1. **GIT\_HOL**

**Objectives**

This hands-on lab was performed to gain practical knowledge of the following Git concepts:

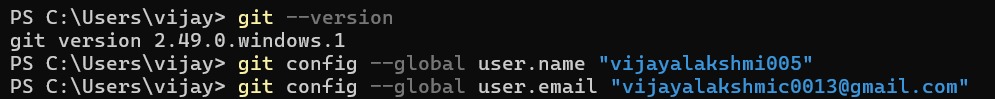
* Basic Git commands: git init, git status, git add, git commit, git push, git pull
* Git configuration setup on a local machine
* Integration of Notepad++ as the default Git editor
* Tracking and managing files in a local Git repository
* Pushing local repository changes to GitHub

**Prerequisites**

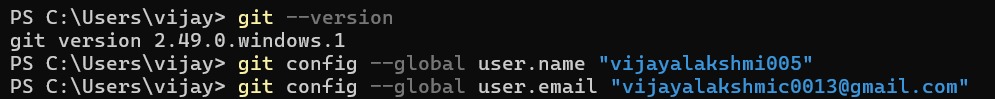
* Git Bash client installed on the local machine
* Notepad++ installed
* GitHub account created (used instead of GitLab for this exercise)

**Step 1: Git Configuration**

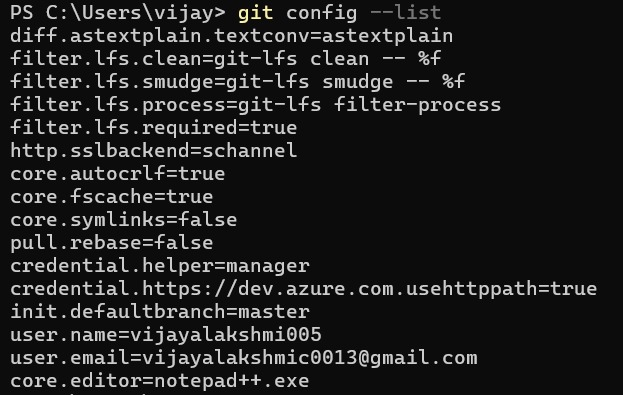
1. **Check Git installation:**



1. **Set user configuration:**

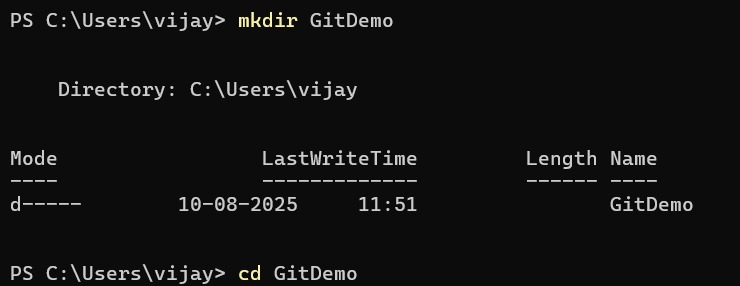


1. **Verify configuration:**



**Step 3: Create & Manage Git Repository**

1. Navigate to Week\_8\_GIT folder and create project directory:



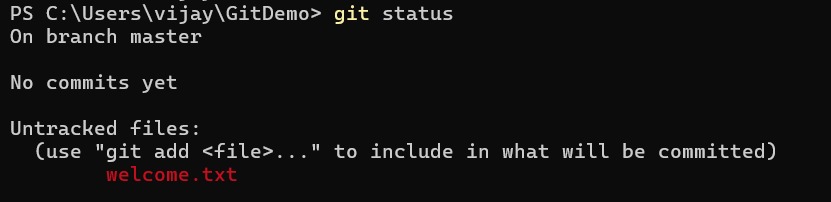
1. Initialize Git repository:



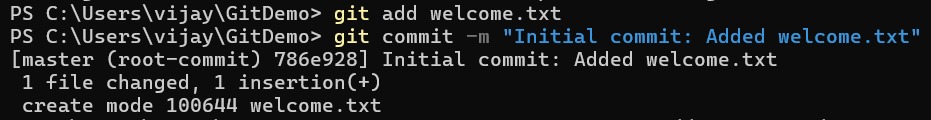
1. Create file and add content:



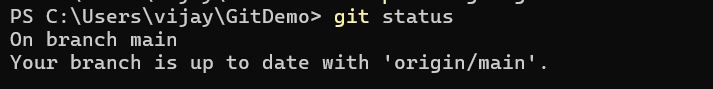
1. Check Git status:



1. Track and commit the file:



1. Check final status:

****

**Step 4: Push to GitHub Repository**



**2. GIT\_HOL**

**Objectives**

* Explain .gitignore
* Explain how to ignore unwanted files using .gitignore

**Prerequisites**

Before starting:

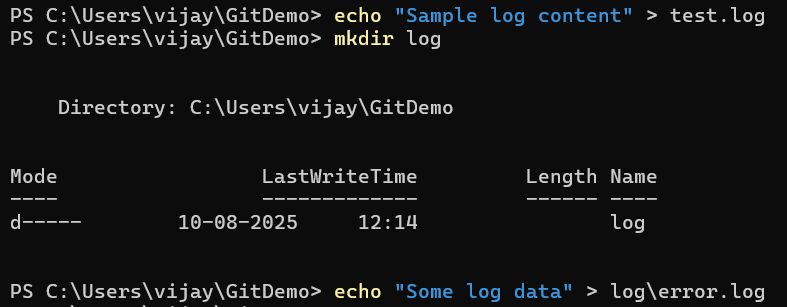
* Git installed and configured (git --version to check).
* Notepad++ integrated as Git’s default editor:
* git config --global core.editor "notepad++"
* A local Git repository and a linked remote (GitHub or GitLab).

**Hands-On Steps**

**Step 1: Navigate to Working Directory**

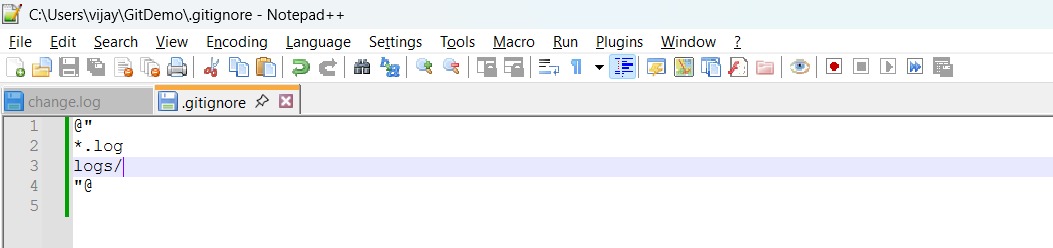


**Step 2: Create Sample Files**

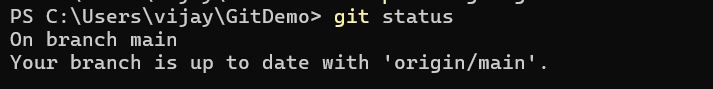
****

**Step 3: Create/Edit .gitignore**

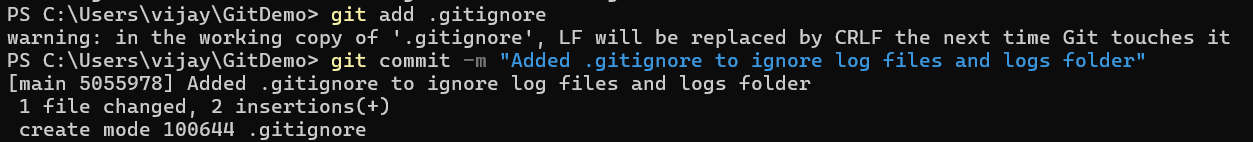
Add the following:

Save and close.

**Step 4: Check Git Status**

****

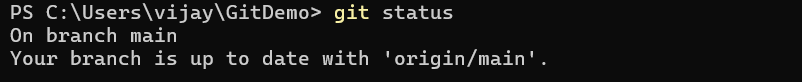
**Step 5: Commit the .gitignore**

****

**Step 6: Push to Remote**



**Output Example**



**3. GIT\_HOL**

**Objectives**

* Understand what .gitignore is and why it’s used.
* Learn how to ignore unwanted files and folders in Git.
* Configure **P4Merge** as a visual diff and merge tool.
* Practice creating branches, making changes, and comparing them visually.

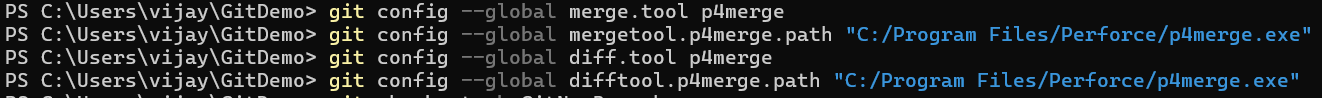
**Prerequisites**

* Git installed and configured on the system.
* **P4Merge** installed and set as Git’s default difftool/mergetool.
* Notepad++ integrated as Git’s default editor.
* A Git repository created locally and linked to a remote (GitHub/GitLab).

**Tasks Performed**

**1. Setting Up P4Merge**

1. Download and install **P4Merge** from  
   <https://www.perforce.com/downloads/visual-merge-tool>
2. Configure Git to use P4Merge:



**2. Creating a Git Repository**

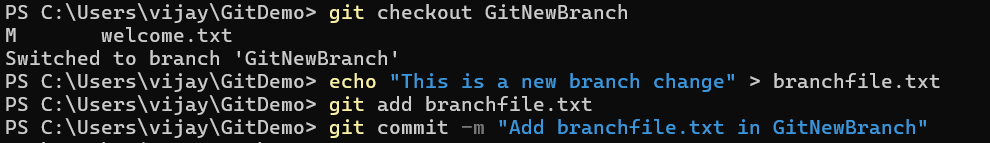
****

**3. Branching and Visual Comparison**

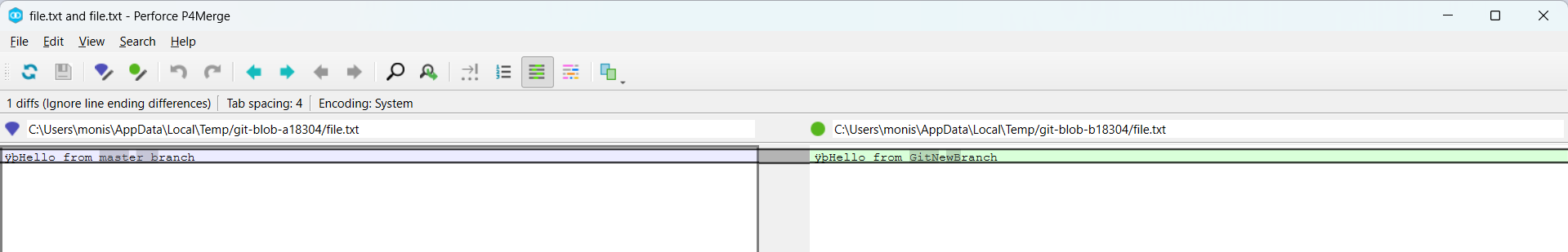
1. Created and switched to a new branch:



1. Created a new file:

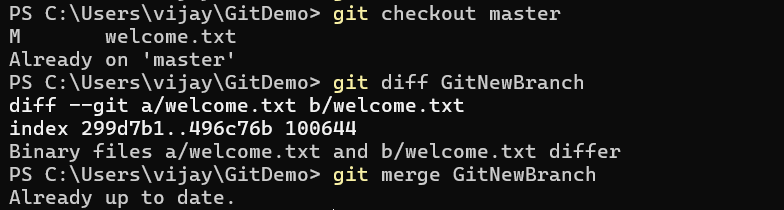
Compared branches visually:

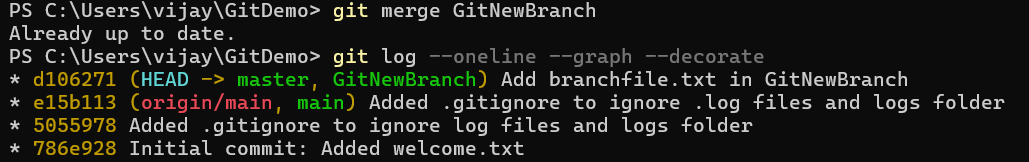
git difftool master GitNewBranch



**5. Merging**

1. Switched back to master and merged branch:

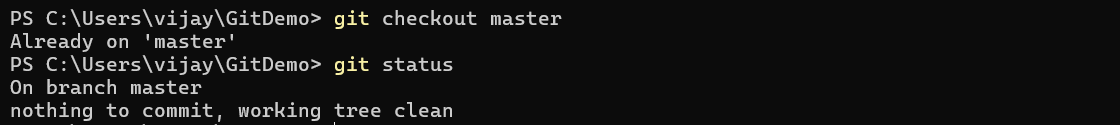




"This lab was performed in a local Git repository with GitHub as the remote host instead of GitLab, but the branching and merging process remains identical."

**4. GIT\_HOL**

**1. Verify if master is clean**



You should see nothing to commit, working tree clean.

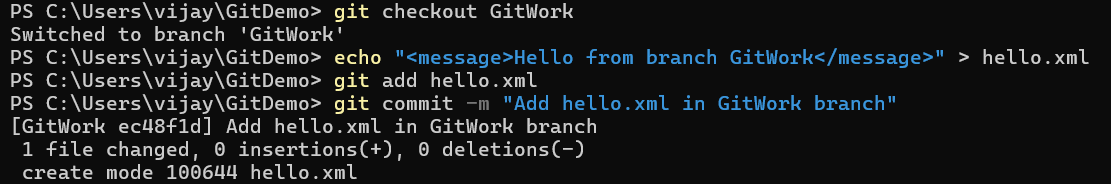
**2. Create branch GitWork and add hello.xml**

git checkout -b GitWork

echo "<message>Hello from branch GitWork</message>" > hello.xml

git add hello.xml

git commit -m "Add hello.xml in GitWork branch"



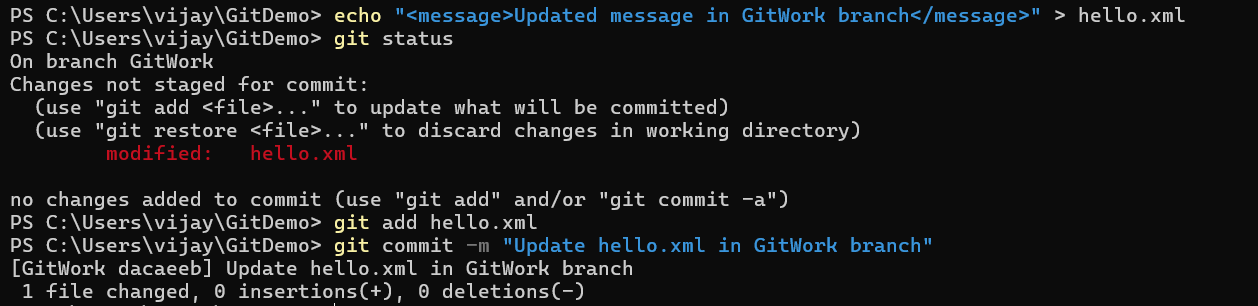
**3. Update hello.xml in branch**

echo "<message>Updated message in GitWork branch</message>" > hello.xml

git status

git add hello.xml

git commit -m "Update hello.xml in GitWork branch"



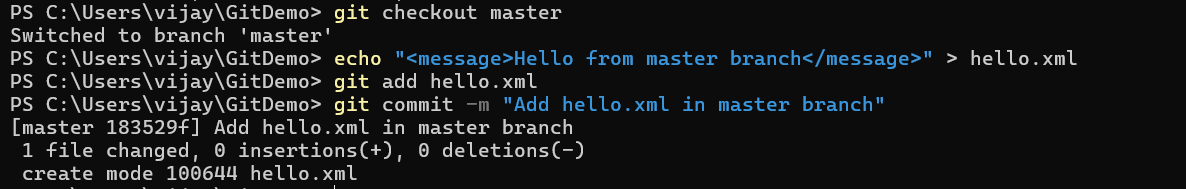
**4. Switch to master and create a different hello.xml**

git checkout master

echo "<message>Hello from master branch</message>" > hello.xml

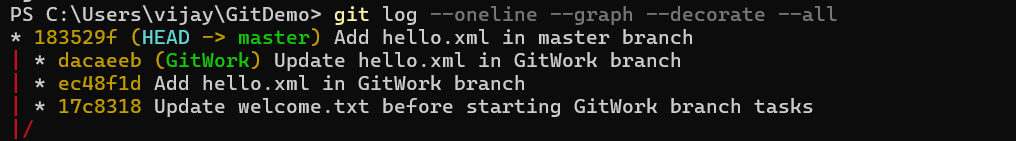
git add hello.xml

git commit -m "Add hello.xml in master branch"



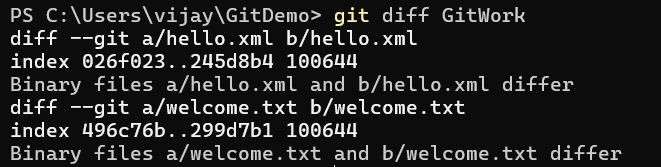
**5. Observe logs**

git log --oneline --graph --decorate –all



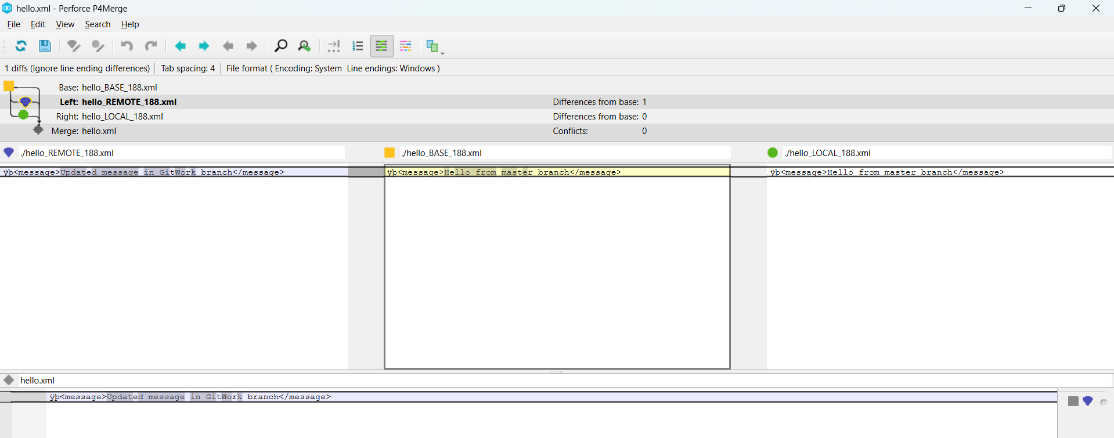
**6. Check differences**

git diff GitWork



**7. Visual diff with P4Merge**

git mergetool



**8. Merge GitWork into master**

git merge GitWork

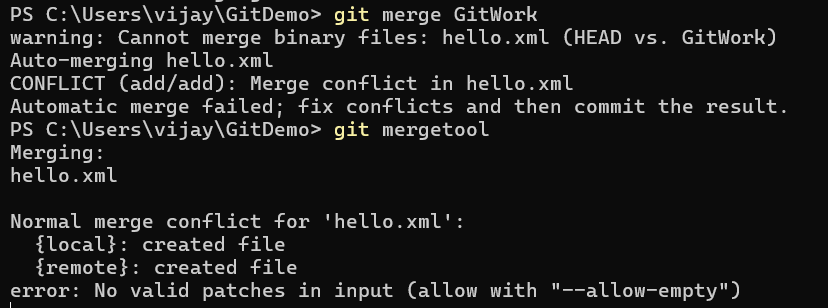
You should get a **merge conflict**.

**9. Resolve the conflict (3-way merge)**

* Open P4Merge automatically by:

git mergetool

* Keep the correct changes manually, save, and close.

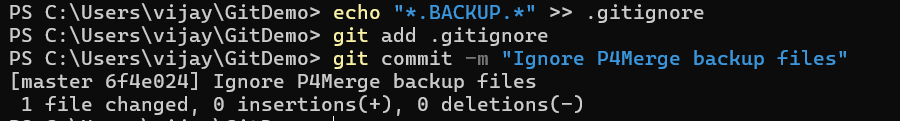


**10. Ignore backup files from P4Merge**

echo "\*.BACKUP.\*" >> .gitignore

git add .gitignore

git commit -m "Ignore P4Merge backup files"



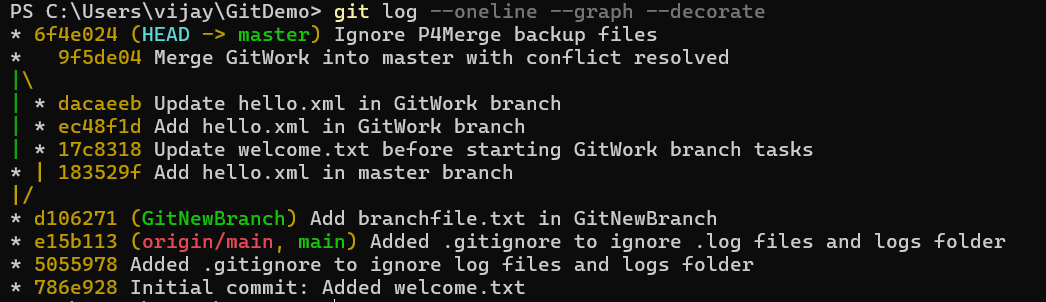
**11. Delete branch**

git branch -d GitWork



**13. Final log**

git log --oneline --graph –decorate



**5. GIT\_HOL**

**Objectives**

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

**In this hands-on lab, you will learn to**

* Execute steps involving clean up and push back to remote Git.

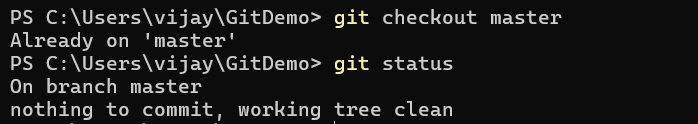
**Estimated time: 10 minutes**

**Steps to Perform**

1. **Verify if master is in a clean state**

git checkout master

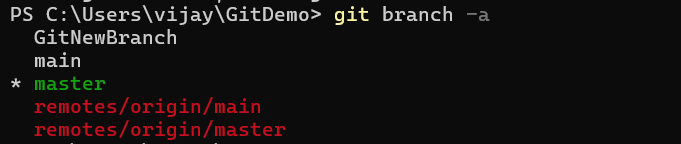
git status



If you see "nothing to commit, working tree clean," you’re good to proceed.

**2️⃣ List all available branches**

git branch -a



The -a option shows both local and remote branches.

**3️⃣ Pull the latest changes from the remote repository**

git pull origin master



This ensures your master is up to date with the remote before pushing.

**4️⃣ Push pending changes from Git-T03-HOL\_002**

If you made changes in a branch during **Git-T03-HOL\_002**, switch to that branch and push it:

If changes were already merged into master, simply:

git push origin master



**5️⃣ Verify changes in the remote repository**

* Go to your GitHub (or GitLab) repository in the browser.
* Check if the pushed commits are visible in the branch or master.
* Confirm file changes are updated.