**Git for Mule (git bash)**

**Git:**

Git is a distributed version control system (VCS) designed for tracking changes in source code during software development. It helps developers coordinate work, manage code history, and collaborate efficiently.

Features:

* Distributed: Each developer has a local copy of the entire repository, allowing them to work offline and independently.
* Speed: Git is fast and efficient, making it suitable for large projects.
* Data Integrity: It ensures the integrity of data by using cryptographic hashing.
* Non-linear Workflows: Git supports branching and merging, enabling flexible development workflows.

**GitHub:**

Definition: GitHub is a web-based Git repository hosting service that builds upon Git’s capabilities.

Purpose: It provides a platform for developers to collaborate, share code, and manage projects.

Features:

* Web Interface: GitHub offers a user-friendly web interface for managing repositories, issues, and pull requests.
* Centralized Hosting: Developers can host their Git repositories on GitHub, making them accessible to others.
* User Management: GitHub includes built-in user management features.
* Marketplace: It has an active marketplace for integrating various tools and services.
* Free and Paid Tiers: GitHub offers both free and paid plans for hosting repositories.

**Step 1 :**

Visit the Official Git Website:

* Go to the official Git website 1 and click the “Download” button for Windows.
* The download will start automatically. If not, you can click the “Click here to download manually” link.

Run the Downloaded File:

* After downloading the executable file, run it.
* You’ll see a pop-up window asking for permission to make changes to your device. Click “Yes” to proceed.
* The Git Setup window will open.

Select Destination Location:

* Click the “Browse…” button to choose the destination location where you want to install Git. By default, it installs to C:\Program Files\Git.
* Click “Next” after selecting your destination location.

Select Components (Optional):

* You can choose additional components to install, such as a desktop icon.
* Ensure that the “Git Bash Here” checkbox is checked.
* Proceed with the default settings if you prefer.

Choose Default Editor (Optional):

* You can select a default text editor to be used by Git. If you have a preferred editor (e.g., Visual Studio Code), choose it here.
* Click “Next.”

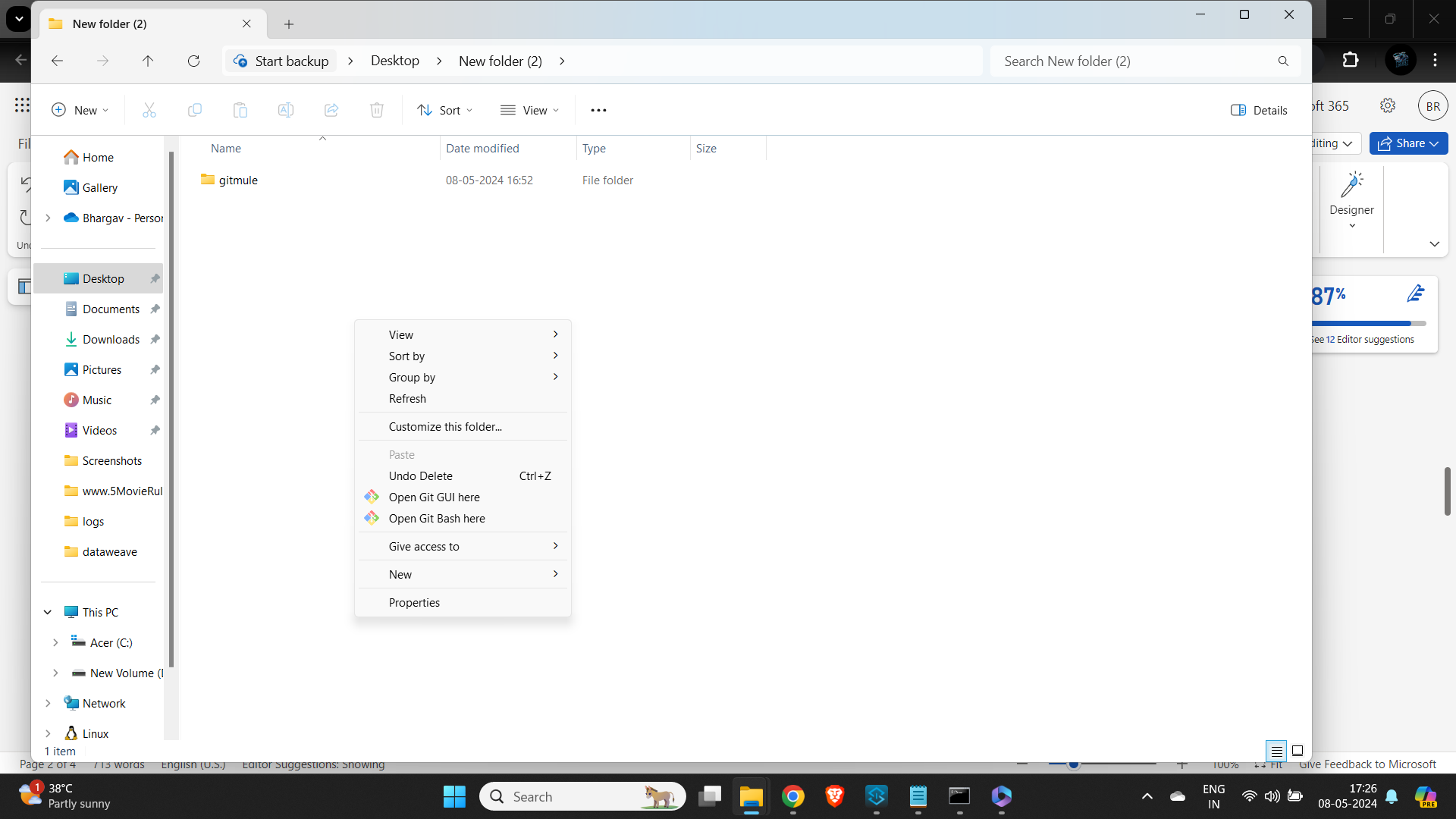
Adjust Initial Branch Name (Optional):

* Git now allows you to adjust the name of the initial branch in new repositories. You can leave it as the default “main” or choose a different name.
* Click “Next.”

Install Git:

* Click “Install” to begin the installation process.
* Once installation is complete, you’ll see a “Completing the Git Setup Wizard” screen.
* Click “Finish.”

Launch Git Bash:

* Git Bash is now installed on your computer. You can launch it by searching for “Git Bash” in your Start menu or by right-clicking in any folder and selecting “Git Bash Here.”
* 

**Step 2:**

Navigate to GitHub:

* Visit the GitHub website by going to GitHub.
* Click “Sign up”:
* On the GitHub homepage, click the “Sign up” button.

Provide Your Details:

* Enter your email address (make sure it’s one you haven’t used for another GitHub account).
* Choose a strong password and type it in.
* Select a unique username for your account.

Verify Your Email:

* GitHub will ask you to verify your email address. Check your inbox for a verification code or link.
* Without a verified email address, you won’t be able to perform certain tasks, such as creating a repository.

Explore GitHub:

* Congratulations! You’ve created your personal GitHub account.

**Step 3:**

Navigate to the New Repository Page:

* In the upper-right corner of any page, click on your profile picture or avatar.
* From the dropdown menu, select “New repository.”

Configure Your Repository:

* You’ll be taken to the “Create a New Repository” page.
* Here are the key settings you can configure:
* Repository Name: Choose a name for your repository.
* Description (Optional): Add a brief description to explain what your project is about.
* Visibility: Decide whether your repository should be public (visible to everyone) or private (only accessible to collaborators).
* Initialize with a README: If you want to create a README file for your project, check this option.
* Add .gitignore: You can choose a template for ignoring specific files (e.g., Python, Java, Node.js, etc.).
* Choose a License: Select a license for your project (optional).

Create the Repository:

* Once you’ve configured the settings, click the “Create repository” button.
* Your new repository will be created, and you’ll be taken to the repository page.

Step 4:

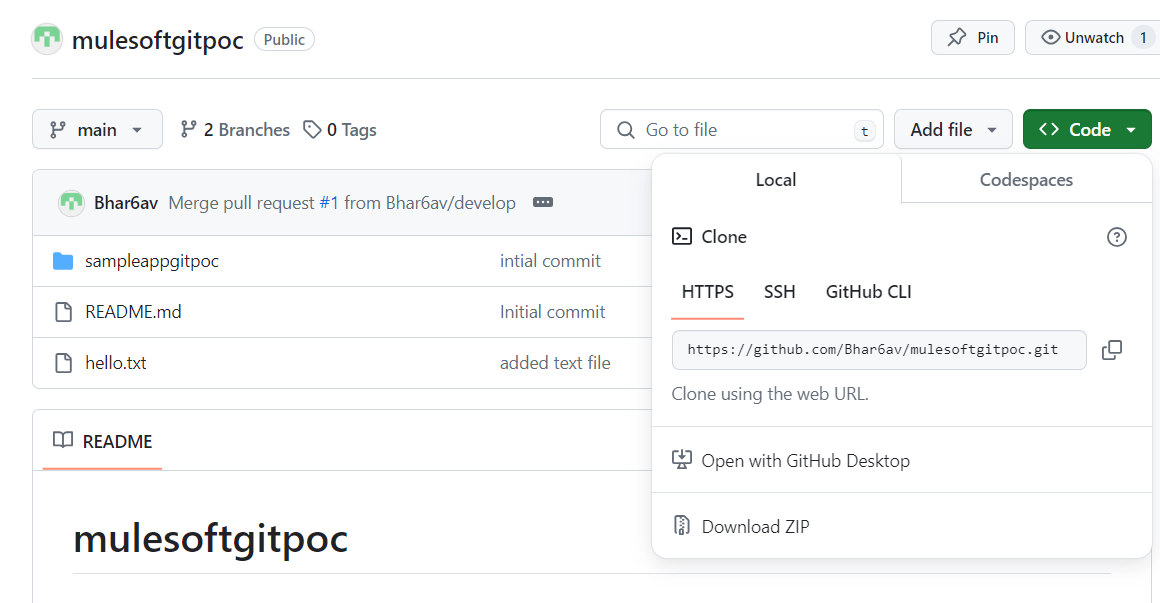
Clone the repository from remote to local

Copy the Repository URL:

First, navigate to the main page of the repository you want to clone on GitHub.com.

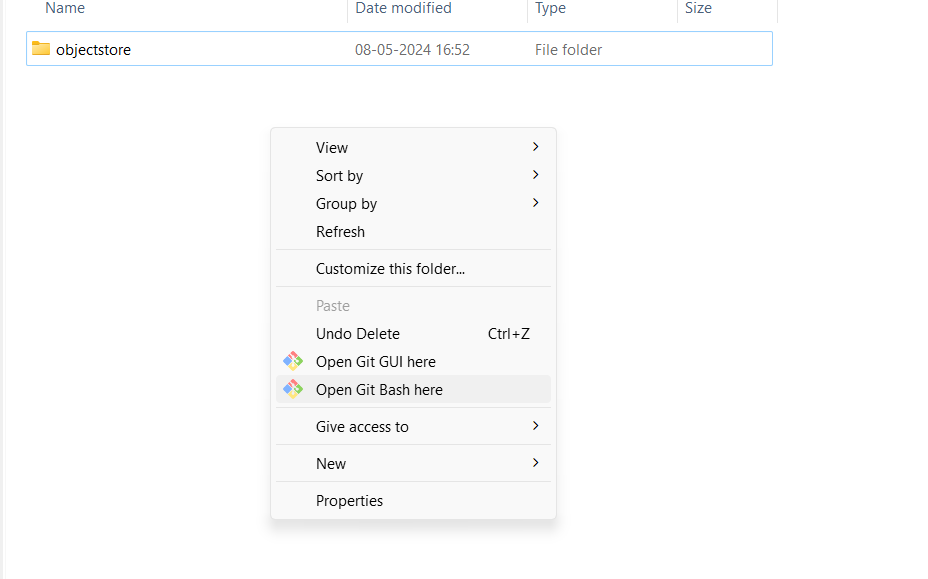
Above the list of files, click on the “Code” button.

Copy the URL for the repository (you can choose either HTTPS or SSH).



Open Terminal or Git Bash:

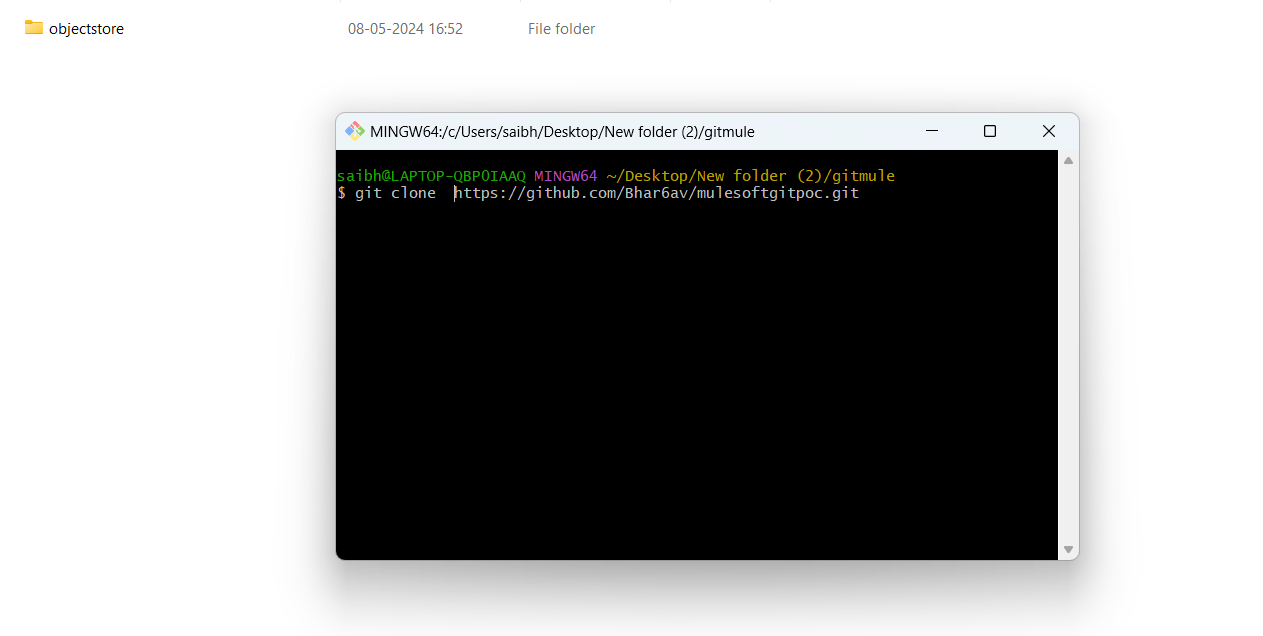
Open your terminal or Git Bash on your local system.



Change Directory:

Use the cd command to navigate to the location where you want to create the cloned directory (local repository).

Clone the Repository:

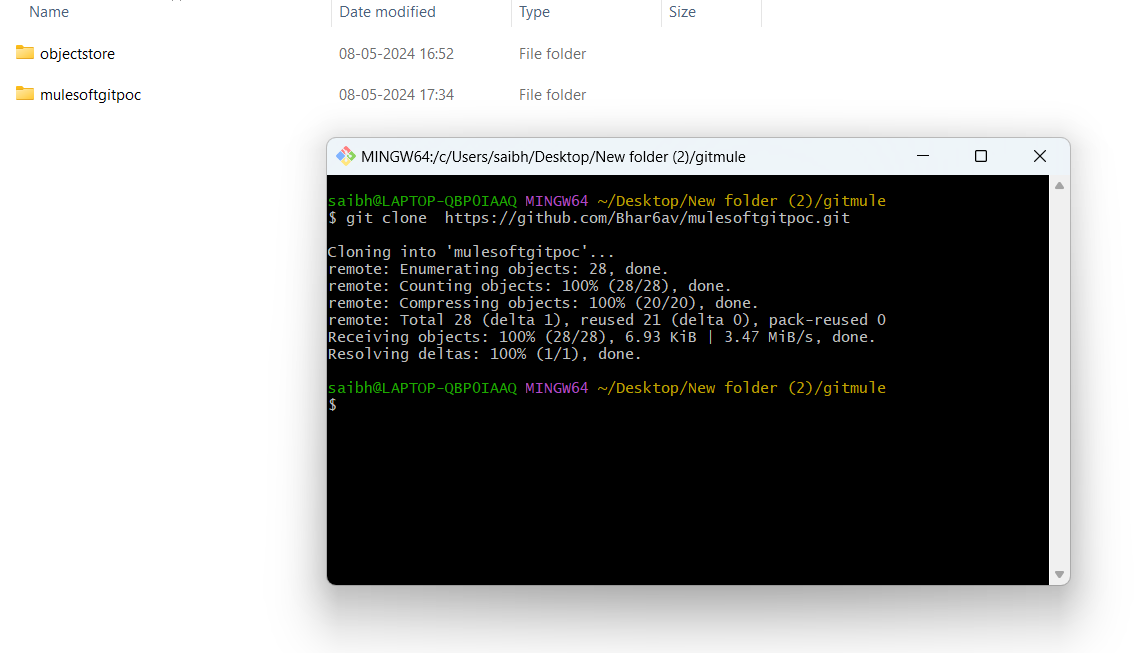


Type the following command, replacing [repository-url] with the URL you copied earlier:

git clone [repository-url]

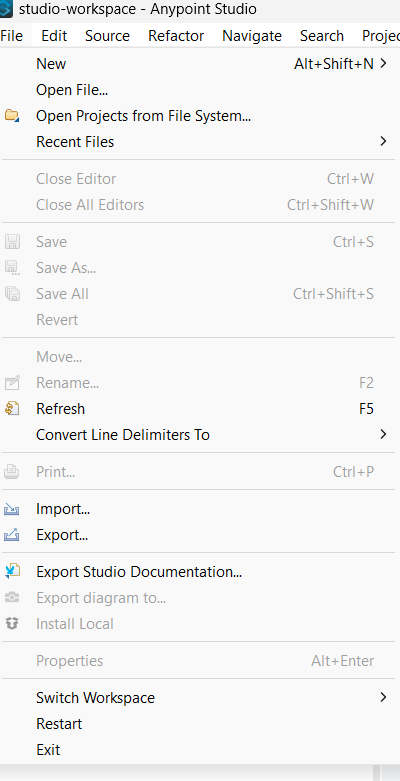
Note: I have already have muleproject in my repo

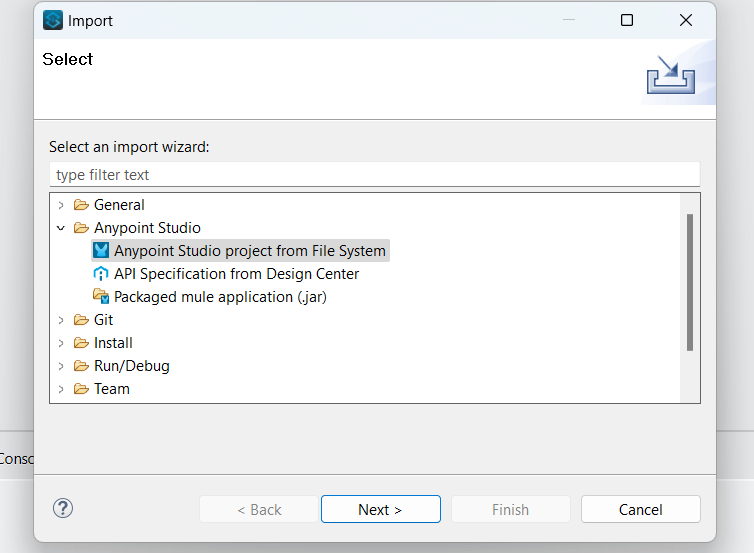
Press Enter to create your local clone.



**Step 6:**

Open anypoint studio to import the project as file system

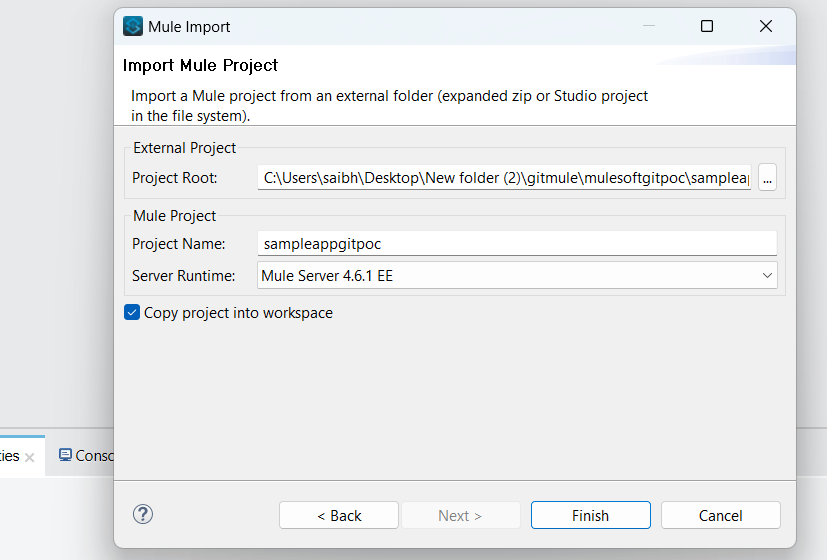




Now browse the folder of your project and click on finish

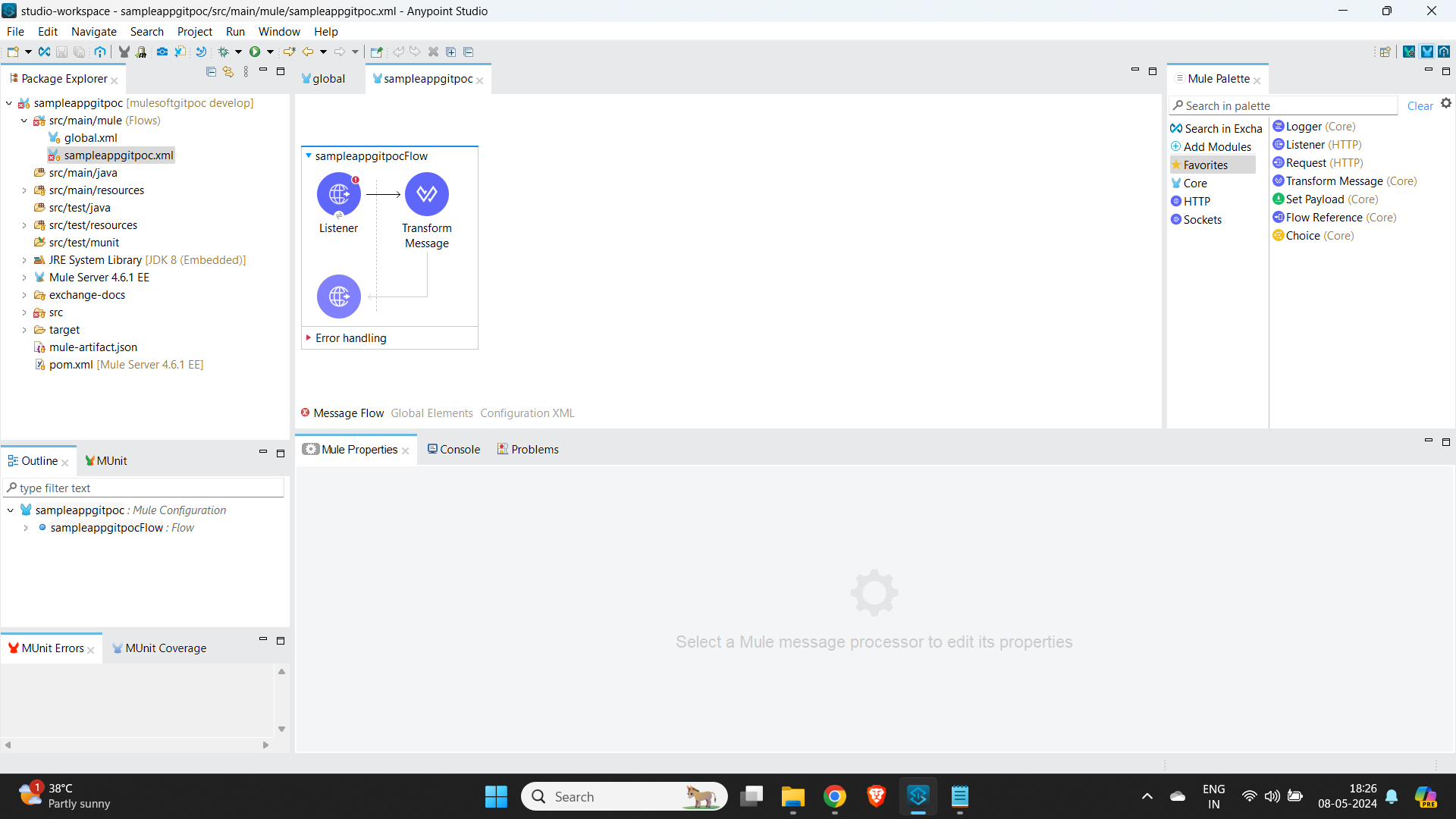
According to your requirment make sure you check or uncheck the copy project tow work space option

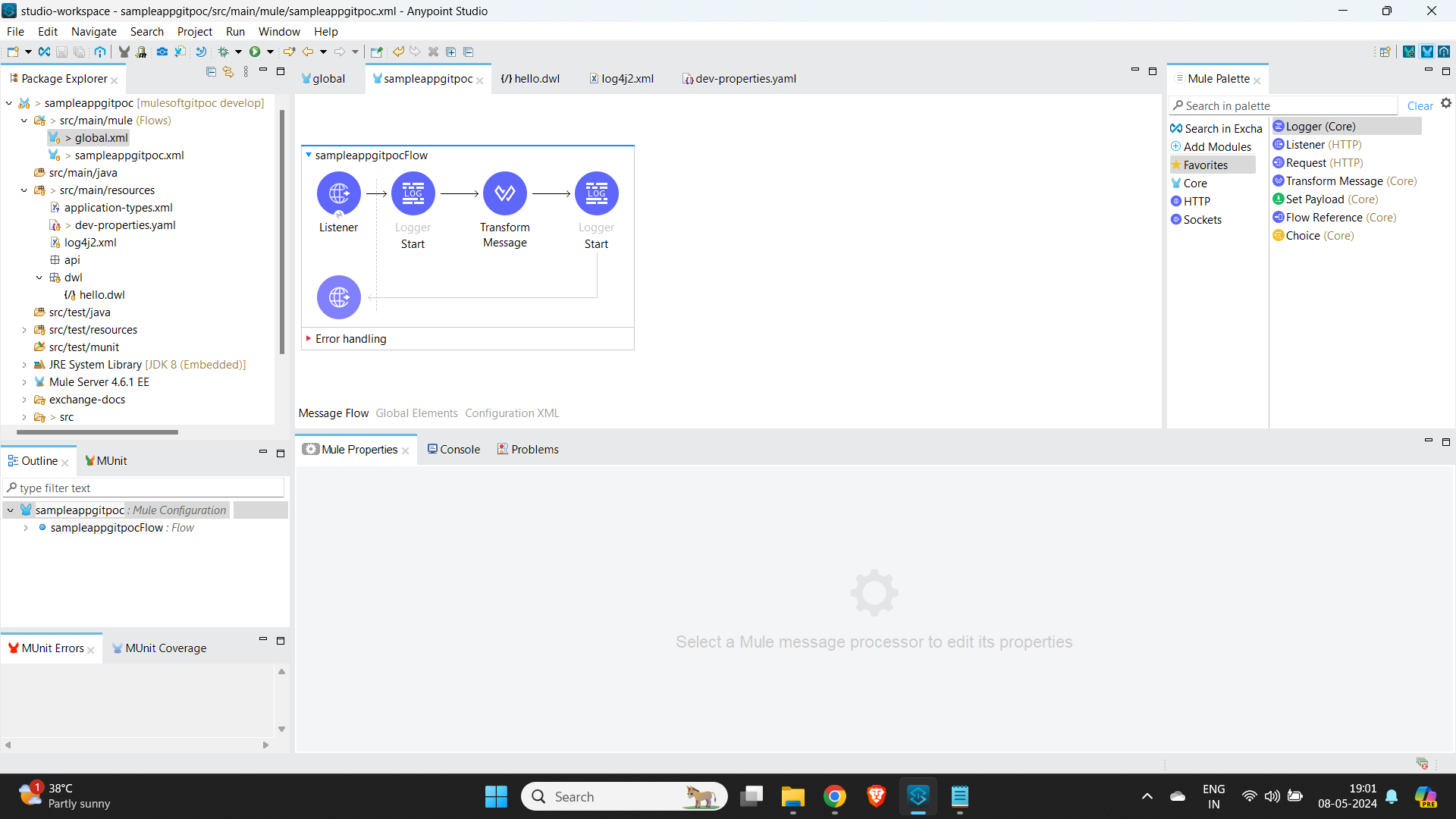
I you leave it check it will creat a copy of the project in the workspace and all the changes you make in the studio will only upload in workspace and not on on yout gitplace. If you uncheck the option all the changes are directly reflected in gitspace folder.



Click on finish you are now ready to make changes to your project.

Step 7:

* Open your project now from studio make the desired changes.
* In my case i want add config prop file and env var and two loggers to the flow
* 



* After making all the required changes save the changes
* Go the git space where the project is present

Git status:

Changes Between the Working Tree and the Index (Staging Area):

* Files that have differences between the current state in your working directory and the last commit (HEAD).
* These are the changes that would be committed if you run git commit.

Changes Between the Index and the Last Commit (HEAD):

* Files that have differences between the staging area (index) and the last commit.
* These are the changes that are staged and ready to be committed.

Untracked Files:

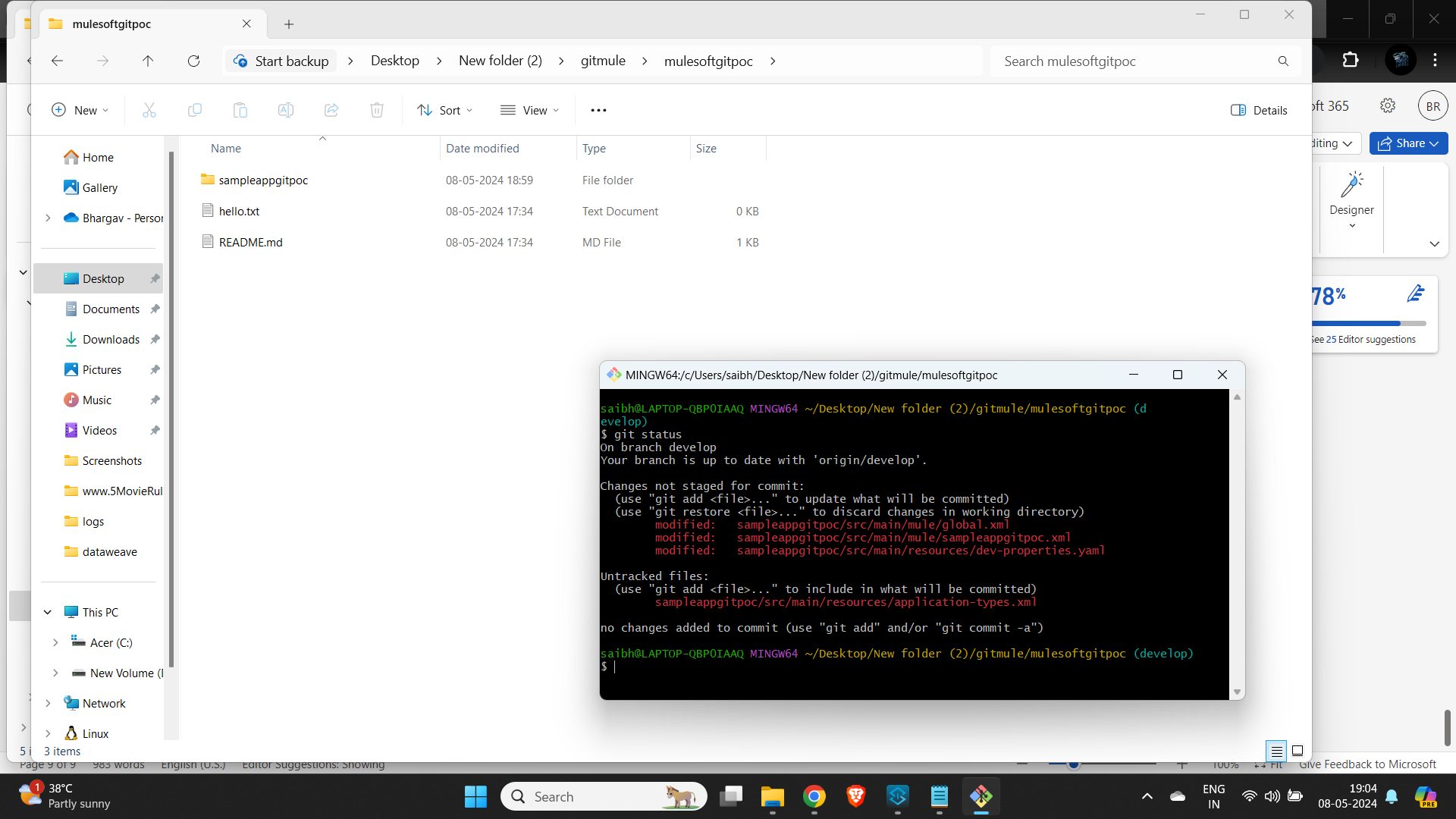
* Files in your working directory that are not tracked by Git (i.e., not part of the repository).
* Git doesn’t consider these files for commits unless you explicitly add them.

Modified:

* Files that are being tracked but modified.

Unmodified:

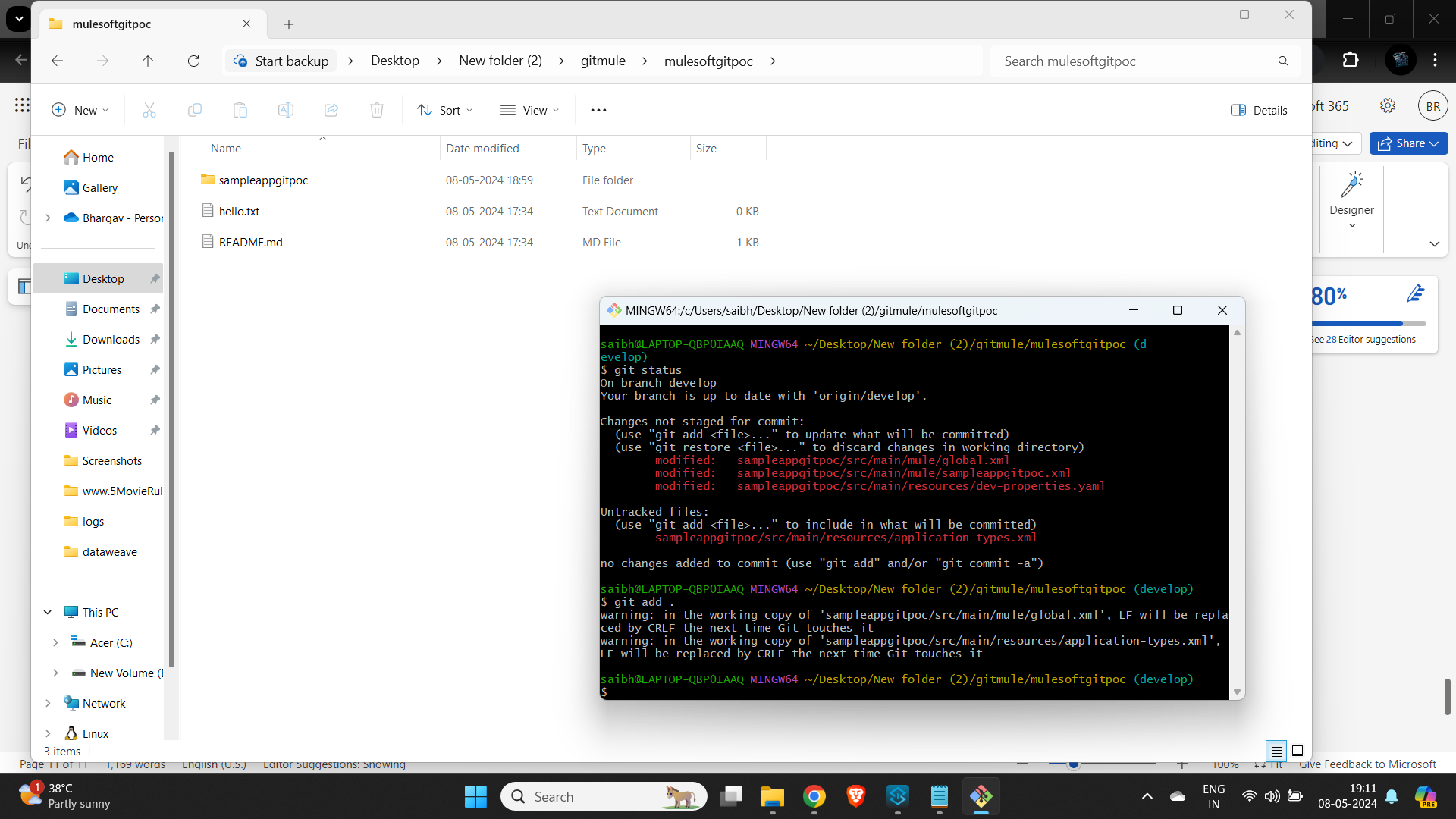
* Files that are being tracked but no changes are made to the file.



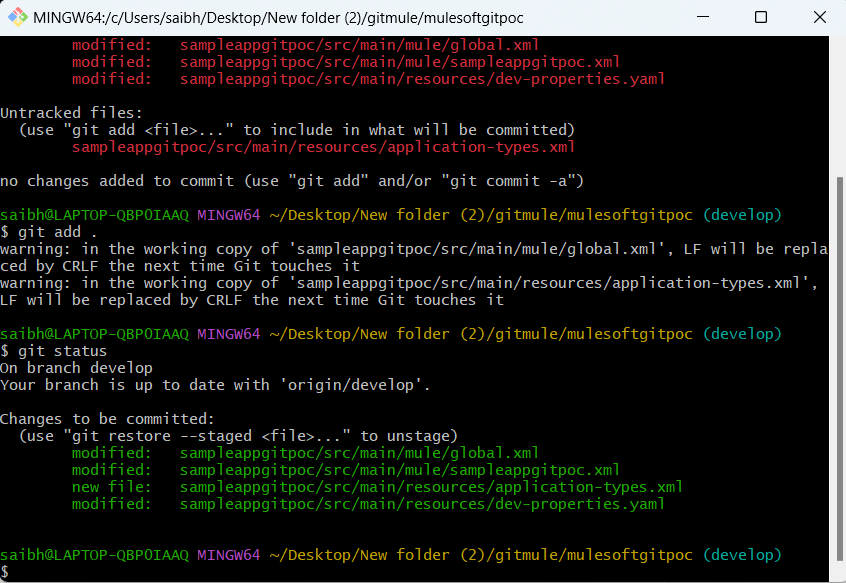
**Step 8:**

git add:

* The git add command stages changes from your working directory (where you make modifications) to the staging area (also known as the index).
* Staging means selecting which changes you want to include in the next commit.
* Now add the files to staging area
* Add all the file by using git add . Command



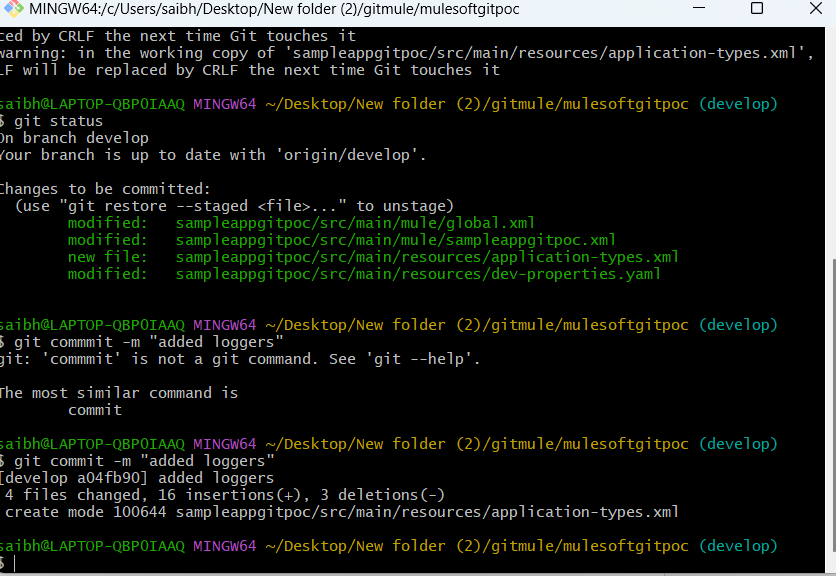
* Now run git status command to make sure that there no unstagged files



* All the files are in staged and ready to commit

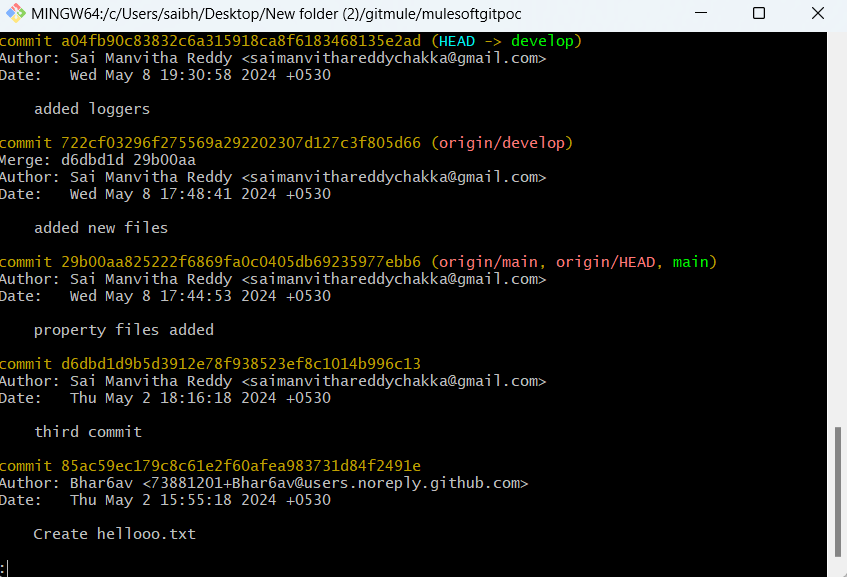
Step8:

**git commit:**

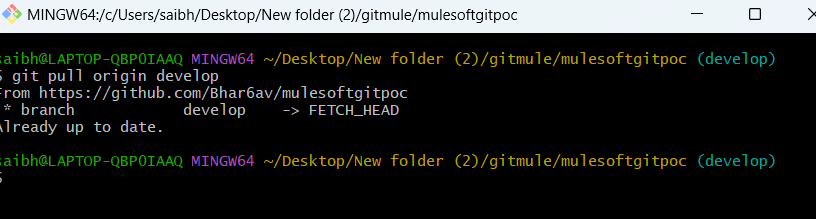
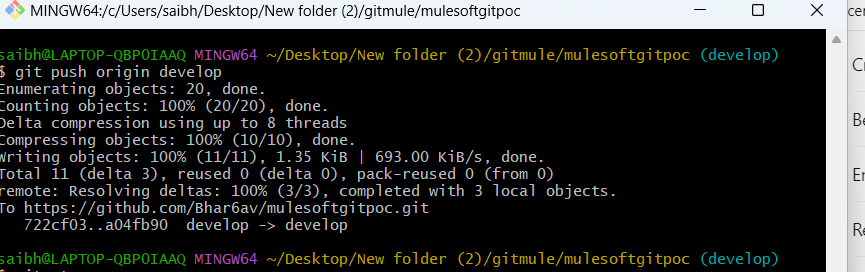
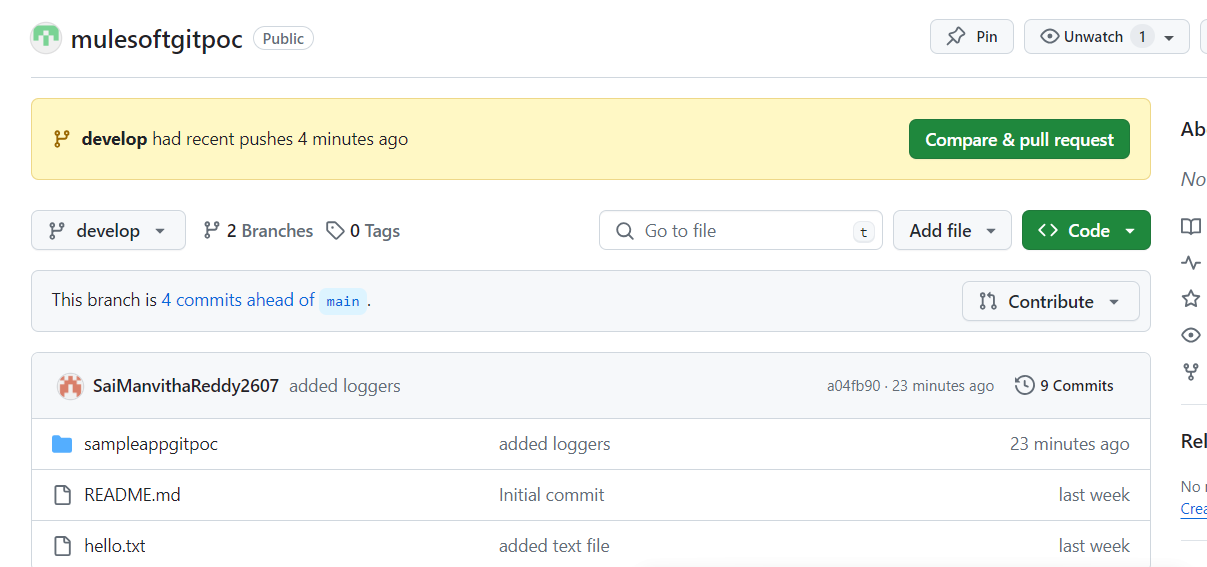
* When you’re ready to save your changes (after using git add to stage them), you use git commit.
* It creates a checkpoint in your project’s history—a snapshot of the changes you’ve made.
* Each commit represents a specific point in time and captures the state of your files.
* To commit your staged changes, use the following command:
* git commit -m "Your commit message here"
* 

git log:

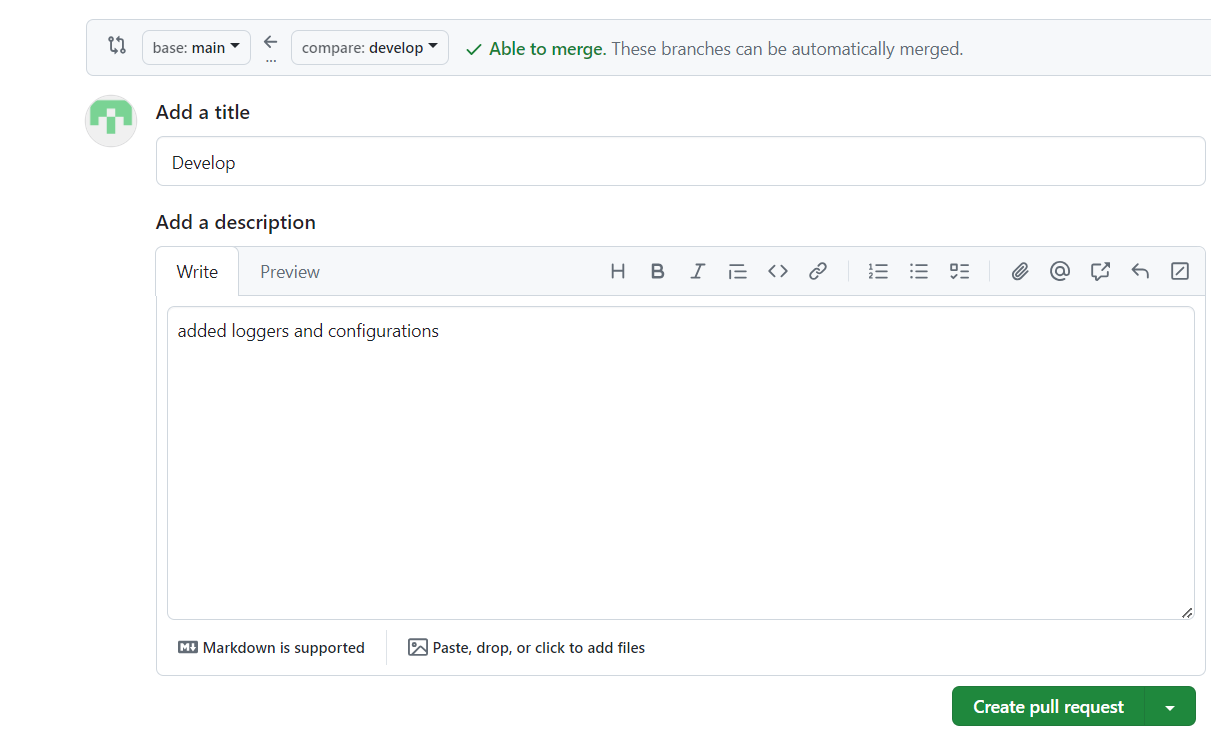
* To see the history of commits in your repository, use: git log

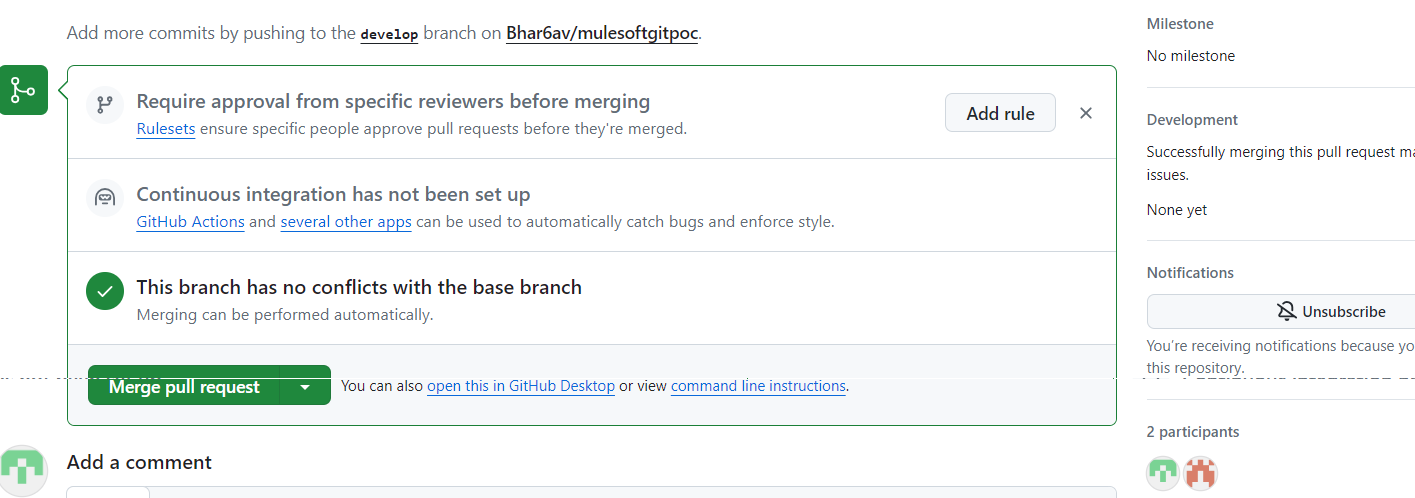
**step 9:**

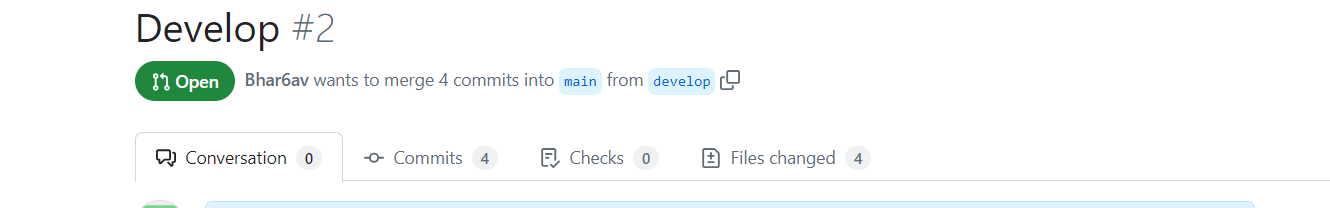
**git push:**

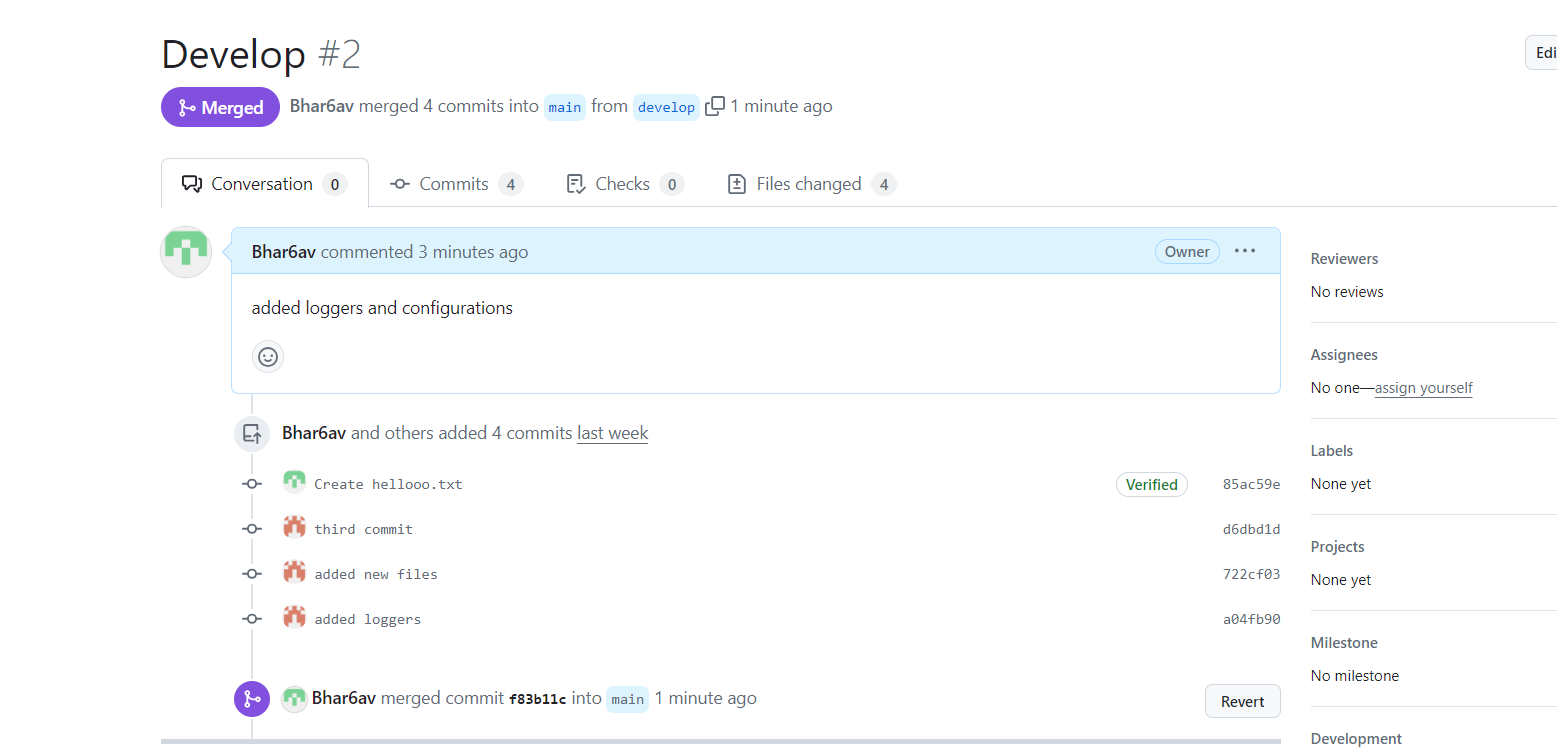
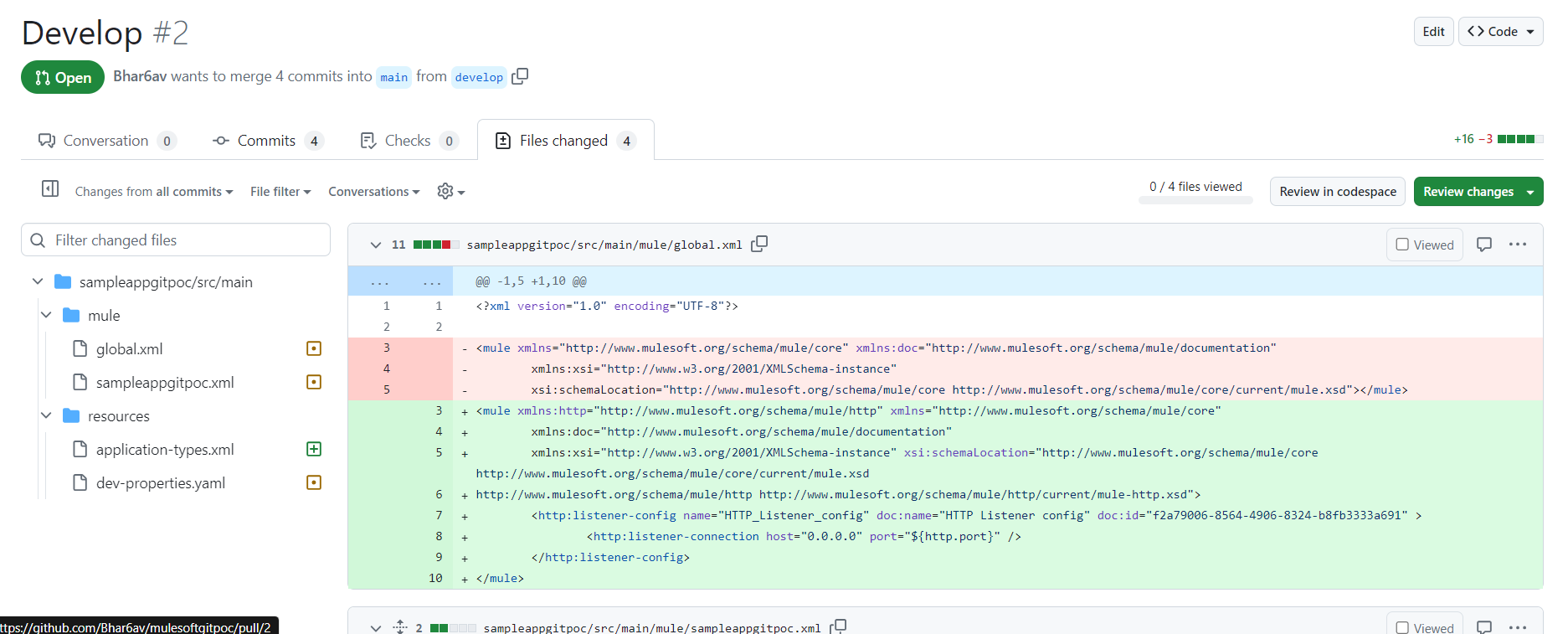
* command is used to upload your local commits from your local Git repository to a remote repository
* Cmd :git push <remote-name> <branch-name>
* For best practice use git pull command to make sure the changes are uptodate in your remote barnch and local branch as there will be many devs working on branch
* 
* **git pull** command
* 
* Now code is pushed to remote repo dev branch
* 

**Git Hub**

* Now let us raise pull request to merge the changes with develop barnch
* Raise pr (pull request in git hub to merge changes )
* 
* Once you raise a PR reviewers must approve the changes inorder to merge





* Select files changed tab to see the changes made
* 
* Once the reviews the changes and meres code now the changes are made into main branch successfully.

**Git commands:**

* To determine the origin (remote repository URL) associated with your local Git repository, you have a few options:

**Using git config:**

* To obtain only the remote URL, run:
* **git config --get remote.origin.url**
* This command will display the URL of the original remote repository.

**Using git remote show:**

* Execute the following command to display information about the remote repository associated with your local repository:
* **git remote show origin**

To view all your configured Git settings, you can use the git config --list command. This command displays a comprehensive list of settings at different levels: system, global, and local. Let’s break it down:

**Global Configuration:**

* Global Git configurations are applied to you as a user and are stored in your home directory.
* They override any system-level settings.
* To view your global Git configuration, run:
* **git config --list –global**

Changing the author information in Git can be done in a few different ways, depending on your specific needs. Let’s explore the options:

**Changing Your Committer Identity Globally:**

* If you want to change your committer name and email for all future commits globally, use the following commands:
* git config --global user.name "John Doe"  
  git config --global user.email "john@doe.org"

**Changing Your Committer Identity per Repository:**

* If you want to set special settings only for a specific repository, omit the --global flag:
* git config user.name "John Doe"  
  git config user.email "john@doe.org"

**Overwriting Author Information for the Next Commit:**

* To change the author information for just the next commit, use the --author flag:
* git commit --author="John Doe <john@doe.org>"

**Editing Past Commits (Be Cautious):**

* Editing past commits rewrites history, so proceed with caution.
* For the very last commit, use:
* git commit --amend --author="John Doe <john@doe.org>"

The git reset command is a powerful tool in Git that allows you to move your repository back to a previous commit, effectively discarding any changes made after that commit. Let’s explore how it works:

**Purpose of git reset:**

* When you want to undo commits or reset the state of your repository, you use git reset.
* It adjusts the HEAD (current branch pointer) and optionally modifies the staging area (index) and working directory.

Modes of git reset:

* soft: Moves the HEAD to the specified commit but keeps your changes in the staging area. Use this when you want to uncommit but keep your changes for recommitting later.
* mixed (default): Moves the HEAD and resets the staging area. Your changes remain in the working directory. Use this when you want to unstage changes.
* hard: Moves the HEAD, resets the staging area, and discards changes in the working directory. Use this when you want to completely discard changes.