sign in</u> button.
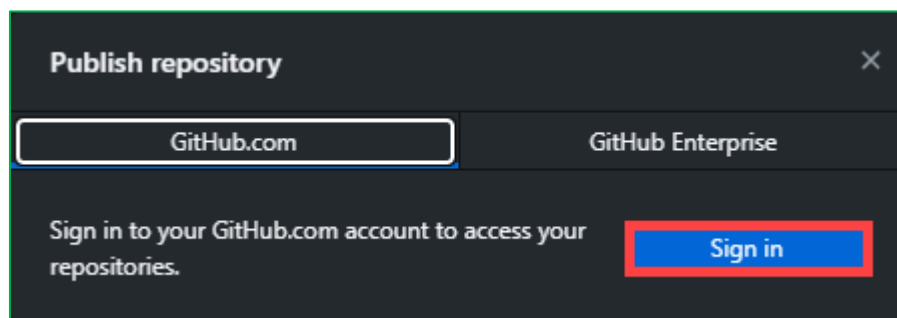

Figure 3.5. Prompt requiring GitHub credentials.

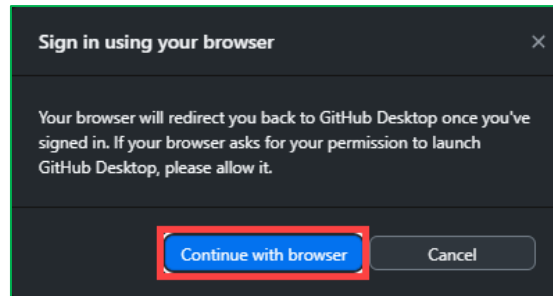6. Another prompt requiring access via web browser appears. Click on the <u>Continue with browser</u> button.

Figure 3.6. Prompt requiring access to a web browser

7. On the web browser, the server provides the sign-in information to link with <u>GitHub Desktop</u>. Check the information is correct and then click on the <u>Continue button.</u>
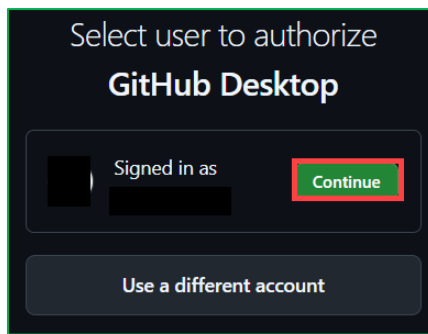

Figure 3.7. Validating sign-in information.

8. The next window displays the authorized permissions for <u>GitHub Desktop</u>. Leave as default and then click on <u>Authorize Desktop</u>.
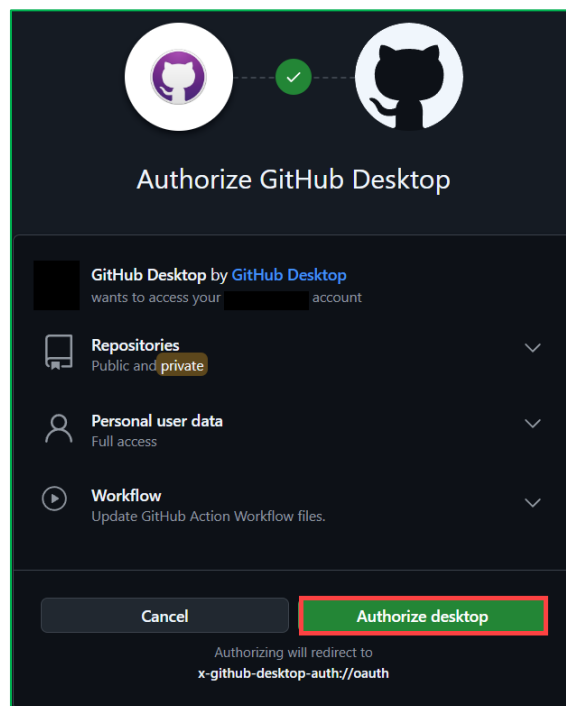

Figure 3.8. GitHub Desktop permissions.

9. The next window requests for your account's password. Enter it and then click on the <u>Confirm</u> button.
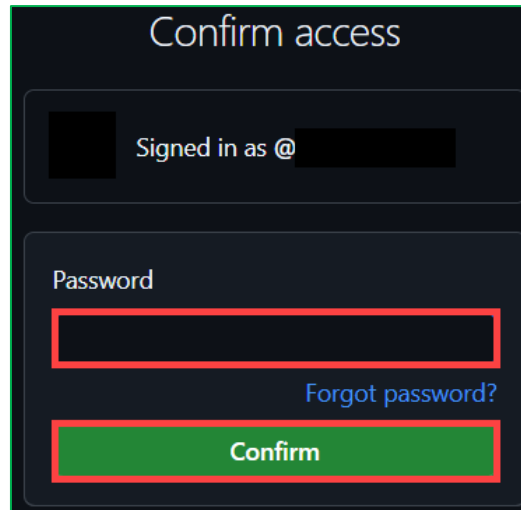

Figure 3.9. GitHub Desktop permissions.

10. Now the web browser requests to open the locally installed <u>GitHub Desktop</u>. Click on the <u>Open</u> button. With this last step, <u>GitHub Desktop</u> should have access to the remote repository.
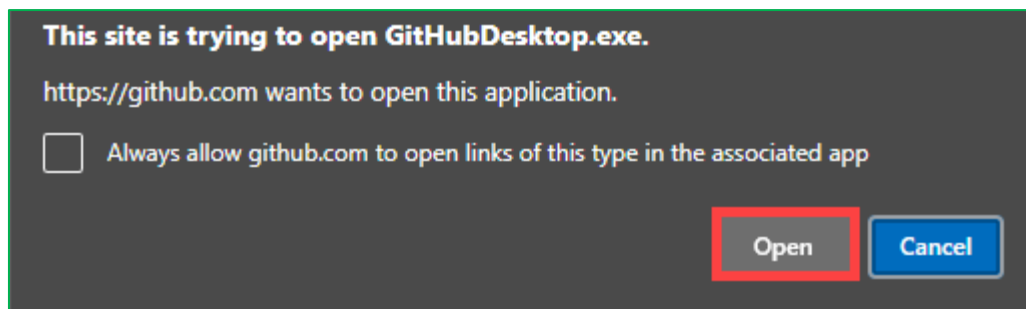

Figure 3.10. GitHub Desktop permissions.

11. Check that the <u>Keep this code private</u> box is not ticked (this will make the repository to be public).
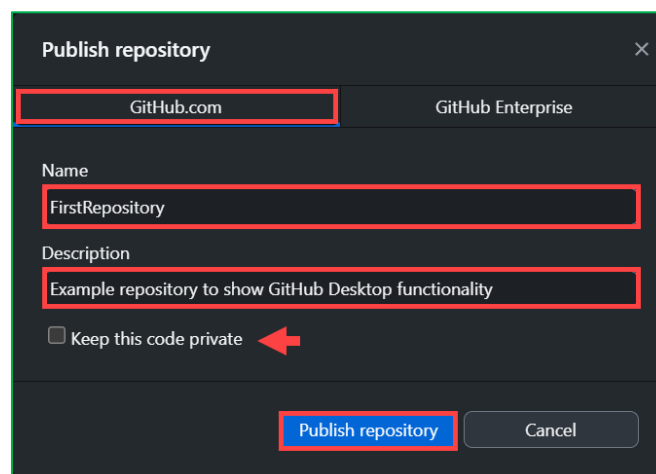

Figure 3.11. Repository publication with GitHub Desktop.

12. Refresh the web browser under the <u>Repositories</u> tab on GitHub's web interface. The new repository should be visible. Click on the name of the repository.
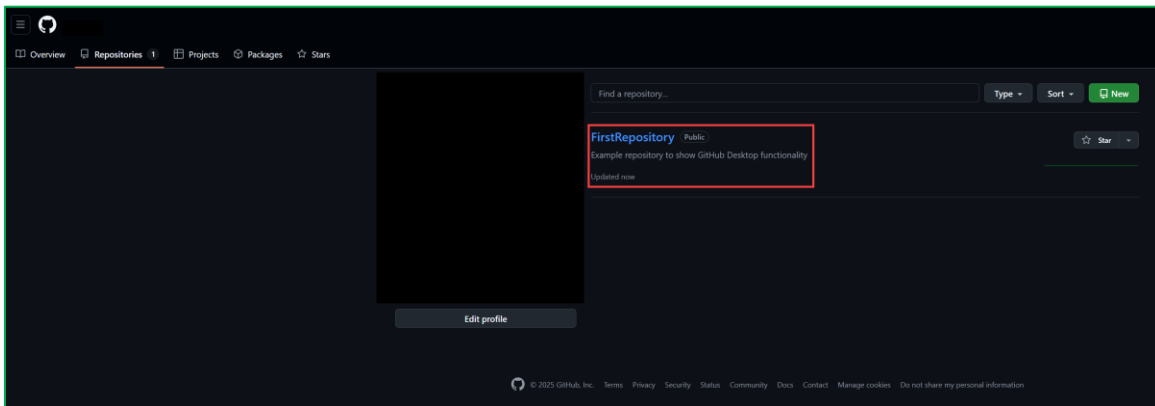

Figure 3.12. Newly created repository on GitHub's web interface.

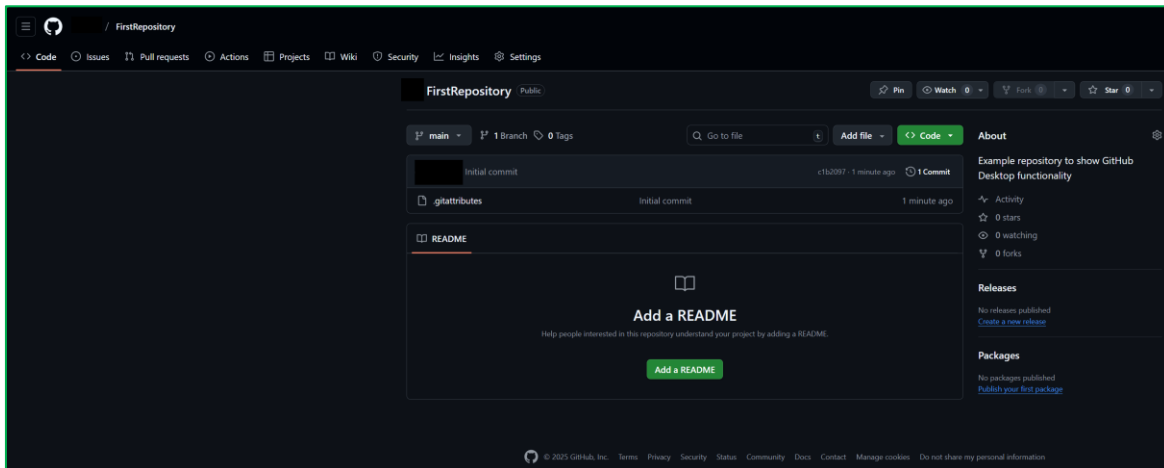13. Once on the repository, there should be no files on it yet.

Figure 3.13. Newly created repository content visualization on GitHub's web interface.

14. On <u>GitHub Desktop</u> right-click in the repository name and select the <u>Show in Explorer</u> option.
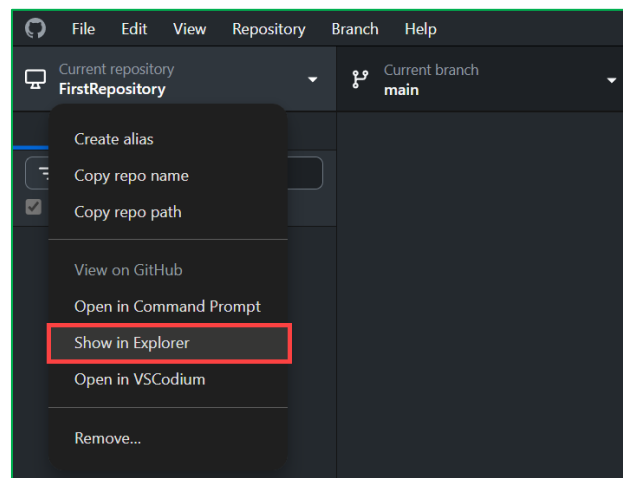

Figure 3.14. Opening the local folder linked to the repository.

15. Under the file explorer, create a text file named Alphabet.txt. Add the characters from A to Z within the file and save it.
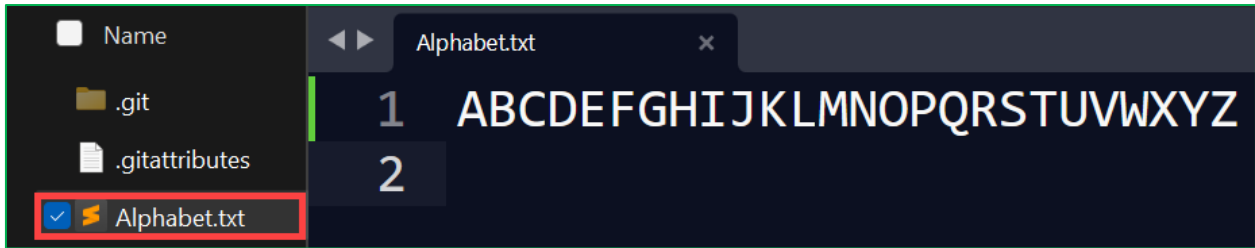

Figure 3.15. Adding a first file into the local folder linked to the repository.

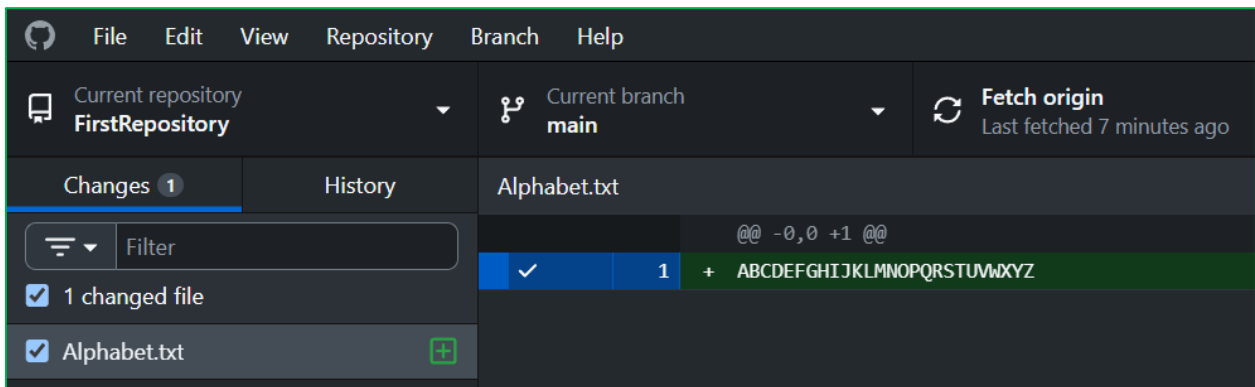16. On GitHub Desktop the newly created file should be displayed.

Figure 3.16. Detection of new local file into the repository.

17. Before uploading the changes into our repository, first we need to provide a commit message. The interface to enter such information is located in the lower left corner of GitHub Desktop. Fill out the message in Figure 2.11 and then click on the Commit 1 file to main button.
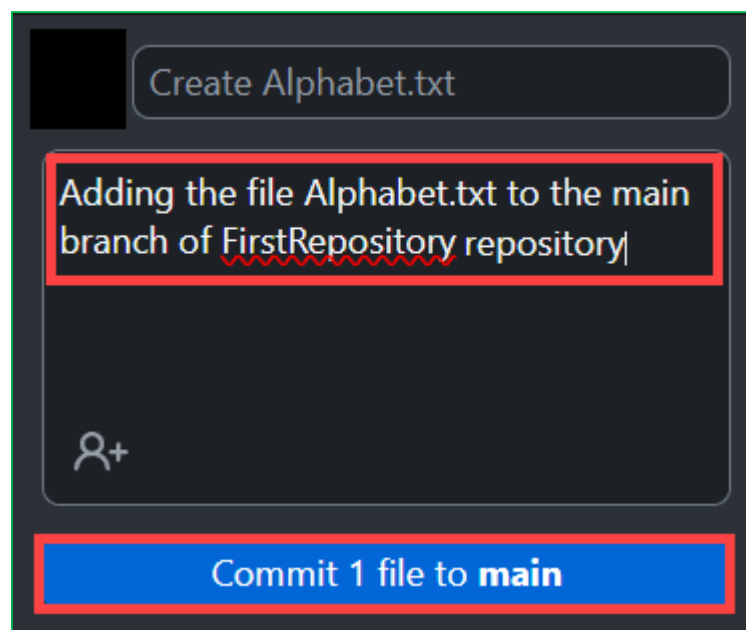

Figure 3.17. Adding a first local file to add into the repository.

18. Once the commit is done, click on the Push origin button. Only after this step (and not before) our local file will be pushed (uploaded) into the repository.
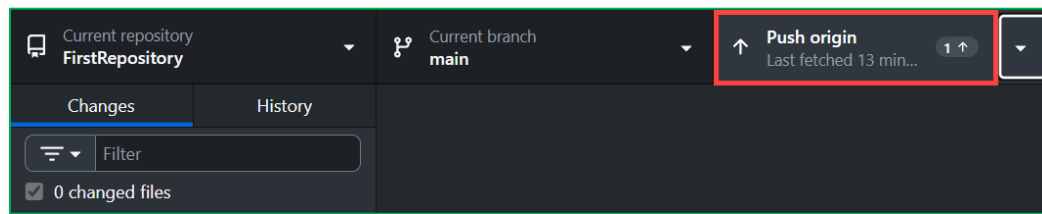

Figure 3.18. Adding a first local file to add into the repository.

19. After refreshing the web interface, the file Alphabet.txt should appear as part of the repository's content.
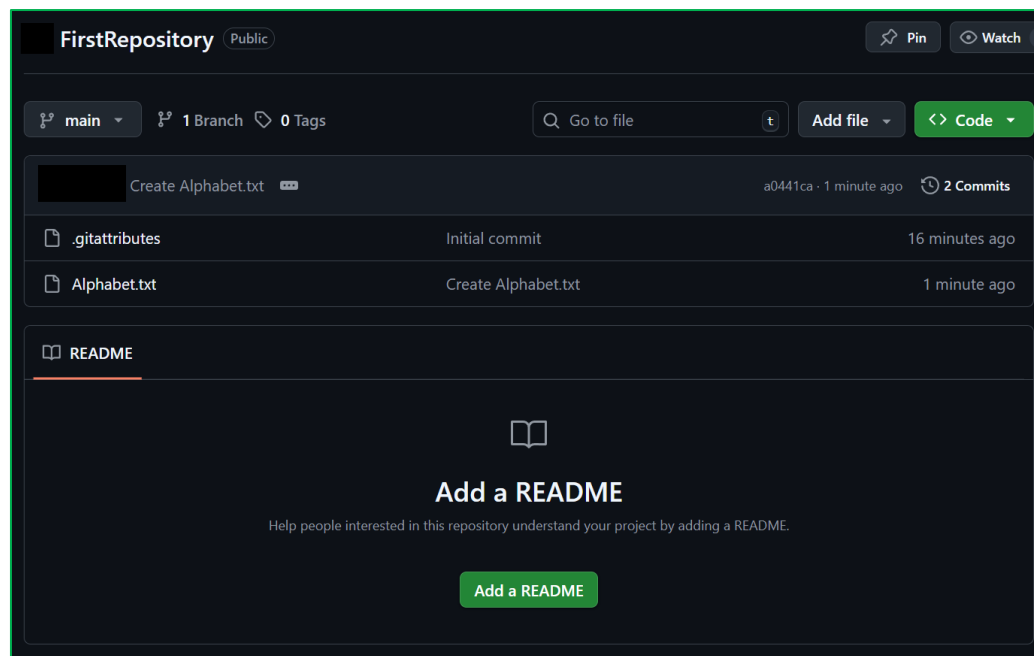

Figure 3.19. Local file pushed into repository.

20. Git has the capability of creating different branches for a single repository. This means that new changes can be tracked and added into a repository in a new branch while having a backup of a version prior to the addition of those changes. In order to add a new branch under GitHub Desktop, click on the Current branch button and the on the New branch button.
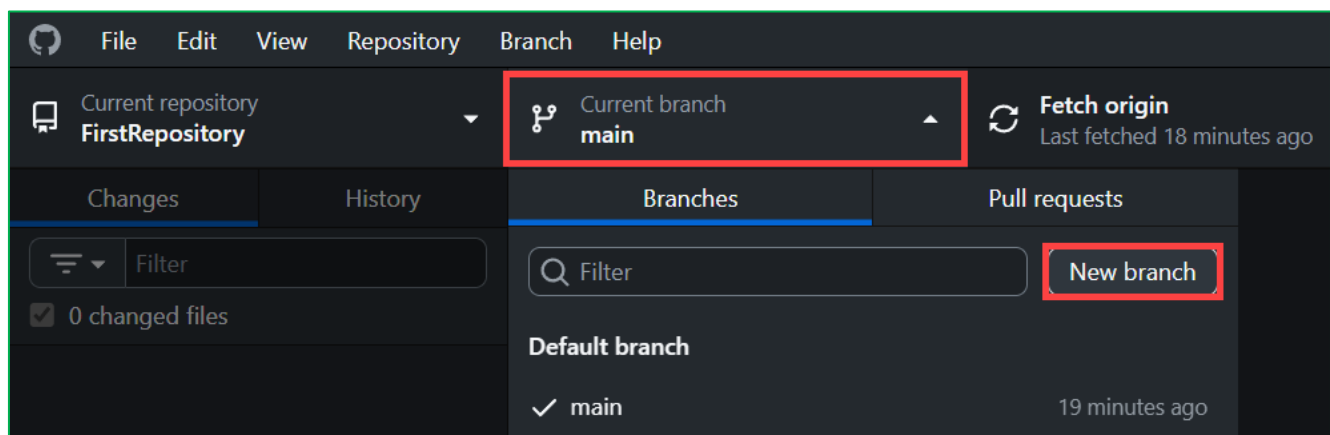

Figure 3.20. Creating a new branch in GitHub Desktop.

21. Enter <u>development</u> as the name of the new branch and then click on the <u>Create branch</u> button.
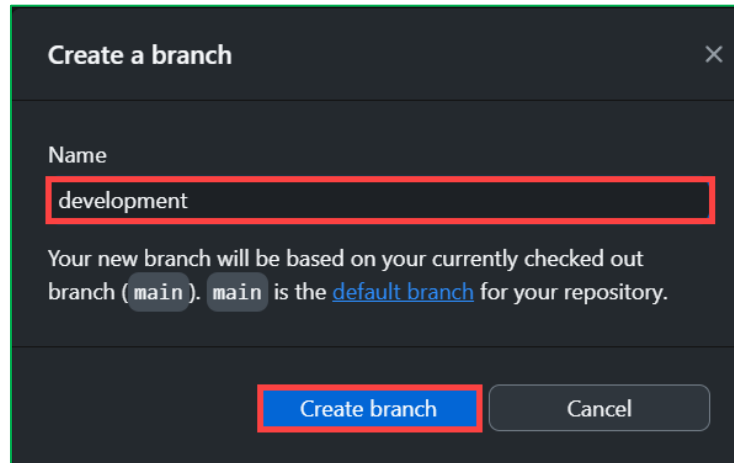

Figure 3.21. Enter the new branch information in GitHub Desktop.

22. Finally, click on the <u>Publish branch</u> button to create the branch in the actual repository.
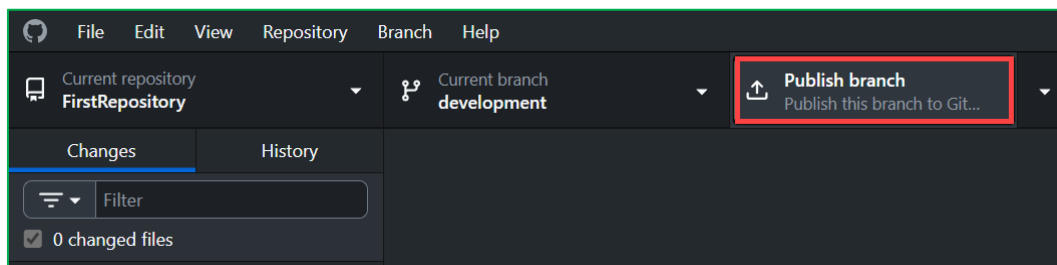

Figure 3.22. Publishing the new branch into repository.

23. Refresh the web browser that is connected to GitHub. Click on the <u>main</u> button and then select the <u>development</u> branch in order to switch to it.
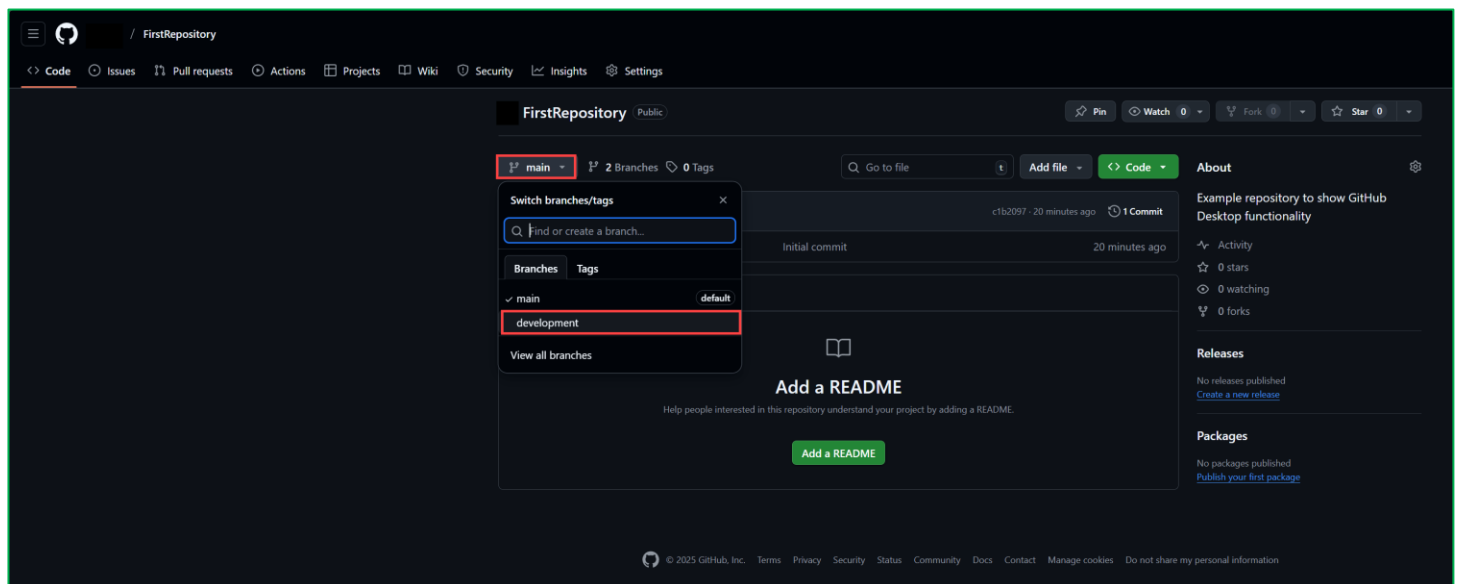

Figure 3.23. Switching to development branch.

24. A new file will be added into the recently created branch (while leaving as-is the main branch for now). In GitHub Desktop right-click again on the repository's name and then click on Show in Explorer.
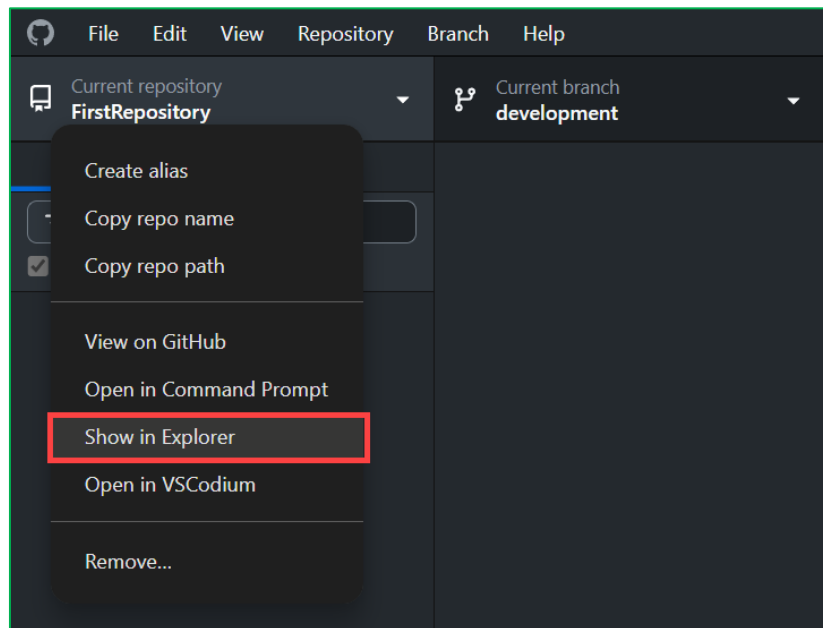

Figure 3.24. Opening the local folder linked to the repository.

25. Under the file explorer, create a text file named Consonants.txt. Add the characters from A to Z (but exclude A, E, I, O, U) within the file and save it.
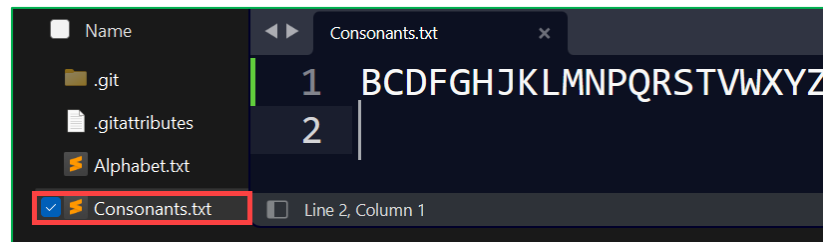

Figure 3.25. Adding a local file to be included into the new branch.

26. Once again provide the commit information (lower left corner on GitHub Desktop) and click on Commit 1 file to development button.
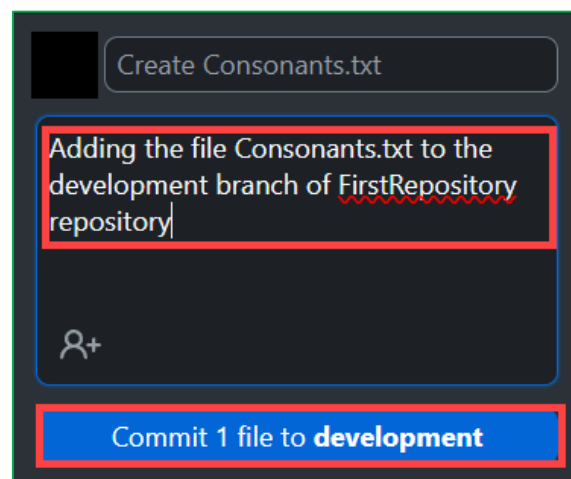

Figure 3.26. Providing commit information for Consonants.txt.

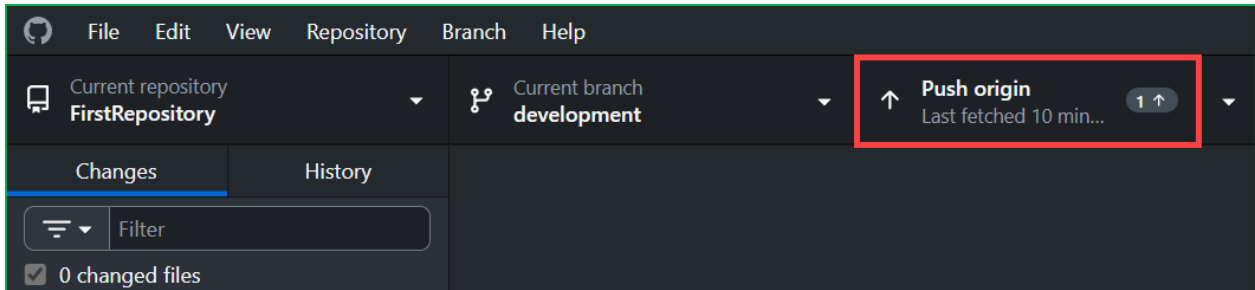27. Once the commit is done, click on the Push origin button.

Figure 3.27. Pushing the newly created file into development branch.

28. In GitHub's web interface, follow Insights > Network. The Network graph on the right shows the history of commits done to each branch of the repository.
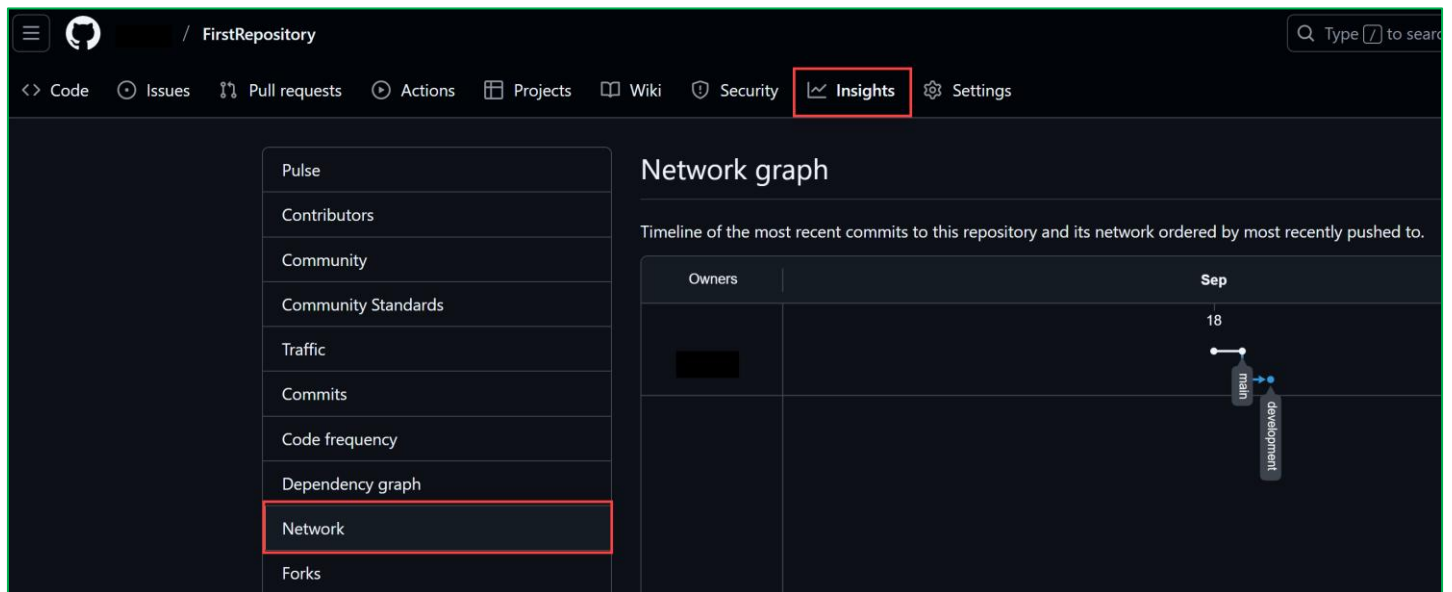

Figure 3.28. Repository's Network graph.

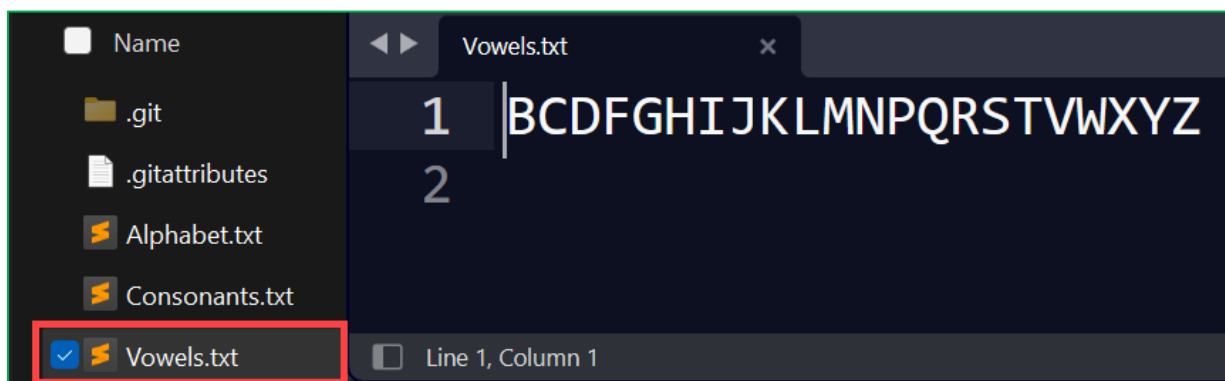29. Create a new local file named Vowels.txt.

Figure 3.29. Creating a third file into the local folder linked to the repository.

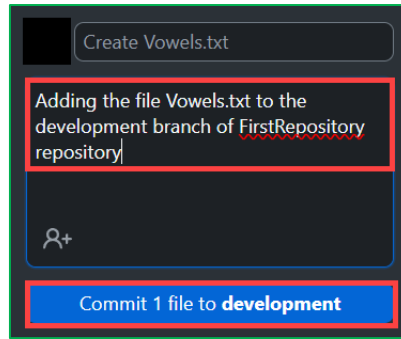30. Add the commit information for the new local file.

Figure 3.30. Providing commit information for Vowels.txt.

31. Switch again to the <u>Network graph</u> in the web interface. The new commit should be displayed.
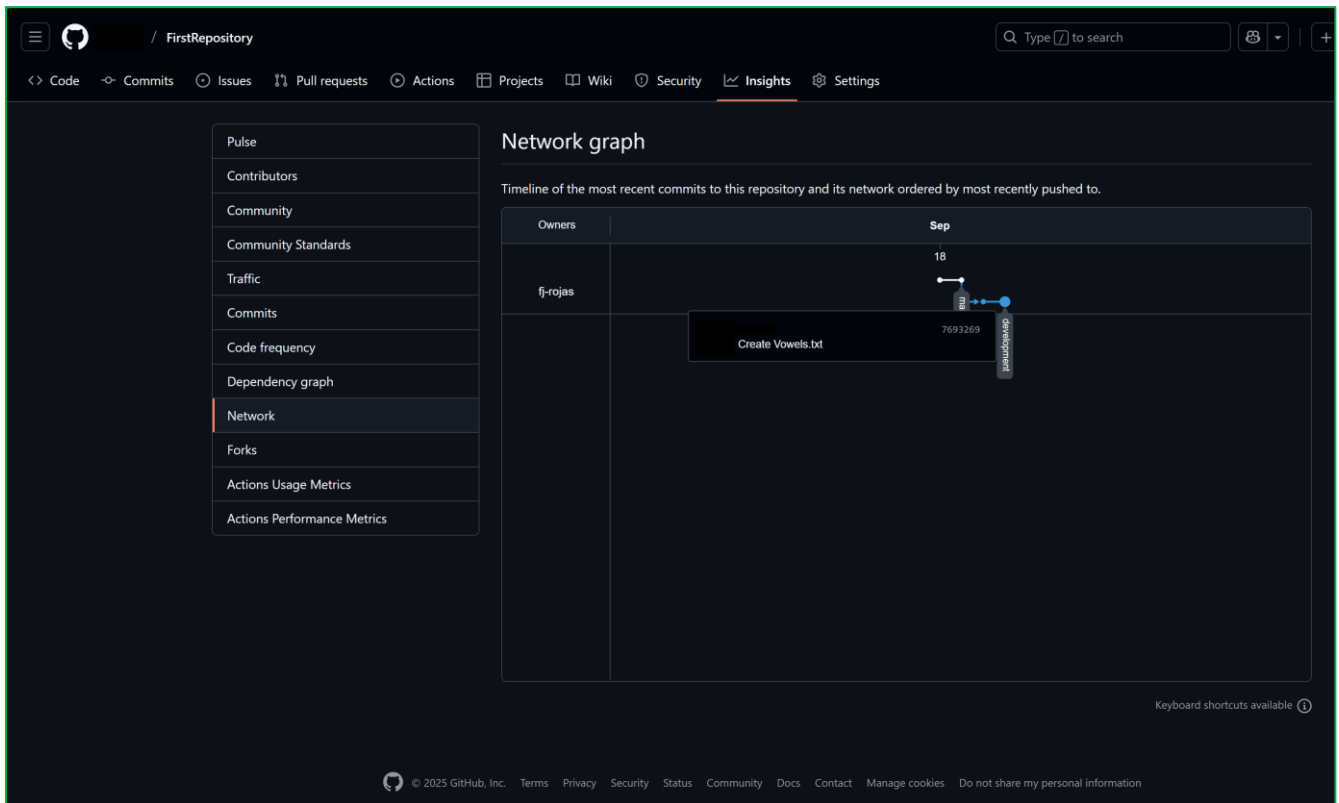

Figure 3.31. Network graph displaying the addition of Vowels.txt.

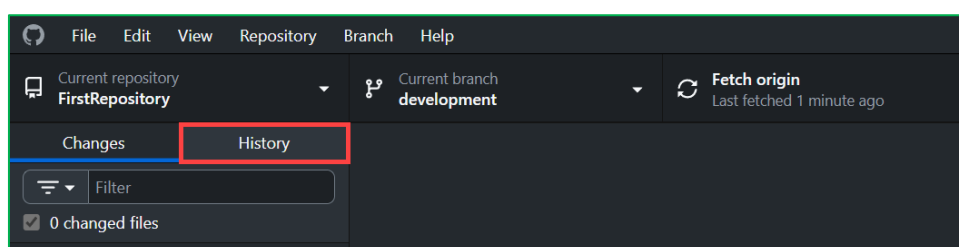32. On <u>GitHub Desktop</u> click on the <u>History</u> tab.

Figure 3.32. Location of History tab on GitHub desktop.

33. The commit history should be displayed. Notice that the Vowels.txt file contains only consonants. Will revert such change and then add the correct file by right-clicking on the problematic commit (the latest one) and then selecting the Revert changes in commit option.
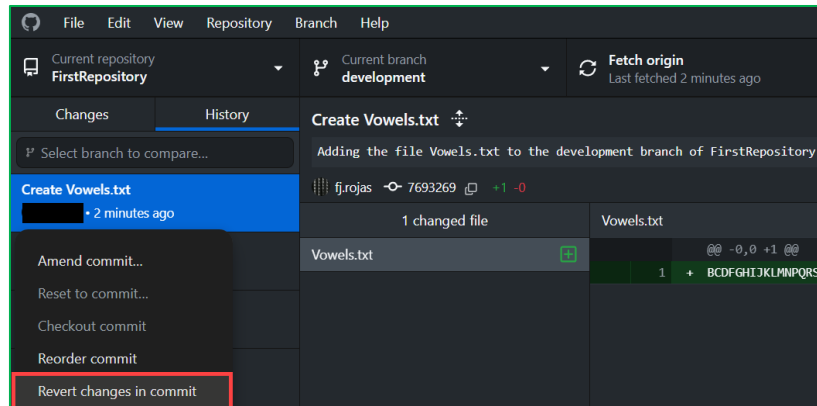

Figure 3.33. Reverting the addition of the Vowels.txt file

34. To actually apply the change in the repository, click on the Push origin button.
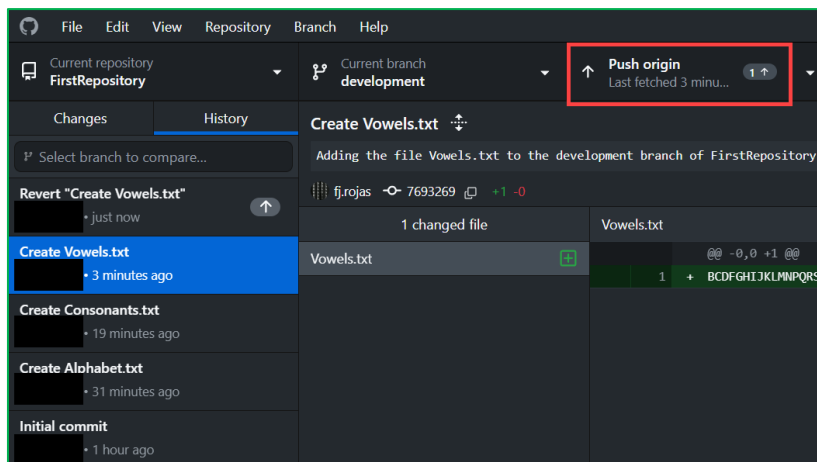

Figure 3.34. Pushing the reversion request.

35. Switch again to the Network graph in the web interface. The new reversion commit now appears in the graph.
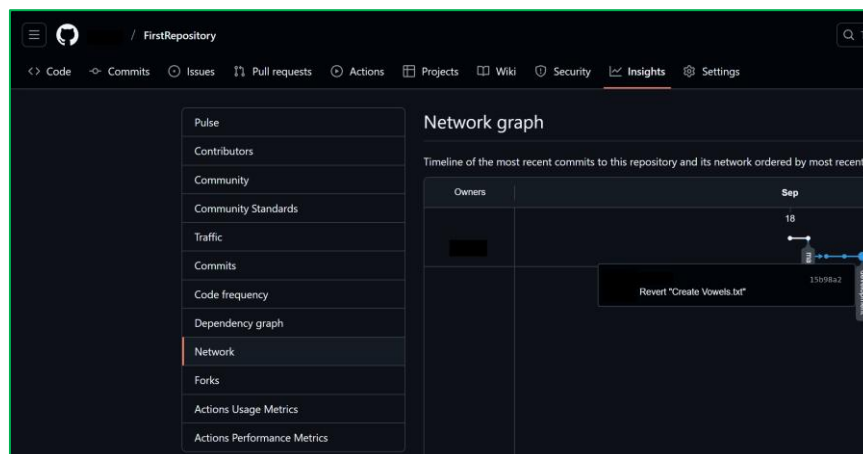

Figure 3.35. Reversion request on the network graph.

36. Now open the Vowels.txt, replace the consonants in the file with vowels. Then commit the change and push it into the repository.
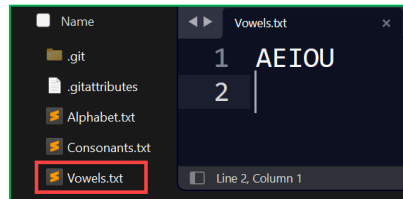

Figure 3.36. Reversion request on the network graph.

37. In the web interface, refresh the Network graph. This last commit with the correct file should be displayed.
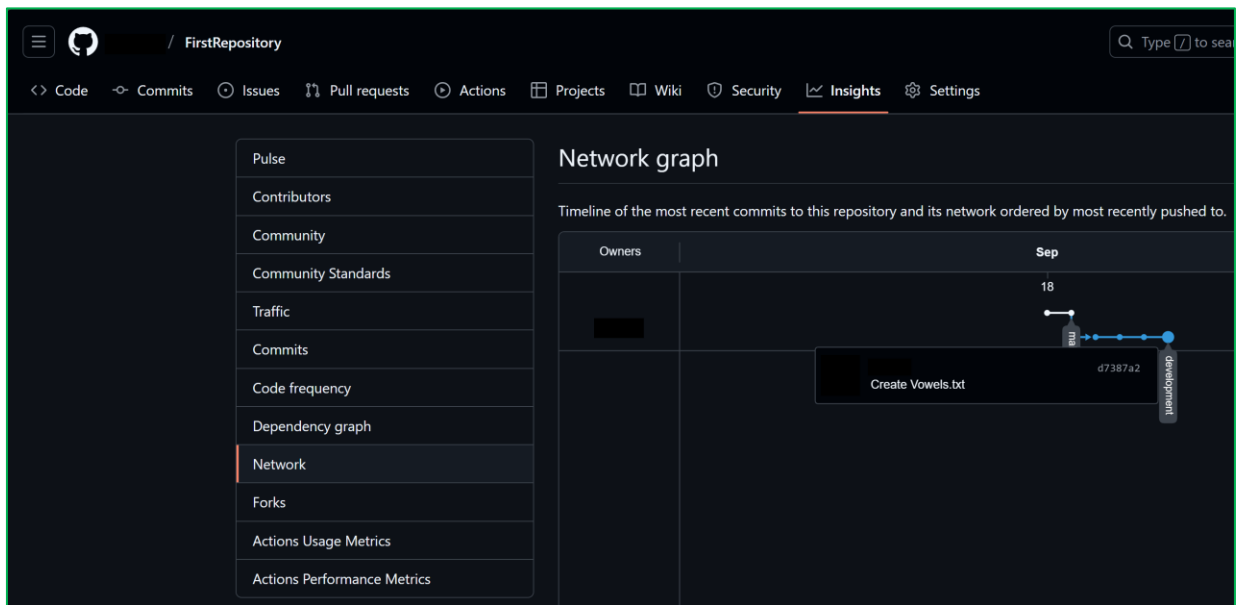

Figure 3.37. Reversion request on the network graph.

38. No more updates will be done on this branch so we will create a pull request in order to include these changes into the main branch. On the web interface go to the Pull requests tab and then click on the New pull request button.
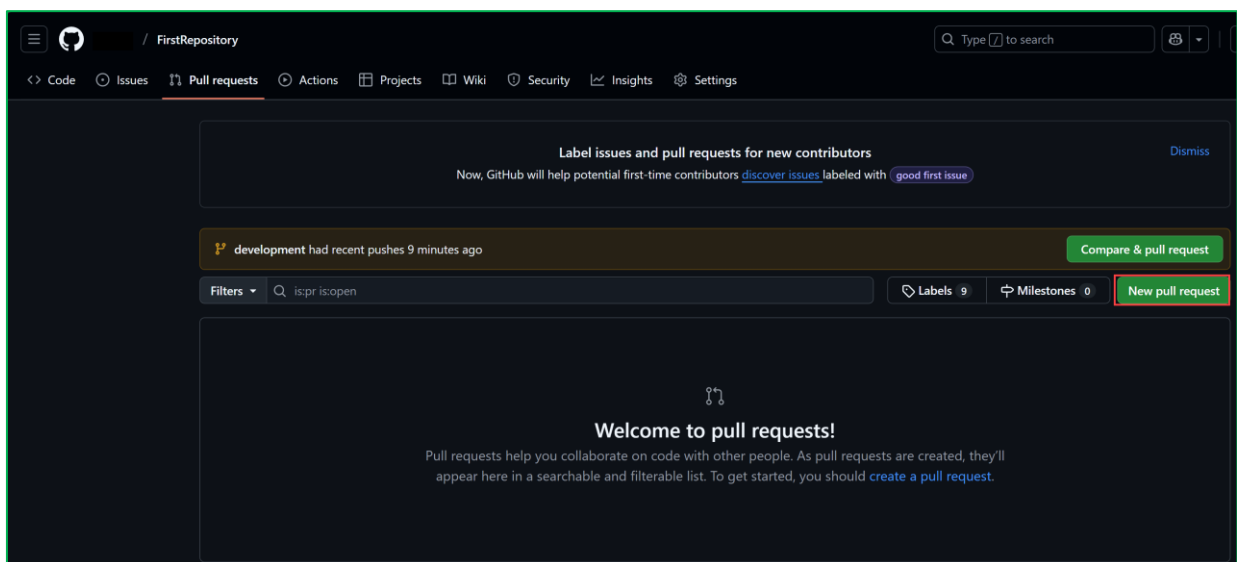

Figure 3.38. Pull requests tab.

39. On the next window, select the development branch in the compare field. This will compare the main branch against the development branch and then highlight the differences between both of them.
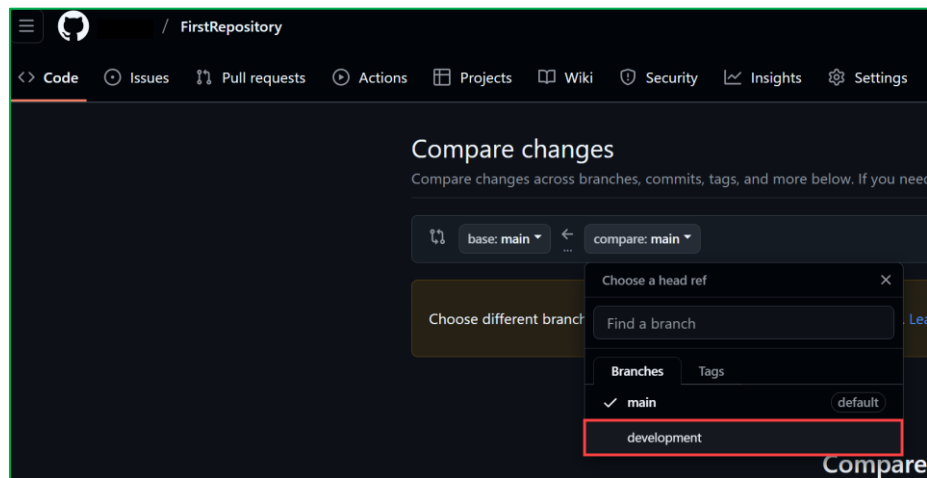

Figure 3.39. Selecting branches to compare.

40. Take a look on the changes displayed and verify they are alright. Then click on the Create pull request button.
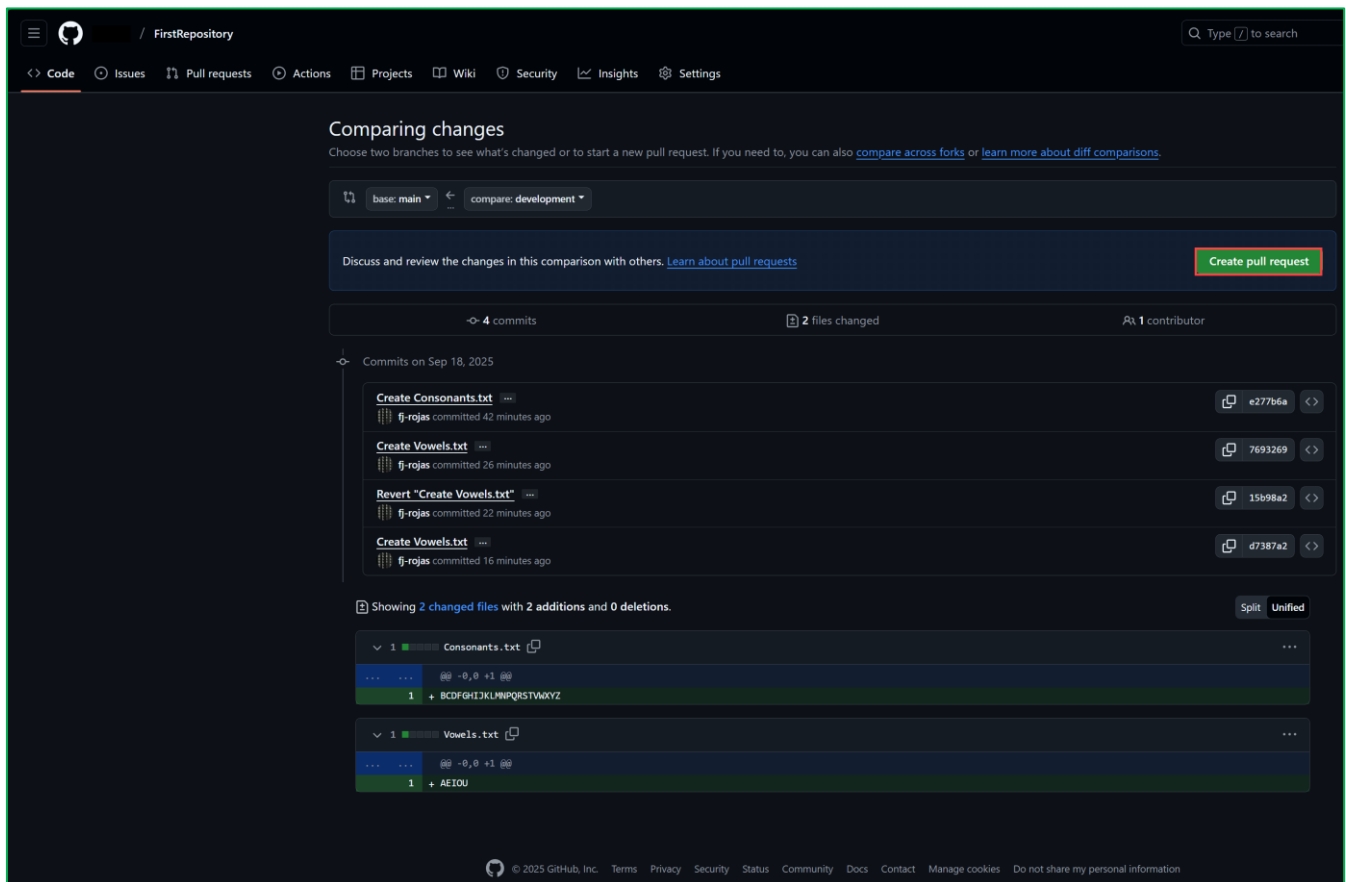

Figure 3.40. Comparing main and development branches contents.

41. On the next window, add a title to the pull request and a description to it. Then click on the <u>Create pull request</u> button.
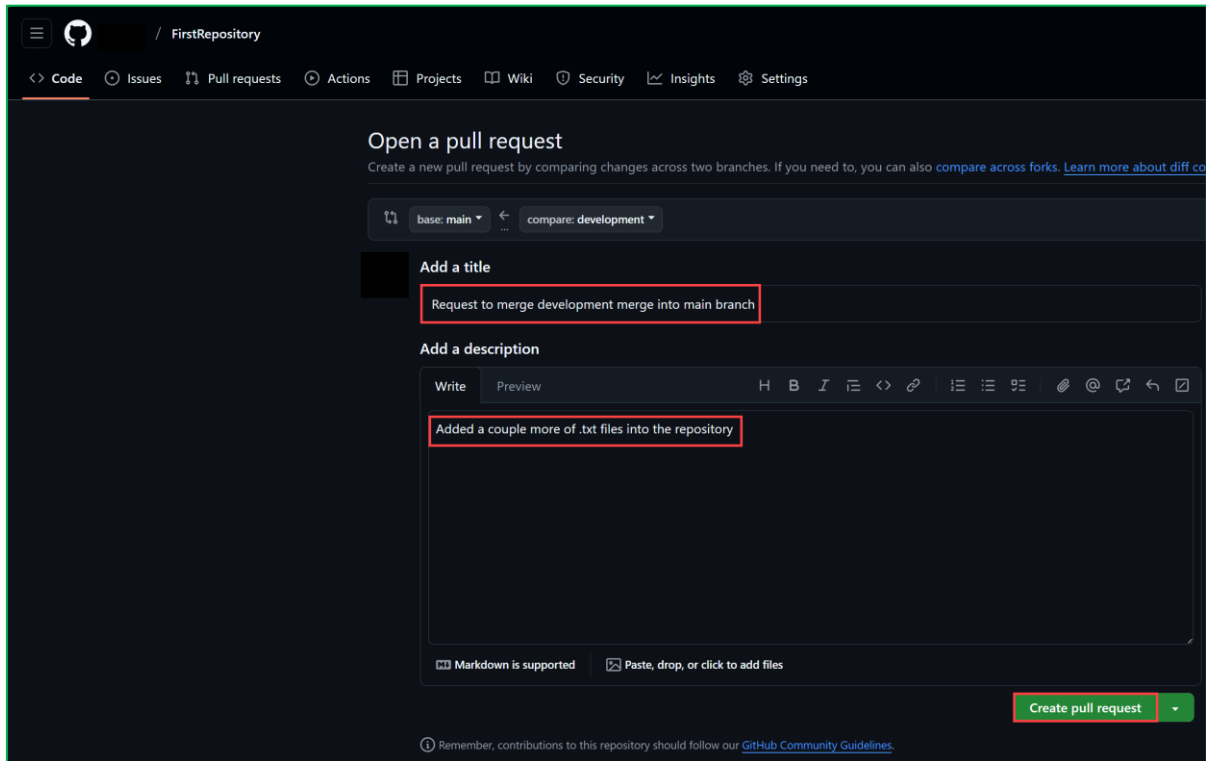

Figure 3.41. Filling out pull request information.

42. One more time, the commit history on the <u>development</u> branch to be merged will appear on-screen. Click on the <u>Merge pull request</u> button.
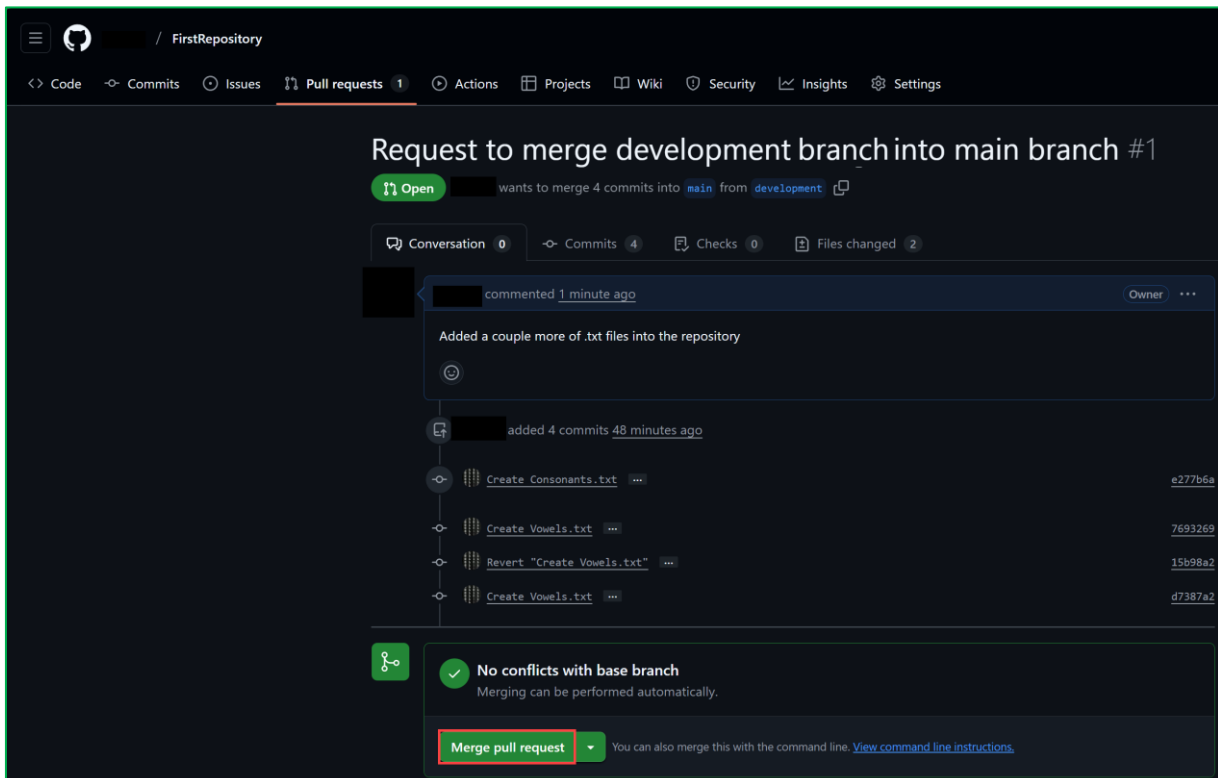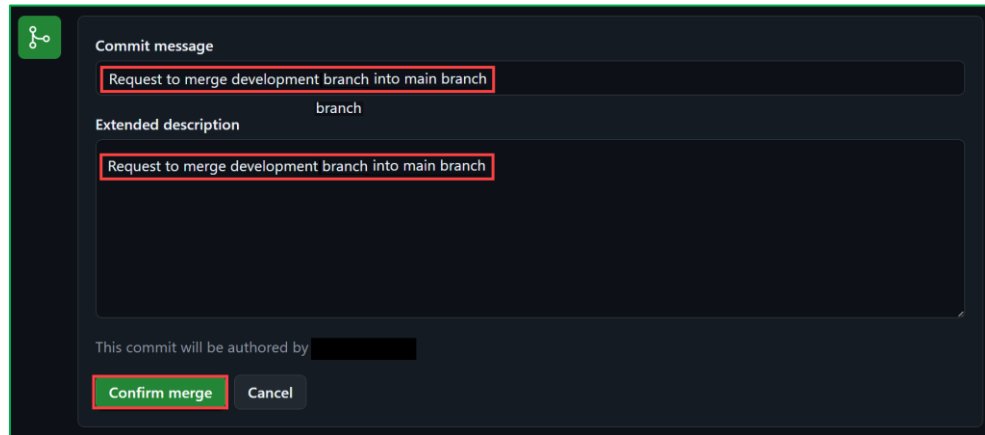

Figure 3.42. Requesting to merge current pull request.

43. Fill out the <u>merge commit</u> information and then click on the <u>Confirm merge</u> button.

Figure 3.43. Merging into the main branch.

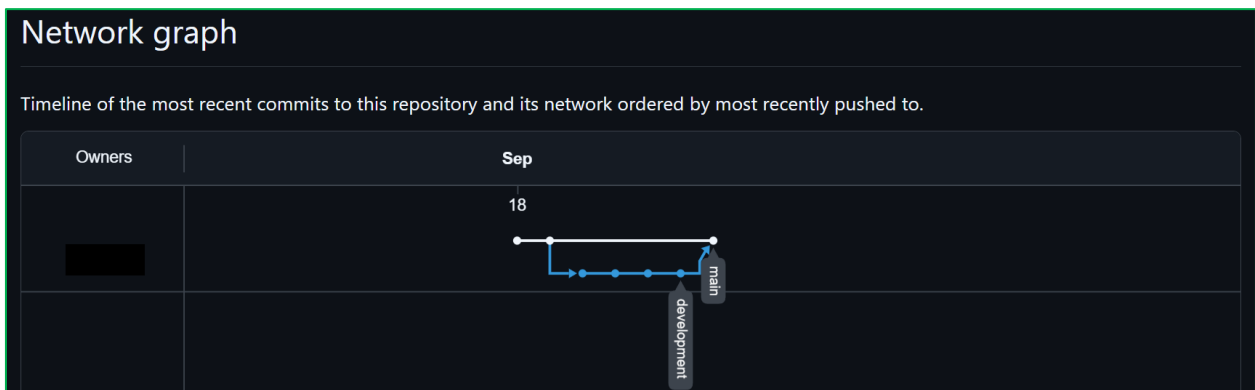44. Switch again to the <u>Network graph</u>. Notice that the <u>development</u> branch now "jumps" into the <u>main</u> one.

Figure 3.44. Final network graph.