**EXERCISE 1: GIT-HOL-1**

**Introduction:**

This foundational Git workshop introduces students to distributed version control systems through practical implementation of core Git functionalities. The laboratory session emphasizes proper development environment setup and familiarization with essential version control operations using modern Git tooling and GitLab platform integration.

**Objective:**

* **Establish Development Environment**: Configure local Git installation with proper user authentication and integrate advanced text editing capabilities through Notepad++ integration for enhanced development workflows.
* **Execute Fundamental Version Control Operations**: Develop proficiency with primary Git operations including repository initialization, change tracking, staging processes, and commit management for effective code versioning.
* **Deploy Remote Repository Integration**: Establish connectivity between local development environments and remote GitLab repositories, enabling distributed development and team collaboration capabilities.

**Implementation Breakdown:**

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

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

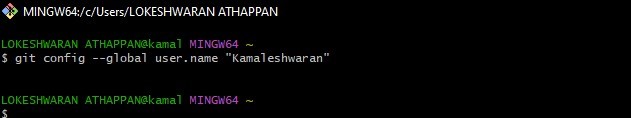
Login to GitLab and create a “GitDemo” project

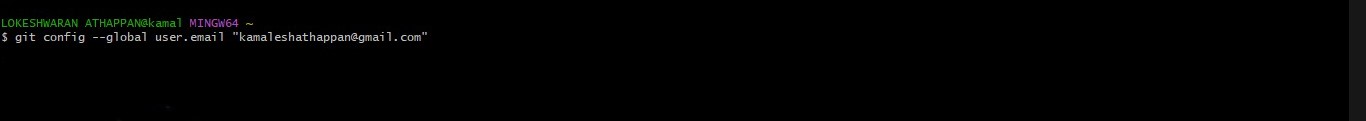
1. To check if Git client is installed properly: Open Git bash shell and execute



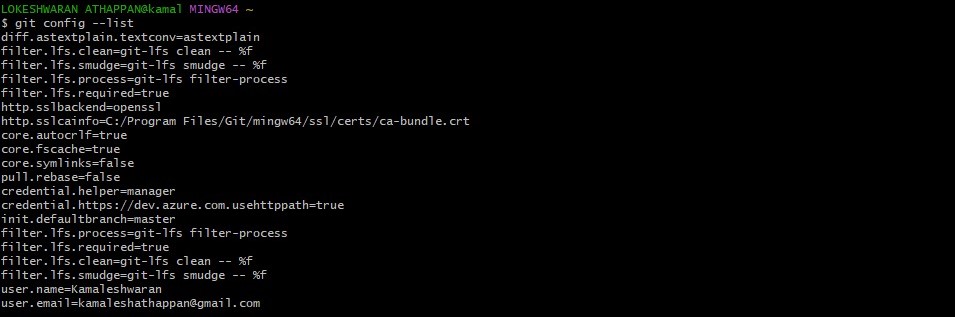
If output shows Git with its version information that indicates, that Git Client installs properly.

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.



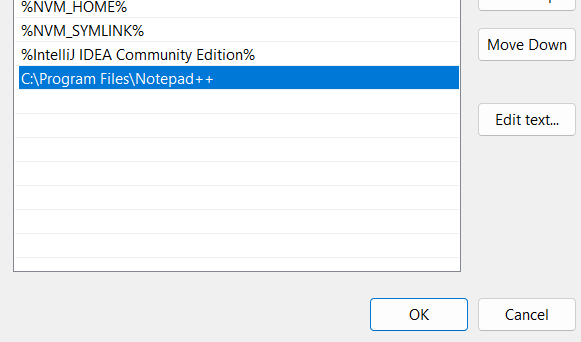
**Step 2: Integrate notepad++.exe to Git and make it a default editor**

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

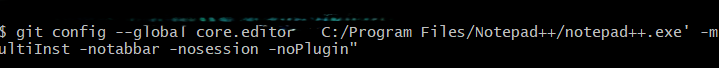
1. To create an alias command for notepad++.exe, execute



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

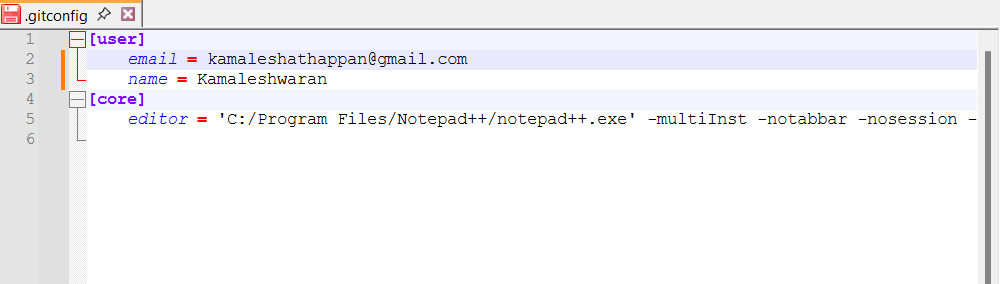


1. To verify if notepad++ is the default editor, execute the command



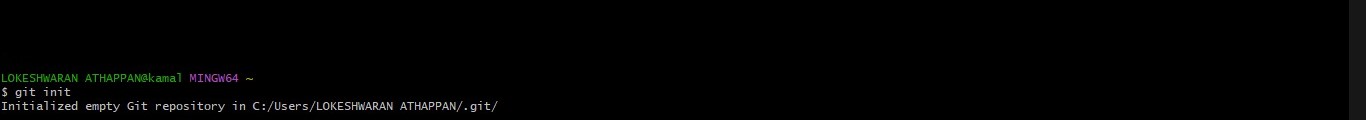
Here ‘-e’ option implies editor

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”.

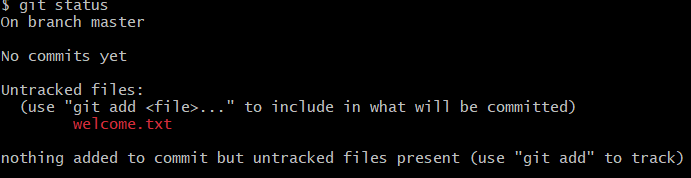
1. To create a file **“welcome.txt”** and add content to the file, execute the command



1. To verify 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



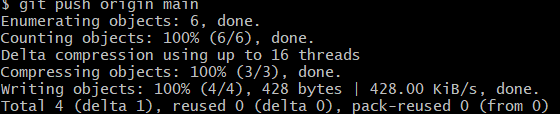
Notepad++ editor will open and to add multi-line comment with default editor

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

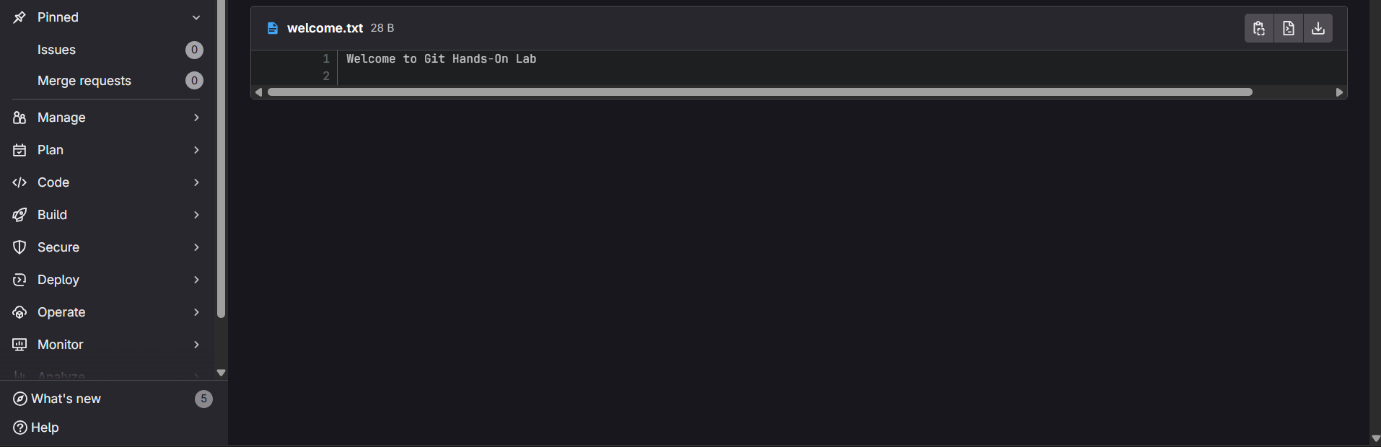
**welcome.txt** is added to the local repository.

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





1. Gitlab



**Conclusion:**

This foundational workshop successfully introduces participants to professional version control practices through hands-on Git implementation. Participants emerge with configured development environments, practical experience in repository management, and established workflows for distributed software development collaboration.

**EXERCISE 2: GIT-HOL-2**

**Introduction:**

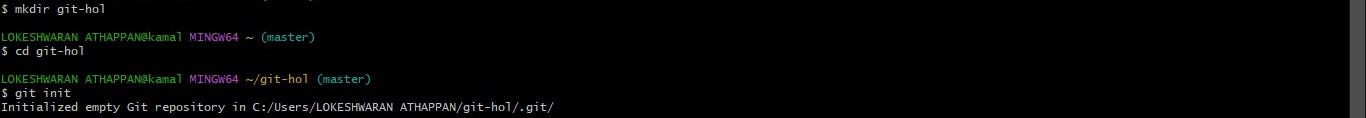
This targeted Git workshop addresses repository content management through advanced file filtering techniques using Git's ignore functionality. The session explores how development teams maintain clean codebases by strategically excluding non-essential files and directories from version control tracking systems.

**Objective:**

* **Analyze File Exclusion Principles**: Explore the theoretical foundations of .gitignore functionality and understand its role in maintaining streamlined repository structures for team-based development.
* **Configure Selective Tracking Rules**: Design and implement .gitignore configurations to systematically exclude specific file patterns (.log files) and directory structures (log directories) from Git version control.
* **Validate Filtering Effectiveness**: Execute comprehensive testing procedures to confirm proper implementation of ignore rules across multiple repository states including working directory, staging area, and remote synchronization.

**Implementation Breakdown:**

**Step 1: Create a test repo and initialise**

****

**Step 2 : Create the files/folders the lab asks to ignore**

****

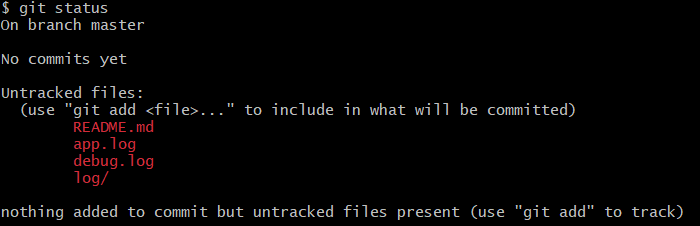
****

****

****

****

**Check status:**

****

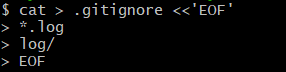
**Step 3 : Create .gitignore to ignore .log files and log folder**

**Open or create .gitignore and add:**

****

****

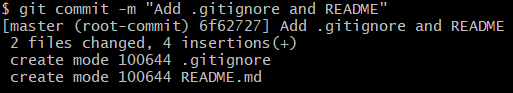
**Command-line write:**

****

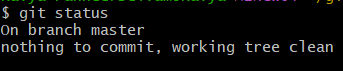
* \*.log ignores any file ending with .log in any folder.
* log/ ignores the directory named log and everything inside it.
* To ignore only a top-level log directory you could use /log/

**Step 4 : Stage & commit**

****

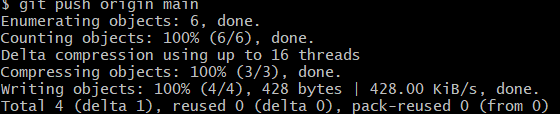
****

**After this commit, run:**

****

**Step 6 : Push to remote**

****

****

**Conclusion:**

This specialized workshop provides participants with advanced repository management capabilities through mastery of Git's file exclusion mechanisms. Graduates will effectively maintain professional repository standards by implementing strategic file filtering, resulting in cleaner collaborative development environments.

**EXERCISE 3: GIT-HOL-3**

**Introduction:**

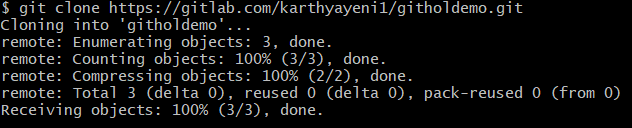
This intermediate Git workshop explores advanced development methodologies through branch-based parallel development strategies. The session demonstrates how modern software teams utilize Git branching to enable simultaneous feature development while maintaining code stability through proper isolation and integration techniques.

**Objective:**

* **Execute Branch Lifecycle Management:** Develop expertise in creating, navigating, and monitoring Git branches while understanding branch pointer relationships and remote branch synchronization.
* **Facilitate Isolated Development Streams:** Practice implementing changes within dedicated feature branches, managing independent codebases, and maintaining separation between experimental and production code.
* **Orchestrate Code Integration Processes:** Perform advanced merge operations utilizing both command-line utilities and sophisticated visual merge tools like P4Merge, followed by proper branch lifecycle completion and audit trail maintenance.

**Implementation:**

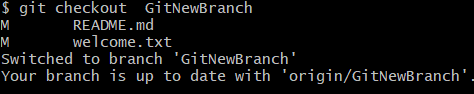
**Step 1 : Start (clone)**

****

****

**Step 2 : Create the branch GitNewBranch and switch to it**

****

****

**Step 3 : List branches (local & remote)**

****

**Step 4 : Make changes on the branch, stage & commit**

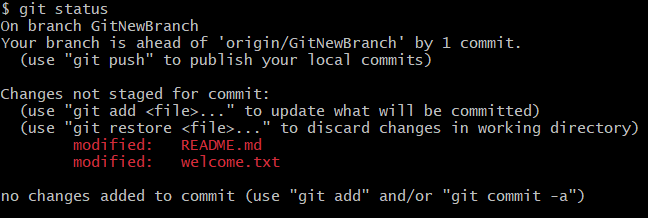
**Example: add a file, stage and commit:**

****

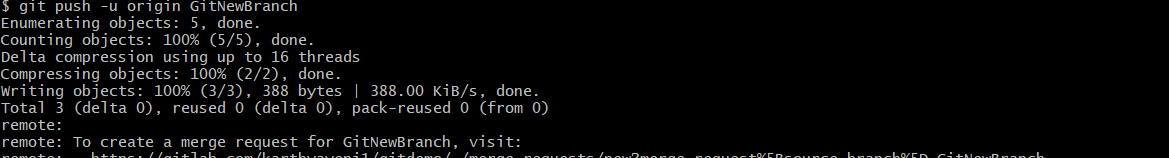
****

****

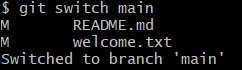
**Check status:**

****

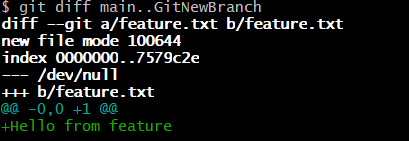
**Pushing:**

****

**Step 5 : Switch back to trunk (main)**

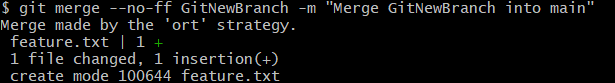
****

**Step 6 : See differences (command-line)**

****

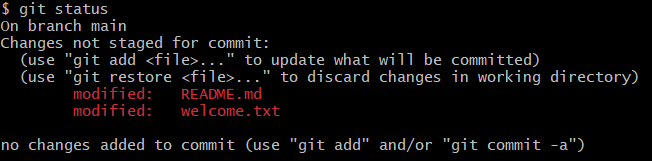
**Step 7 : Merge the source branch into trunk**

**Fast, non-fast-forward-safe merge:**

****

**If conflicts happen**

**git status to list conflicted files.**

****

**Mark resolved:**

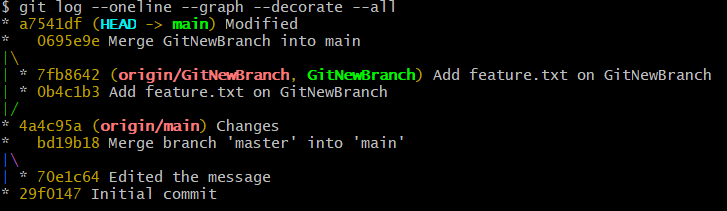
****

**Finish the merge (if merge):**

****

**Step 8 : Inspect merge history**

**After merge:**

****

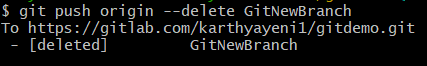
**Step 9 : Delete the branch (local and remote)**

**After confirming everything is merged:**

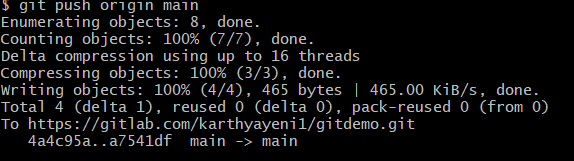
**Local delete:**

****

**Remote delete:**

****

**Step 10 : Push trunk changes to remote**

****

**Conclusion:**

This intermediate workshop establishes advanced collaborative development competencies through comprehensive branch management training. Participants will confidently implement parallel development strategies, manage feature isolation, and execute safe code integration procedures essential for modern agile development teams.

**EXERCISE 4 : GIT-HOL-4**

**Introduction:**

This advanced Git workshop tackles complex collaborative development challenges through systematic merge conflict resolution methodologies. The laboratory simulates realistic multi-developer scenarios where simultaneous file modifications require sophisticated conflict detection, analysis, and resolution using professional development tools.

**Objective:**

* **Diagnose Complex Merge Conflicts:** Develop skills in identifying, analyzing, and categorizing merge conflicts that occur when parallel development streams modify identical code sections or file structures.
* **Utilize Professional Resolution Tools:** Gain expertise with industry-standard conflict resolution utilities including Git's native diff capabilities and advanced 3-way merge visualization tools for complex conflict scenarios.
* **Execute Comprehensive Resolution Protocols:** Implement complete conflict resolution workflows encompassing conflict detection, manual resolution procedures, verification processes, and post-resolution repository maintenance including ignore file updates.

**Step 1 : Set Up a Local Folder**

****

****

****

**Step 2 : Link Local Repo to GitHub**

1. **Copy the repository URL from GitHub (HTTPS).**
2. **In Git Bash:**

****

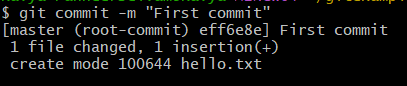
**Step 3 : Create First File and Push**

**Create file:**

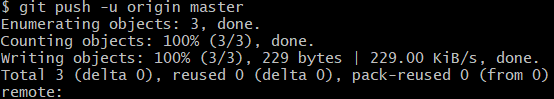
****

**Stage & Commit:**

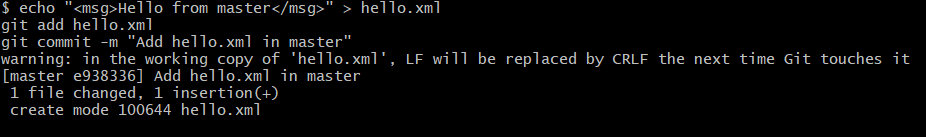
****

****

**Push:**

****

**Step 4 : Edit hello.xml in Master Branch**

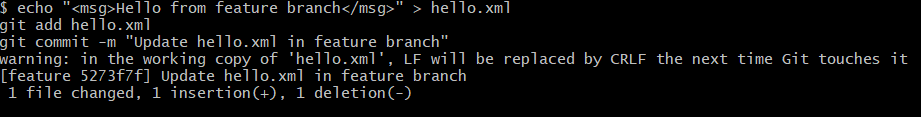
****

**Step 5 : Create a New Branch and Modify hello.xml**

**Create branch:**

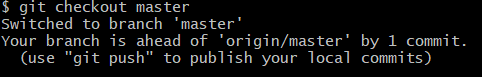
****

**Edit hello.xml with a different message:**

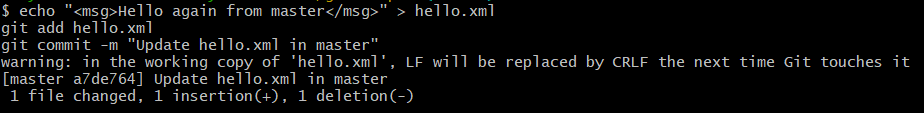
****

**Step 6 : Switch Back to Master and Modify hello.xml**

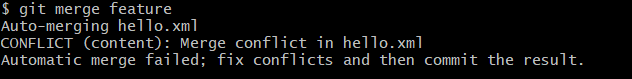
**Switch to master:**

****

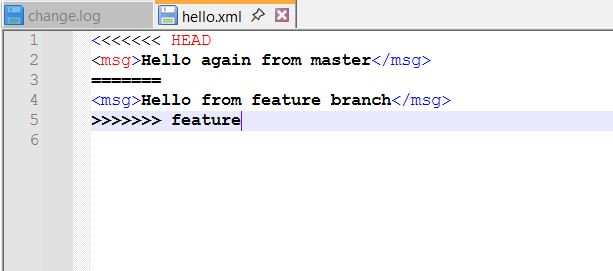
**Update file differently:**

****

**Step 7 : Merge and Resolve Conflict**

****

**Conflict Solving:**

****

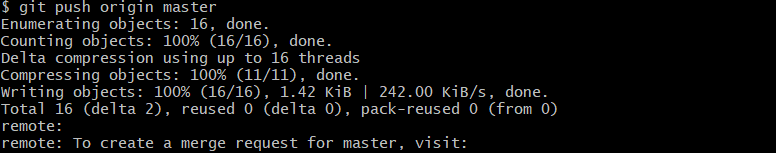
**Stage the resolved file:**

****

**Commit the merge:**

****

**Step 8 : Pushing the changes**

****

**Conclusion:**

This advanced workshop delivers critical collaborative development skills for managing complex merge scenarios encountered in professional software development environments. Participants will confidently navigate challenging conflict situations using industry-standard tools and protocols, ensuring successful team collaboration in demanding development projects.

**EXERCISE 5 : GIT-HOL-5**

**Introduction:**

This capstone Git workshop concentrates on professional repository management practices essential for maintaining healthy distributed development environments. The session emphasizes systematic cleanup procedures and robust remote synchronization strategies that ensure consistent code availability across distributed development teams.

**Objective:**

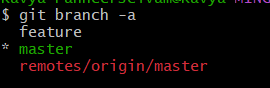
* **Assess Repository Health Status:** Develop systematic approaches for evaluating repository cleanliness, branch organization, and overall system state before executing synchronization or maintenance operations.
* **Implement Distributed Synchronization Workflows:** Master bidirectional synchronization processes including remote change integration and local modification distribution to maintain consistent codebase versions across team members.
* **Execute Verification and Validation Procedures:** Establish comprehensive verification protocols to ensure successful remote integration, validate change visibility, and confirm proper distributed version control functionality.

**Implementation:**

**Step 1: Verify Master Branch is in Clean State**

****

**Step 2: List All Available Branches**

****

**Step 3: Pull the Remote Git Repository to Master**

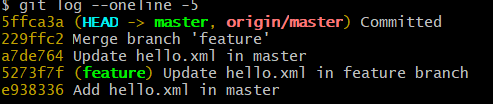
****

****

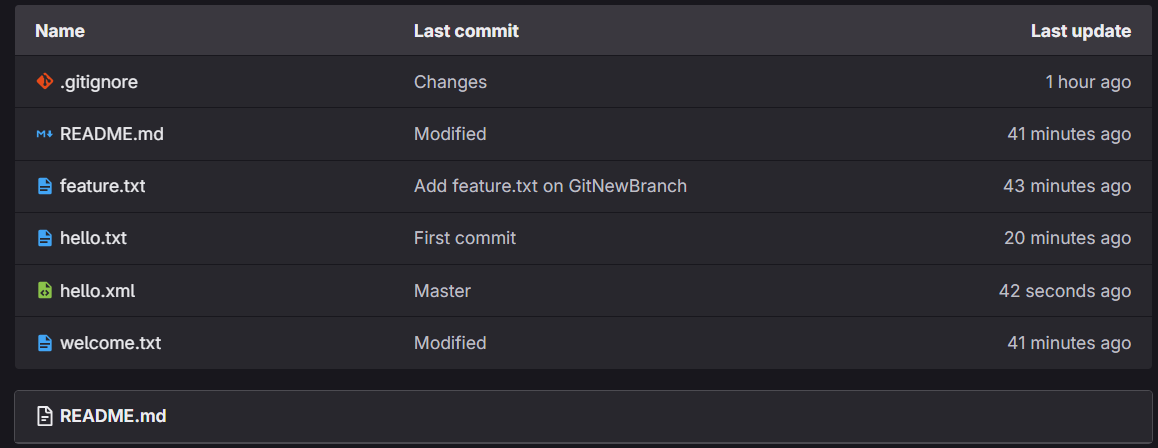
**Step 4: Push Pending Changes to Remote Repository**

****

**Step 5: Observe Changes in Remote Repository**

****

**Check on Gitlab**

****

**Conclusion:**

This capstone workshop consolidates the complete Git learning experience by establishing professional-grade repository maintenance standards and distributed collaboration protocols. Participants will demonstrate mastery of enterprise-level version control practices, enabling them to effectively lead and contribute to sophisticated collaborative development initiatives.