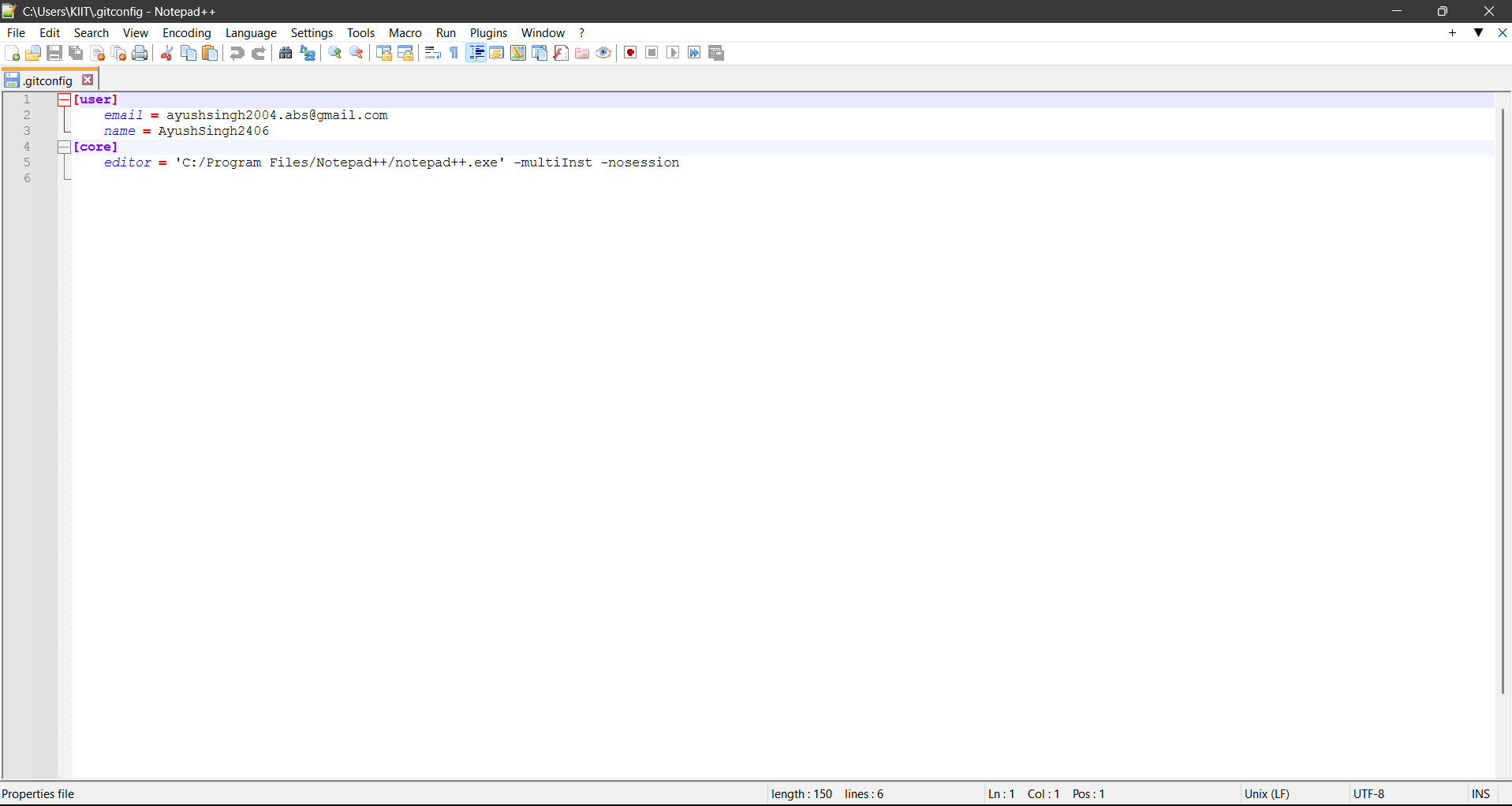
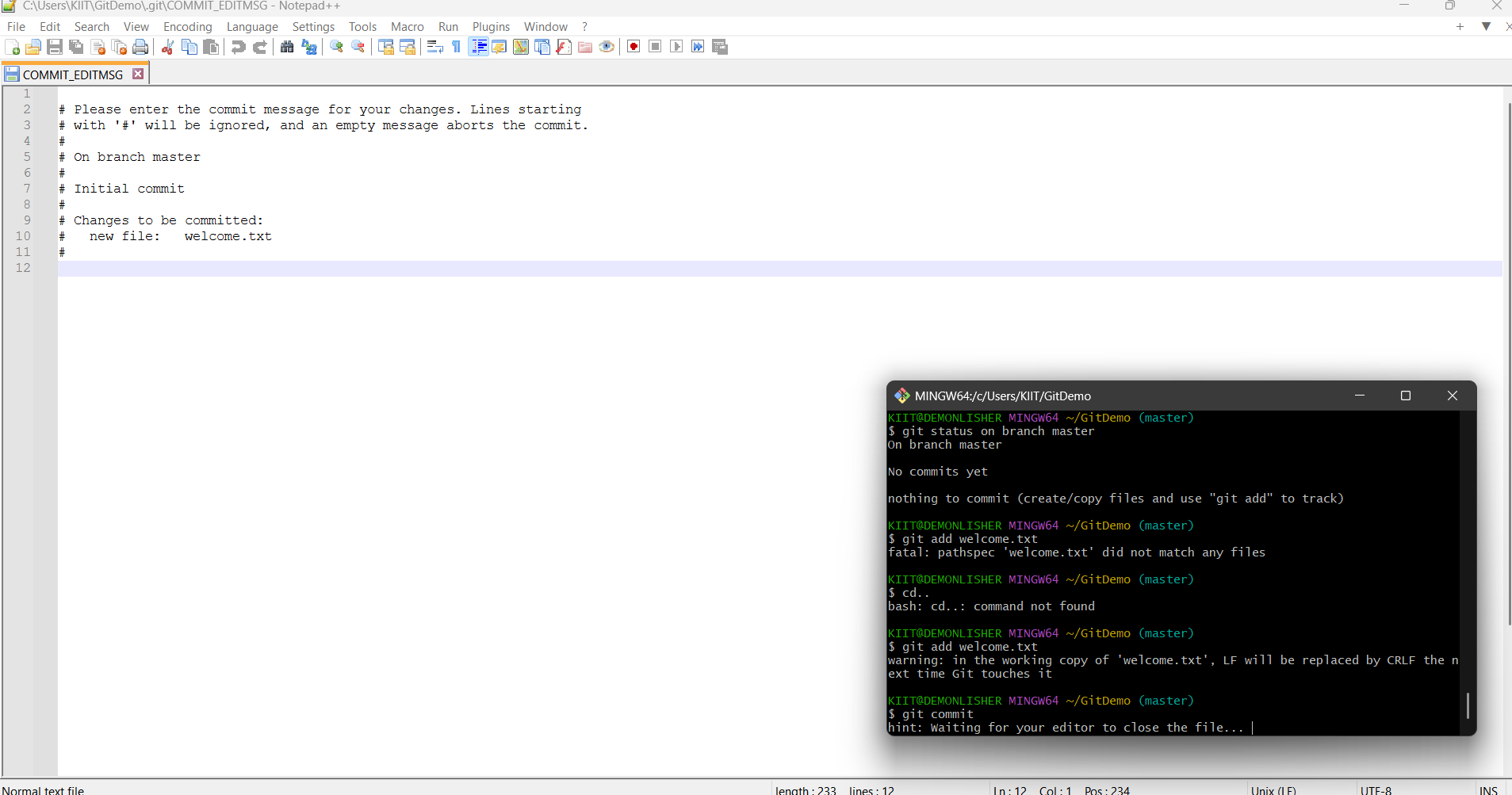
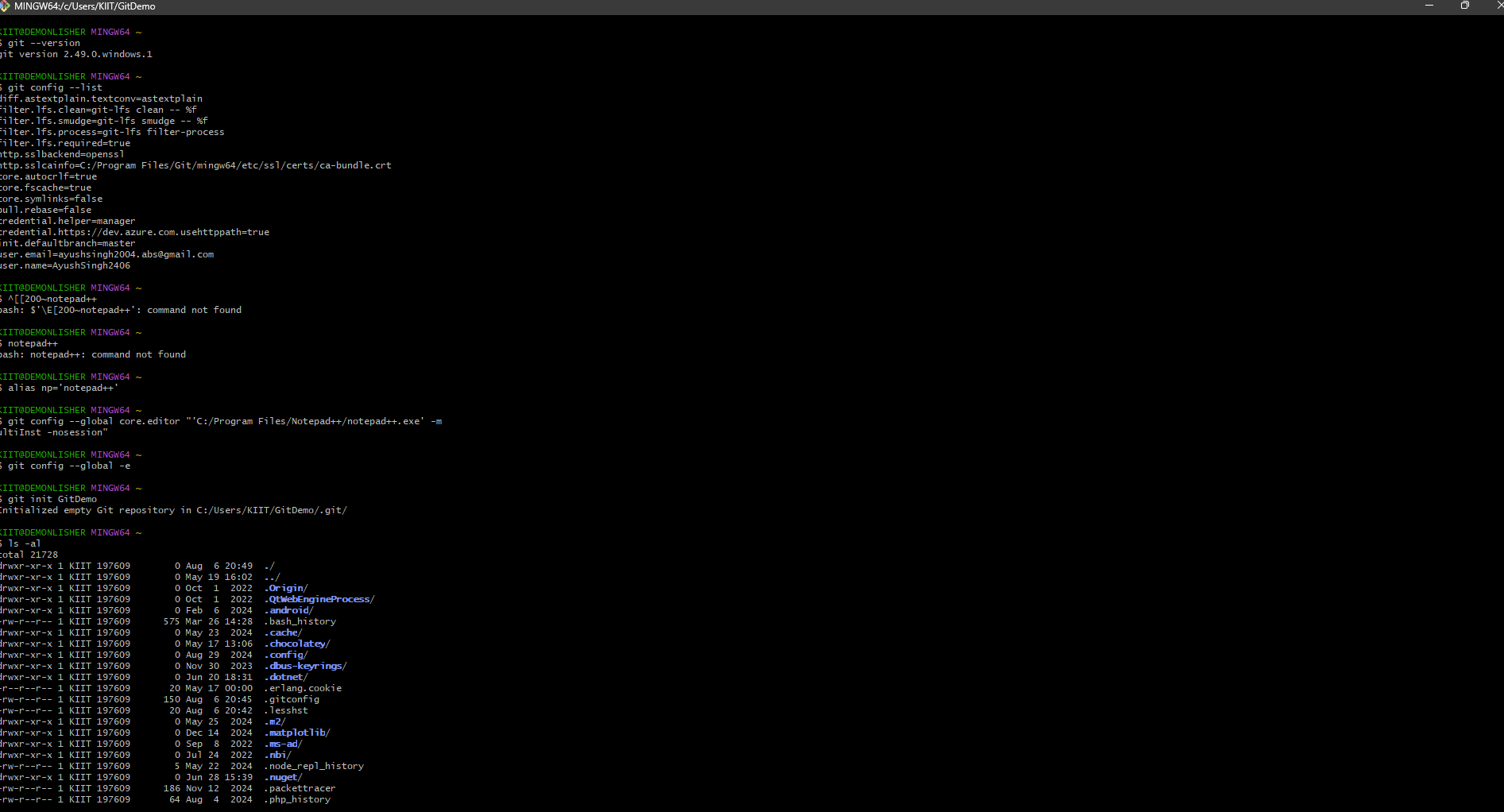
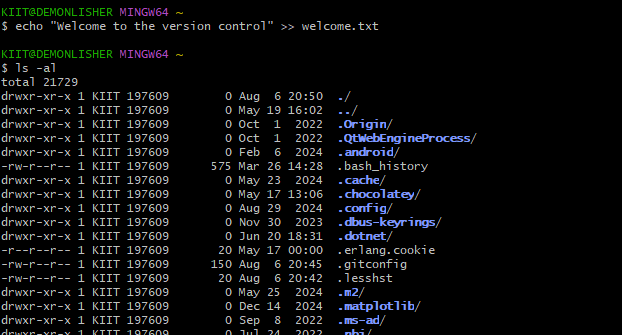
**HANDS ON 1: SETUP AND BASICS**

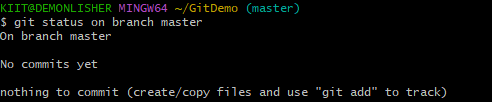


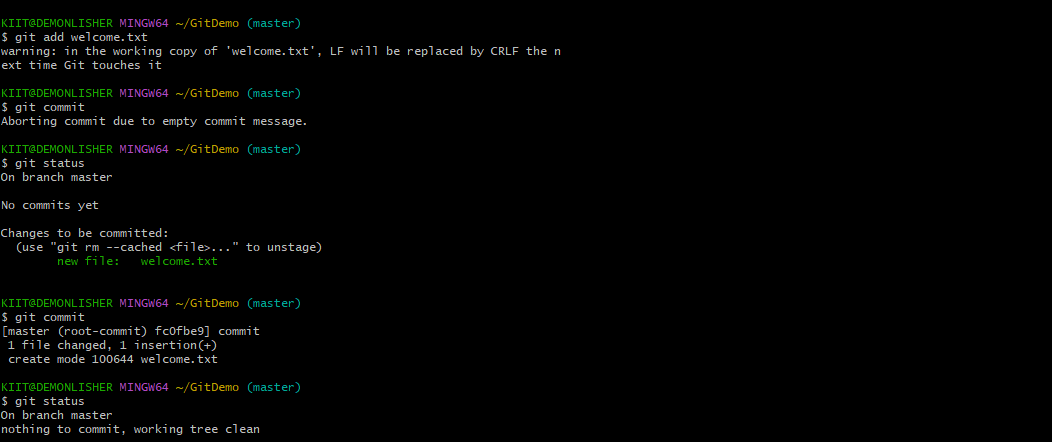


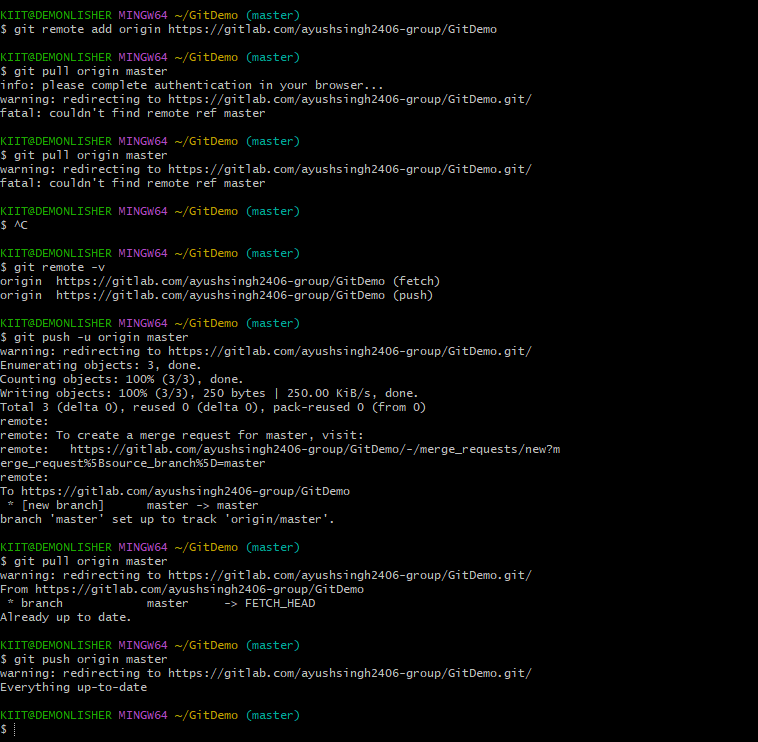




git5







**HANDS ON 2: Git Ignore Command**

# 1. What is .gitignore?

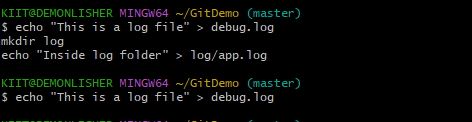
.gitignore is a special file used in Git to tell Git which files or directories to ignore in a project.  
  
Key Points:  
- Git does not track files or folders listed in this file.  
- It helps avoid committing temporary files, build artifacts, and sensitive configurations.  
- It improves collaboration by keeping the repository clean and relevant.

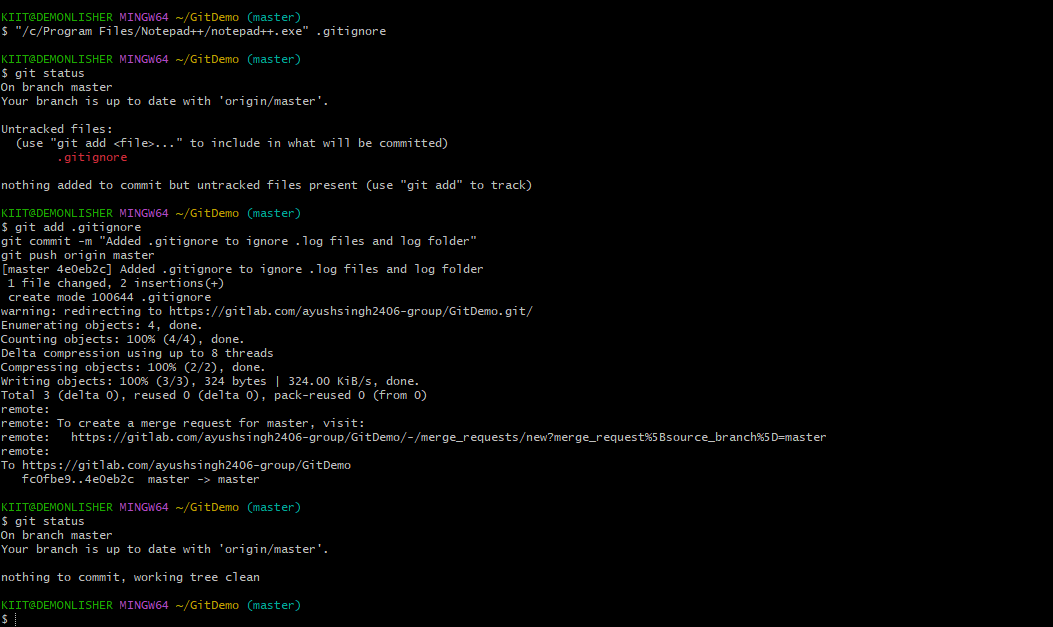
Common examples of files to ignore:

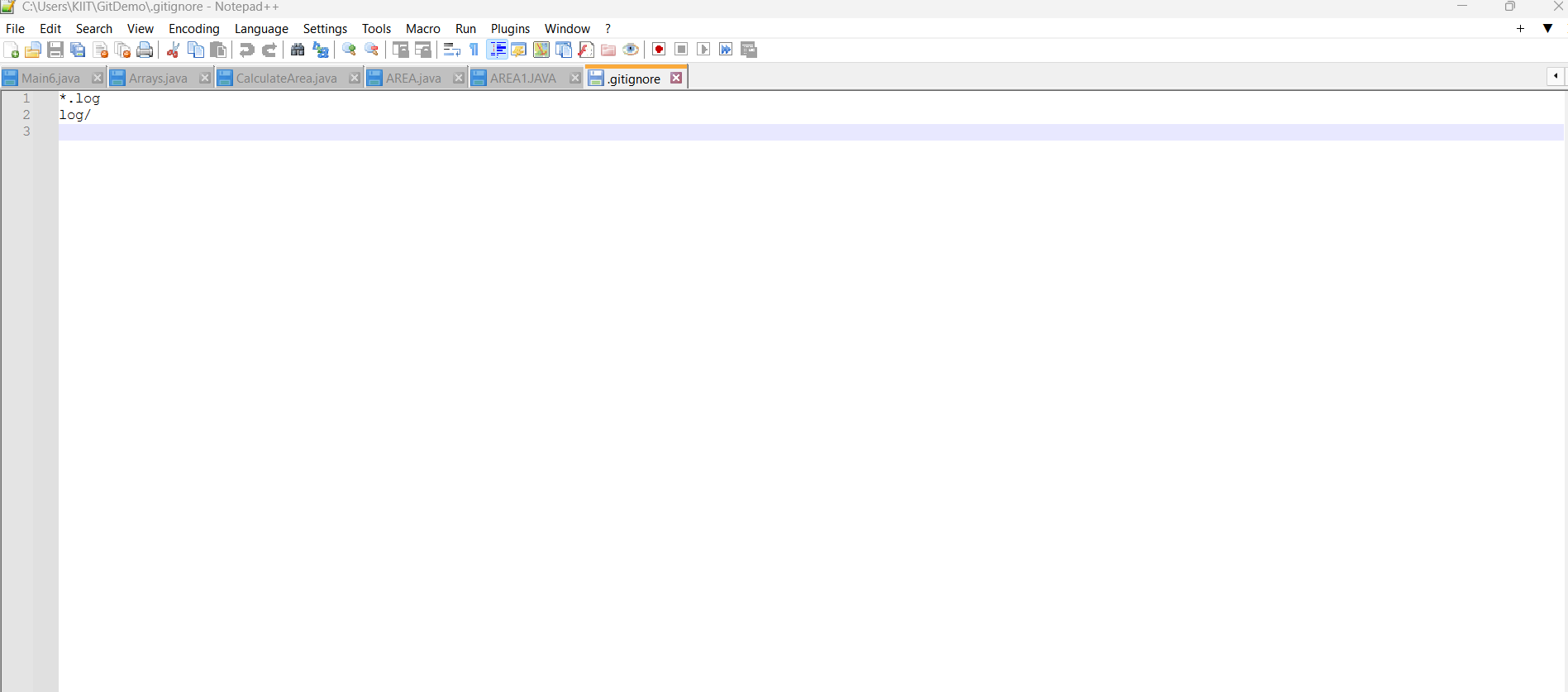
|  |  |
| --- | --- |
| File/Folder | Why it is ignored |
| \*.log | Log files generated during runtime |
| \*.class | Compiled Java bytecode |
| node\_modules/ | Dependencies installed via npm |
| .env | Contains sensitive config or API keys |
| \_\_pycache\_\_/ | Python cache files |
| .vscode/, .idea/ | IDE-specific project configs |

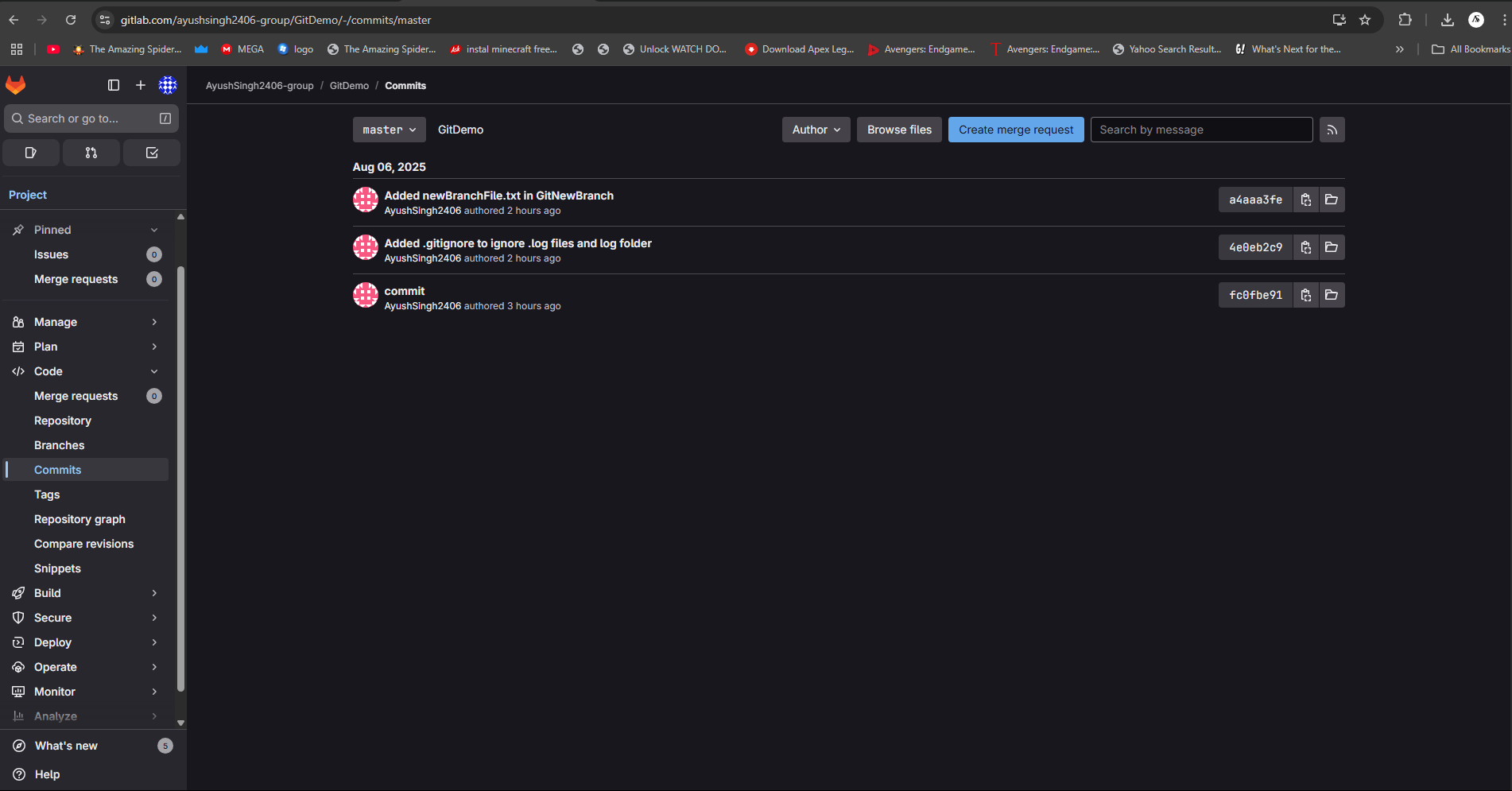
# 2. How to ignore unwanted files using .gitignore

Step 1: Create a .gitignore file  
Use the command: touch .gitignore  
  
Step 2: Add file or folder patterns inside .gitignore. Example:  
 \*.log # Ignore all .log files  
 log/ # Ignore the entire log folder  
 .env # Ignore sensitive environment config file  
 \*.tmp # Ignore all temporary files  
  
Step 3: Save and commit the .gitignore file:  
 git add .gitignore  
 git commit -m "Added .gitignore file"  
  
Note: If a file is already tracked, Git will not ignore it automatically. To untrack it:  
 git rm --cached filename









**HANDS ON 3: BRANCHING AND MERGING**

# 1. Explain Branching and Merging

Branching in Git allows you to create separate lines of development. A branch represents an independent version of the codebase where you can add features, fix bugs, or experiment safely.  
  
Merging is the process of combining changes from one branch into another, typically from a feature branch into the main branch (like master or main). This integrates the development work and keeps the codebase up to date.

Benefits of Branching:

* - Work on multiple features independently.
* - Avoid interfering with the main codebase.
* - Easier collaboration among team members.

Types of Merging:

* - Fast-forward merge
* - Three-way merge

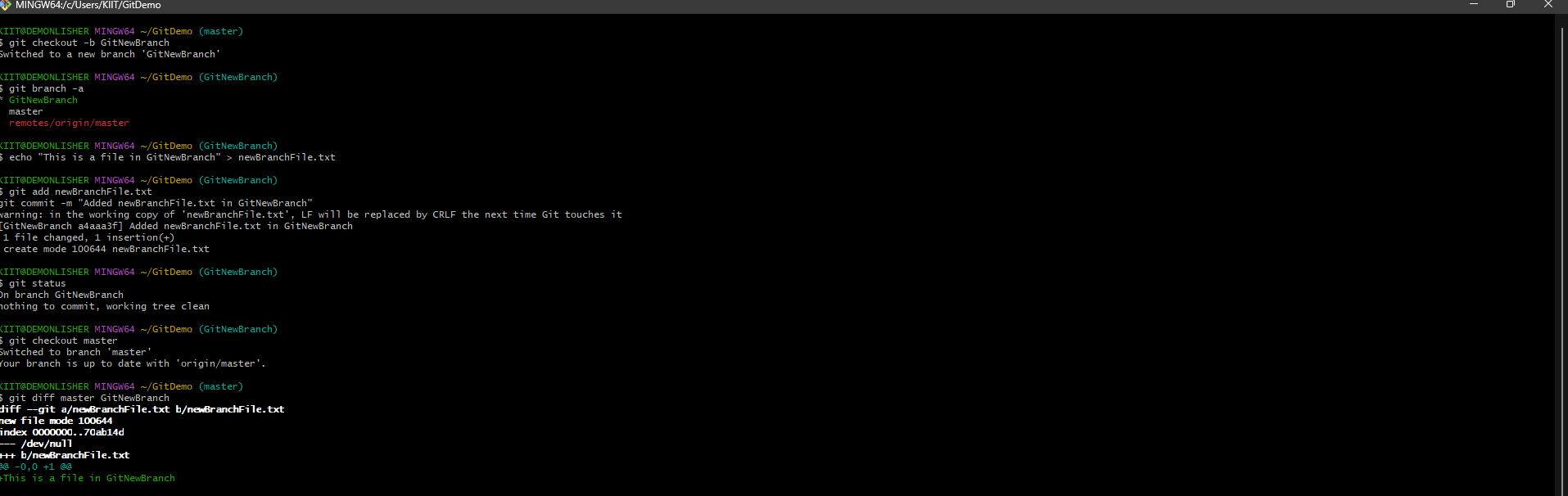
# 2. Explain about Creating a Branch Request in GitLab

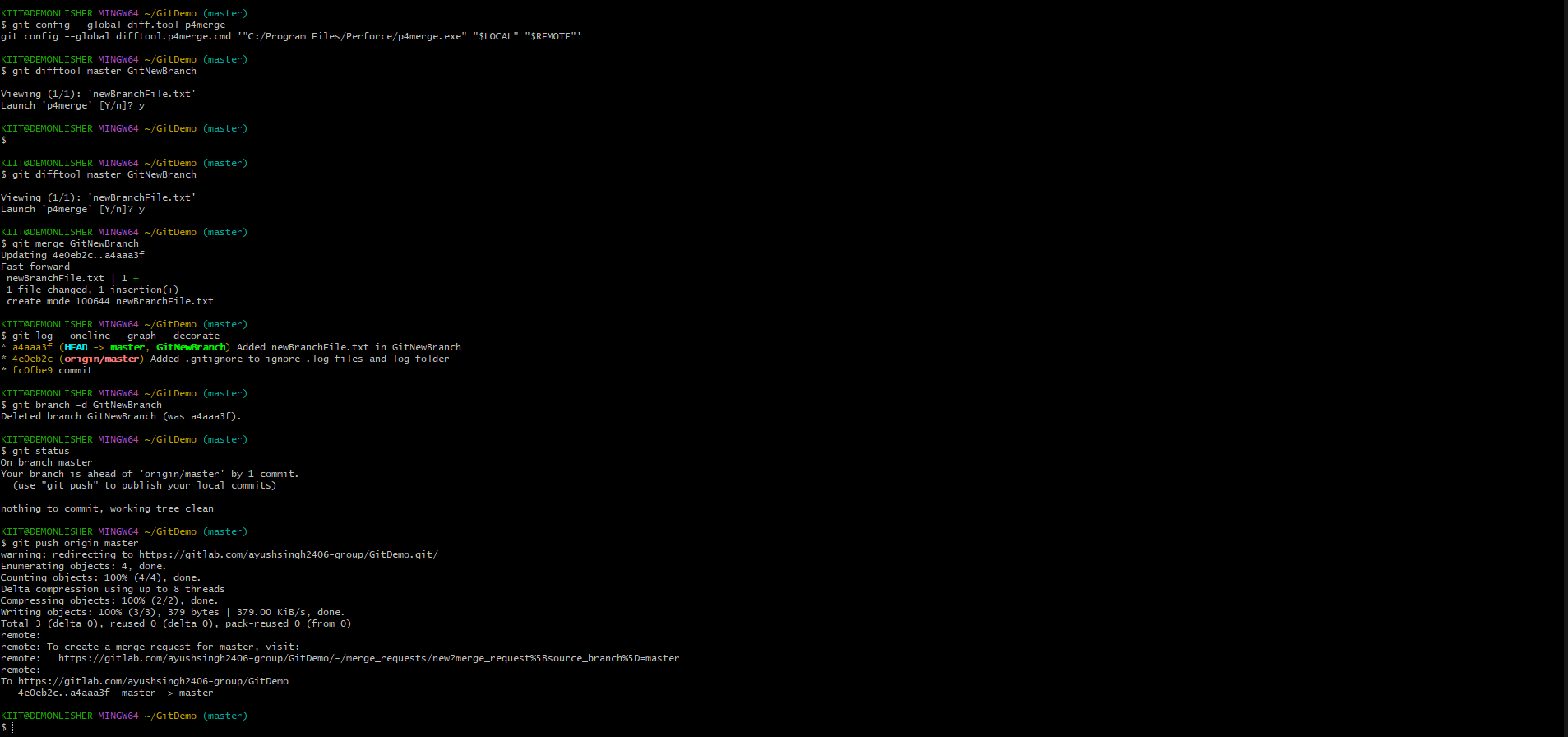
In GitLab, there is no separate concept of a 'branch request'. Instead, you create a branch manually or via the GitLab interface and start working on it.  
  
Steps to create a branch:  
1. Go to your GitLab project.  
2. Click on 'Repository' > 'Branches'.  
3. Click 'New branch'.  
4. Name your branch and choose the base branch (typically 'master' or 'main').  
5. Click 'Create branch'.  
Now you can push your changes to this new branch.

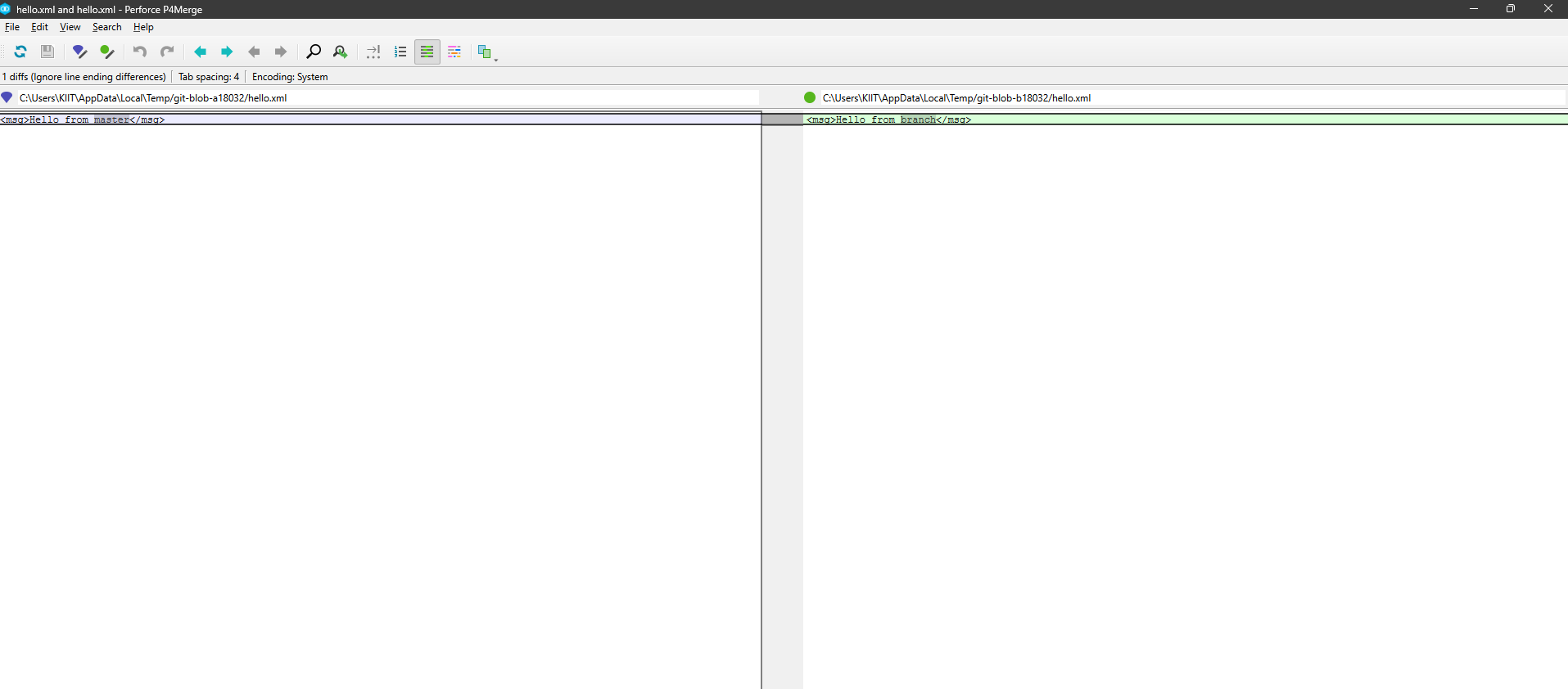
# 3. Explain about Creating a Merge Request in GitLab

A Merge Request (MR) in GitLab is used to propose changes from one branch into another and initiate a code review and approval process.

Steps to create a Merge Request:  
1. Push your feature or topic branch to GitLab.  
2. Go to your project repository.  
3. Click on 'Merge Requests' > 'New merge request'.  
4. Select source branch (your feature branch) and target branch (usually master or main).  
5. Click 'Compare branches and continue'.  
6. Add a title, description, and reviewers.  
7. Click 'Create merge request'.  
After review, the changes can be merged if approved.







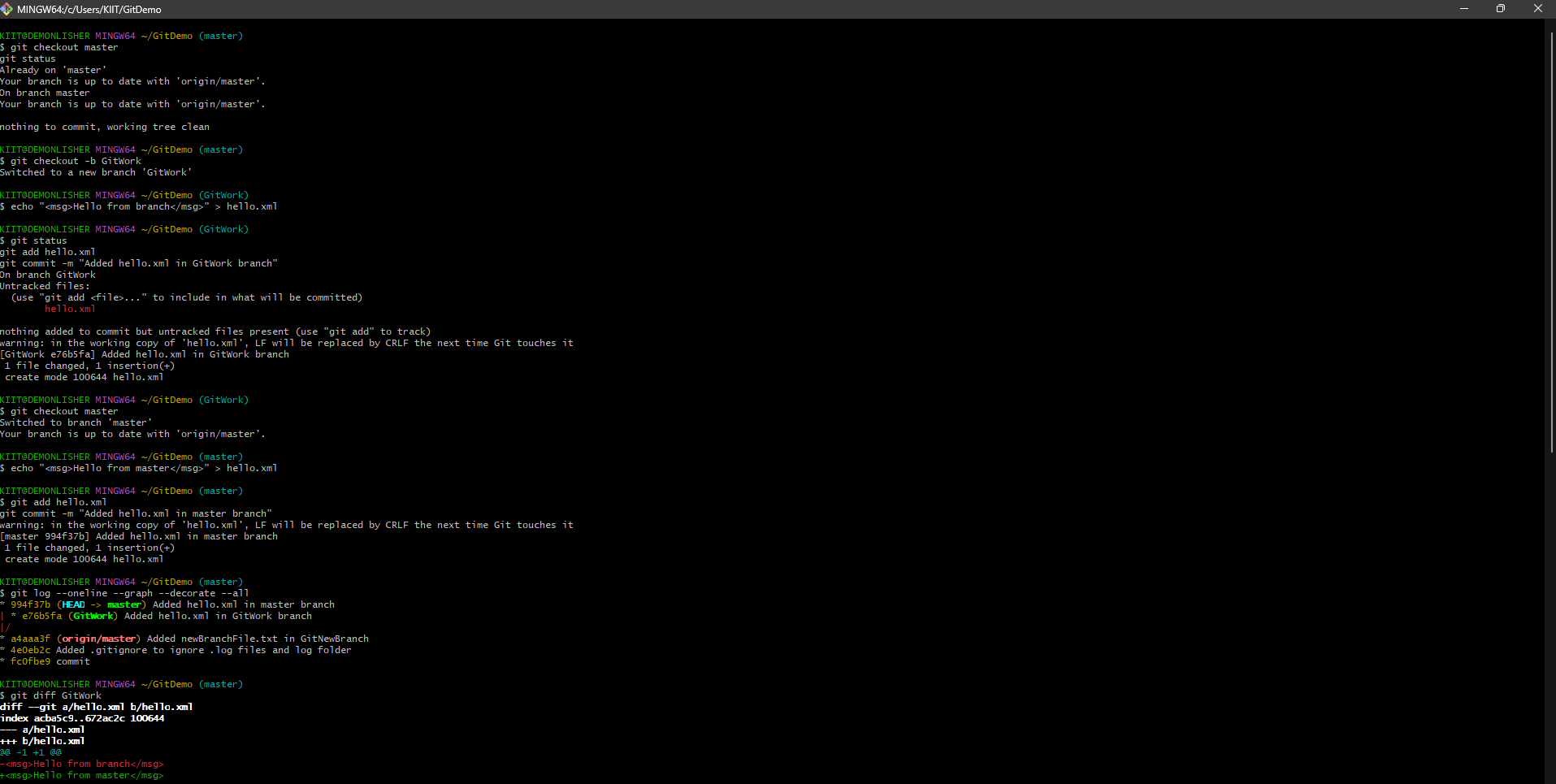
**HANDS ON 4: MERGE CONFLICT**

# How to Resolve the Conflict During Merge

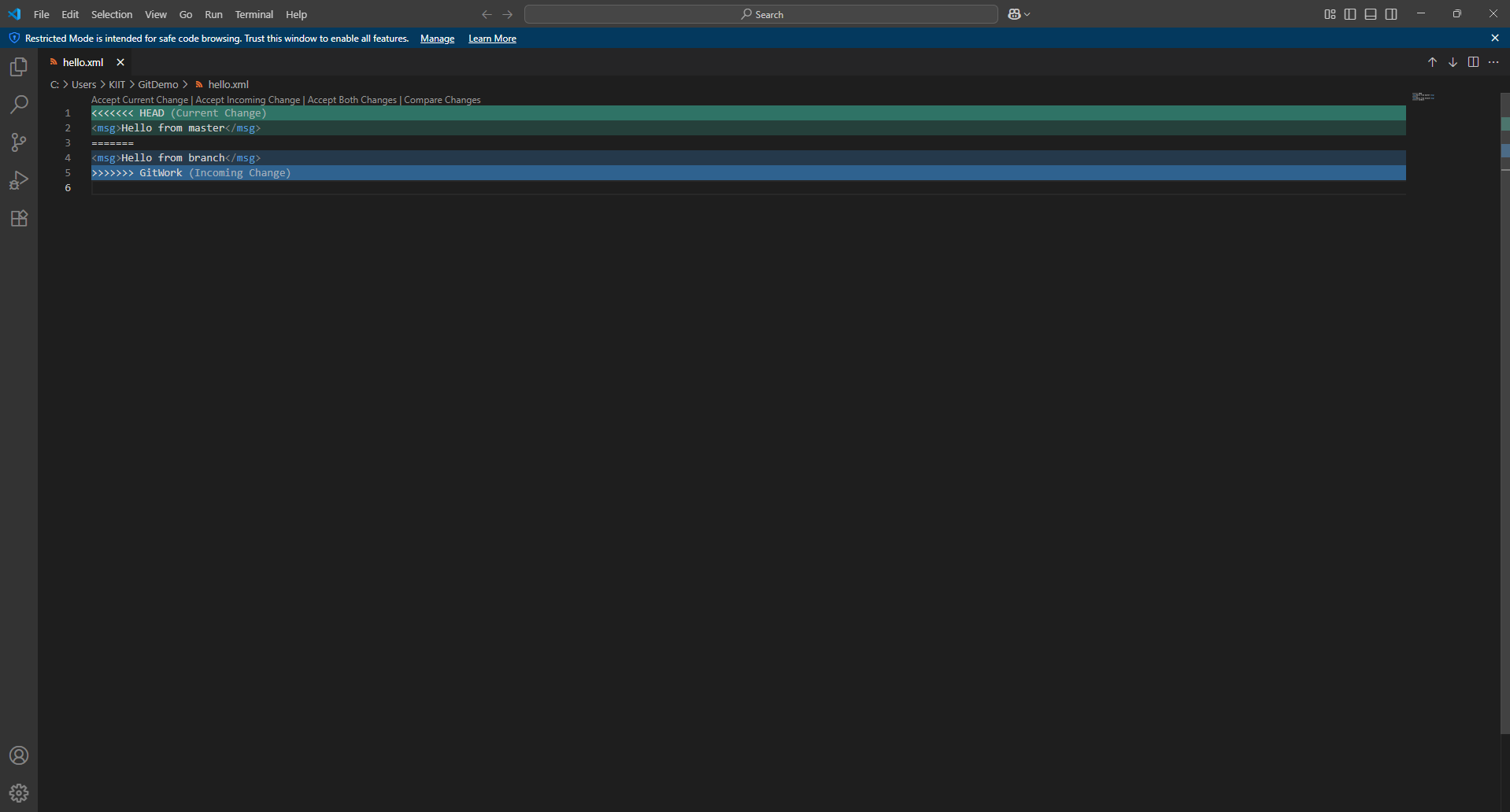
A merge conflict occurs when two branches have changes to the same line(s) in a file or when one branch deletes a file that the other has modified.  
  
Git cannot automatically determine which changes to keep, so manual intervention is needed.

1. Steps to resolve a merge conflict:

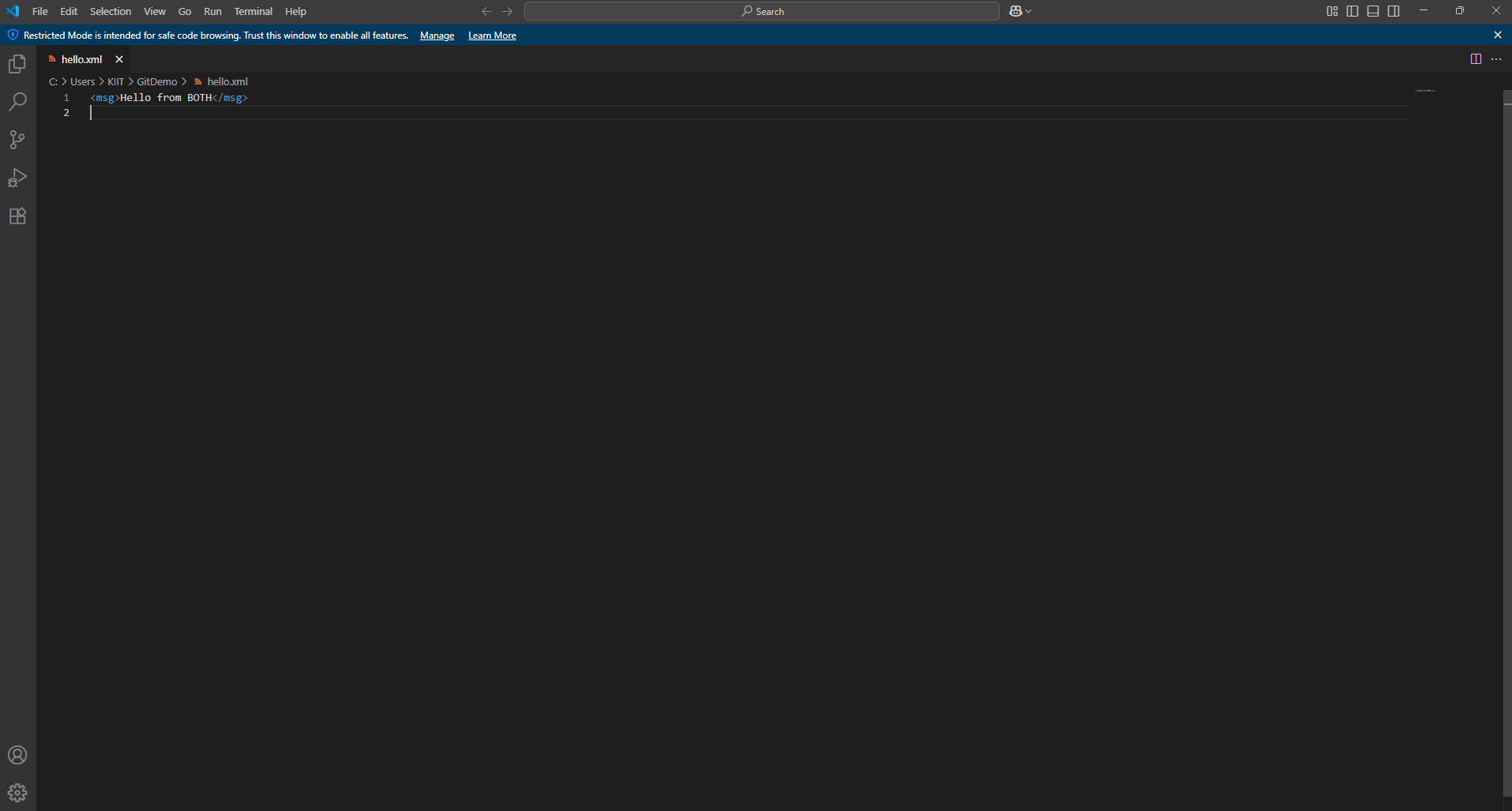
* Start a merge that leads to a conflict:
* git merge <branch-name>
* Git will stop the merge and indicate which files have conflicts.
* Open the conflicted file. You'll see conflict markers like:
* <<<<<<< HEAD  
  Changes from the current branch  
  =======  
  Changes from the branch being merged  
  >>>>>>> branch-name
* Manually edit the file to keep the desired content and remove conflict markers.
* Mark the conflict as resolved:
* git add <conflicted-file>
* Commit the merge:
* git commit -m "Resolved merge conflict"
* Optional: Use a 3-way merge tool (e.g., VS Code, P4Merge) to simplify conflict resolution.

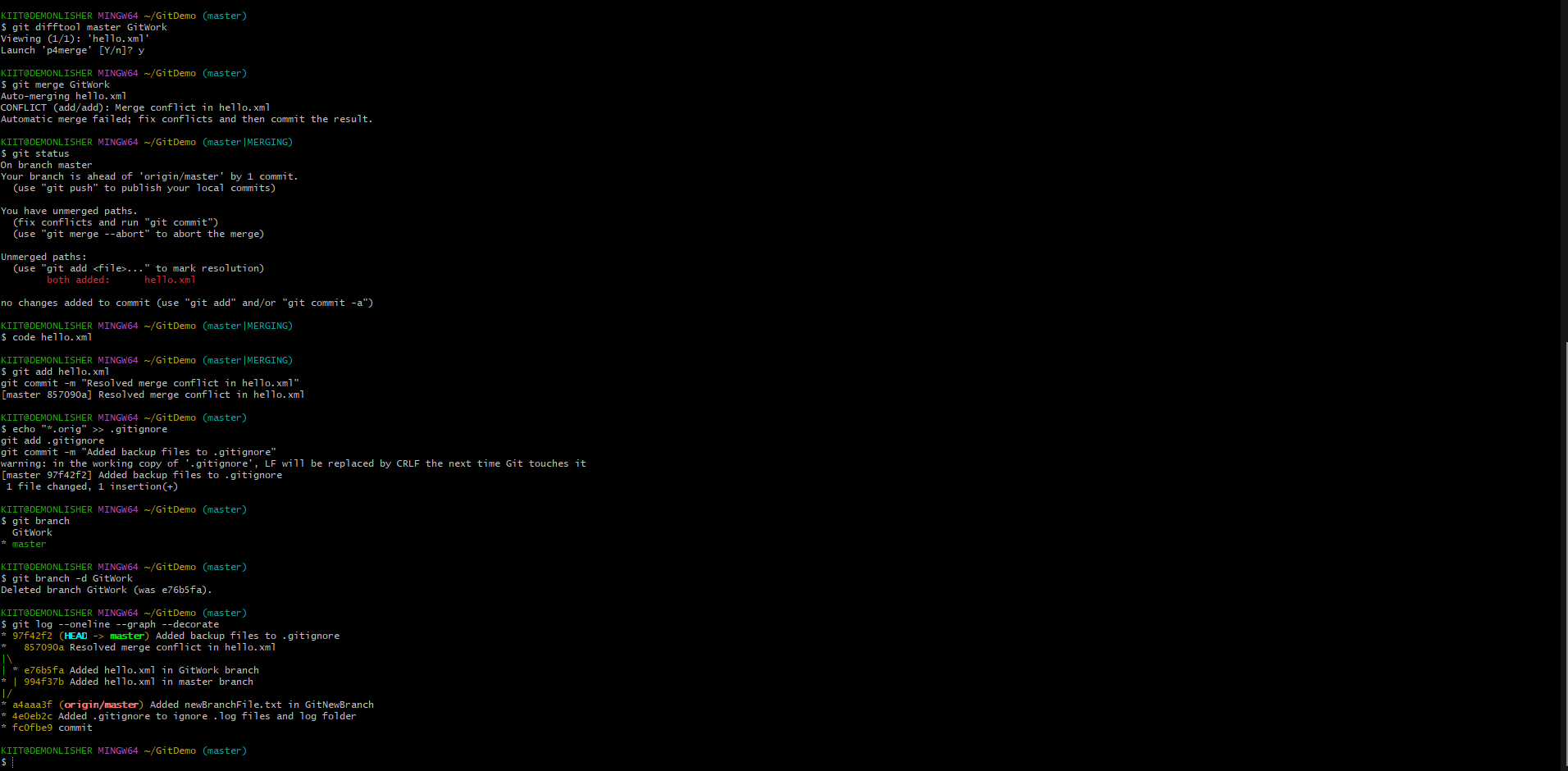


Conflict Found:



Conflict resolved:





**HANDS ON 5: CLEANUP AND PUSH BACK**

# How to Clean Up and Push Back to Remote Git

Cleaning up and pushing back to a remote Git repository involves ensuring your local repository is up to date, removing unnecessary branches, committing any remaining work, and synchronizing with the remote repository.

1. Steps to clean up and push to remote Git:

* 1. Check if your working directory is clean:
* git status
* 2. Add and commit any pending changes:
* git add .
* git commit -m "Your commit message"
* 3. Delete any branches that have already been merged:
* git branch -d branch-name
* (use -D for force delete if needed)
* 4. Pull the latest changes from the remote:
* git pull origin master
* 5. Push your changes to the remote repository:
* git push origin master

