1. **Objectives**

Familiar with Git commands like git init, git status, git add, git commit, git push, and git pull.

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

* Setup your machine with Git Configuration
* Integrate notepad++.exe to Git and make it a default editor
* Add a file to source code repository

**Prerequisites**

* Install Git Bash client in your machine

Notes\*:

|  |
| --- |
| Please follow the below steps for creating a free account in GitHub.  Don’t use cognizant credentials to login to GitHub. |

Estimated time to complete this lab: **30 minutes.**

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

To create a new repository, signup with GitLab and register your credentials

Login to GitLab and create a “GitDemo” project

1. To check if Git client is installed properly: Open Git bash shell and execute



If output shows Git with its version information that indicates, that Git Client installs properly.

1. To configure user level configuration of user ID and email ID execute



1. To check if the configuration is properly set, execute the following command.



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

1. To check, if notepad++.exe execute from Git bash



If Git bash could not able to recognize notepad++ command that implies notepad++.exe is note added to the environment path variable.

To add path of notepad++.exe to environment variable, go to control panel -> System -> Advanced System settings. Go to Advanced tab -> Environment variables -> Add path of notepad++.exe to the path user variable by clicking on “Edit”



1. Exit Git bash shell, open bash shell and execute



Now, notepad++ will open from Git bash shell

1. To create an alias command for notepad++.exe, execute



It will open notepad++ from bash shell, and create a user profile by adding the line in notepad++



1. To configure the editor, execute the command



1. To verify if notepad++ is the default editor, execute the command



Here ‘-e’ option implies editor

It will show the entire global configuration as shown below,



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

1. Open Git bash shell and create a new project “**GitDemo**” by executing the command



1. Git bash initializes the “**GitDemo**” repository. To verify, execute the command



It will display all the hidden files in the Git “working directory”.

1. To create a file **“welcome.txt”** and add content to the file, execute the command



1. To verify if the file “welcome.txt” is created, execute



1. To verify the content, execute the command



1. Check the status by executing



Now the file **“welcome.txt”** is available in Git “working directory”

1. To make the file to be tracked by Git repository, execute the command



1. To add multi line comments, we are opening default editor to comment. Execute the command



Notepad++ editor will open and to add multi-line comment with default editor

1. To check if local and “Working Directory” git repository are same, execute git status



**welcome.txt** is added to the local repository.

1. Signup with GitLab and create a remote repository **“GitDemo”**
2. To pull the remote repository, execute

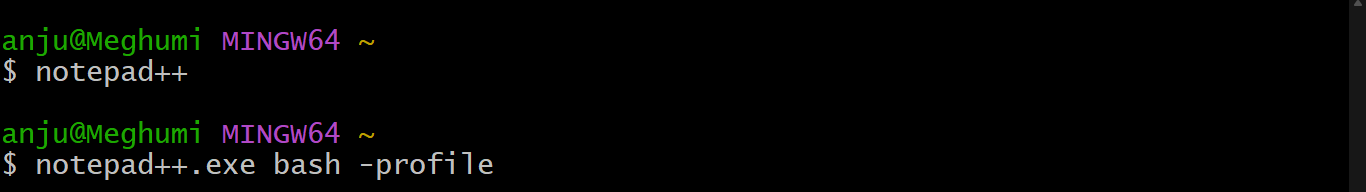
git pull origin master

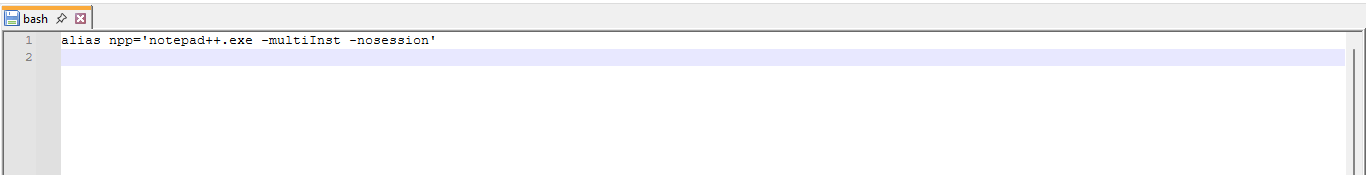
1. To push the local to remote repository, execute

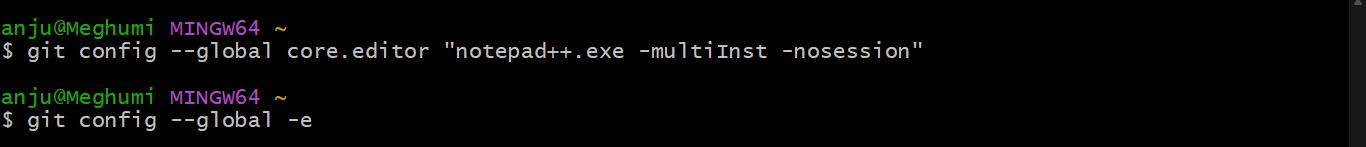
git push origin master

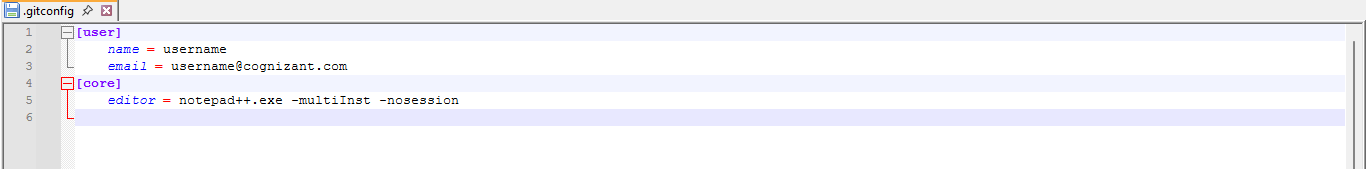
Answer:

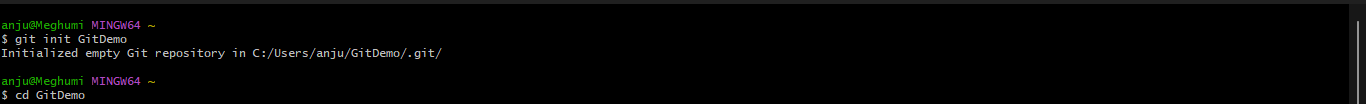


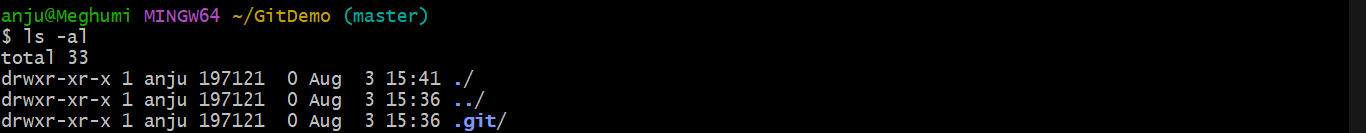


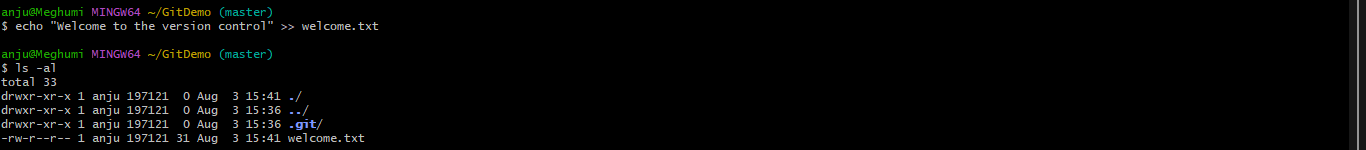


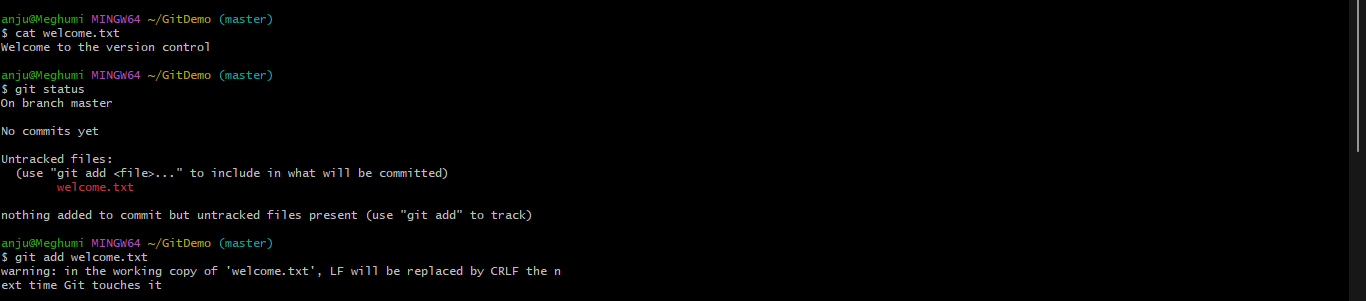


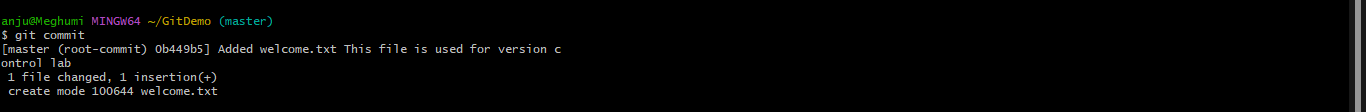


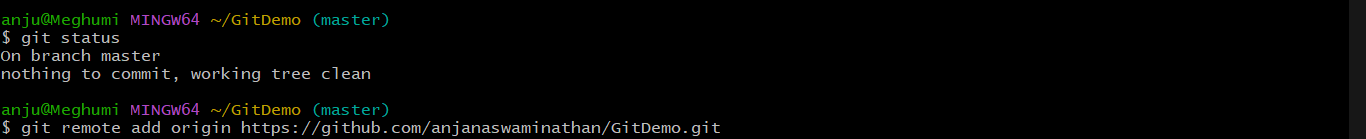


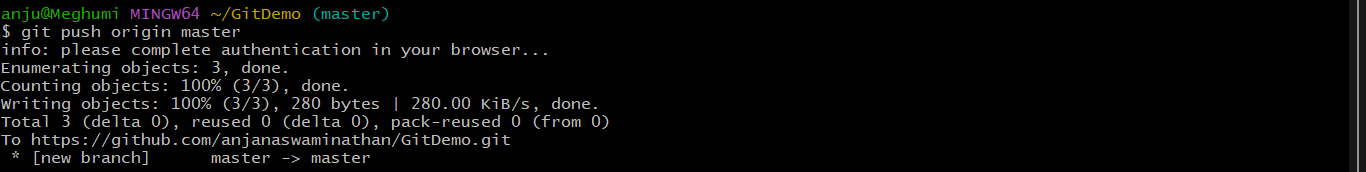












2. **Objectives**

* Explain git ignore
* Explain how to ignore unwanted files using git ignore

In this hands-on lab, you will learn how to:

* Implement git ignore command to ignore unwanted files and folders

**Prerequisites**

The following are the pre-requisites to complete this hands-on lab:

* Setting up Git environment
* Integrate notepad++ as a default editor
* A Git repository in the local system and a remote repository in GitLab

Notes\*:

|  |
| --- |
| Please follow the below steps for creating a free account in GitHub.  Do not use cognizant credentials to login to GitHub. |

Estimated time to complete this lab: **20 minutes.**

Create a **“.log”** file and a **log folder** in the working directory of Git. Update the **.gitignore** file in such a way that on committing, these files (.log extensions and log folders) are ignored.

Verify if the git status reflects the same about working directory, local repository and git repository.

Answer:

**1. Explain .gitignore**

.gitignore is a file used in Git to tell the system **which files or folders it should ignore** (not track or commit).  
This is helpful when you have files that are temporary, auto-generated, or not meant to be shared (like logs, caches, or configuration files).

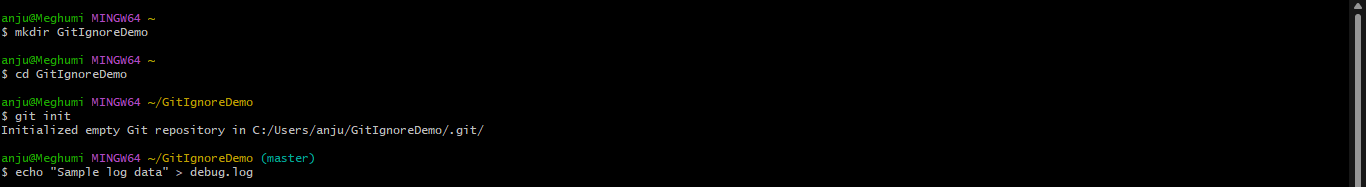
When Git sees the .gitignore file, it follows the instructions inside it and skips tracking any files that match the patterns written there.

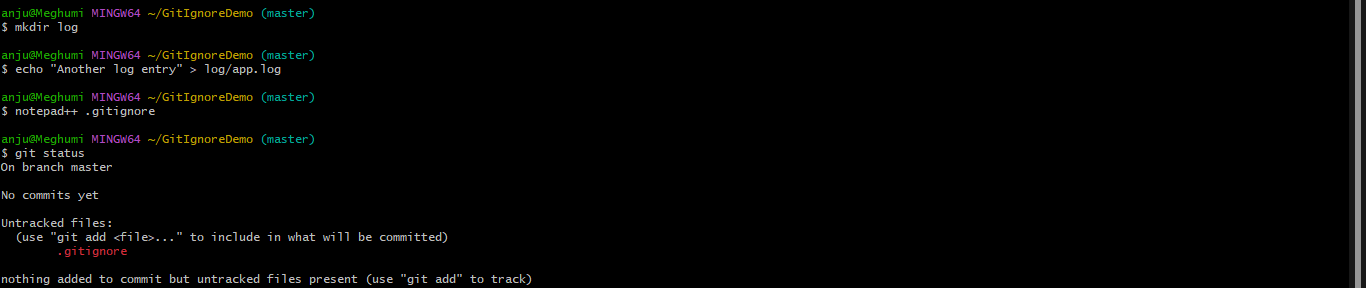
**2. How to Ignore Unwanted Files Using .gitignore**

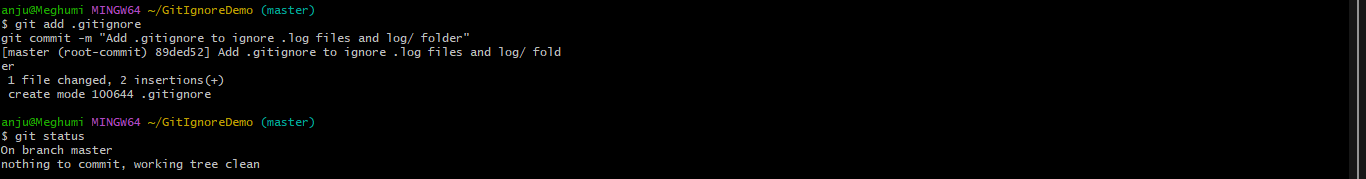
To ignore unwanted files:

1. **Create a .gitignore file** in the root folder of your Git project.
2. **Write the names or patterns** of files or folders you want Git to ignore.
3. Git will automatically **skip those files** during add/commit operations.

Once added to .gitignore, Git won't show these files in git status, and they won’t be committed to the repository.







3. **Objectives**

* Explain branching and merging
* Explain about creating a branch request in GitLab
* Explain about creating a merge request in GitLab

In this hands-on lab, you will learn how to:

* Construct a branch, do some changes in the branch, and merge it with master (or trunk)

**Prerequisites**

The following are the pre-requisites to complete this hands-on lab:

* Setting up Git environment with P4Merge tool for Windows

Notes\*:

|  |
| --- |
| Please follow the below steps for creating a free account in GitHub.  Do not use cognizant credentials to login to GitHub. |

Estimated time to complete this lab: **30 minutes.**

Please follow the instruction to complete the hands-on. Each instruction expects a command for the Git Bash.

**Branching:**

1. Create a new branch **“GitNewBranch”.**
2. List all the local and remote branches available in the current trunk. Observe the “\*” mark which denote the current pointing branch.
3. Switch to the newly created branch. Add some files to it with some contents.
4. Commit the changes to the branch.
5. Check the status with **“git status”** command.

**Merging:**

1. Switch to the master
2. List out all the differences between trunk and branch. These provide the differences in command line interface.
3. List out all the visual differences between master and branch using **P4Merge tool**.
4. Merge the source branch to the trunk.
5. Observe the logging after merging using **“git log –oneline –graph –decorate”**
6. Delete the branch after merging with the trunk and observe the git status.

**Answer:**

**1. Branching and Merging in Git**

**Branching**

* **Branching means creating a separate line of development from the main code (usually called master or main).**
* **You can work on new features or bug fixes without affecting the main codebase.**
* **Example: git branch feature-login creates a branch named feature-login.**

**Merging**

* **Merging means taking the changes from one branch (like feature-login) and combining them into another, usually into master.**
* **This helps bring new features or updates into the main project after testing.**
* **Command: git merge feature-login (while on master)**

**2. Creating a Branch Request in GitLab**

**A Branch Request is not a technical term in GitLab, but it usually means creating a new branch in the GitLab interface.**

**Steps:**

1. **Go to your GitLab project.**
2. **Click on Repository → Branches.**
3. **Click New Branch.**
4. **Enter the name of the new branch (e.g., GitNewBranch).**
5. **Select the branch to base it on (e.g., master).**
6. **Click Create Branch.**

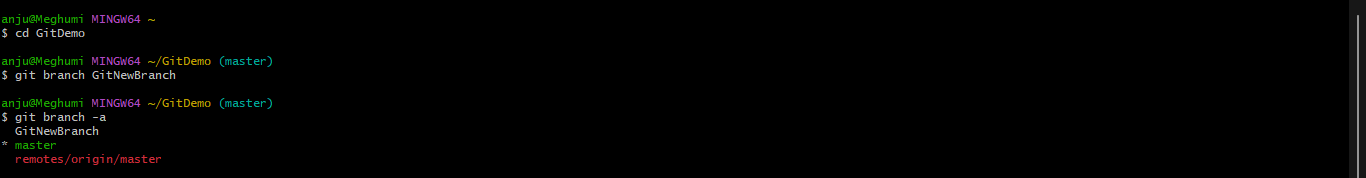
**Now you can push code changes to this branch using Git Bash.**

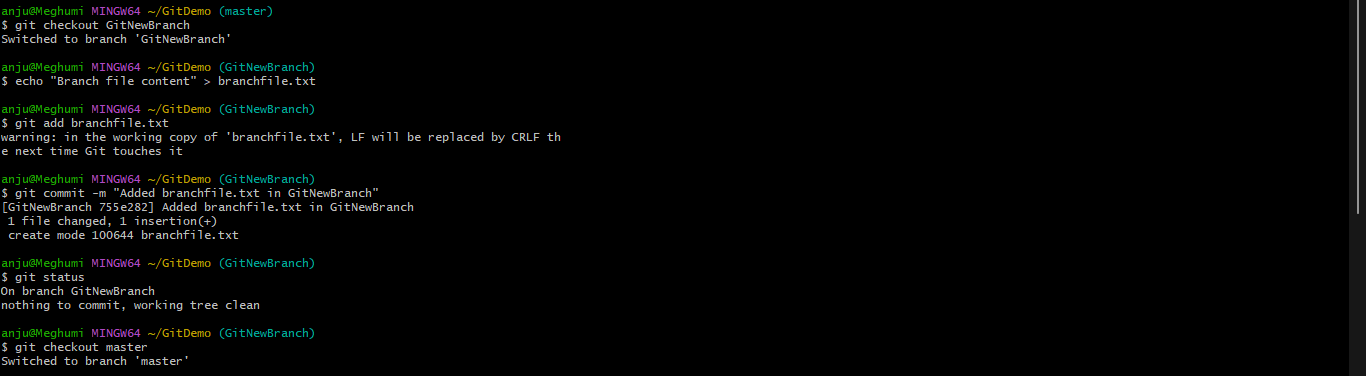
**3. Creating a Merge Request in GitLab**

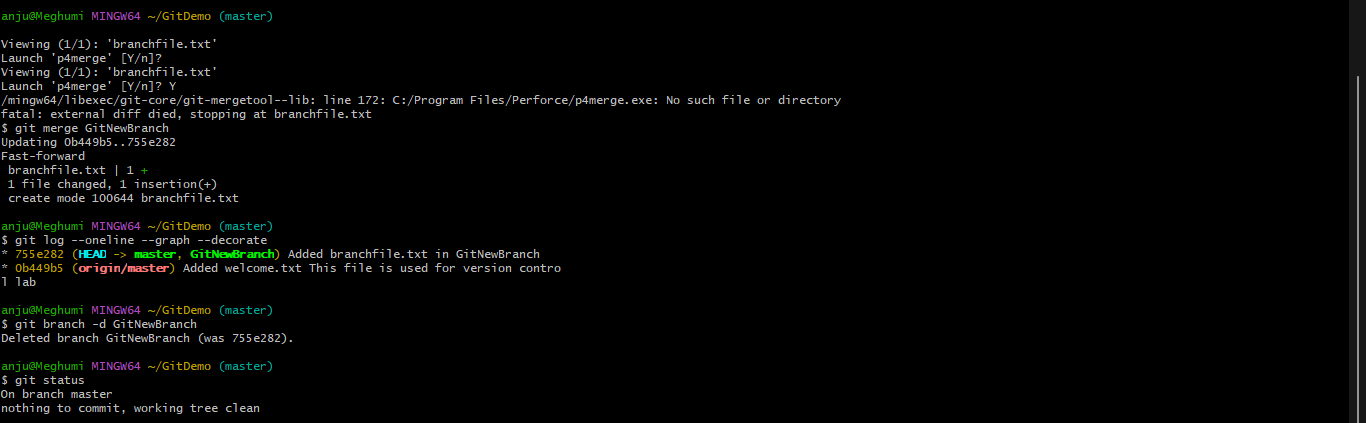
**A Merge Request (MR) is a way to ask GitLab to review and merge code from one branch to another (e.g., from GitNewBranch into master).**

**Steps:**

1. **Go to your GitLab project.**
2. **Click Merge Requests → New Merge Request.**
3. **Select:**
   * **Source branch: The one with your changes (e.g., GitNewBranch)**
   * **Target branch: Usually master or main**
4. **Click Compare branches and continue.**
5. **Fill in a title and description for your MR.**
6. **Click Create Merge Request.**
7. **After code review or approval, click Merge to complete.**

****

****

****

4. **Objectives**

* Explain how to resolve the conflict during merge.

In this hands-on lab, you will learn how to:

* Implement conflict resolution when multiple users are updating the trunk (or master) in such a way that it results into a conflict with the branch’s modification.

**Prerequisites**

The following are the pre-requisites to complete this hands-on lab:

* Hands-on ID: **“Git-T03-HOL\_001”**

Notes\*:

|  |
| --- |
| Please follow the below steps for creating a free account in GitHub.  Do not use cognizant credentials to login to GitHub. |

Estimated time to complete this lab: **30 minutes.**

Please follow the instructions to complete the hands-on. Each instruction expect a command for the Git Bash.

1. Verify if master is in clean state.
2. Create a branch **“GitWork”.** Add a file “hello.xml”.
3. Update the content of “hello.xml” and observe the status
4. Commit the changes to reflect in the branch
5. Switch to master.
6. Add a file **“hello.xml”** to the master and add some different content than previous.
7. Commit the changes to the master
8. Observe the log by executing **“git log –oneline –graph –decorate –all”**
9. Check the differences with Git diff tool
10. For better visualization, use P4Merge tool to list out all the differences between master and branch
11. Merge the bran to the master
12. Observe the git mark up.
13. Use 3-way merge tool to resolve the conflict
14. Commit the changes to the master, once done with conflict
15. Observe the git status and add backup file to the .gitignore file.
16. Commit the changes to the .gitignore
17. List out all the available branches
18. Delete the branch, which merge to master.
19. Observe the log by executing **“git log –oneline –graph –decorate”**

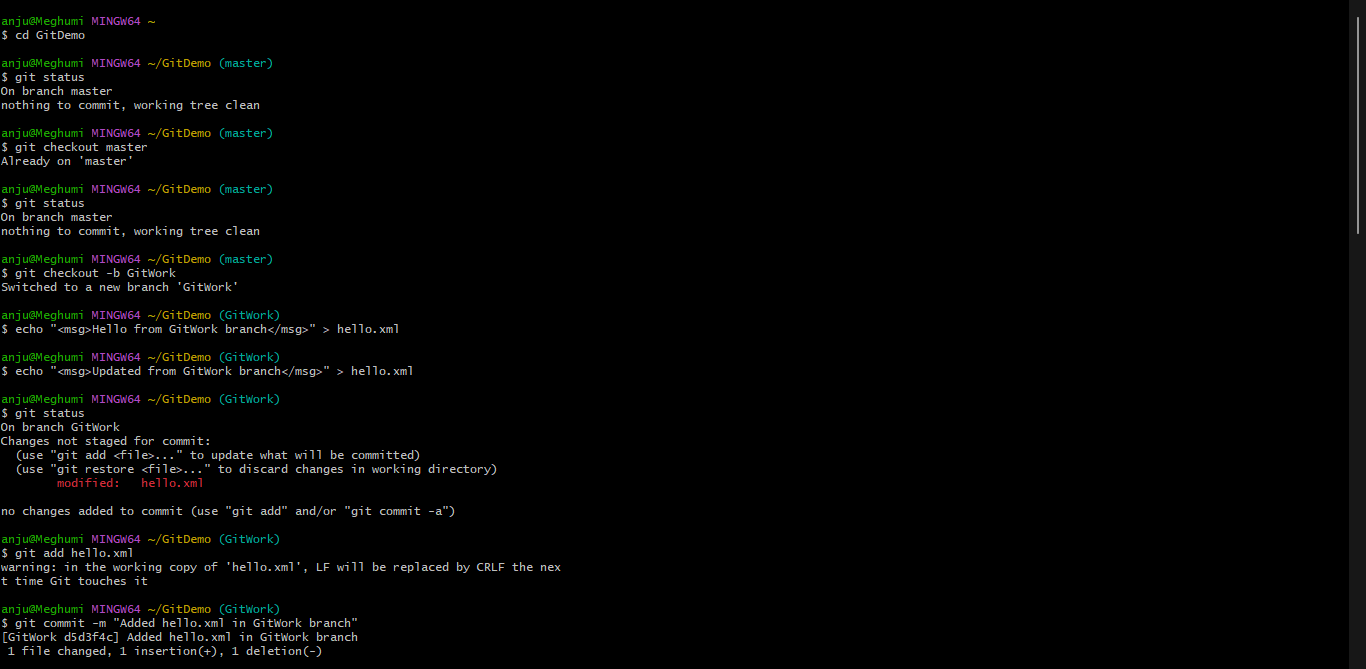
Answer:

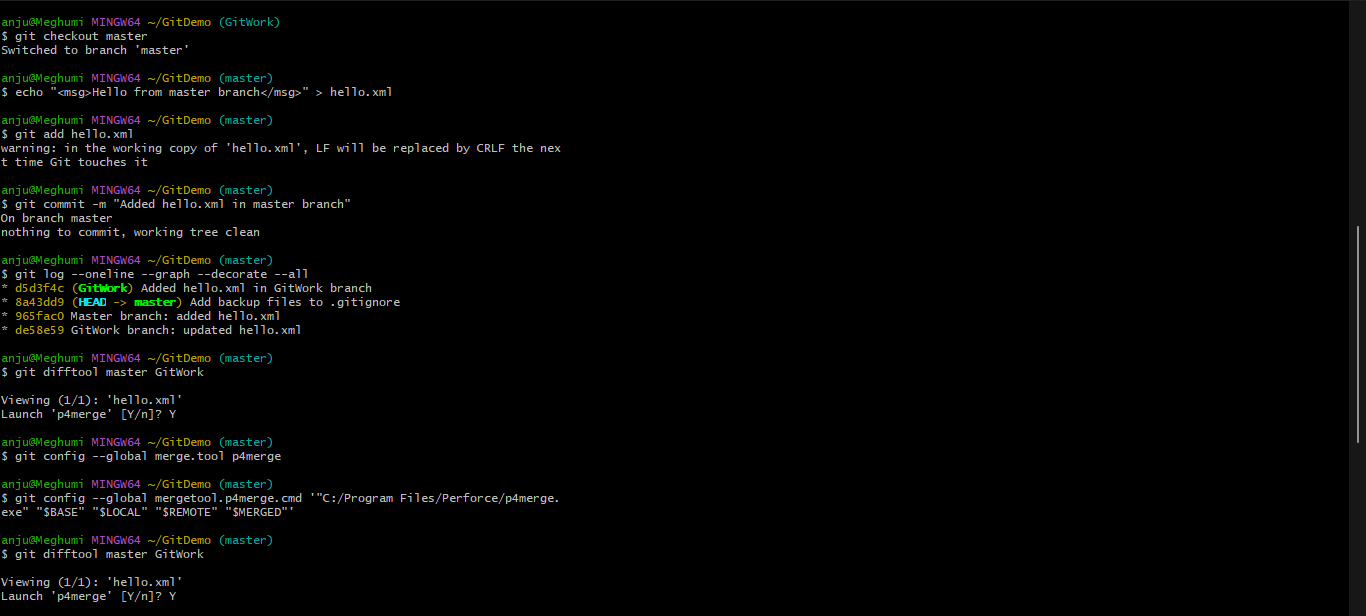
**What is a Merge Conflict?**

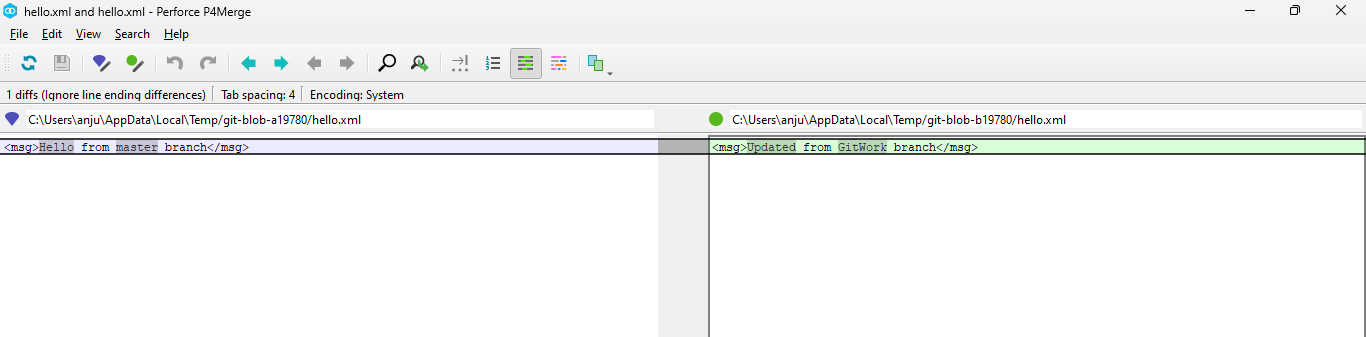
A **merge conflict** happens when Git is unable to automatically combine changes from two branches. This usually occurs when two people (or branches) change the same part of a file in different ways.

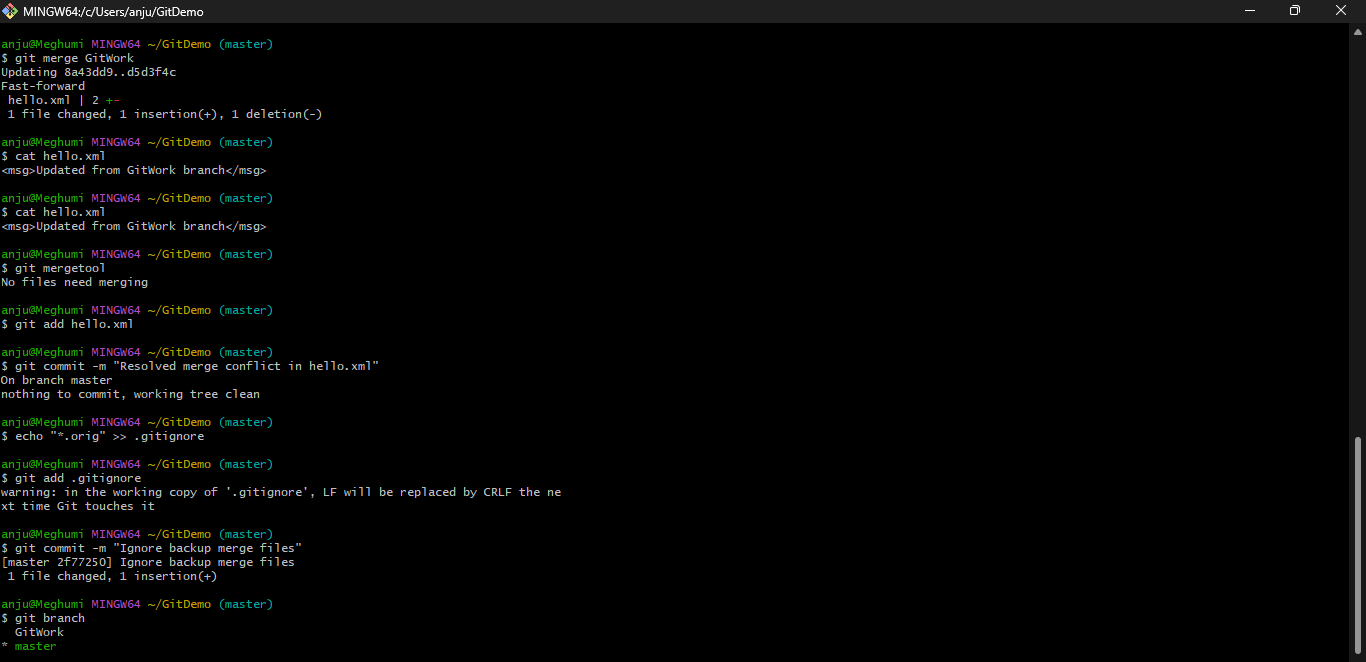
**How to Resolve a Merge Conflict :**

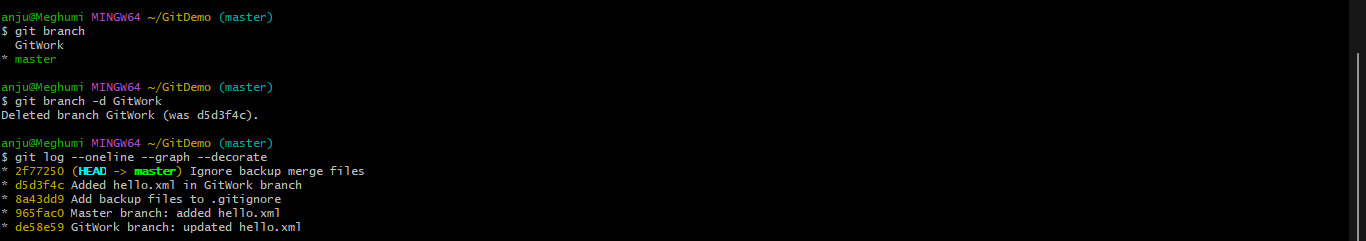
1. **Start the merge process**: You attempt to combine two branches. Git will try to merge them automatically.
2. **Conflict is detected**: If Git cannot decide which change to keep, it marks the file as conflicted and pauses the merge.
3. **Identify the conflicting files**: You check which files have conflicts. Git clearly shows you which files need attention.
4. **Open the file with conflict**: Inside the file, Git marks the conflicting sections using special markers. These markers show both versions of the conflicting code.
5. **Manually resolve the conflict**: You read the conflicting changes and decide what the final version should be. You can choose one version, or combine both.
6. **Remove the conflict markers**: After choosing the correct version, you remove Git’s conflict indicators from the file to clean it up.
7. **Mark the conflict as resolved**: Once the conflict is resolved, you let Git know that you have handled the problem.
8. **Complete the merge process**: Finally, you save your changes and complete the merge. The branches are now successfully merged.











5. **Objectives**

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

In this hands-on lab, you will learn how to:

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

**Prerequisites**

The following are the pre-requisites to complete this hands-on lab:

* Hands-on ID: **“Git-T03-HOL\_002”**

Notes\*:

|  |
| --- |
| Please follow the below steps for creating a free account in GitHub.  Do not use cognizant credentials to login to GitHub. |

Estimated time to complete this lab: **10 minutes.**

Please follow the instructions to complete the hands-on. Each instruction expects a command for the Git Bash.

1. Verify if master is in clean state.
2. List out all the available branches.
3. Pull the remote git repository to the master
4. Push the changes, which are pending from **“Git-T03-HOL\_002”** to the remote repository.
5. Observe if the changes are reflected in the remote repository.

Answer:

**How to clean up and push back to remote Git**

**1. Verify clean working state**

* Run git status to check if there are any uncommitted changes.
* If there are changes, either commit them using git add . and git commit -m "message", or discard them.
* This ensures your local branch is “clean” before pushing.

**2. Switch to the correct branch**

* Use git checkout <branch-name> to move to the branch you want to update (e.g., main or your feature branch).

**3. Pull latest changes from remote**

* Run git pull origin <branch-name> to sync your local branch with the latest version from the remote repository.
* This helps avoid conflicts when pushing.

**4. Push your branch to remote**

* If your branch has new commits, run:

git push origin <branch-name>

* This sends your local commits to the remote repository (GitHub, GitLab, etc.).

**5. Verify on remote**

* Open your repository in GitHub/GitLab.
* Check that your commits and files are visible on the correct branch





