	Pimpri Chinchwad Education Trust's **Pimpri Chinchwad College of Engineering (PCCoE)** (An Autonomous Institute) Affiliated to Savitribai Phule Pune University(SPPU) ISO 21001:2018 Certified by TUV	

Course: DevOps Laboratory	**Code:** BIT26VS01
Name: Amar Vaijinath Chavan	**PRN:** 124B2F001
Assignment 2: Demonstrate the ability to use different git commands to working with local repository, remote repository and log operation	

Aim: To comprehensively demonstrate the use of Git commands to manage a project's lifecycle, including local tracking, remote synchronization, branching strategies, and history manipulation.

Objectives:

1. To understand the detailed three-tier architecture of Git (Working, Staging, Repository).
2. To master branching, merging, and rebasing for professional parallel development.
3. To implement remote operations for team collaboration and synchronization.
4. To learn advanced undo and history inspection techniques.

Theory:

1. The Logic of Distributed Version Control Git is a Distributed Version Control System (DVCS) where every developer's machine contains a full backup of the project history. This architecture allows for lightning-fast local operations and ensures data integrity. The primary goal of Git is to allow developers to experiment without the risk of destroying stable code.

2. Understanding the Staging Area (The Index) The Staging Area is a unique Git feature that acts as a middle ground between your working directory and the final repository. It allows you to selectively choose which changes should be part of the next commit, enabling clean and logical commit histories rather than bulk saving everything.

3. The Master-Worker Distributed Model Git's distributed nature means every developer has a local "Source of Truth". Unlike centralized systems, this allows for redundant backups across the team and enables complex operations like rebasing and local branching without affecting the shared remote repository until the developer is ready to push.

4. Merge Conflict Anatomy A merge conflict occurs when Git's auto-merge algorithm cannot decide between two competing changes. This typically happens when the same line in a file is modified differently on two branches. The conflict markers (<<<<, ====, >>>>) are Git's way of pausing the process and handing "Quality Control" back to the developer to ensure no critical logic is lost during integration.

5. The Logic of Reverting Merges Reverting a merge commit is complex because a merge has multiple "parents" (source branches). Using the -m 1 flag is essential because it tells Git which

parent branch should be considered the "mainline" to keep. This allows a team to quickly back out of a faulty integration while keeping the project history intact for future audits.

6. **History Rewriting with Reset** The git reset command is a powerful tool for correcting local mistakes before they are shared. By using the --soft flag, a developer can "undo" a commit while keeping all their work in the staging area. This is often used to combine multiple small, messy commits into a single, clean "atomic" commit that is easier for teammates to review.

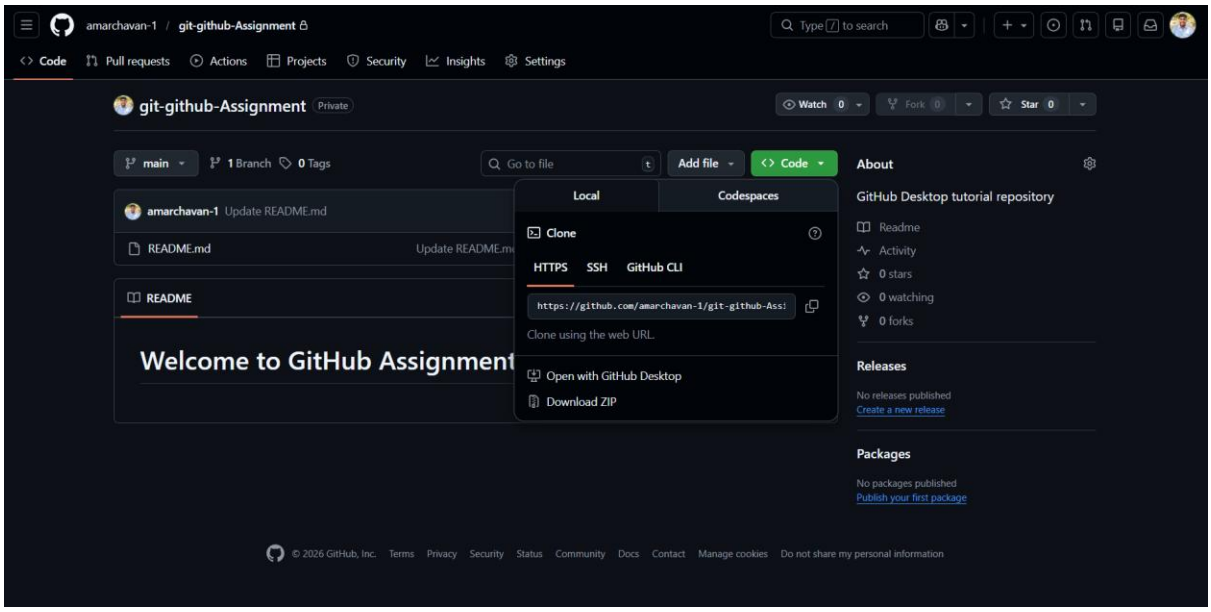
7. **Branching, Merging, and Rebasing**

- **Branching:** Creates an independent path for feature development.
- **Merging:** Combines history from two branches (creates a merge commit).
- **Rebasing:** Moves the entire feature branch so it begins on the tip of the master branch, creating a cleaner, linear history.

Practical Procedure / Detailed Execution Steps:

Step 1: Remote Setup and Initialization

1. **GitHub Setup:** Log in to GitHub and create a new public repository.



2. **Local Workspace:**
   mkdir assignment2 && cd assignment2
   git init # Initialize the local repository

```
C:\Users\Amar\Desktop\DevOps\devops_lab>mkdir assignment2

C:\Users\Amar\Desktop\DevOps\devops_lab>cd assignment2

C:\Users\Amar\Desktop\DevOps\devops_lab\assignment2>git init
Initialized empty Git repository in C:/Users/Amar/Desktop/DevOps/devops_lab/assignment2/.git/
```

3. **Linking Remote:** Connect the local repo to the GitHub URL.
   git remote add origin https://<github_url>

Step 2: Local Workflow and Basic Tracking

1. **File Creation:** Add a basic file and verify its status.
   git status # View untracked files
2. **Staging and Committing:**
   git add .
   git commit -m "<add commit message>"

```
C:\Users\Amar\Desktop\DevOps\devops_lab\assignment2>git remote add origin https://github.com/amarchavan-1/git-github-Assignment.git

C:\Users\Amar\Desktop\DevOps\devops_lab\assignment2>git remote -v
origin  https://github.com/amarchavan-1/git-github-Assignment.git (fetch)
origin  https://github.com/amarchavan-1/git-github-Assignment.git (push)

C:\Users\Amar\Desktop\DevOps\devops_lab\assignment2>git branch

C:\Users\Amar\Desktop\DevOps\devops_lab\assignment2>

C:\Users\Amar\Desktop\DevOps\devops_lab\assignment2>echo "DevOps Assignment 2" > readme.md

C:\Users\Amar\Desktop\DevOps\devops_lab\assignment2>git status
On branch main

No commits yet

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        readme.md

nothing added to commit but untracked files present (use "git add" to track)

C:\Users\Amar\Desktop\DevOps\devops_lab\assignment2>git add .

C:\Users\Amar\Desktop\DevOps\devops_lab\assignment2>git commit -m "Initial commit: Added readme"
[main (root-commit) 578cc45] Initial commit: Added readme
 1 file changed, 1 insertion(+)
 create mode 100644 readme.md
```

Step 3: Branching and Forced Conflict Generation

1. **Feature Branching:**
   git branch feature-dev
   git switch feature-dev
2. **Simultaneous Edits:** Edit file on the branch, then switch to main and create a second file.
3. Edit the same conflict.txt on feature-dev.

```
C:\Users\Amar\Desktop\DevOps\devops_lab\assignment2>git branch feature-dev

C:\Users\Amar\Desktop\DevOps\devops_lab\assignment2>git checkout feature-dev
Switched to branch 'feature-dev'

C:\Users\Amar\Desktop\DevOps\devops_lab\assignment2>echo "Feature work in progress" >> readme.md

C:\Users\Amar\Desktop\DevOps\devops_lab\assignment2>git add . && git commit -m "feat: added development notes"
[feature-dev edca7fe] feat: added development notes
 1 file changed, 1 insertion(+)

C:\Users\Amar\Desktop\DevOps\devops_lab\assignment2>git checkout main
Switched to branch 'main'

C:\Users\Amar\Desktop\DevOps\devops_lab\assignment2>echo "MAIN EDIT" > conflict.txt

C:\Users\Amar\Desktop\DevOps\devops_lab\assignment2>git add . && git commit -m "Conflict file: main version"
[main 5ce701e] Conflict file: main version
 1 file changed, 1 insertion(+)
 create mode 100644 conflict.txt

C:\Users\Amar\Desktop\DevOps\devops_lab\assignment2>git checkout feature-dev
Switched to branch 'feature-dev'

C:\Users\Amar\Desktop\DevOps\devops_lab\assignment2>echo "FEATURE EDIT" > conflict.txt

C:\Users\Amar\Desktop\DevOps\devops_lab\assignment2>git add . && git commit -m "Conflict file: feature version"
[feature-dev 72b6cc8] Conflict file: feature version
 1 file changed, 1 insertion(+)
 create mode 100644 conflict.txt
```

Step 4: Manual Conflict Resolution

1. **Triggering the Merge:** Switch to main and attempt to integrate the branch.
   git checkout main
   git merge feature-dev
   Result: Automatic merge fails.
2. **Manual Fix:** Open conflict.txt, remove markers (<<<<, ====, >>>>), and keep the desired text.
3. **Finalizing Resolution:**
   Bash
   git add conflict.txt
   git commit -m "resolved merge conflict manually"
4. Check Logs of commits

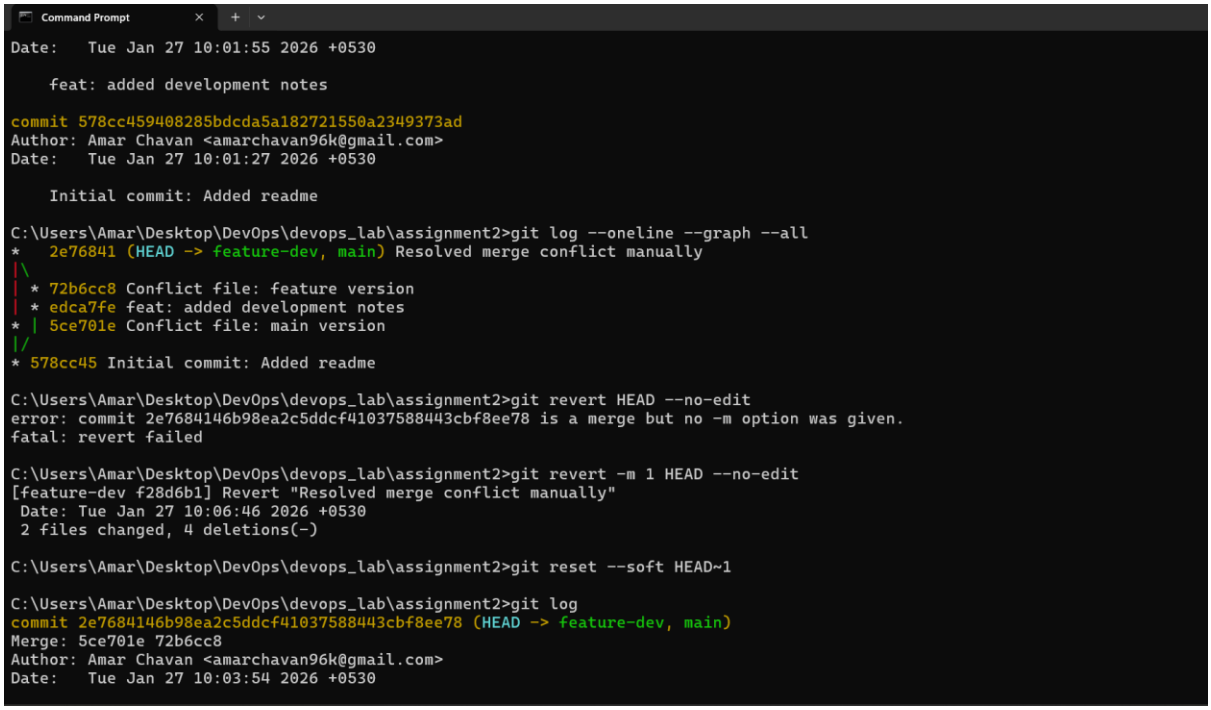
```
C:\Users\Amar\Desktop\DevOps\devops_lab\assignment2>git checkout main
Switched to branch 'main'

C:\Users\Amar\Desktop\DevOps\devops_lab\assignment2>git merge feature-dev
Auto-merging conflict.txt
CONFLICT (add/add): Merge conflict in conflict.txt
Automatic merge failed; fix conflicts and then commit the result.

C:\Users\Amar\Desktop\DevOps\devops_lab\assignment2>git add conflict.txt

C:\Users\Amar\Desktop\DevOps\devops_lab\assignment2>nano conflict.txt

C:\Users\Amar\Desktop\DevOps\devops_lab\assignment2>git status
On branch main
All conflicts fixed but you are still merging.
  (use "git commit" to conclude merge)

Changes to be committed:
        modified:   conflict.txt
        modified:   readme.md

Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        modified:   conflict.txt
```

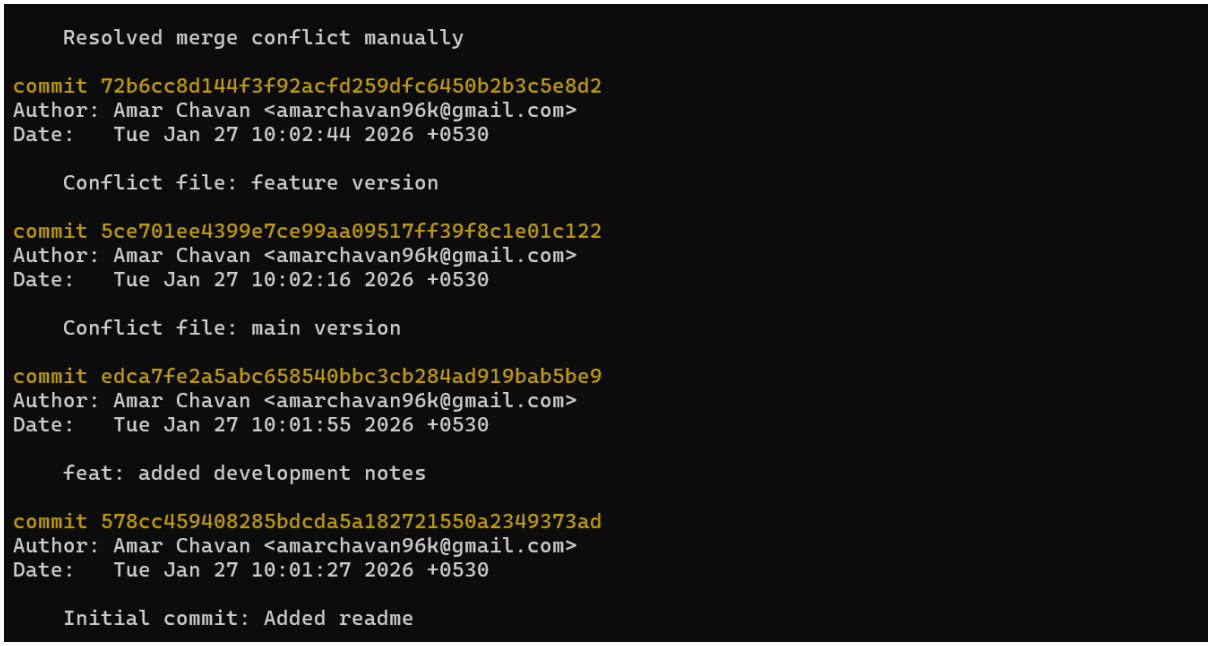
```
C:\Users\Amar\Desktop\DevOps\devops_lab\assignment2>git add conflict.txt

C:\Users\Amar\Desktop\DevOps\devops_lab\assignment2>git commit -m "Resolved merge conflict manually"
[main 2e76841] Resolved merge conflict manually

C:\Users\Amar\Desktop\DevOps\devops_lab\assignment2>git branch
  feature-dev
* main

C:\Users\Amar\Desktop\DevOps\devops_lab\assignment2>git log
commit 2e7684146b98ea2c5ddcf41037588443cbf8ee78 (HEAD -> main)
Merge: 5ce701e 72b6cc8
Author: Amar Chavan <amarchavan96k@gmail.com>
Date:   Tue Jan 27 10:03:54 2026 +0530

    Resolved merge conflict manually

commit 72b6cc8d144f3f92acfd259dfc6450b2b3c5e8d2 (feature-dev)
Author: Amar Chavan <amarchavan96k@gmail.com>
Date:   Tue Jan 27 10:02:44 2026 +0530

    Conflict file: feature version

commit 5ce701ee4399e7ce99aa09517ff39f8c1e01c122
Author: Amar Chavan <amarchavan96k@gmail.com>
Date:   Tue Jan 27 10:02:16 2026 +0530

    Conflict file: main version

commit edca7fe2a5abc658540bbc3cb284ad919bab5be9
Author: Amar Chavan <amarchavan96k@gmail.com>
Date:   Tue Jan 27 10:01:55 2026 +0530
```

```
commit 578cc459408285bdcda5a182721550a2349373ad
Author: Amar Chavan <amarchavan96k@gmail.com>
Date:   Tue Jan 27 10:01:27 2026 +0530

    Initial commit: Added readme

C:\Users\Amar\Desktop\DevOps\devops_lab\assignment2>git checkout feature-dev
Switched to branch 'feature-dev'

C:\Users\Amar\Desktop\DevOps\devops_lab\assignment2>git rebase main
Successfully rebased and updated refs/heads/feature-dev.

C:\Users\Amar\Desktop\DevOps\devops_lab\assignment2>git log
commit 2e7684146b98ea2c5ddcf41037588443cbf8ee78 (HEAD -> feature-dev, main)
Merge: 5ce701e 72b6cc8
Author: Amar Chavan <amarchavan96k@gmail.com>
Date:   Tue Jan 27 10:03:54 2026 +0530

    Resolved merge conflict manually

commit 72b6cc8d144f3f92acfd259dfc6450b2b3c5e8d2
Author: Amar Chavan <amarchavan96k@gmail.com>
Date:   Tue Jan 27 10:02:44 2026 +0530

    Conflict file: feature version

commit 5ce701ee4399e7ce99aa09517ff39f8c1e01c122
Author: Amar Chavan <amarchavan96k@gmail.com>
Date:   Tue Jan 27 10:02:16 2026 +0530

    Conflict file: main version

commit edca7fe2a5abc658540bbc3cb284ad919bab5be9
Author: Amar Chavan <amarchavan96k@gmail.com>
```

Step 5: Advanced History Operations

1. **History Inspection:**
   git log --oneline --graph --all
2. **The "Merge Revert" (Failed vs. Successful):**
   git revert HEAD --no-edit
   git revert -m 1 HEAD --no-edit
3. **Soft Reset:** Undo the revert while keeping the code staged.
   git reset --soft HEAD~1
   git log # Verification of removed commit

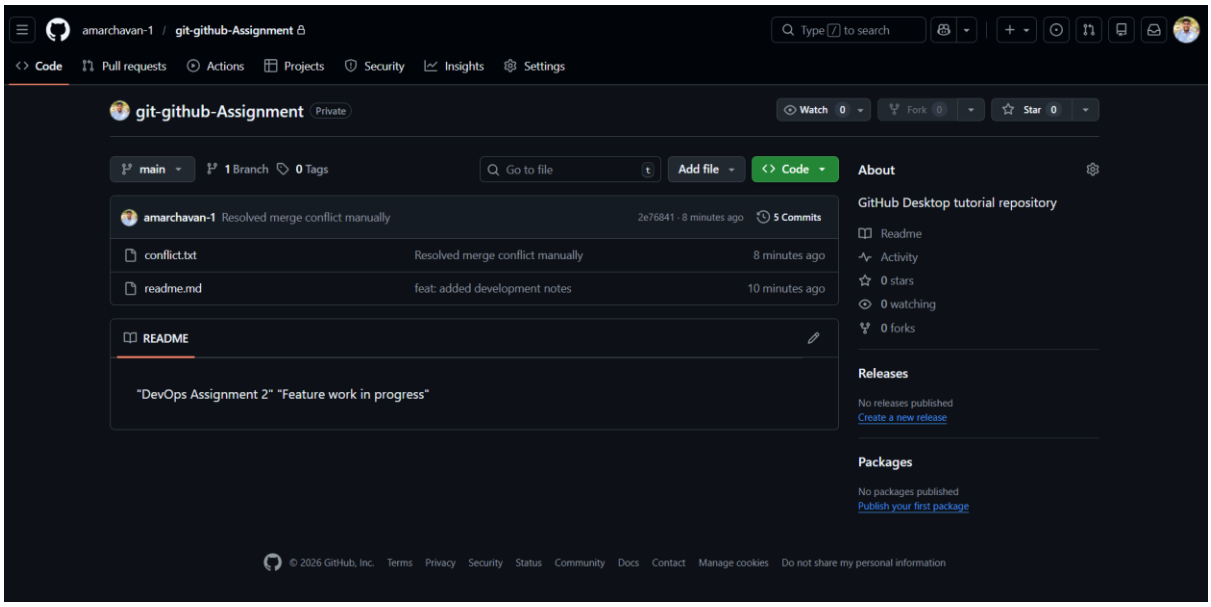




Step 6: Remote Synchronization

Finalize the assignment by syncing all local history changes to the GitHub cloud.
git pull origin main
git push -u origin main --force

```
C:\Users\Amar\Desktop\DevOps\devops_lab\assignment2>git pull origin main
remote: Enumerating objects: 6, done.
remote: Counting objects: 100% (6/6), done.
remote: Compressing objects: 100% (3/3), done.
remote: Total 6 (delta 0), reused 3 (delta 0), pack-reused 0 (from 0)
Unpacking objects: 100% (6/6), 1.23 KiB | 35.00 KiB/s, done.
From https://github.com/amarchavan-1/git-github-Assignment
 * branch            main       -> FETCH_HEAD
 * [new branch]      main       -> origin/main
fatal: refusing to merge unrelated histories

C:\Users\Amar\Desktop\DevOps\devops_lab\assignment2>git push -u origin main --force
Enumerating objects: 15, done.
Counting objects: 100% (15/15), done.
Delta compression using up to 4 threads
Compressing objects: 100% (9/9), done.
Writing objects: 100% (15/15), 1.31 KiB | 335.00 KiB/s, done.
Total 15 (delta 1), reused 0 (delta 0), pack-reused 0 (from 0)
remote: Resolving deltas: 100% (1/1), done.
To https://github.com/amarchavan-1/git-github-Assignment.git
 + 84ffbc3...2e76841 main -> main (forced update)
branch 'main' set up to track 'origin/main'.
```



Conclusion

The successful execution of this assignment demonstrates a comprehensive mastery of the **Distributed Version Control System (DVCS)** workflow, which is a core pillar of modern DevOps. By implementing a **Feature Branching Strategy**, we established a safe environment for parallel development, ensuring that the main branch remains stable while experimental work is isolated. The manual resolution of **Merge Conflicts** highlighted the critical role of the developer in managing code integrity when automated systems reach their logical limits.

Furthermore, the application of advanced commands like **revert -m 1** and **reset --soft** proved essential for maintaining a clean, professional project history that is both auditable and reversible. Finally, synchronizing these complex local changes with a **Remote GitHub Repository** using push --force confirmed the ability to manage the global "Source of Truth" in a distributed team environment. Ultimately, these Git proficiencies provide the technical foundation necessary for building automated **CI/CD pipelines**, where version control triggers the entire build, test, and deployment lifecycle.