**Git Hands-On Lab Documentation**

**Title:** Version Control with Git – Hands-On Lab

**Objective:**  
This lab introduces fundamental Git commands and workflows, guiding users through Git configuration, editor integration, and repository management using GitLab.

**Lab Objectives**

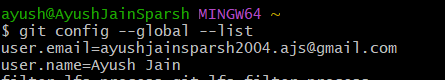
By the end of this lab, you will be able to:

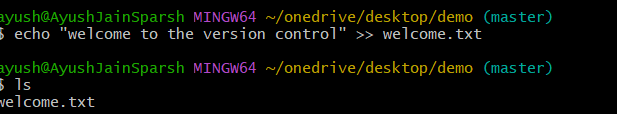
1. Configure Git on your local machine
2. Integrate Notepad++ as the default Git editor
3. Create and manage a Git repository
4. Push code to a remote GitLab repository

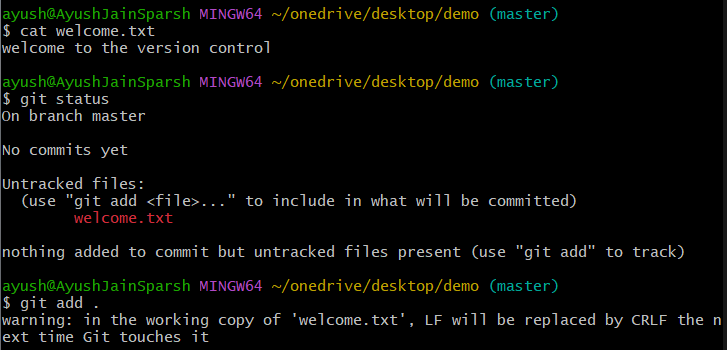
**Steps :**

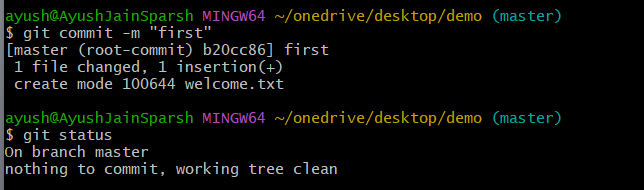
1. Git Configuration Setup
2. Integrate Notepad++ as Default Git Editor
3. Create and Manage Git Repository
4. Push to GitLab

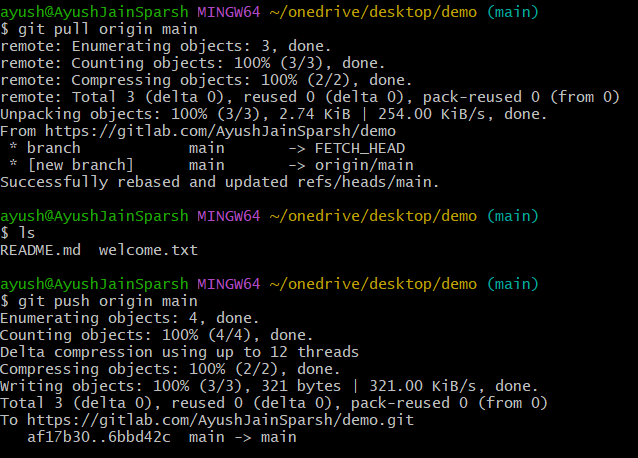
**Output Snapshots :**

****

****

****

****

****

**Result:**

- Configured Git and set up your identity

- Integrated Notepad++ as your default Git editor

- Created a local Git repository and committed changes

- Pushed your code to a remote GitLab repository

**Git Hands-On Lab Documentation**

**Objective:**  
Understand the purpose and functionality of .gitignore

1. Learn how to exclude specific files and folders from Git tracking
2. Implement .gitignore to ignore .log files and log folders

**Lab Objectives**

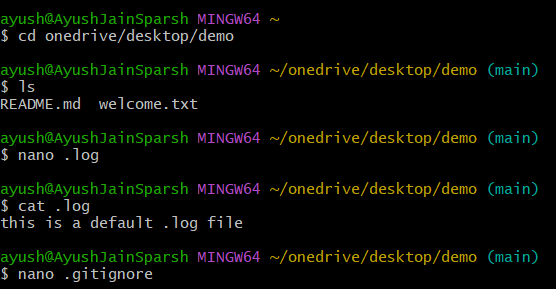
By the end of this lab, you will be able to:

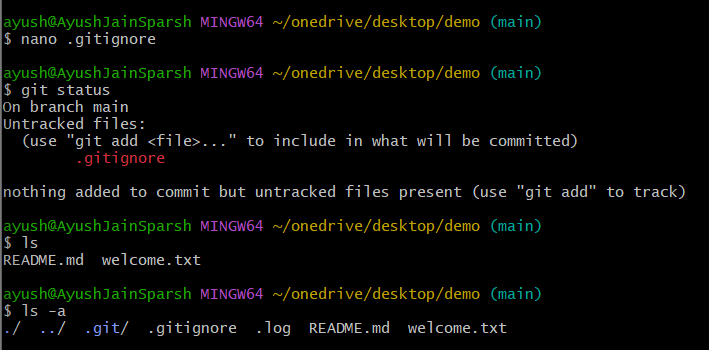
1. Initialise .gitignore file to stop unwanted files like .log to get store in gitlab

**Steps :**

1. Initialize Git Repository (if not already done)
2. Create Files and Folders to Ignore
3. Create/Edit .gitignore File
4. Verify .gitignore Behavior

**Output Snapshots :**

****

****

**Git Hands-On Lab Documentation**

**Objective:**  
 Understand Git branching and merging workflows

1. Learn how to create a branch and merge it into the master (trunk)
2. Explore GitLab's branch and merge request features
3. Use P4Merge for visual diffing during merges

**Prerequisites**

Ensure the following are ready before starting:

* Git environment installed and configured
* P4Merge tool installed and set as Git’s diff/merge tool
* A local Git repository initialized
* A remote GitLab repository connected

**What is Branching in Git?**

Branching allows developers to diverge from the main codebase (usually master or main) to work on features or fixes independently.

Benefits:

- Isolated development

- Parallel workflows

- Safe experimentation

**What is Merging in Git?**

Merging integrates changes from one branch into another. Typically, feature branches are merged into master after testing.

**Lab Objectives**

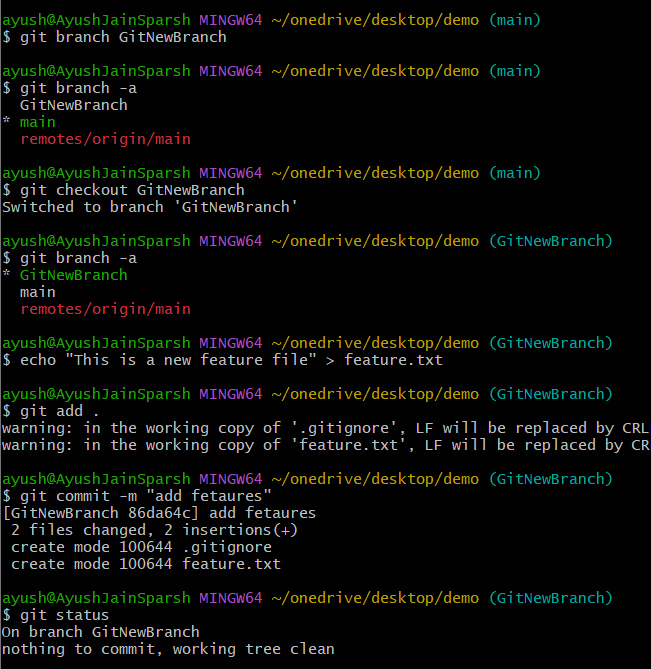
By the end of this lab, you will be able to:

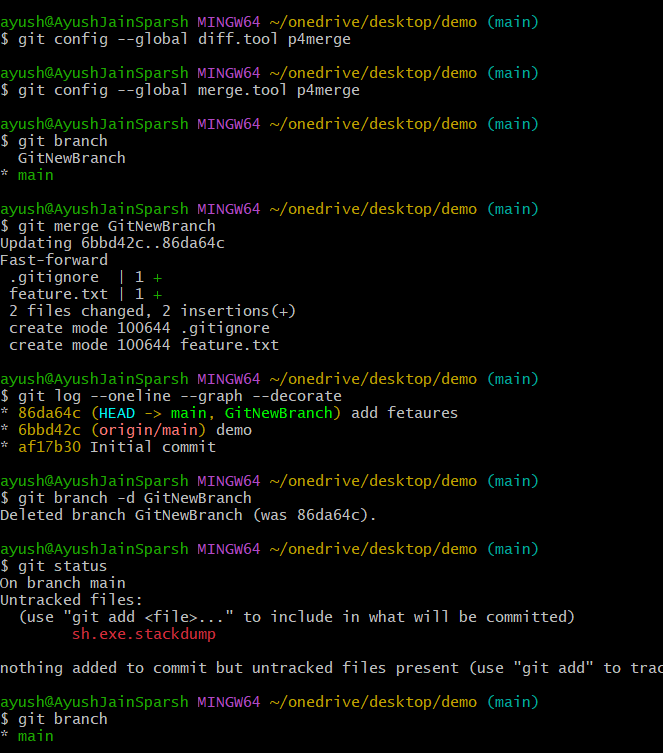
1. Initialise .gitignore file to stop unwanted files like .log to get store in gitlab

**Steps :**

1. Create a New Branch
2. Switch to the New Branch Add Files and Make Changes
3. Commit the ChangesCheck Git Status
4. Switch Back to Master & List CLI Differences
5. View Visual Differences with p4merge
6. Merge the branches

**Output Snapshots :**

****

****

**Git Hands-On Lab Documentation**

**Objective:**  
- Understand how merge conflicts occur in Git

- Learn how to resolve conflicts using Git and a 3-way merge tool

- Use P4Merge for visual conflict resolution

- Update .gitignore to exclude backup files

**Prerequisites**

Before starting, ensure the following are ready:

* Git environment installed and configured
* P4Merge tool installed and integrated with Git
* Hands-on ID: Git-T03-HOL\_001
* A local Git repository initialized and connected to a remote

**What is a Merge Conflict?**

A merge conflict occurs when Git cannot automatically reconcile differences between two branches. This typically happens when the same file is modified in both branches in overlapping lines.

**Lab Objectives**

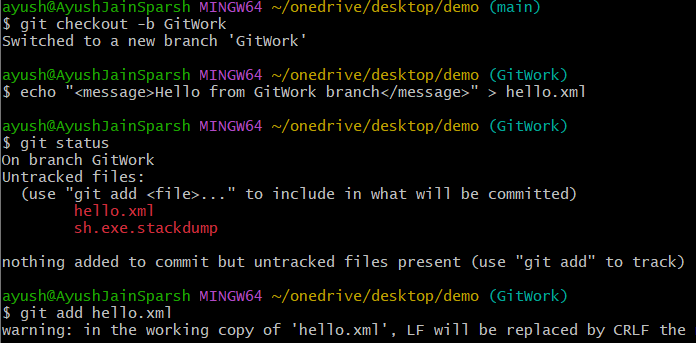
By the end of this lab, you will be able to:

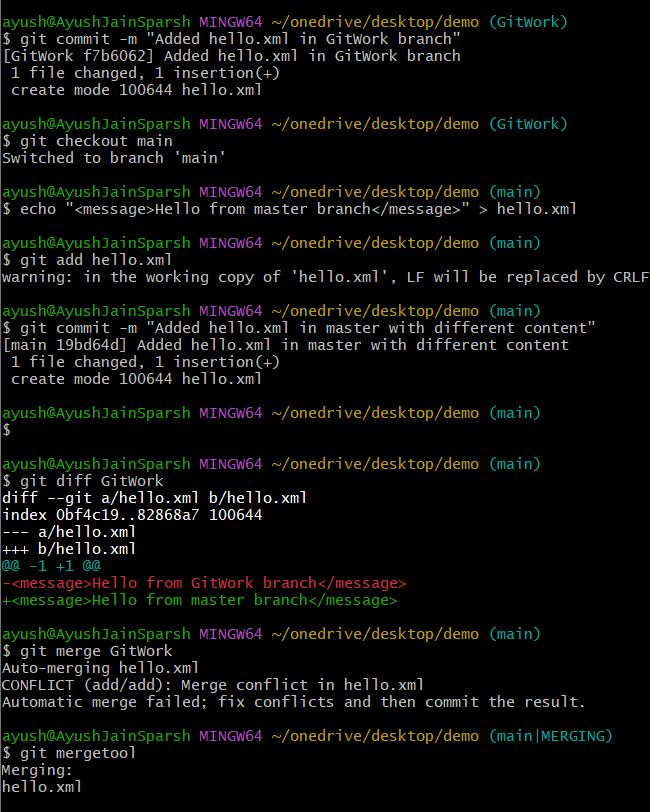
1. Initialise .gitignore file to stop unwanted files like .log to get store in gitlab

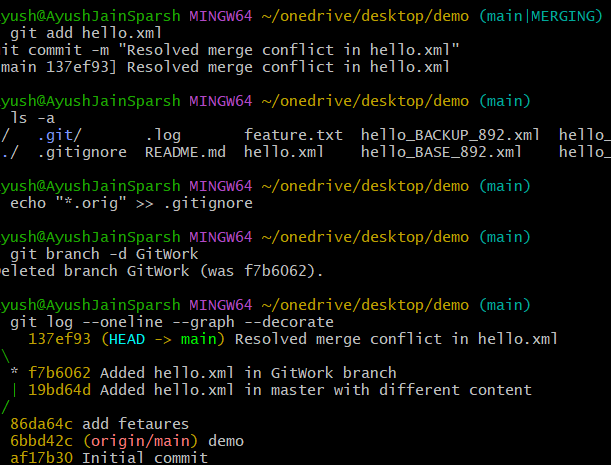
**Steps :**

1. Verify main is clean
2. Create a new branch. Add and modify hello.xml in branch
3. Commit the ChangesCheck Git Status
4. Switch Back to main and add conflicting hello.xml in main
5. Commit changes and check difference
6. Merge the branches
7. Resolve conflict using 3 way merge tool
8. Add backup file to .gitignore
9. Commit and list all branches and delete final log

**Output Snapshots :**

****

****

****

**Git Hands-On Lab Documentation**

**Objective:**  
 Ensure the local repository is clean and synchronized

* Push pending changes to the remote repository
* Confirm successful update on GitHub

**Prerequisites**

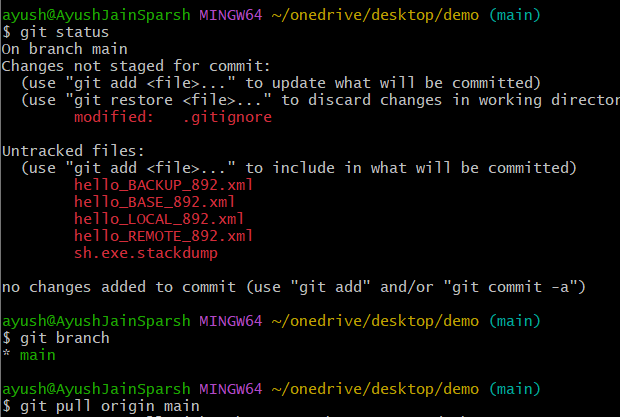
Before starting, make sure:

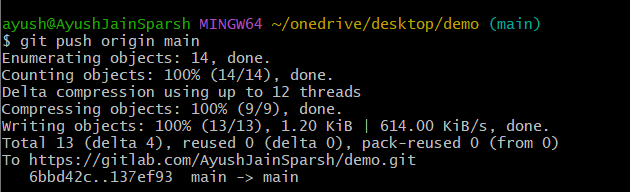
1. Git is installed and configured
2. You have a GitHub account
3. Hands-on ID: Git-T03-HOL\_002
4. Remote repository is already set up and linked

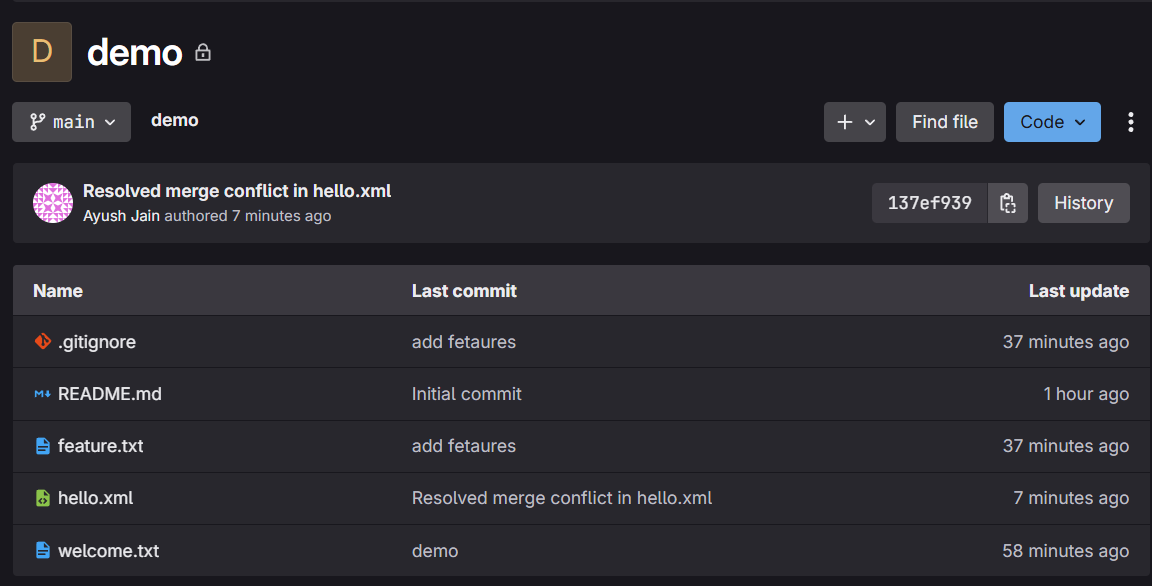
**Steps :**

1. Verify main is clean
2. List all available branches
3. Pull remote repo to main
4. Push pending changes to remote
5. Verify changes on Gitlab

**Output Snapshots :**

****

****

****