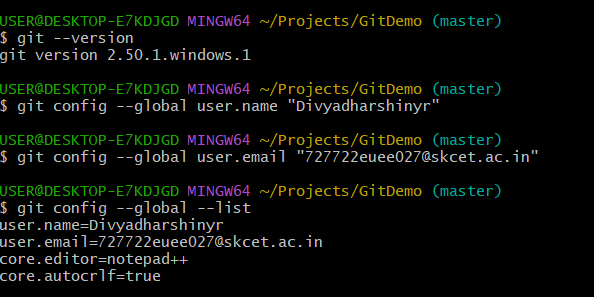
**WEEK – 08**

**GIT Hands-On**

**1.GIT HOL**

**Step 1: Setup your machine with Git Configuration**

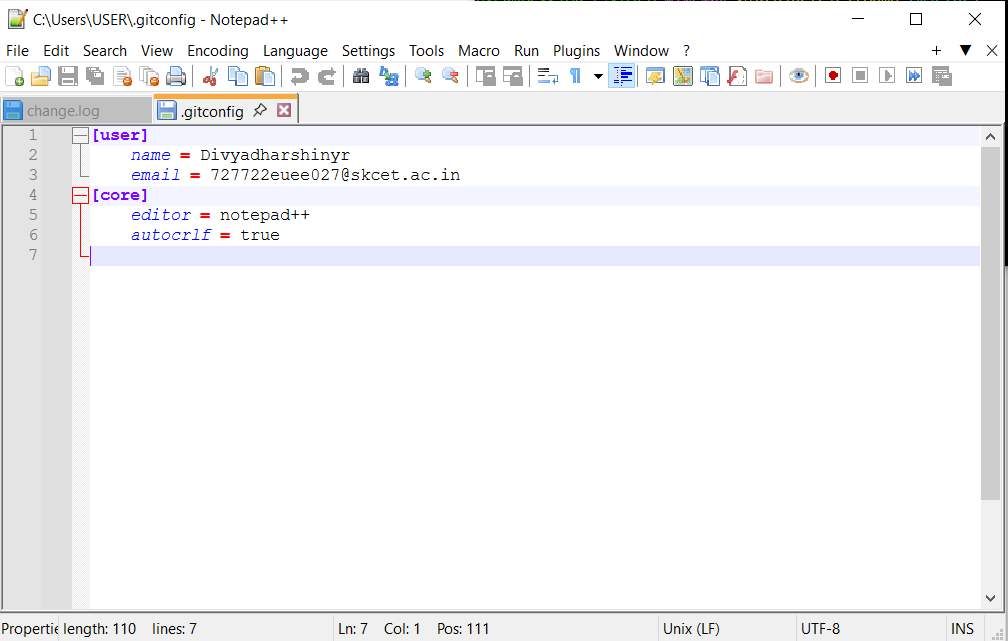
* Created “GitDemo” Project
* Checked the git version
* Configured user level configuration of user ID and email ID



**Step 2: Integrate notepad++.exe to Git and make it a default editor**

* Made notepad++ as default editor
* Verified if notepad++ is the default editor

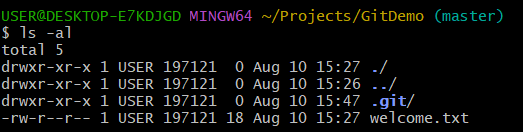




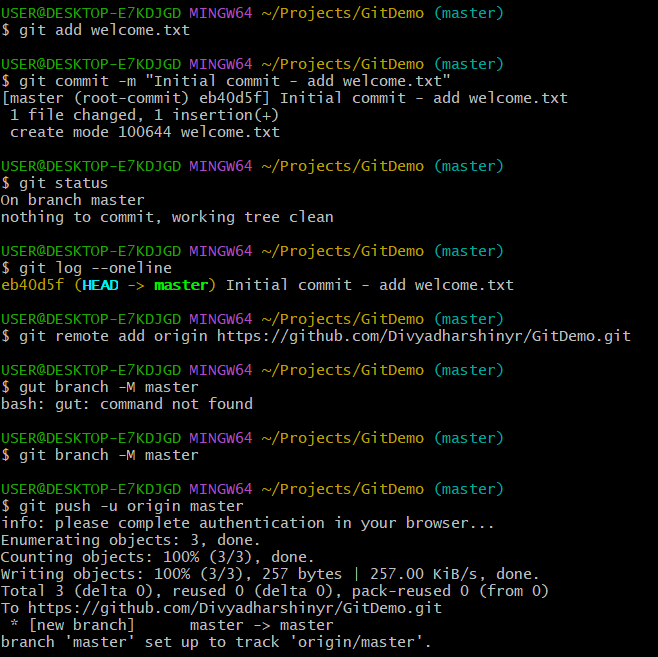
**Step 3: Add a file to source code repository**

****

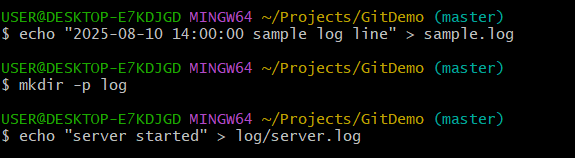


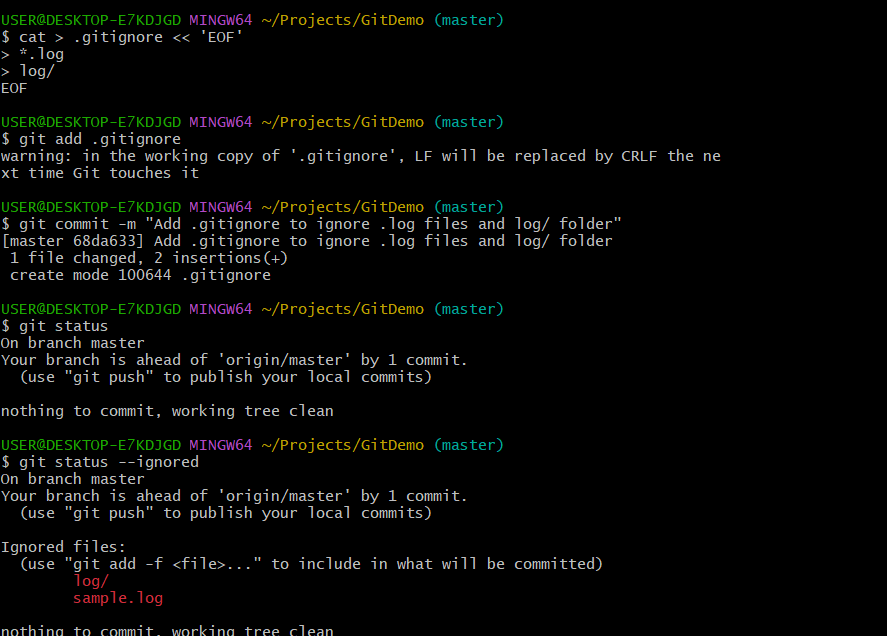






**2.GIT HOL**





**Explanation:**

* Created a .log file named sample.log in the working directory.
* Created a log directory containing a server.log file.
* Added a .gitignore file with the patterns:
  + \*.log → ignores all files with .log extension.
  + log/ → ignores the log directory and its contents.
* Committed the .gitignore file to the repository.
* Ran git status to verify the ignore rules.
* Output showed sample.log and log/ under *Ignored files*, confirming they are excluded from version control.
* Working tree reported as clean, meaning no tracked changes remain.

**3. GIT HOL**

**1.Explain branching and merging**

* Branching in Git is the process of creating an independent line of development within the same repository. Each branch can have its own commits without affecting the main branch, allowing multiple features or fixes to be developed in parallel.
* Merging is the process of integrating changes from one branch into another. When a branch’s work is complete, it is merged into the target branch (e.g., master or main), combining the commit histories and applying changes.

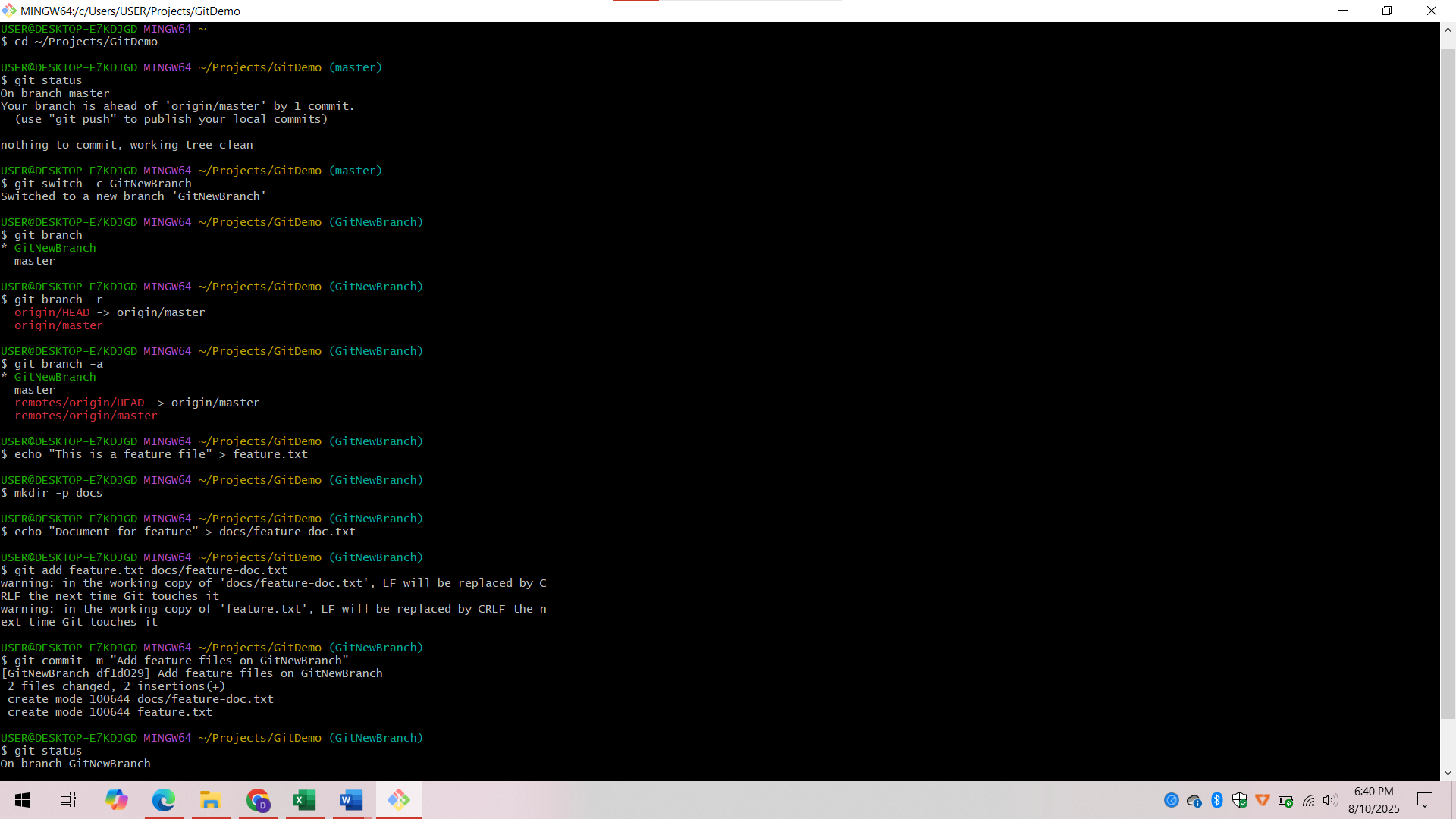
**2.Explain about creating a branch request in GitLab**

* In GitLab, creating a branch request means making a new branch in the repository to work on a specific feature, fix, or task without affecting the main branch.
* This can be done from the Repository to Branches section in GitLab by clicking New branch, entering a branch name, and specifying the branch to base it on (usually main or master).
* The new branch can then be used to push commits related to that specific work.

**3.Explain about creating a merge request in GitLab**

* A Merge Request (MR) in GitLab is a way to propose merging changes from one branch into another.
* After pushing commits to a branch, navigate to Merge Requests in GitLab and click “New merge request.”
* Select the source branch (working branch) and the target branch (e.g., main or master), review the changes, and submit the request.
* The merge request allows team members to review code, discuss changes, and approve before merging into the target branch.

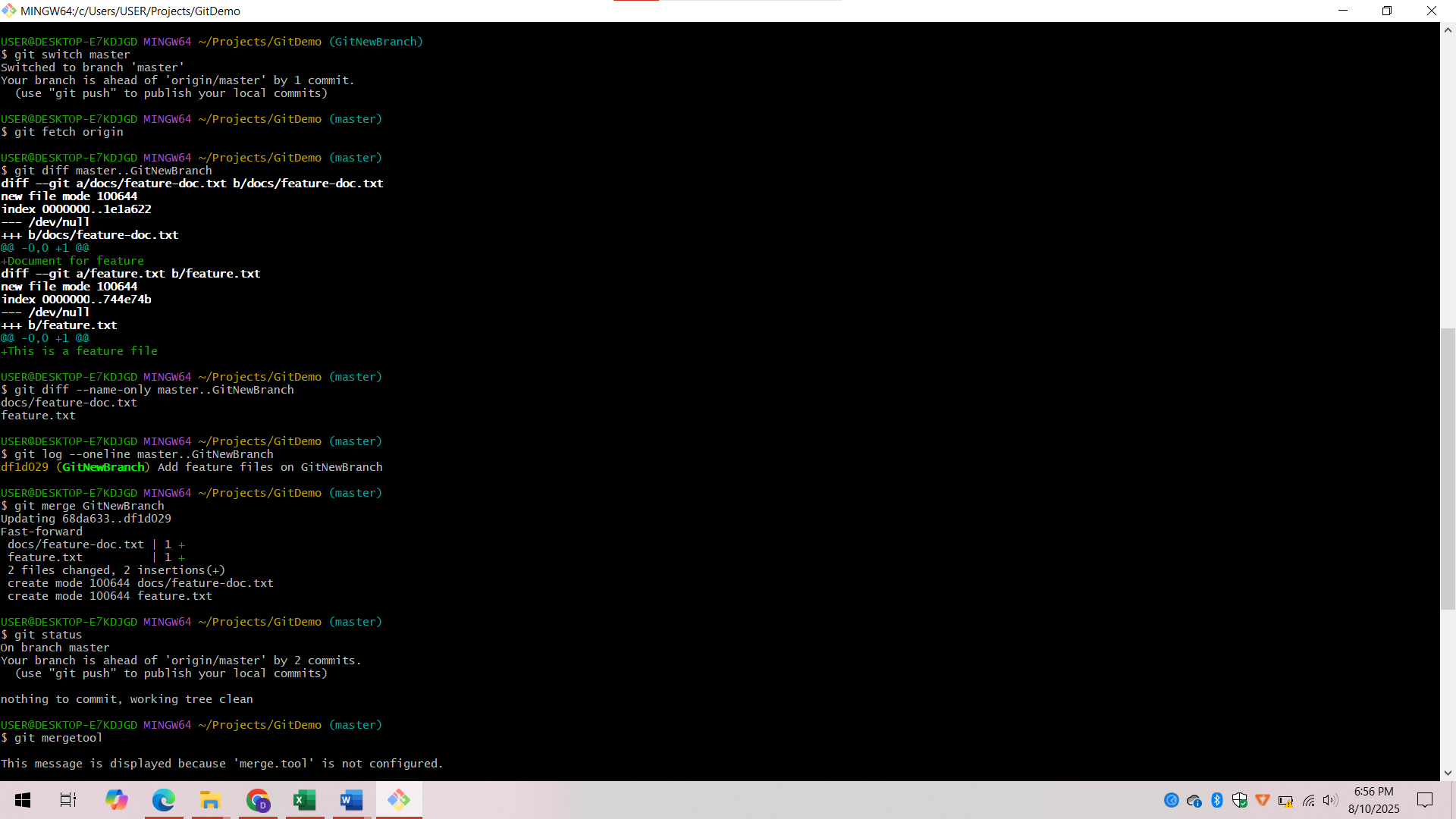
**Branching:**

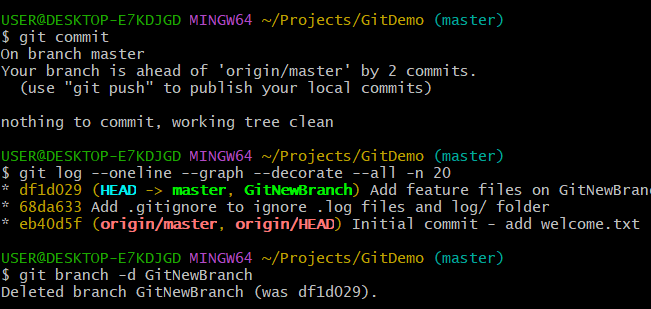


**Branching Process**

* Began on the main branch (master or main) to ensure the latest version of the project is used as the base.
* Creates a new branch with a descriptive name for the specific feature or task to be developed.
* Switched to the newly created branch to start isolated work without affecting the main branch.
* Added as required for the task.
* Stages the modified files and commit them to the branch, ensuring changes are recorded in version history.

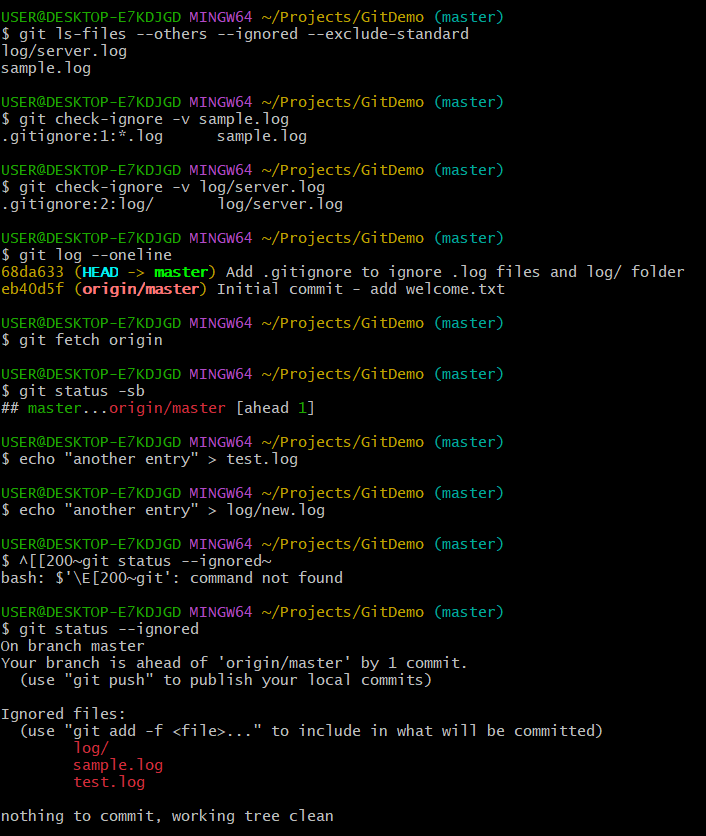
**Merging:**





**Merging Process**

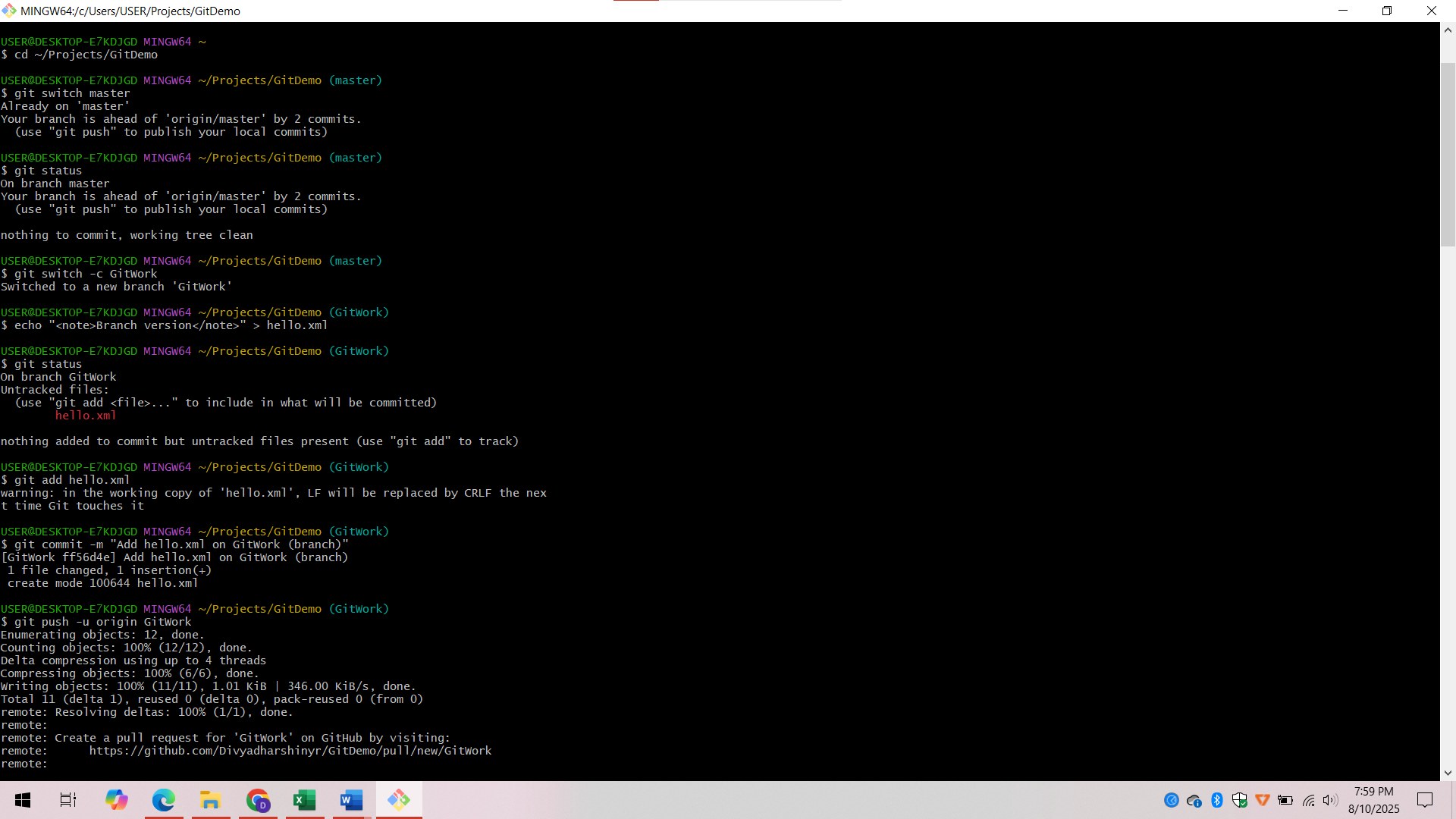
* Switched to the target branch (e.g., master) into which changes will be merged.
* Review and compare the differences between the target branch and the source branch to confirm readiness for merging.
* Executed the merge command to integrate the source branch changes into the target branch.
* Commit the merge to finalize integration of changes into the target branch.
* Deleted the source branch locally and remotely if it is no longer required to keep the repository clean.

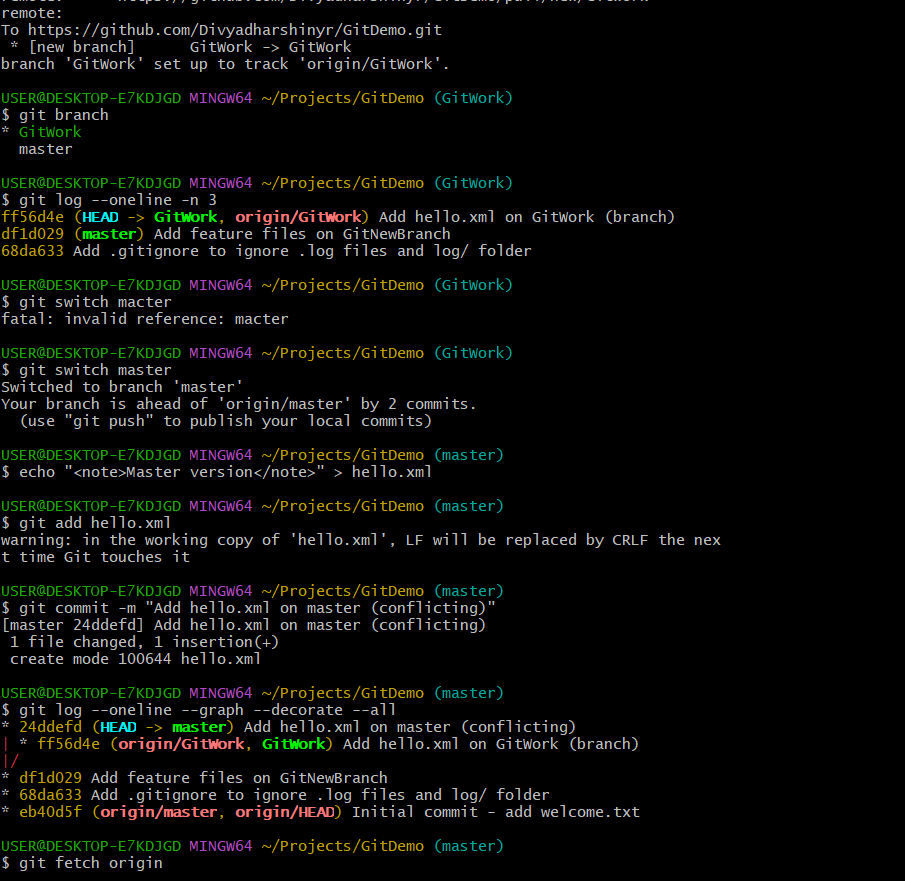
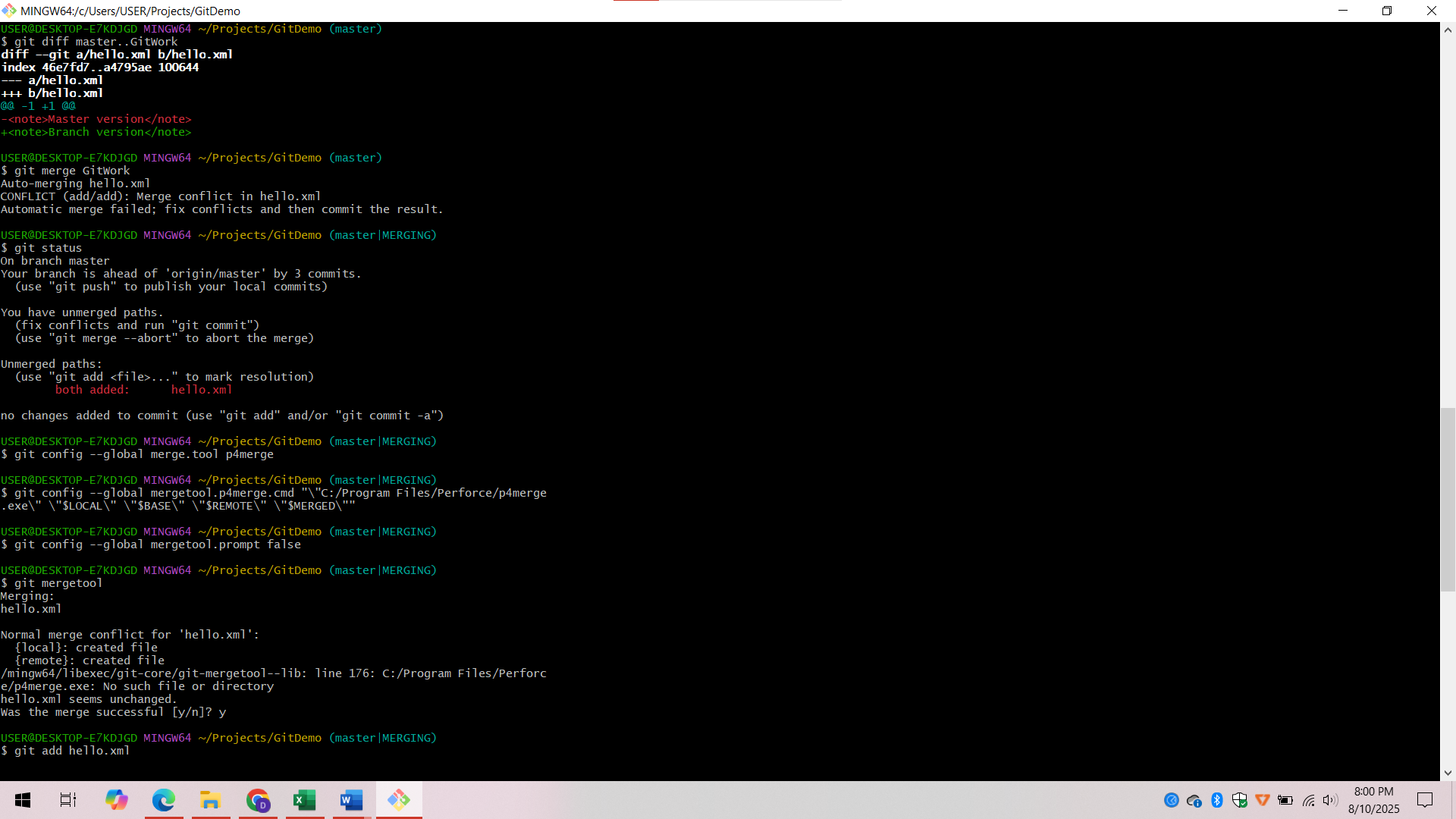


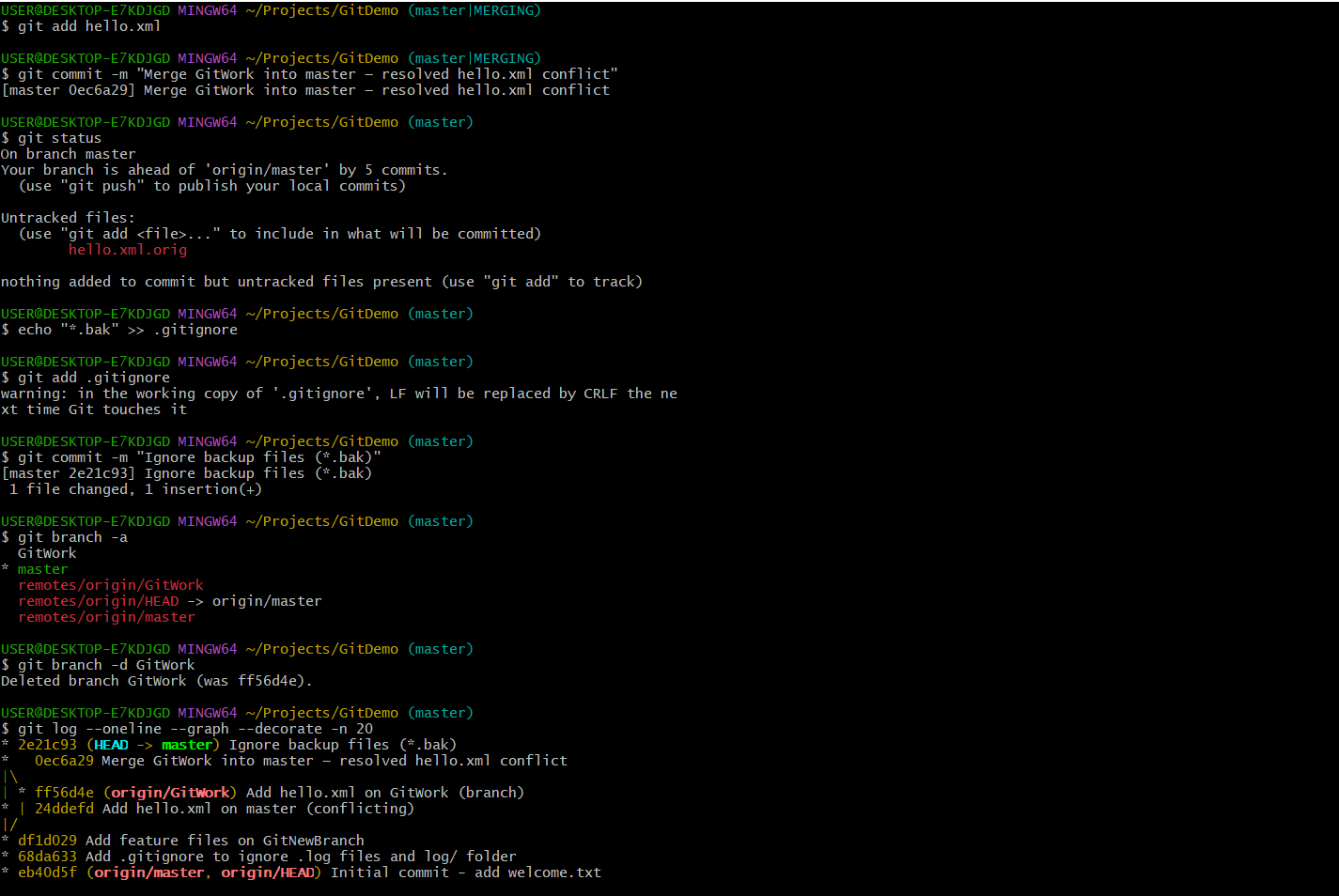
**4.GIT HOL**

**1.Explain how to resolve the conflict during merge.**

* During a merge, if the same section of a file is changed in both branches, Git marks it as a conflict.
* Use git status to identify conflicted files listed under *Unmerged paths*.
* Open the conflicted files; conflict markers (<<<<<<<, =======, >>>>>>>) indicate the differing changes.
* Edit the file to keep the correct content, removing the conflict markers.
* Stage the resolved files using git add <filename>.
* Finalize the merge by committing with git commit.
* Verify with git status to ensure no conflicts remain.
* Optionally, use a visual merge tool like P4Merge to simplify resolution.





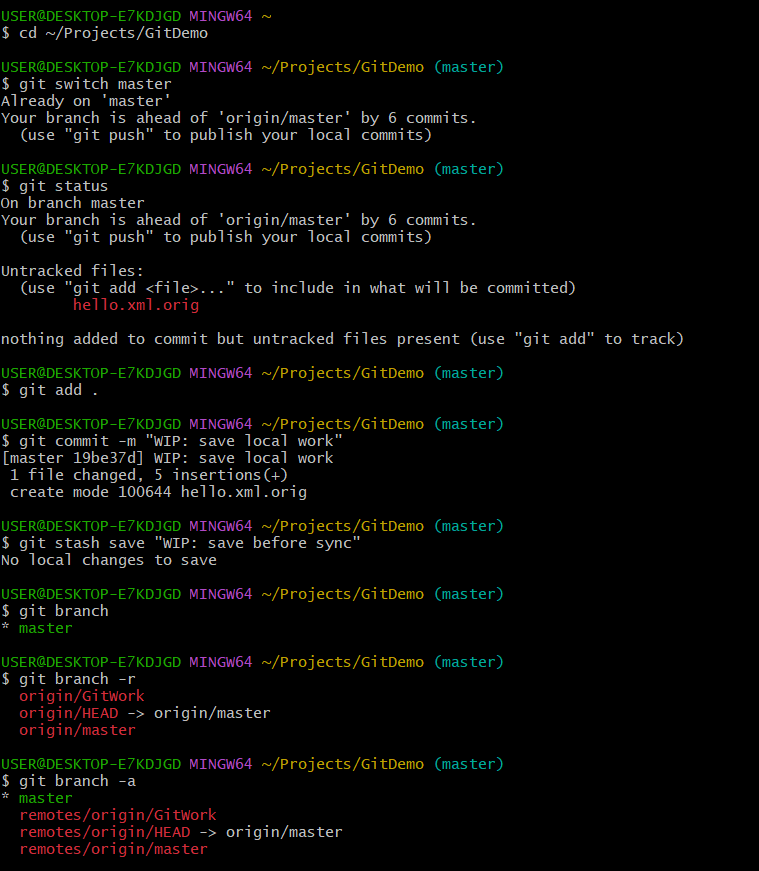
**Explanation:**

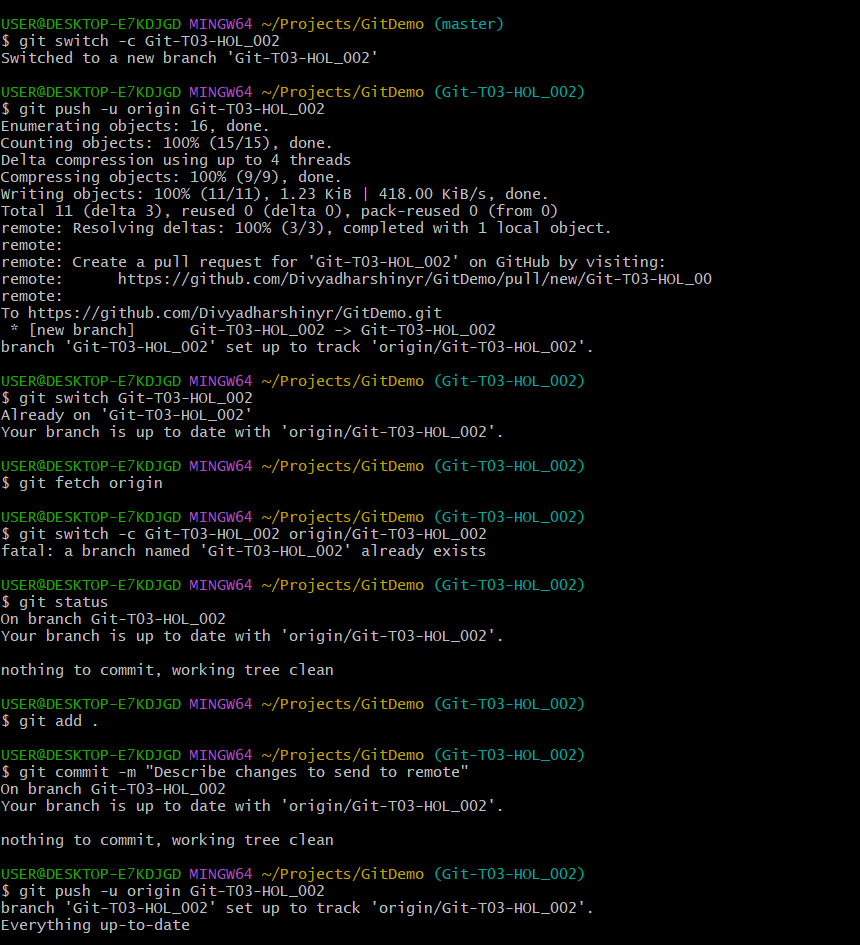
* Verified the master branch was clean and created a new branch GitWork to add hello.xml with specific content.
* Switched back to master and added a different hello.xml with conflicting content to create an intentional merge conflict.
* Compared branch differences using Git commands and P4Merge for visual inspection.
* Merged GitWork into master, resolved the conflict using a 3-way merge tool, and committed the resolved file.
* Added .gitignore entries to exclude backup files such as \*.orig.
* Deleted the merged branch and verified the merge in the commit history.

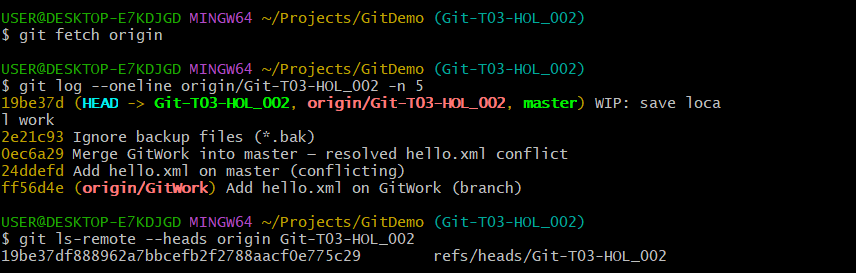
**5.GIT HOL**

**1.Explain how to clean up and push back to remote Git**

* Remove untracked files using git clean or manually delete them.
* Add unnecessary or temporary file patterns to .gitignore to prevent tracking.
* Delete unused local branches with git branch -d <branch>.
* Stage the cleanup changes using git add.
* Commit the cleanup changes with git commit.
* Push the updated commits and branch changes to the remote repository using git push.







**Explanation:**

* Checked the list of local and remote branches using git branch -a.
* Confirmed that Git-T03-HOL\_002 did not exist in either local or remote branches.
* Created a new branch Git-T03-HOL\_002 from the latest master branch.
* Made necessary changes and staged them using git add.
* Committed the changes to the new branch with git commit.
* Pushed the branch to the remote repository using git push -u origin Git-T03-HOL\_002