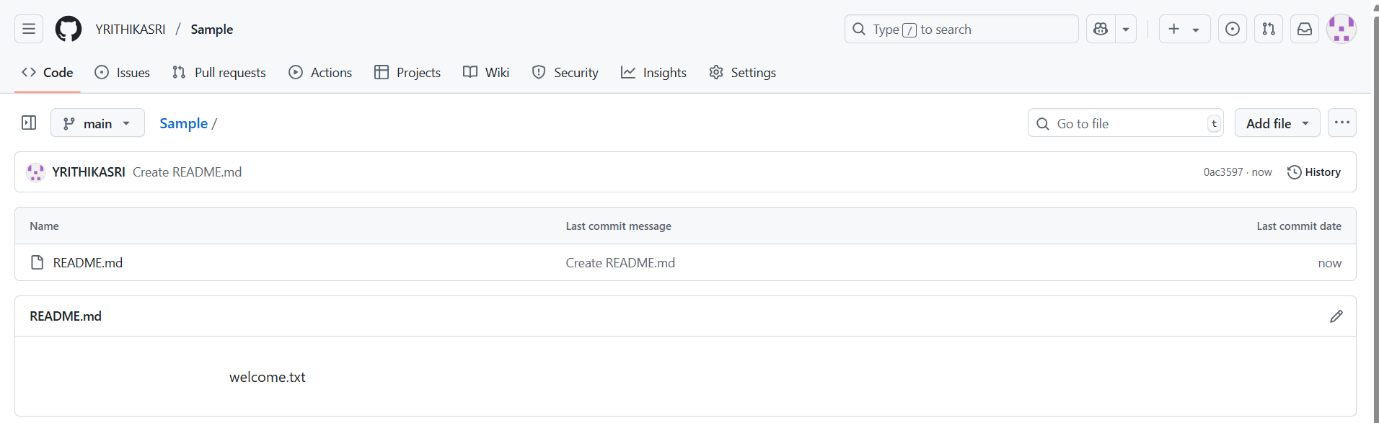
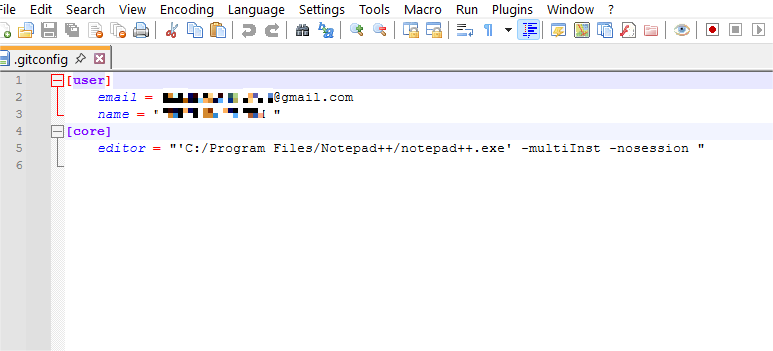
**HANDSON 1: GIT CONFIGURATION**

**OUTPUT:**

****

****

**HANDSON 2: IGNORE COMMAND**

**README**

# .gitignore in Git

## What is .gitignore?

- The .gitignore file is a special file in Git that tells Git which files or directories to ignore.

- Ignored files are not tracked by Git and will not appear in commits.

- This is useful for excluding temporary files, build artifacts, logs, and sensitive information from version control.

## Why use `.gitignore`?

- To keep the repository clean and free from unnecessary files.

- To prevent committing files that are machine-specific or environment-specific.

- To avoid pushing sensitive data such as passwords, API keys, or configuration files.

## How to Ignore Files Using `.gitignore`

1. Create a .gitignore file in the root of your Git repository.

2. Add patterns for files or directories you want to ignore.

   Examples:

   Ignore all .log files

      \*.log

   Ignore the log directory

      /log/

   Ignore a specific file

    secret.txt

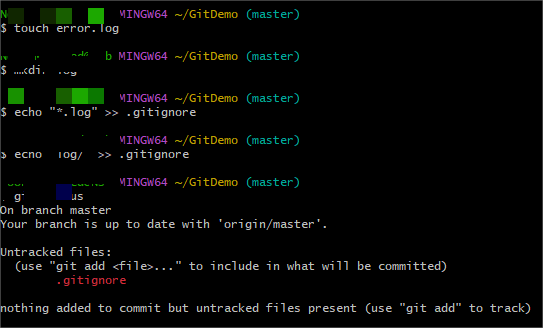
3. Save the .gitignore file.

4. Stage and commit the .gitignore file:

    git add .gitignore

    git commit -m "Added .gitignore to exclude unwanted files"

**OUTPUT**

****

**HANDSON 3: WORKING WITH BRANCH**

**README**

# Branching and Merging in Git & Merge Requests in GitLab

## Branching in Git

- \*\*Definition\*\*: A branch in Git is a separate line of development that allows you to work on features or fixes without affecting the main codebase.

- \*\*Purpose\*\*:

  - Work independently on new features.

  - Experiment without risking production code.

  - Collaborate efficiently in teams.

### Creating a Branch

git branch new-branch

This creates a new branch but does not switch to it.

To switch:

git checkout new-branch

Or create and switch in one step:

git checkout -b new-branch

## Merging in Git

Definition: Merging combines changes from one branch into another.

Process:

Switch to the target branch (e.g., master or main).

### Merge the source branch:

git merge source-branch

### Merge Conflicts:

Occur when changes overlap between branches.

Must be resolved manually before completing the merge.

Creating a Branch Request in GitLab

Branch Request: In GitLab, you typically create a branch in the remote repository to work on a feature or issue.

### Steps:

Go to your GitLab project.

Navigate to Repository → Branches.

Click New branch.

Enter the branch name and select the source branch (usually main or master).

Click Create branch.

### Creating a Merge Request in GitLab

Merge Request (MR): A request to merge changes from one branch into another in GitLab.

### Purpose:

Review code before merging.

Discuss and collaborate on changes.

Ensure quality and approval process.

## Steps:

In GitLab, go to your project.

Click Merge requests → New merge request.

Select the source branch (the one with your changes) and the target branch (where changes should go).

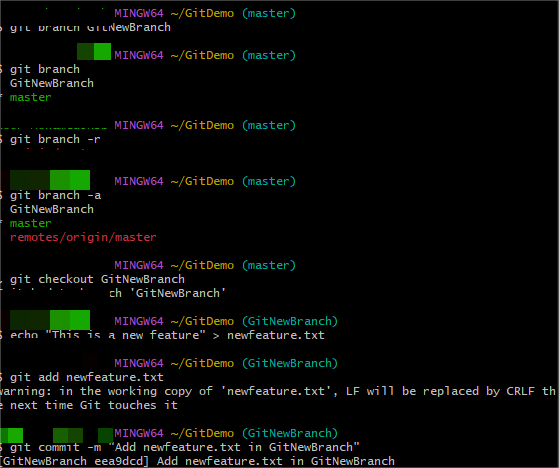
Add a title and description explaining the changes.

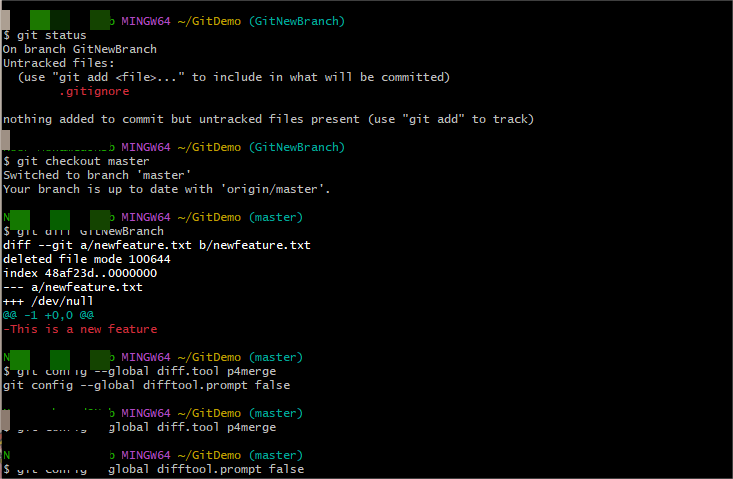
Assign reviewers or approvers.

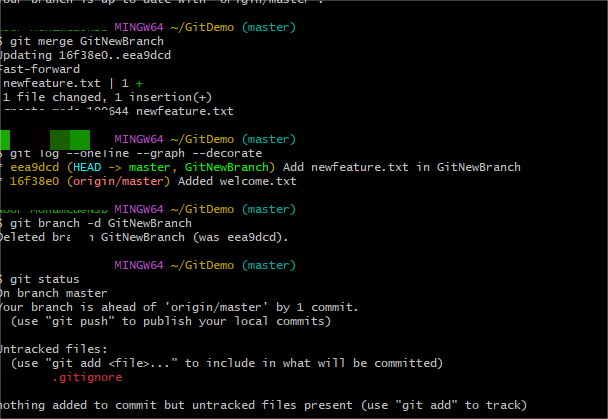
Click Submit merge request.

After review and approval, the MR can be merged into the target branch.

**OUTPUT**

****





**HANDSON 4: CONFLICT RESOLUTION**

**README**

# How to Resolve Merge Conflicts in Git

## What is a Merge Conflict?

A merge conflict occurs when Git cannot automatically combine changes from two branches.

Common causes:

- The same line in a file was modified in both branches.

- A file was deleted in one branch but modified in another.

---

## Steps to Resolve a Merge Conflict

### 1. Identify the Conflict

Run:

git merge branch-name

If there’s a conflict, Git stops merging and displays:

CONFLICT (content): Merge conflict in <file>

Automatic merge failed; fix conflicts and then commit the result.

Check conflicted files:

git status

### 2. Open the Conflicted Files

Git marks the conflicting sections as:

Code from your current branch

Code from the branch you are merging

branch-name

HEAD → Changes from your branch.

branch-name → Changes from the branch being merged.

### 3. Resolve the Conflict

Manually edit the file to:

Keep only your changes.

Keep only the other branch’s changes.

Combine both changes.

Example resolved content:

Final merged version of the code.

### 4. Mark as Resolved

After editing:

git add <file>

### 5. Complete the Merge

Finish the merge:

git commit

Git creates a merge commit and finalizes the process.

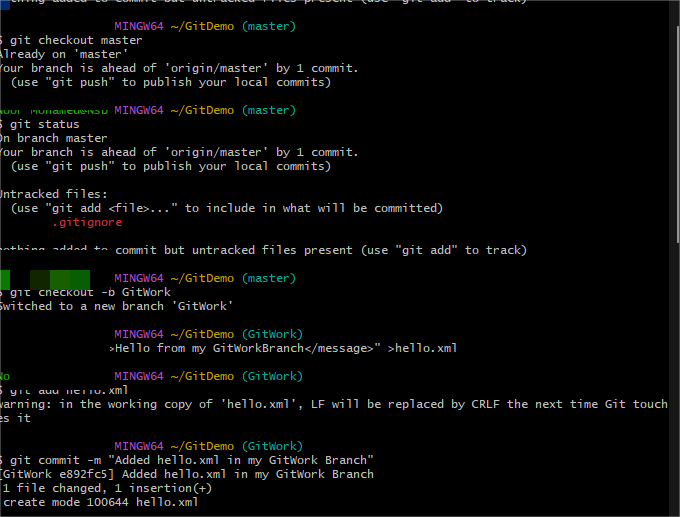
Optional: Use a Merge Tool

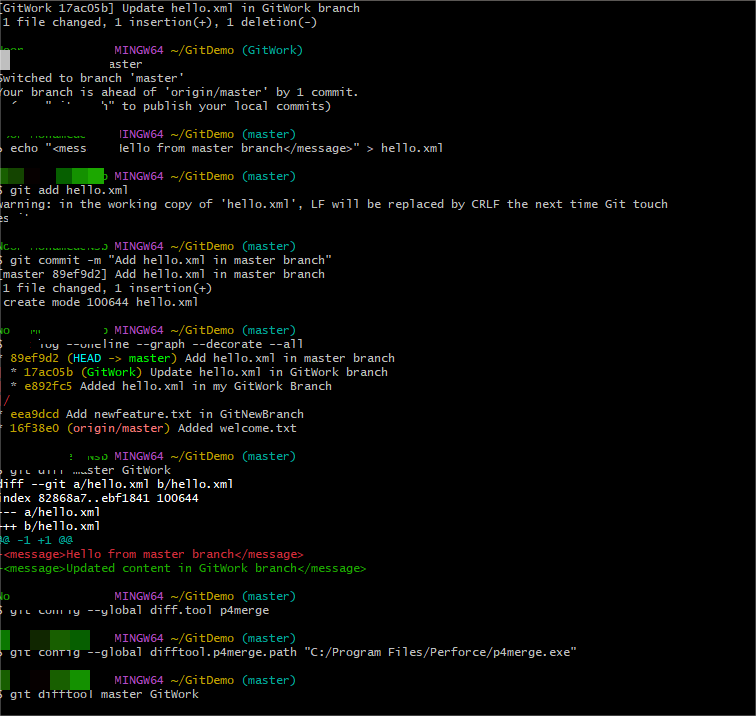
For visual conflict resolution:

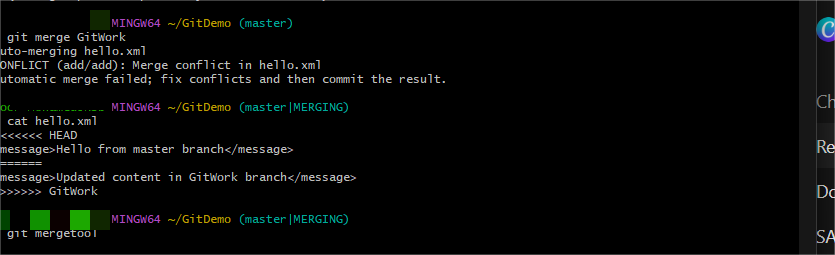
git mergetool

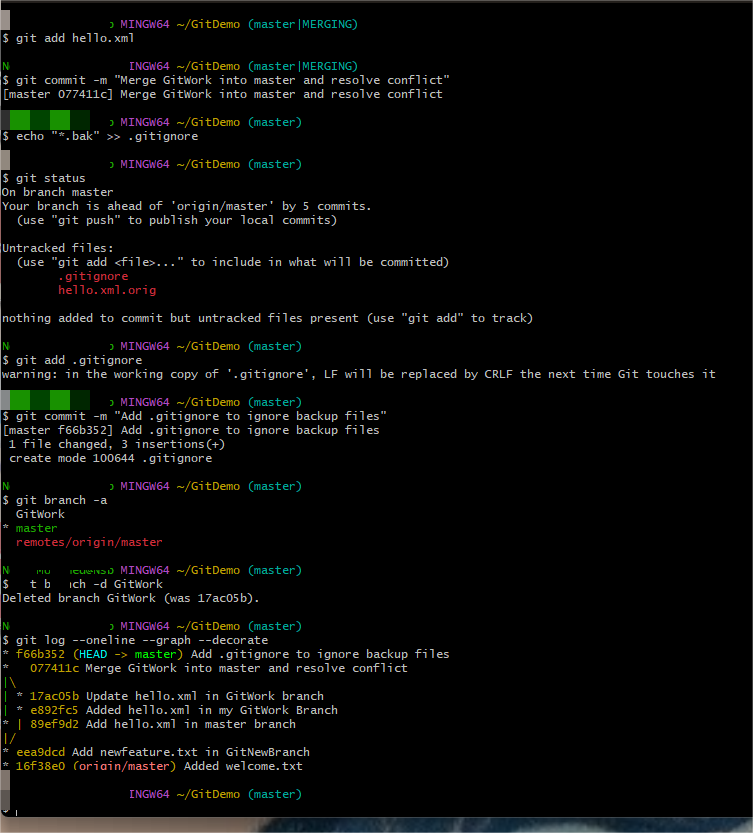
Tools like P4Merge, VS Code, or Meld make this easier.

**OUTPUT**

****

****

****

****

**HANDSON 5: CLEANUP AND PUSH BACK**

**README**

# How to Clean Up and Push Back to Remote Git

## Purpose

Over time, a Git repository may accumulate unnecessary files, outdated branches, or unmerged changes.

Cleaning up ensures the repository stays organized and that only relevant changes are pushed to the remote.

---

## 1. Clean Up Local Changes

### Remove Untracked Files and Folders

To delete files not tracked by Git:

git clean -f

To delete untracked files and directories:

git clean -fd

Note: This is irreversible — files will be permanently deleted.

### 2. Delete Unused Local Branches

List local branches:

git branch

Delete a branch that has been merged:

git branch -d branch-name

Force-delete an unmerged branch:

git branch -D branch-name

### 3. Prune Stale Remote Tracking Branches

To remove references to branches that no longer exist on remote:

git fetch --prune

### 4. Pull the Latest Changes

Ensure your local repository is up to date:

git pull origin main

(Replace main with your default branch name if different.)

### 5. Stage and Commit Updated Files

git add .

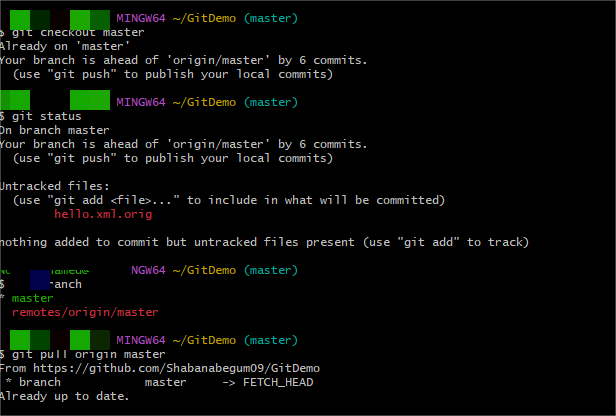
git commit -m "Cleanup: removed unused files and branches"

### 6. Push Back to Remote

Push the cleaned-up repository to the remote:

git push origin main

**OUTPUT**

****