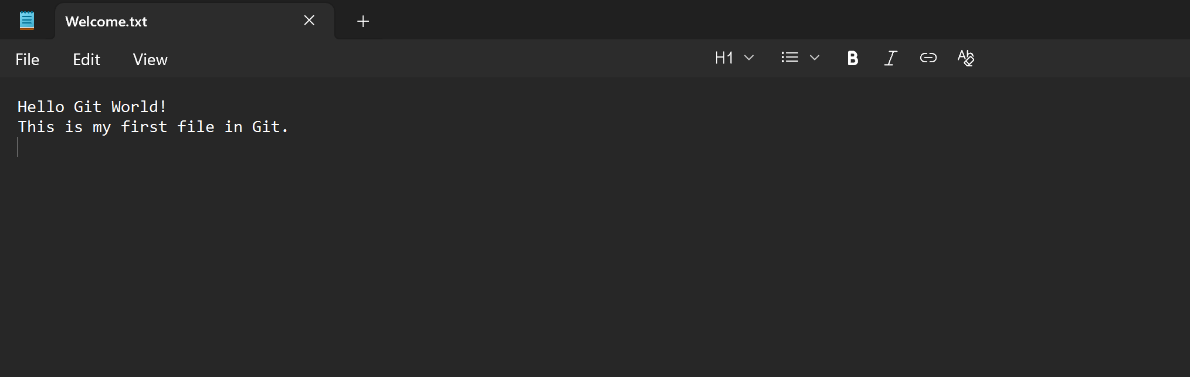
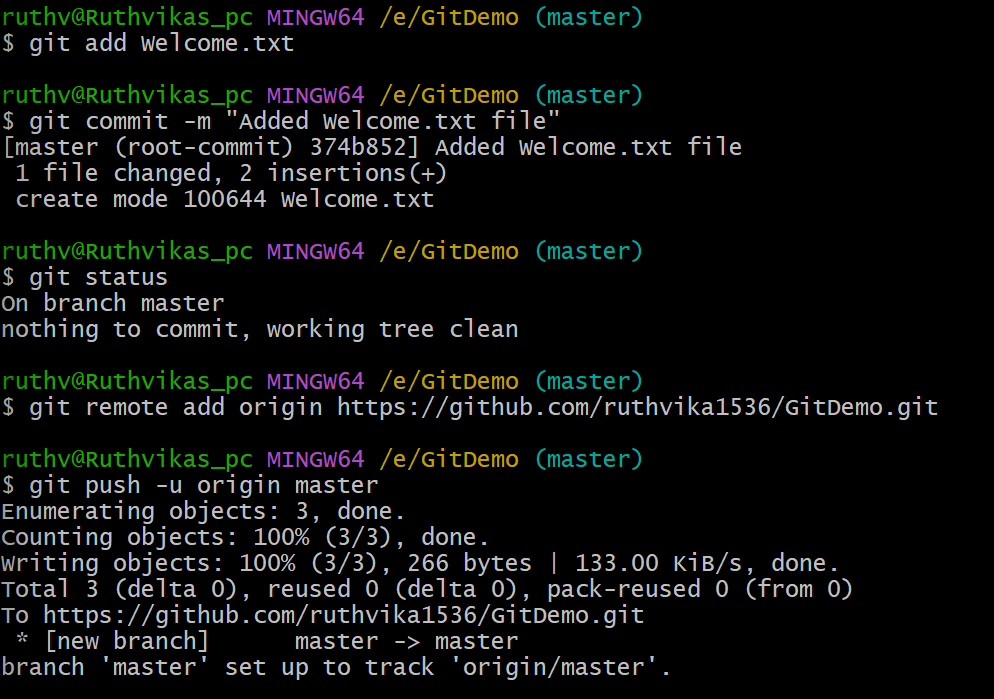
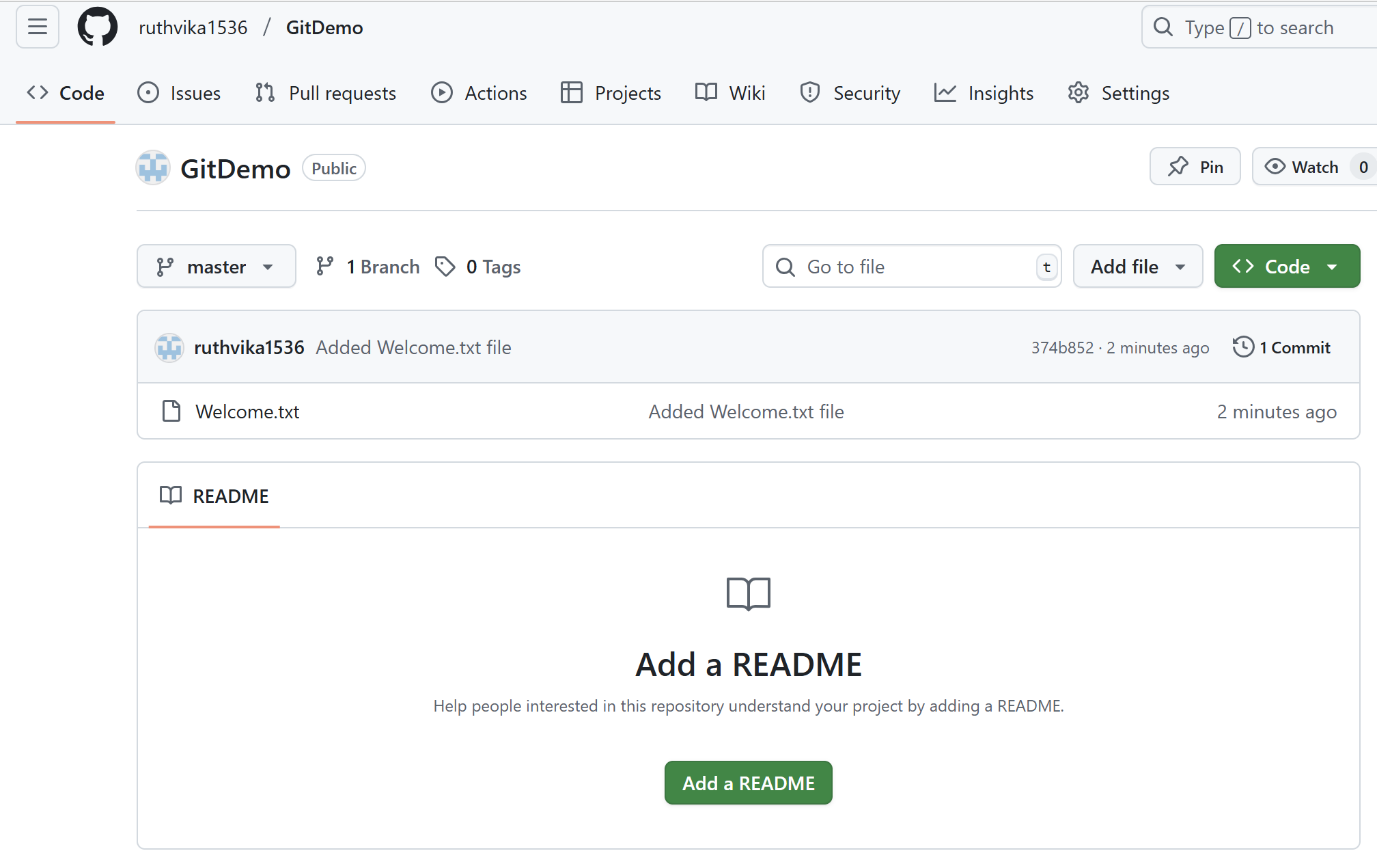
**Week-8**

**React**

**Exercise 1:**

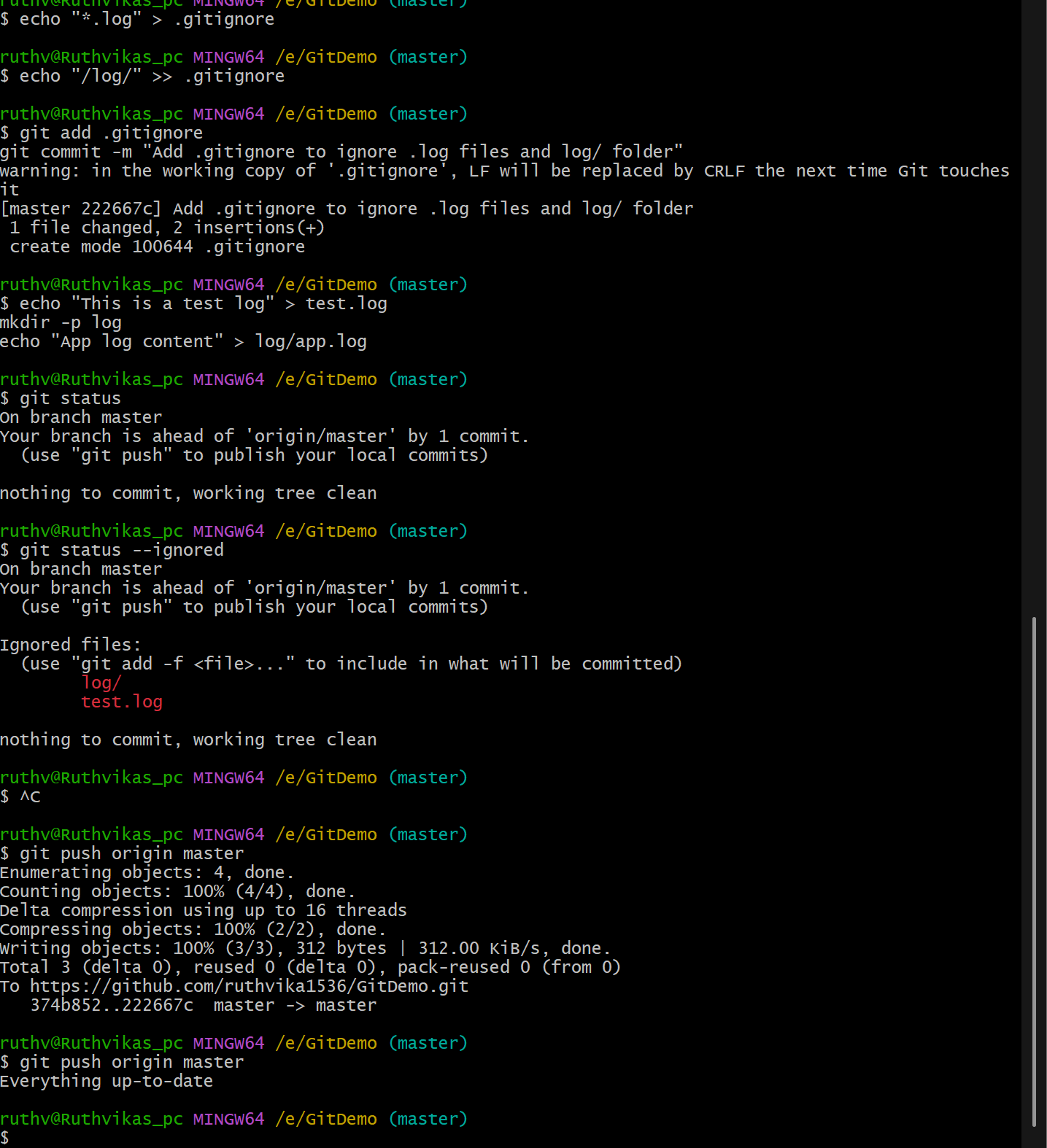
1. Check Git is installed or not and check its Version.
2. To configure user level configuration of user ID and email ID execute.
3. To check if the configuration is properly set, execute the following command.
4. Git config –global --list
5. Create a folder with the name GitDemo and create a welcome.txt file in it with the contents as followed.
6. Now we Add the file to the staging area.
7. Commit the file to the local repository with a message.
8. Check Git status to confirm a clean working directory.
9. Push the changes to the remote repository.
10. Push explicitly to the origin master branch.

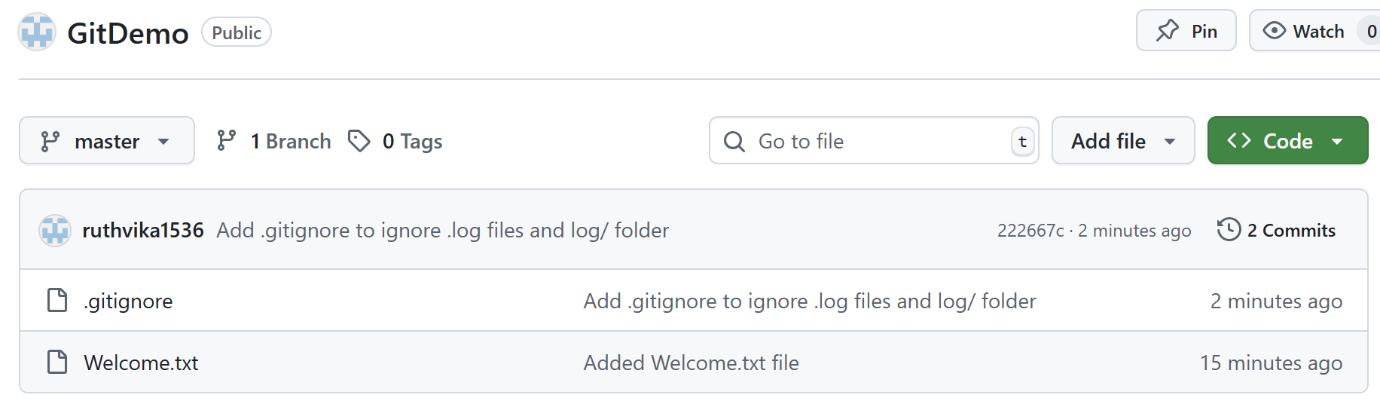




**Exercise 2:**

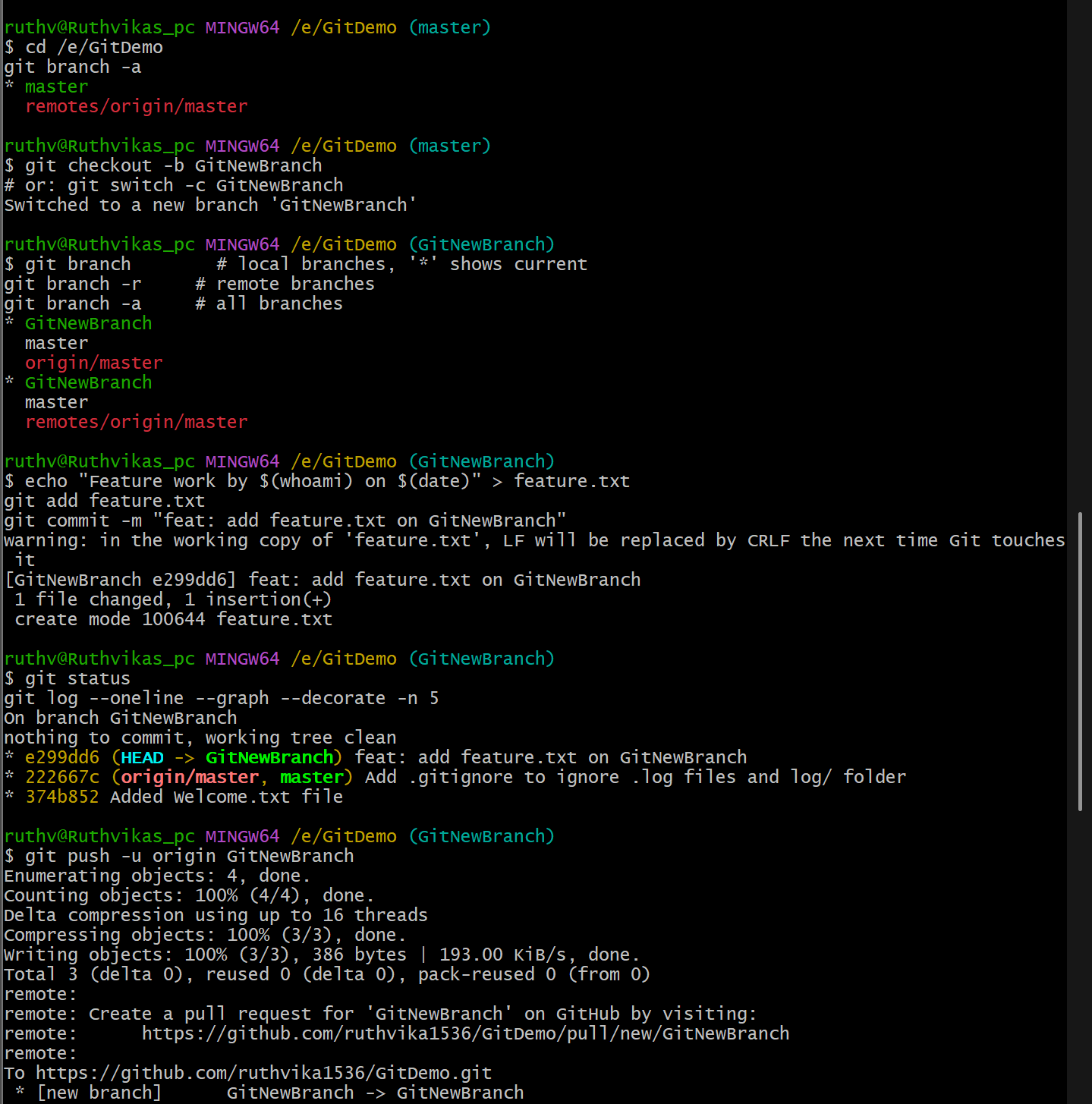
1. Ensure you are inside the correct Git repository folder.
2. Create a .log file in the working directory (e.g., test.log).
3. Create a folder named log in the working directory.
4. Inside the log folder, create a sample file (optional, just to test ignoring).
5. Create a .gitignore file in the repository root if it doesn’t already exist.
6. Add the following rules in .gitignore:
7. \*.log (to ignore all .log files)
8. log/ (to ignore the log folder)
9. Save the .gitignore file.
10. Check the Git status to verify that .log files and the log folder are untracked and ignored.
11. Use the ignored files listing to confirm (git status --ignored).
12. Stage and commit the .gitignore file to the repository.
13. Push the commit to the remote repository.

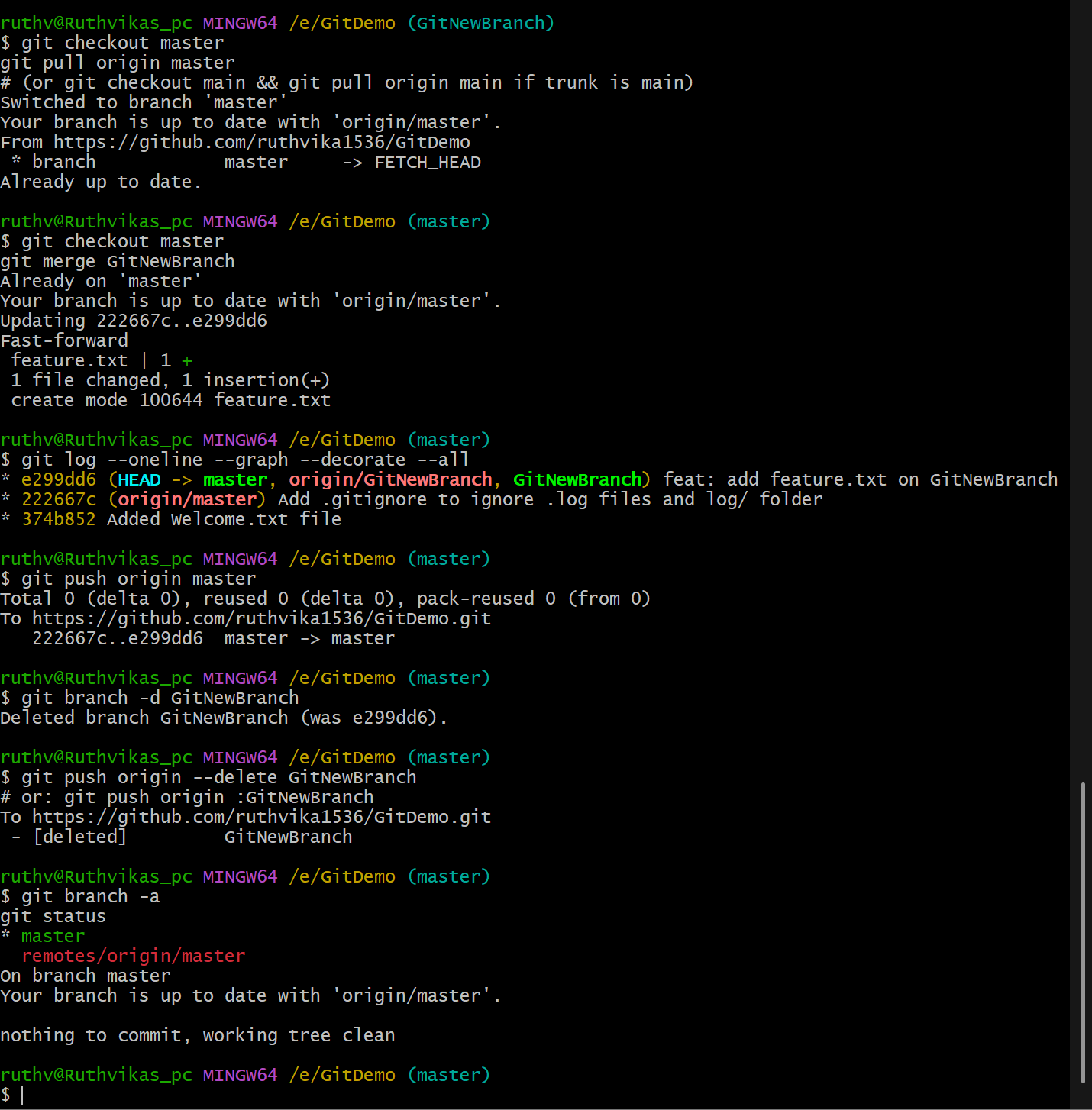


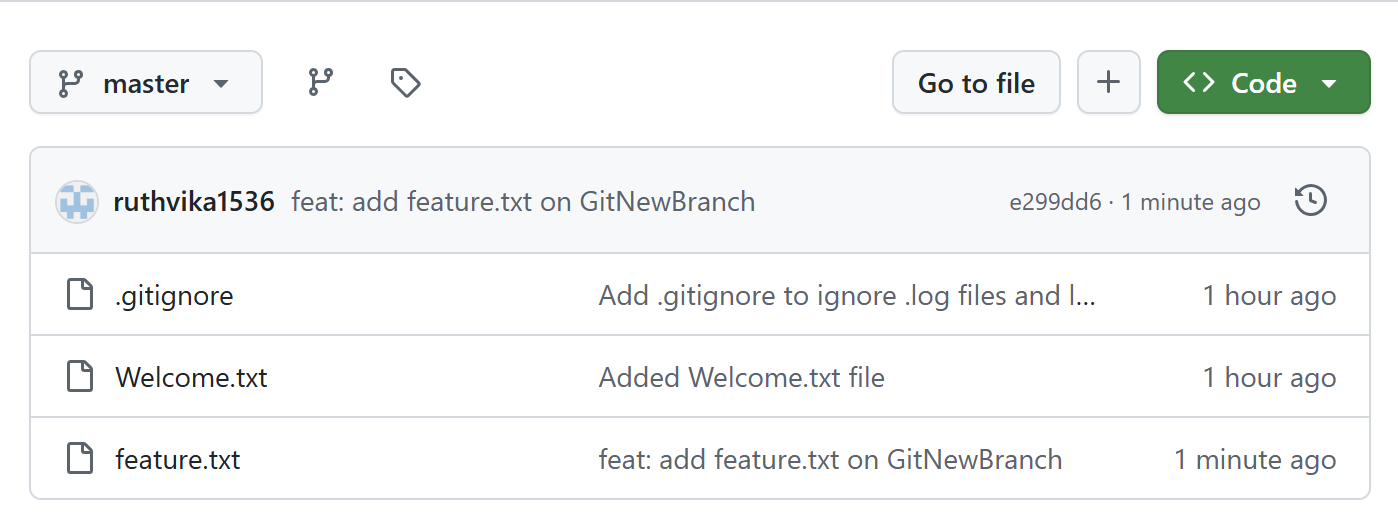


**Exercise 3:**

1. Ensure you are inside the correct Git repository folder.
2. Create a new branch named GitNewBranch.
3. List all local and remote branches to verify the new branch is created.
4. Switch to the GitNewBranch.
5. Create a new file named features.txt and add some content to it.
6. Stage and commit the new file to the branch.
7. Check the Git status to confirm the branch is up to date.
8. Switch back to the master branch.
9. Compare the differences between master and GitNewBranch in the CLI.
10. View the differences visually using the merge/diff tool (e.g., P4Merge).
11. Merge GitNewBranch into master.
12. View the commit history with a graph (git log --oneline --graph --decorate).
13. Delete the GitNewBranch locally after merging.
14. Push the updated master branch to the remote repository.

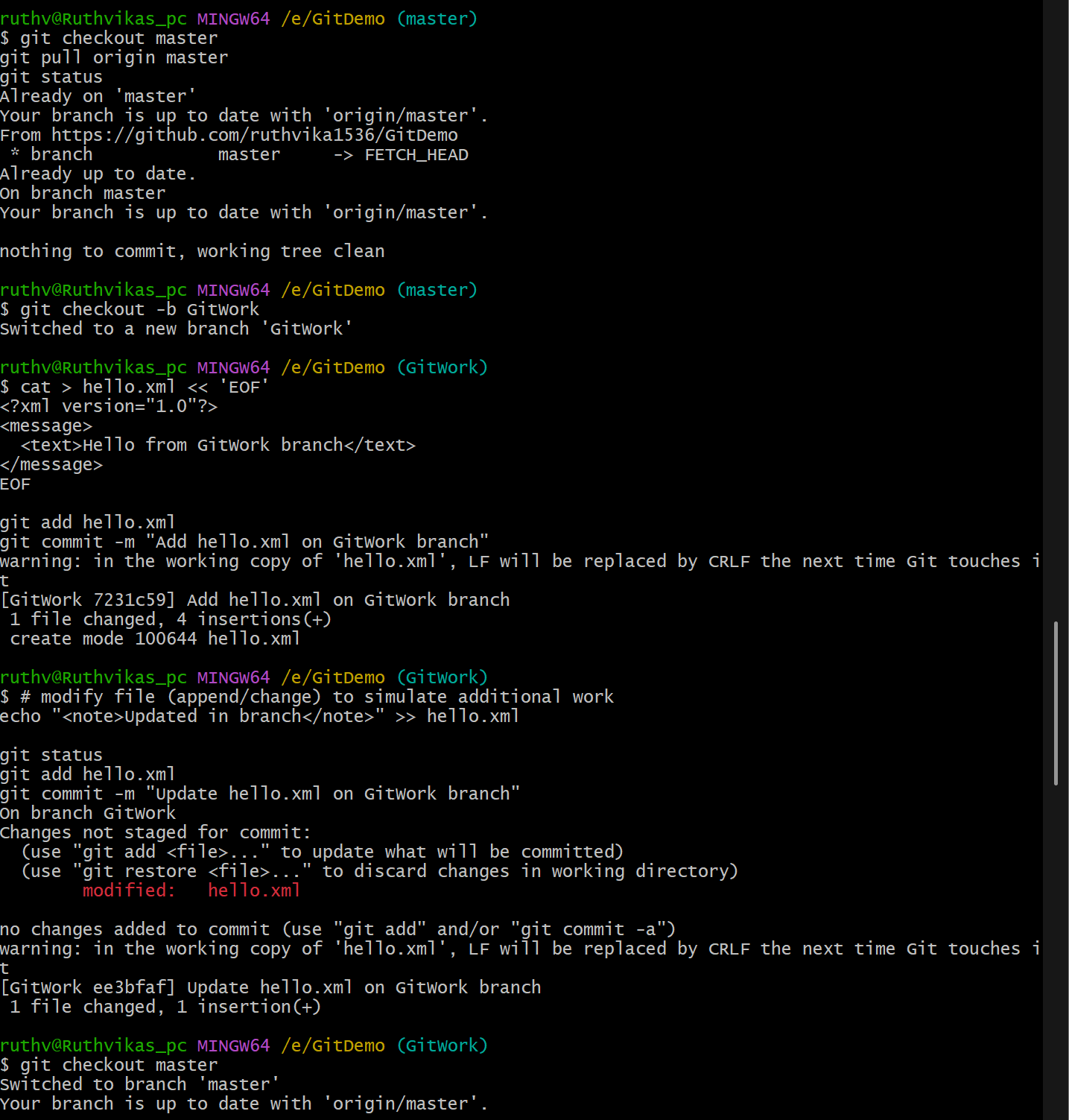


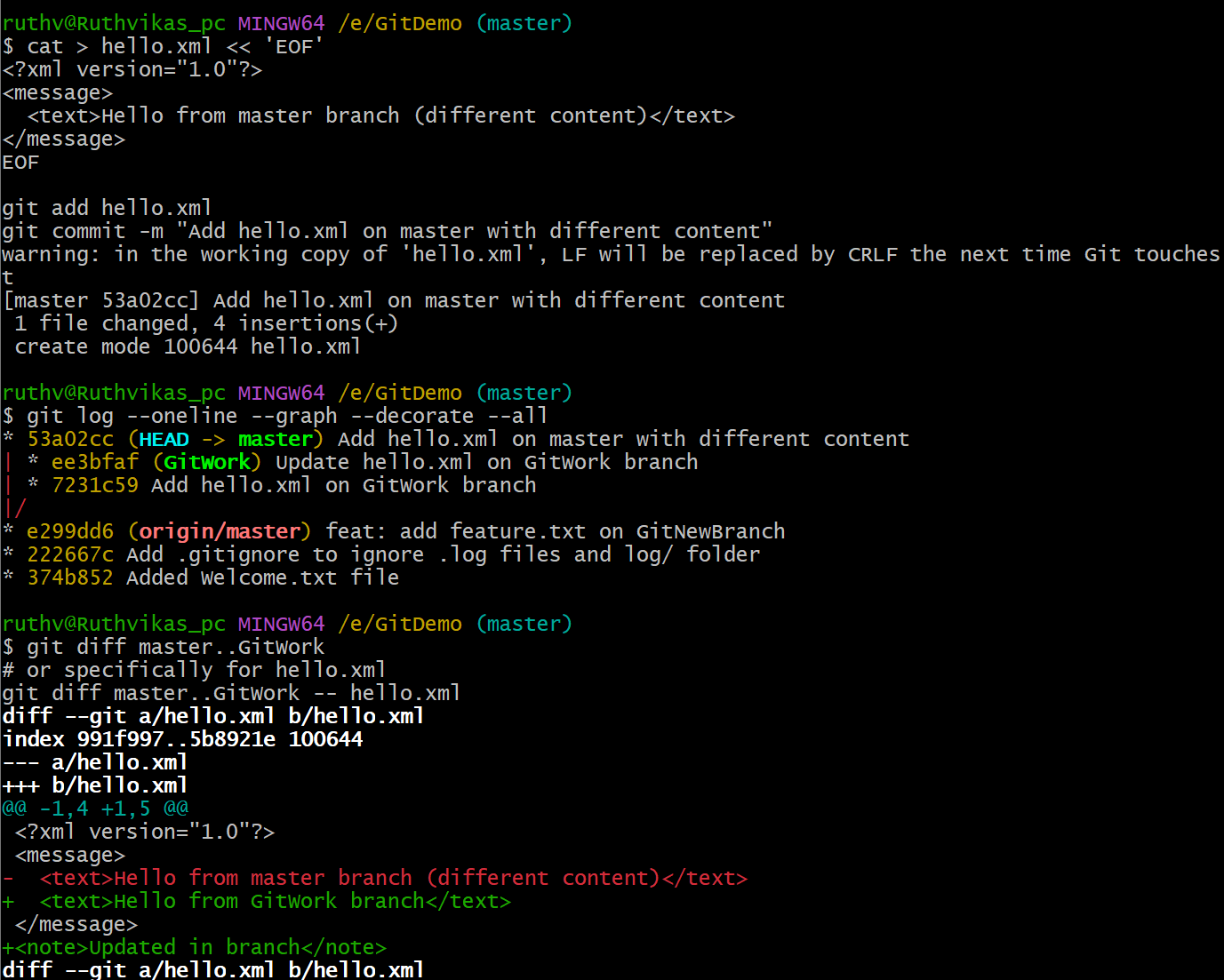
****

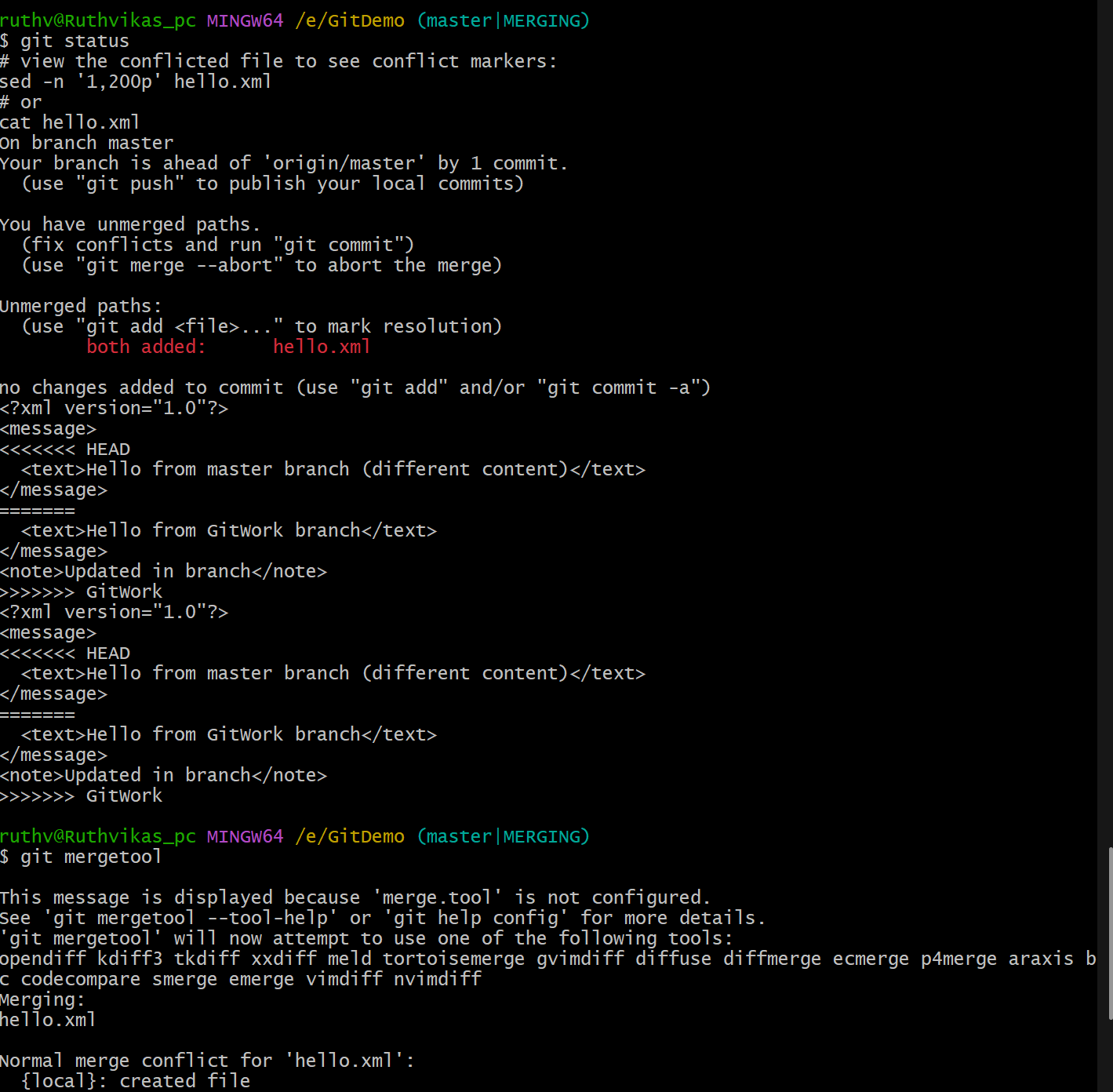
****

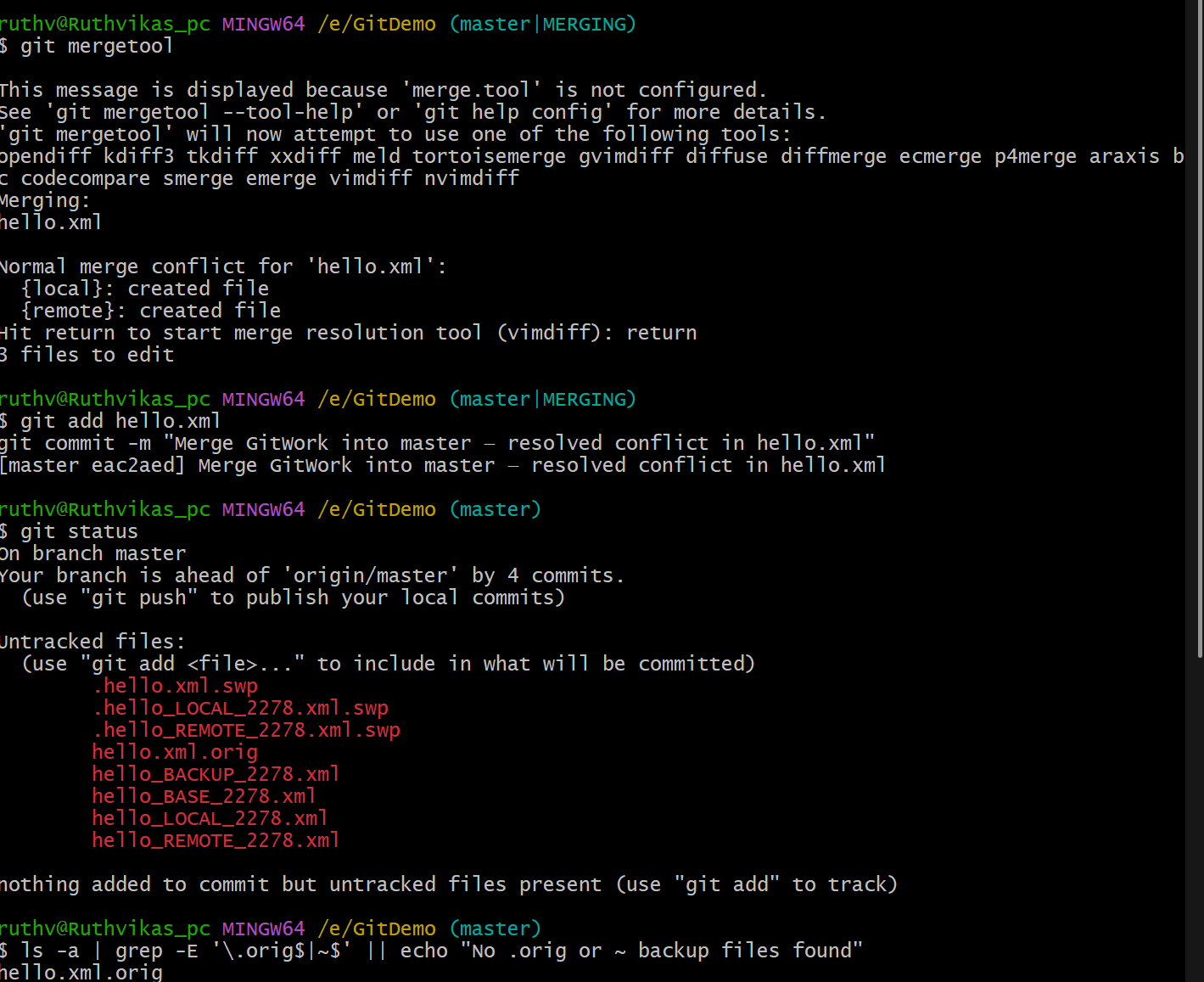
**Exercise 4:**

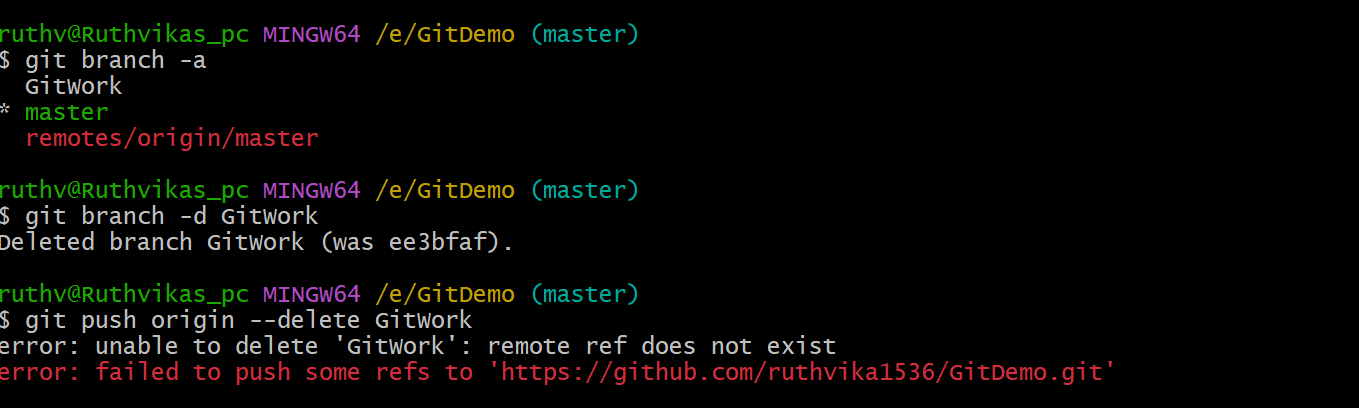
1. Verify that the master branch is in a clean state.
2. Create a new branch named GitWork.
3. Add a file named hello.xml in the GitWork branch and write some content.
4. Stage and commit the hello.xml changes to the branch.
5. Switch to the master branch.
6. Create another hello.xml file in master with different content.
7. Stage and commit the file in master.
8. View the commit history with all branches using git log --oneline --graph --decorate --all.
9. Compare differences between master and GitWork using the CLI diff tool.
10. View differences visually using the merge tool (e.g., P4Merge).
11. Merge GitWork into master (which causes a merge conflict).
12. Open the merge tool to resolve the conflict in hello.xml.
13. Stage and commit the resolved file to master.
14. Update .gitignore to ignore backup files.
15. Commit the .gitignore changes.
16. List all branches to verify.
17. Delete the GitWork branch locally after merging.
18. View the commit history again to confirm the merge.

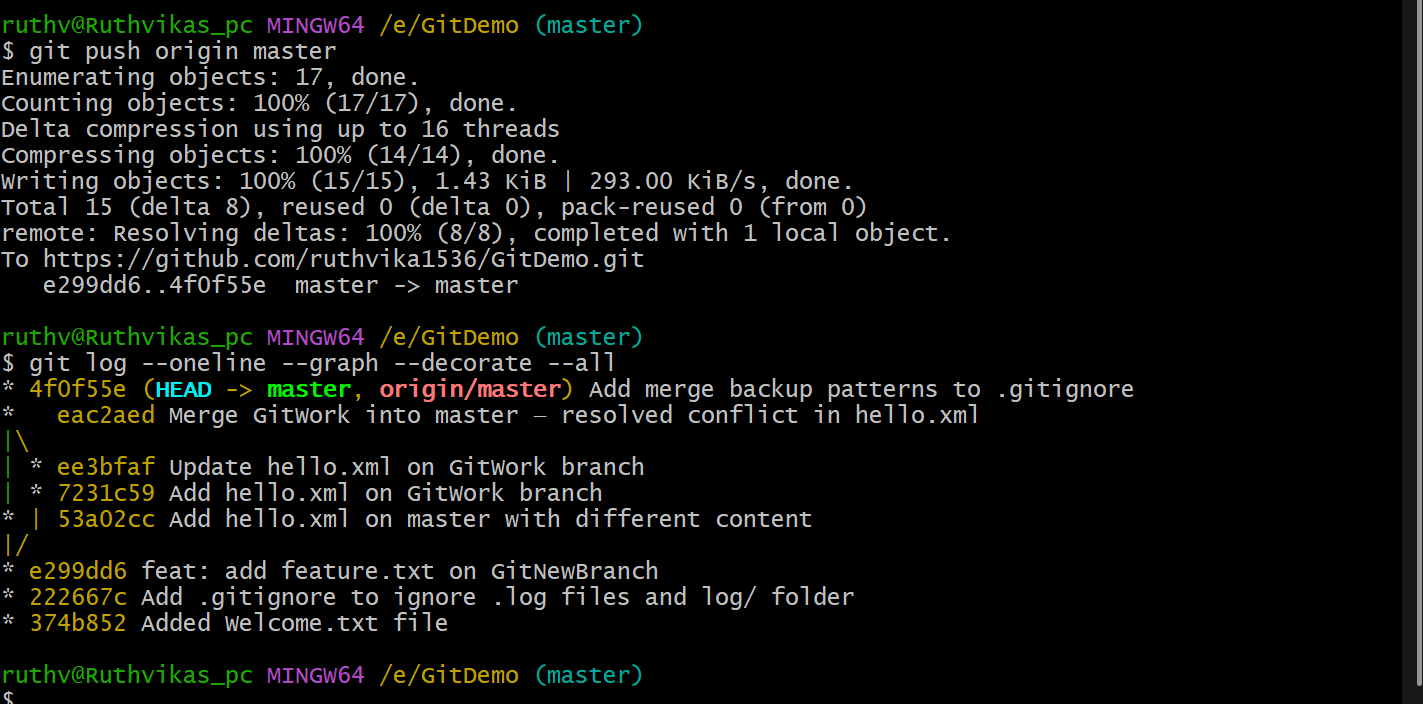


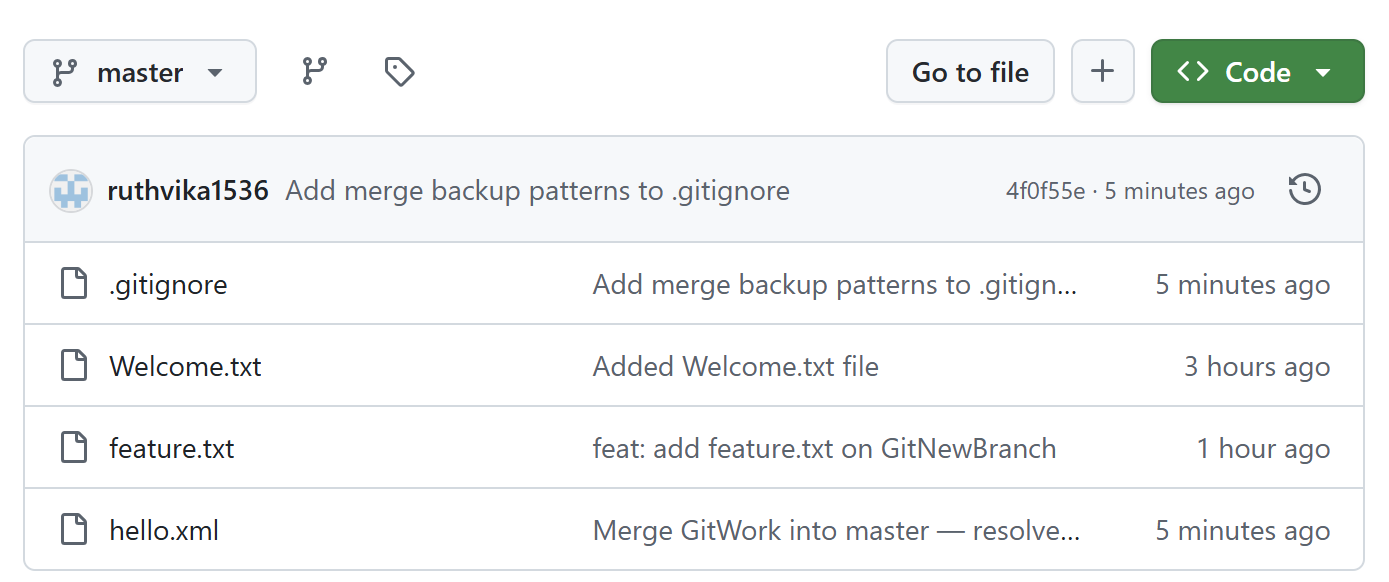






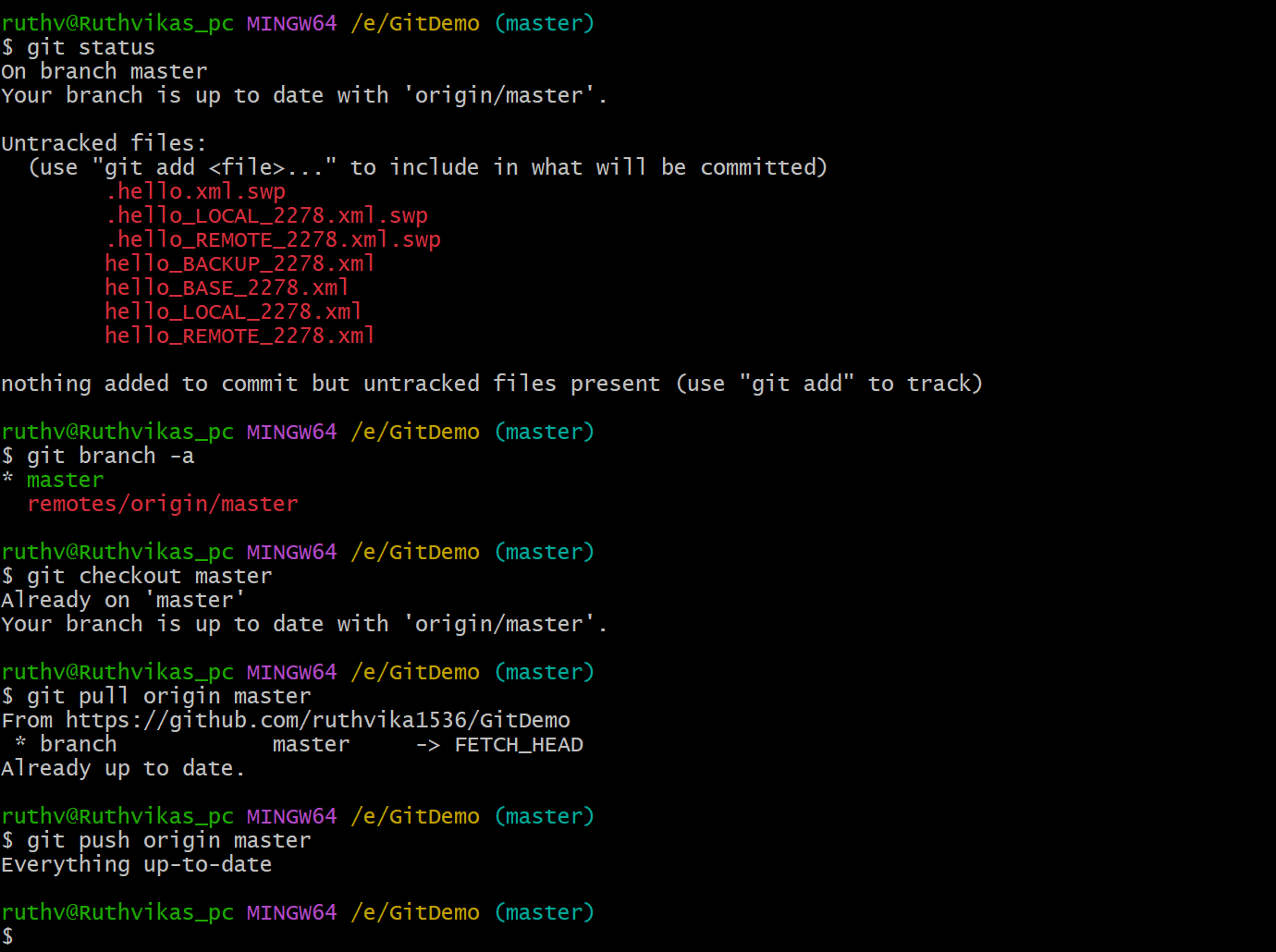






**Exercise 5:**

1. Verify that the master branch is in a clean state.
2. List all available branches in the repository.
3. Pull the latest changes from the remote repository into master.
4. Push any pending local changes from the previous hands-on to the remote repository.
5. Verify that the changes are reflected on the remote repository (e.g., on GitHub).



**Conclusion:**

Across all the hands-on exercises, we learned to create and manage files in Git, use .gitignore to exclude unwanted files, create and switch branches, merge changes, and resolve conflicts using merge tools.  
We also practiced cleaning up branches, synchronizing local and remote repositories, and maintaining a clean, conflict-free project history.

**-- THE END --**