**SCM (Source Code Management) and Git**

**1. SCM (Source Code Management):**

* **What is SCM?**
  + SCM refers to tools and practices used to manage changes in source code, track versions, and handle code revisions.
  + SCM systems help manage the history of code changes and collaborate among multiple developers working on a project.

**Key Benefits of Using SCM:**

* **Collaboration:** Multiple developers can work on the same project simultaneously without overwriting each other’s work.
* **Version Control:** Tracks all code changes, allowing developers to go back to previous versions if needed.
* **Code Review:** Changes can be reviewed before merging into the main codebase.
* **Risk Mitigation:** Prevents loss of code and ensures that any mistakes or bugs can be traced back and corrected.
* **Improved Productivity:** Developers can work in parallel, switch between versions, and merge changes easily.

**Git: A Distributed SCM Tool**

**1. Git Overview:**

* **What is Git?**
  + Git is a **distributed version control system**. It helps track changes to code, manage versions, and collaborate on development.
  + Git is **incremental**, meaning it stores only the changes made (not the whole file each time).

**2. Core Concepts in Git:**

* **Commit:**
  + A commit is like a **snapshot** of your project at a specific point in time.
  + Each commit in Git records changes to the files and is assigned a unique ID (commit hash).
* **Incremental Versioning:**
  + Git tracks only changes (additions, deletions, modifications) rather than copying the entire file, making it more efficient.
  + **Example:**  
    If you're traveling from **Tvm** (Thiruvananthapuram) to **Kazhakootam**, you only note the changes (Tvm to Kazhakootam). On another trip, you record all stops (Tvm → Chakka → Infosys → Tvm) but avoid repeating the same information.
* **Branching:**
  + Git allows developers to **branch** off the main codebase, make changes, and then merge them back. This lets developers work on different features simultaneously without interfering with each other's code.
  + **Feature Branching:** Each new feature is developed in a separate branch, keeping the main codebase stable.
* **Repository:**
  + A Git repository is where your project’s **version history** is stored. It holds all the commits, branches, and information about your project.
* **Version History:**
  + Git keeps a detailed **history** of all changes made to the codebase. This helps you track progress, go back to previous states, and find bugs introduced in past commits.

**Benefits of Using Git:**

* **Collaboration:** Multiple developers can contribute to the same project without conflict.
* **Version Control:** Git keeps track of changes and allows reverting to previous versions of the code.
* **Code Review:** Changes can be reviewed before being merged into the main branch.
* **Risk Mitigation:** Mistakes are easier to identify and fix since all changes are recorded.
* **Improved Productivity:** Git allows for parallel development, easy switching between versions, and faster project management.

**Installing Git:**

**For Ubuntu (Debian-based systems):**

* To install Git:
* sudo apt install git -y

**For Fedora (Red Hat-based systems):**

* To install Git:
* sudo dnf install git -y

**Basic Git Commands:**

1. **git init**: Initializes a new Git repository in your project directory.
2. **git add .**: Stages all changes in your project for the next commit.
3. **git commit -m "message"**: Commits your staged changes with a message describing the changes.
4. **git push**: Pushes the local commits to the remote repository (e.g., GitHub).
5. **git pull**: Pulls the latest changes from the remote repository.
6. **git status**: Shows the current status of your files (modified, staged, untracked, etc.).
7. **git log --oneline**: Shows a simplified commit history with each commit’s hash and message.
8. **git branch**: Lists the branches in your repository.
9. **git checkout branch-name**: Switches to the specified branch.
10. **git merge branch-name**: Merges changes from the specified branch into the current branch.

### Getting & Creating Projects

| **Command** | **Description** |
| --- | --- |
| git init | Initialize a local Git repository |
| git clone ssh://git@github.com/[username]/[repository-name].git | Create a local copy of a remote repository |

### Basic Snapshotting

| **Command** | **Description** |
| --- | --- |
| git status | Check status |
| git add [file-name.txt] | Add a file to the staging area |
| git add -A | Add all new and changed files to the staging area |
| git commit -m "[commit message]" | Commit changes |
| git rm -r [file-name.txt] | Remove a file (or folder) |
| git remote -v | View the remote repository of the currently working file or directory |

### Branching & Merging

| **Command** | **Description** |
| --- | --- |
| git branch | List branches (the asterisk denotes the current branch) |
| git branch -a | List all branches (local and remote) |
| git branch [branch name] | Create a new branch |
| git branch -d [branch name] | Delete a branch |
| git push origin --delete [branch name] | Delete a remote branch |
| git checkout -b [branch name] | Create a new branch and switch to it |
| git checkout -b [branch name] origin/[branch name] | Clone a remote branch and switch to it |
| git branch -m [old branch name] [new branch name] | Rename a local branch |
| git checkout [branch name] | Switch to a branch |
| git checkout - | Switch to the branch last checked out |
| git checkout -- [file-name.txt] | Discard changes to a file |
| git merge [branch name] | Merge a branch into the active branch |
| git merge [source branch] [target branch] | Merge a branch into a target branch |
| git stash | Stash changes in a dirty working directory |
| git stash clear | Remove all stashed entries |
| git stash pop | Apply latest stash to working directory |

### Sharing & Updating Projects

| **Command** | **Description** |
| --- | --- |
| git push origin [branch name] | Push a branch to your remote repository |
| git push -u origin [branch name] | Push changes to remote repository (and remember the branch) |
| git push | Push changes to remote repository (remembered branch) |
| git push origin --delete [branch name] | Delete a remote branch |
| git pull | Update local repository to the newest commit |
| git pull origin [branch name] | Pull changes from remote repository |
| git remote add origin ssh://git@github.com/[username]/[repository-name].git | Add a remote repository |
| git remote set-url origin ssh://git@github.com/[username]/[repository-name].git | Set a repository's origin branch to SSH |

### Inspection & Comparison

| **Command** | **Description** |
| --- | --- |
| git log | View changes |
| git log --summary | View changes (detailed) |
| git log --oneline | View changes (briefly) |
| git diff [source branch] [target branch] | Preview changes before merging |

**Restoring**

git restore . for restoring from staging to working directory

git restore –workspace for restoring from local repository to working directory

git restore –staged for restoring from local repository to staging

**Commit**: A commit is like a snapshot of your project at a specific point in time. Each

commit in Git records changes to the files and is assigned a unique ID (commit hash).

**Tag**: Git tags are markers used to highlight specific commits in the history of your Git

repository.

**Hooks**:

It is a way to trigger other functions. Git comes with a set of script. Script can automatically

run at every meaningful phase.

**Index**: File is where git stores staging area information

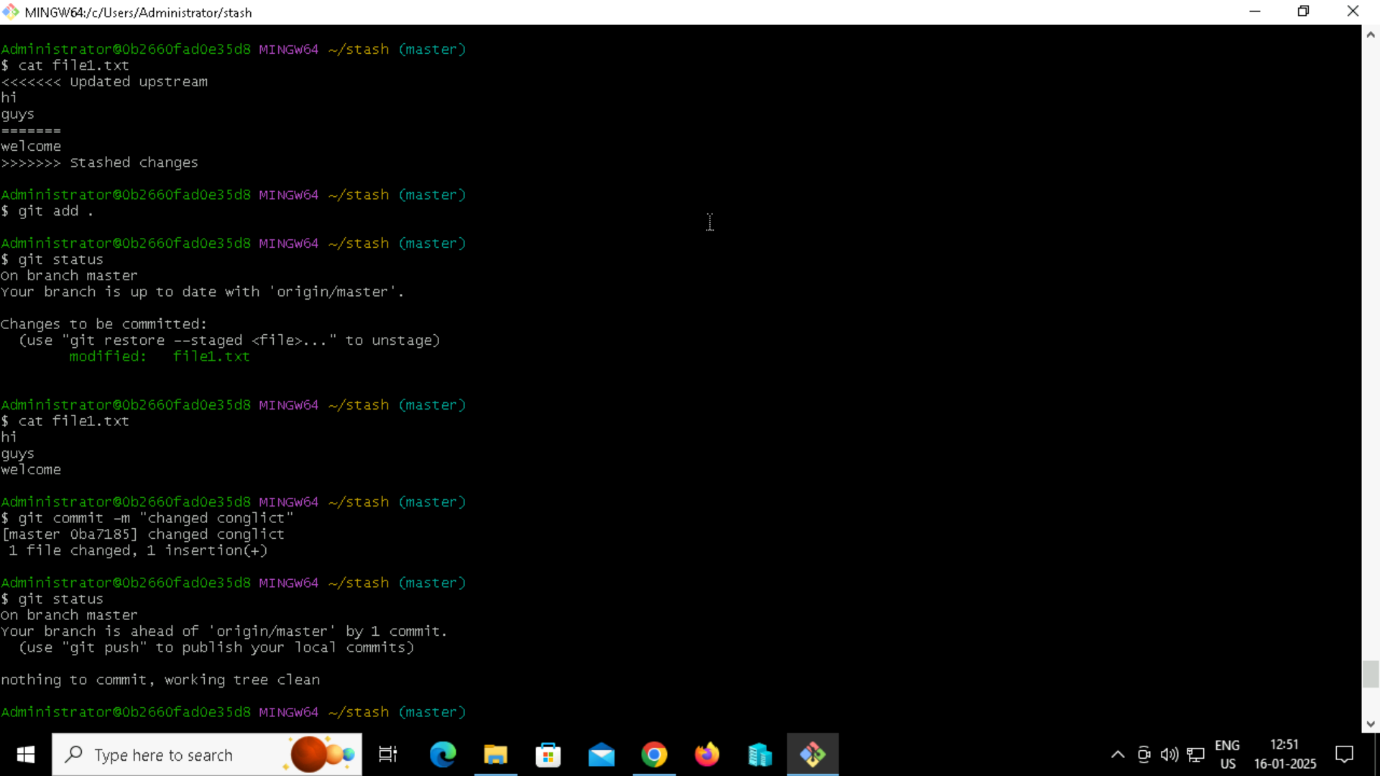
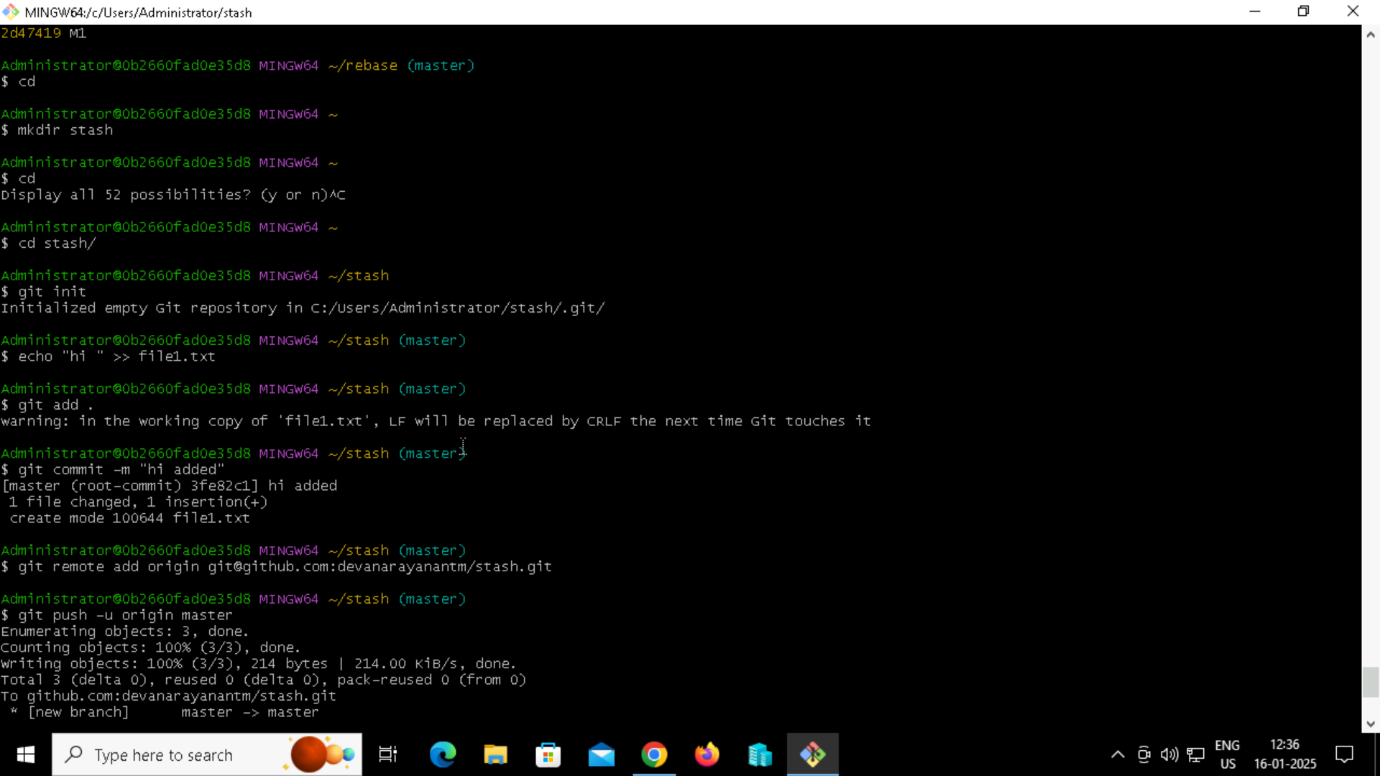
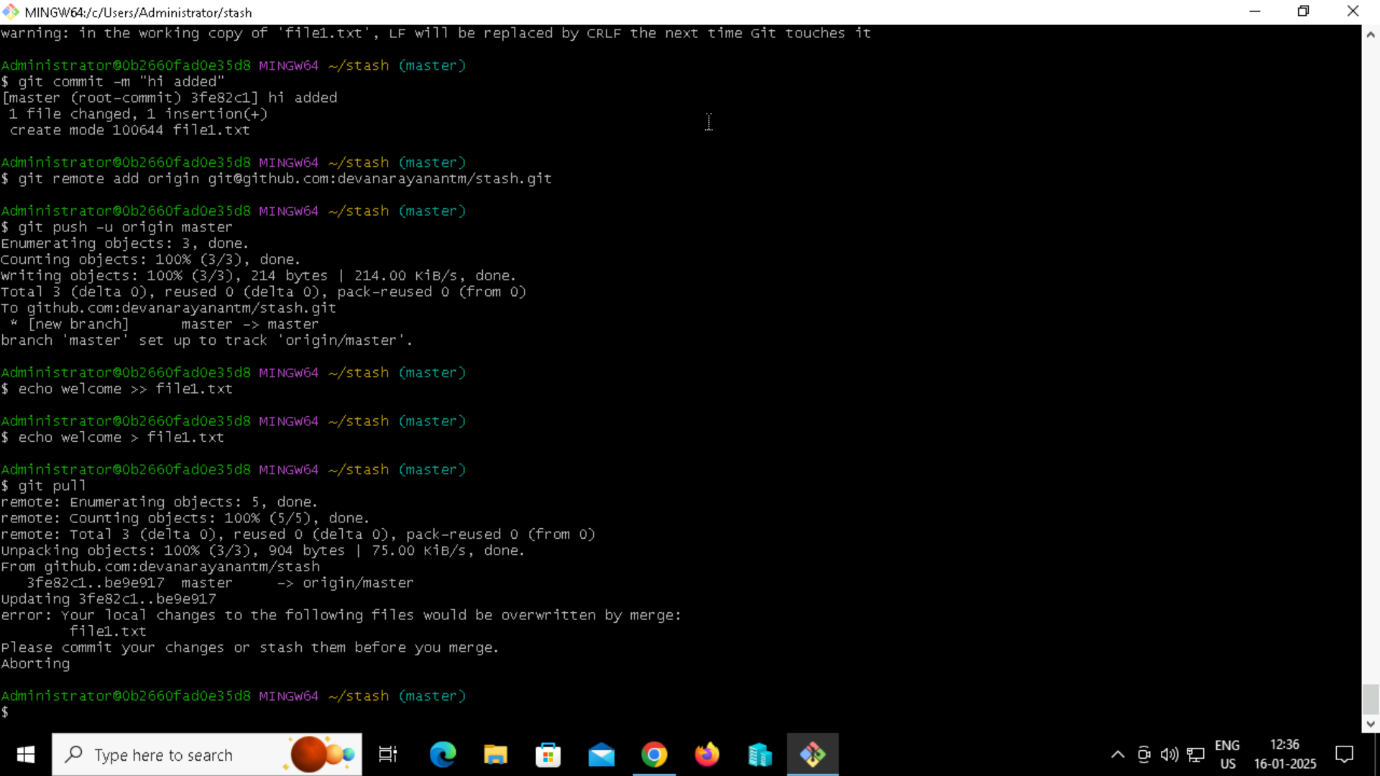
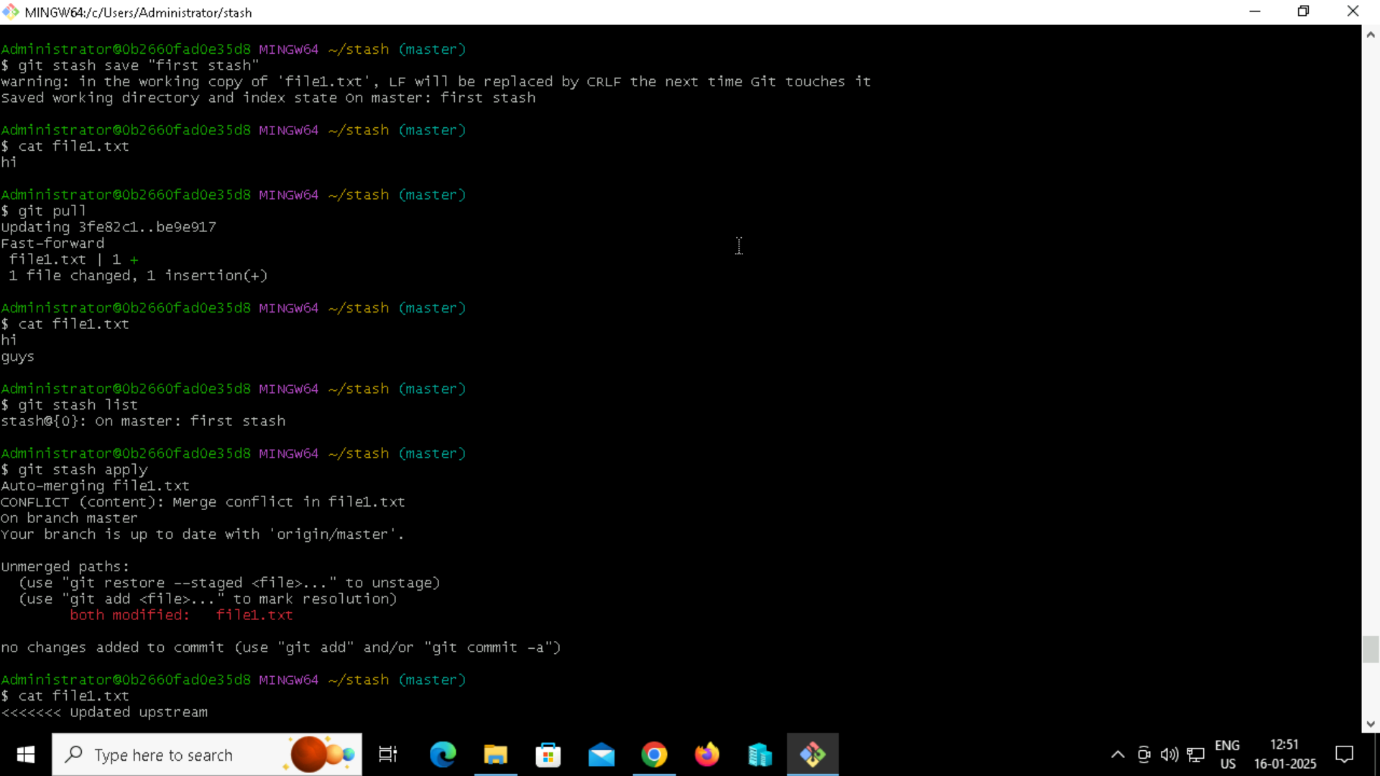
16/01/2025

git commit –- amend

This command helps to edit the commit message of the last message. The below screenshot

displays the commit messaged modified

**Stashing**



**Reset:**

**Soft :** moves the branch pointer (effects only in local repository)

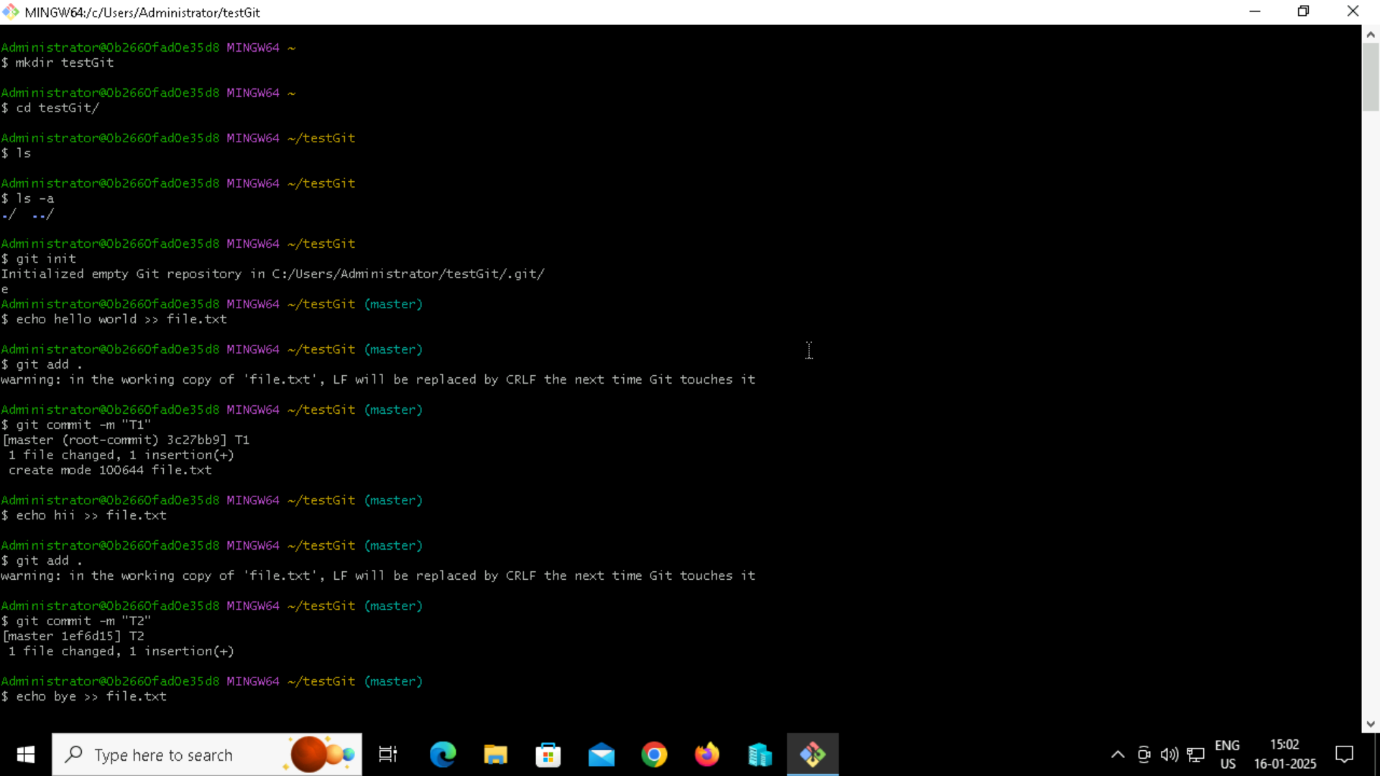
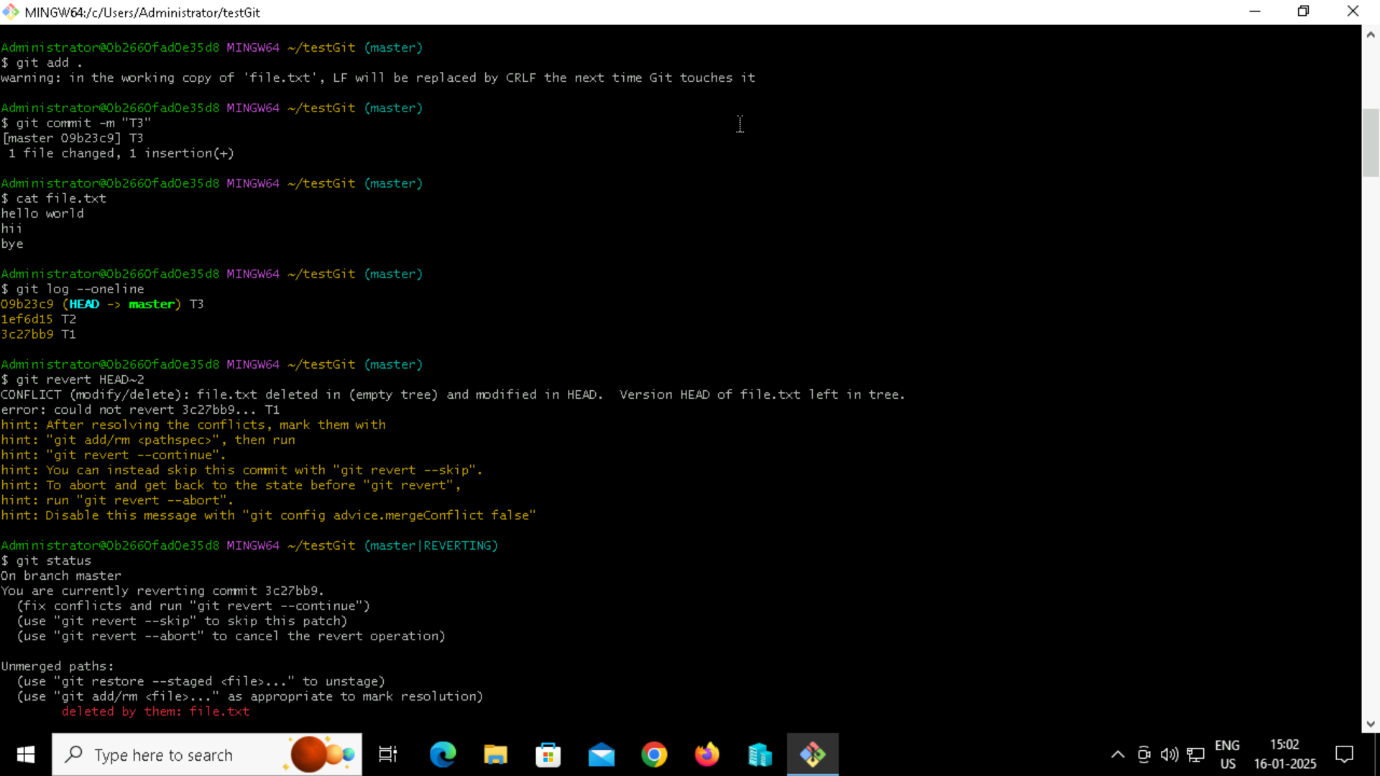
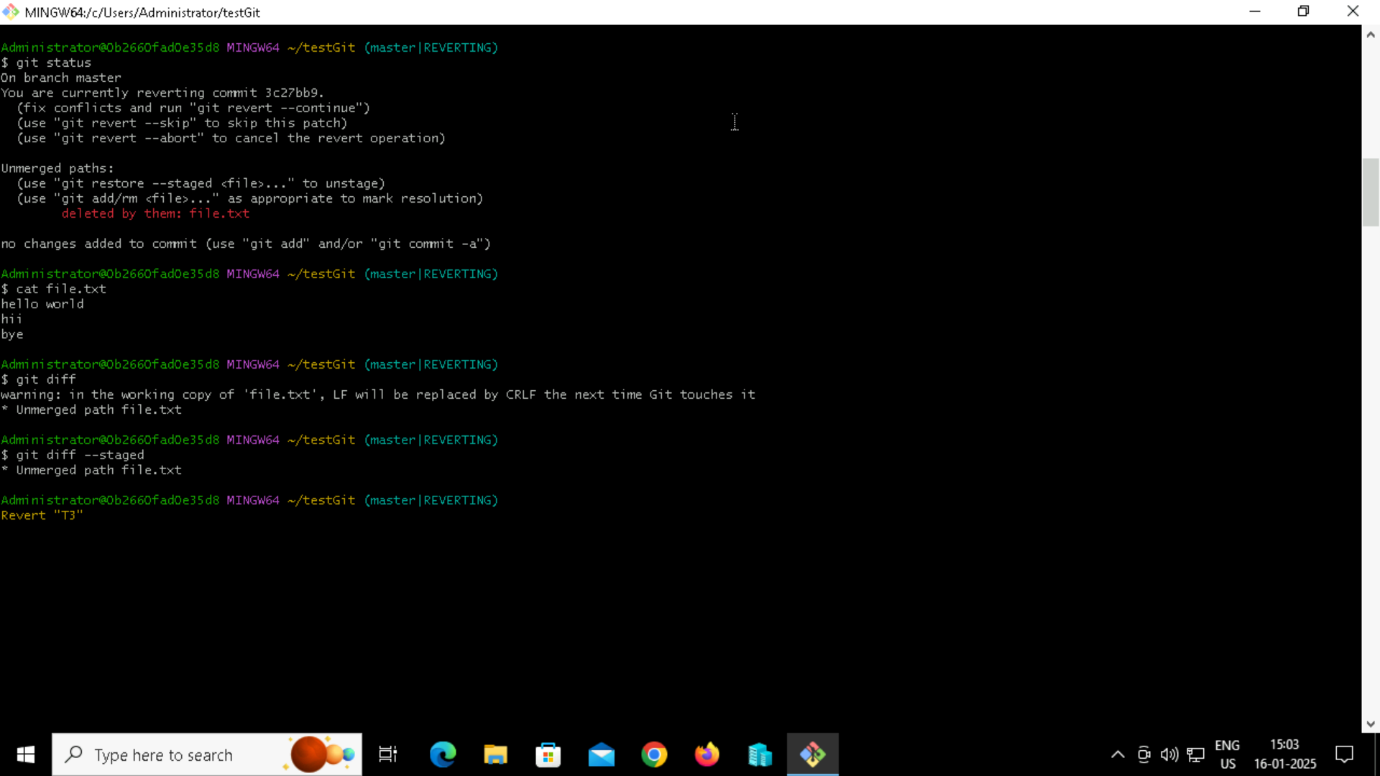
You will remove the last commit from the current branch, but the file changes will stay in working

tree and in index

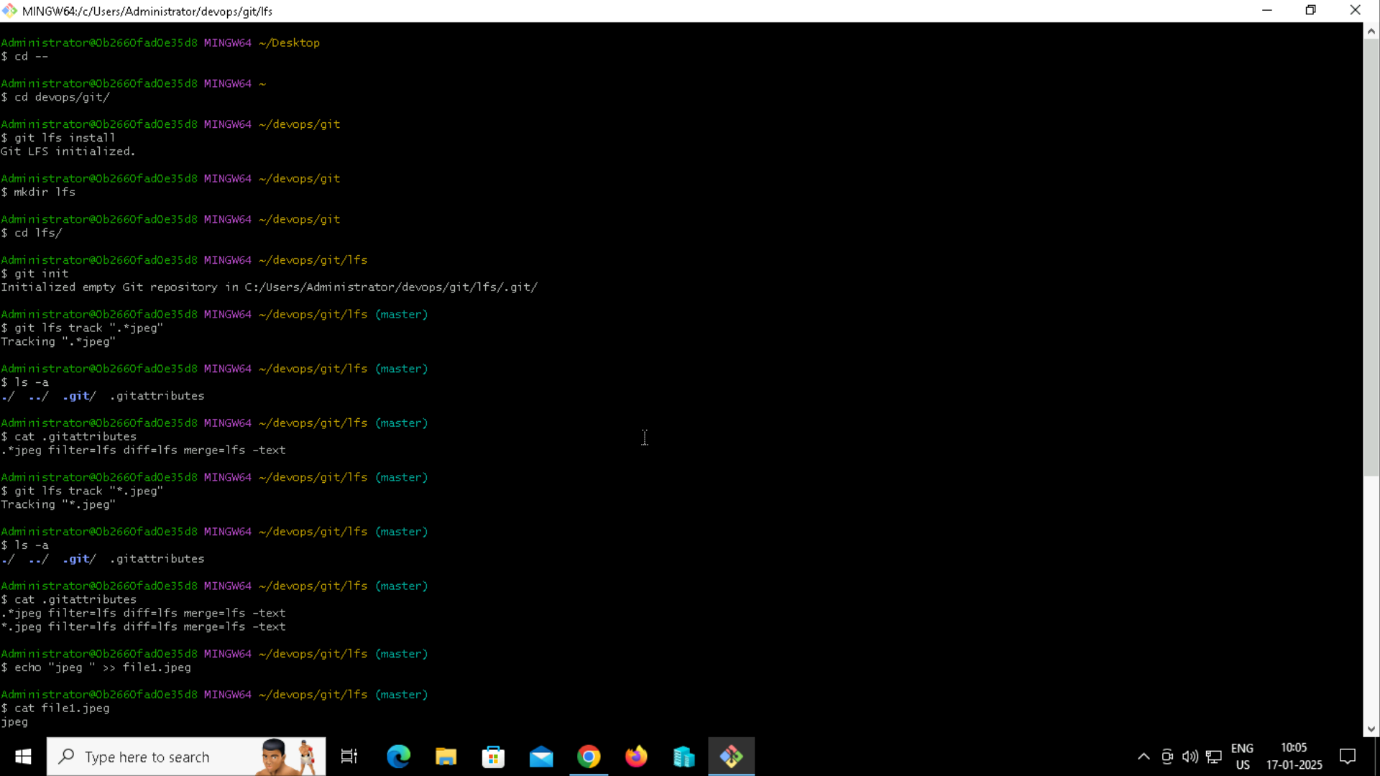
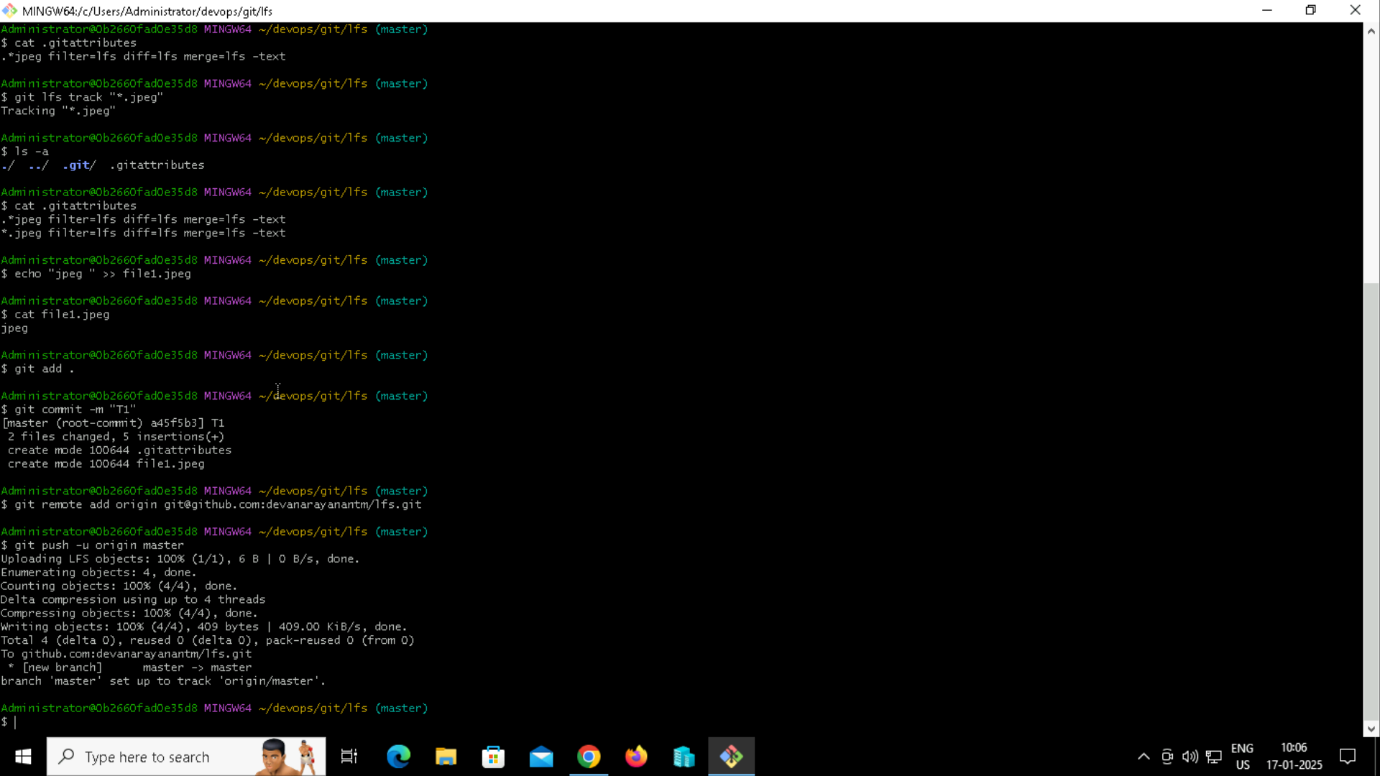
**Hard:** it removes in all stages

**Mixed:** k mode You will still keep the changes in your working tree but not on index

**Revert**



LFS



SIGNED COMMIT

