PRACTICAL NO. 2

Exploring Git and GitHub

LOB2	To obtain knowledge of Version Control Systems to effectively track changes with Git, GitHub and understand their best practices in team environments.
LO2	Demonstrate the use of Git and GitHub to manage version control in projects and compare different workflows.

Git, Git Hub and Repositories –

Git is a distributed open-source version control software/ system that helps developers to keep track of changes made to files and directories leading to ease of collaboration on software projects. This feature is crucial for maintaining a detailed history of the project's evolution and offers an easy way for reverting changes if necessary. This is possible, as each developer has a copy of the codebase, and changes are synchronized between them. This allows for a more flexible and decentralized development process, as developers can work on their changes without worrying about interfering with the work of others.
Git is designed to be fast and efficient, even with very large codebases. It uses a content-addressable file system to store all versions of files in the repository, which allows for quick access and easy retrieval of past/ earlier versions of code.

GitHub is a Git repository hosting service that provides a web-based graphical interface (GUI). It helps every team member work together on a project from anywhere, making it easy to collaborate. GitHub is one place where project managers and developers coordinate, track, and update their work, so projects stay transparent and on schedule. The packages can be published privately, within the team, or publicly for the open-source community. Downloading packages from GitHub enables them to be used and reused. GitHub helps all team members stay on the same page and stay organized.

A git repository can be seen as a database containing all the information needed to retain and manage the revisions and history of a project. In git, repositories are used to retain a complete copy of the entire project throughout its lifetime.

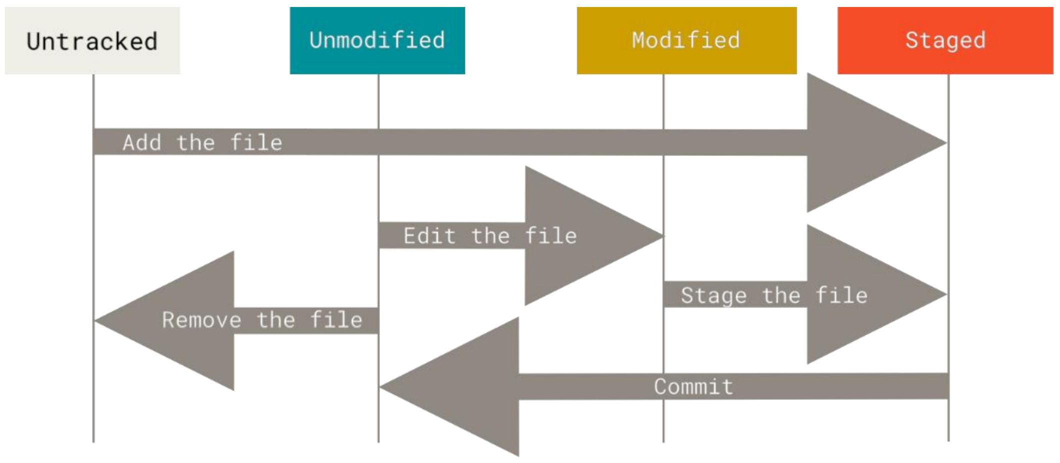
Git maintains a set of configuration values within each repository such as the repository user's name and email address. Unlike the file data or other repository metadata, configuration settings are not propagated from one repository to another during a clone, or fork, or any other duplication operation. Instead of this, git manages and stores configuration settings on a per-site, per-user, and per-repository basis.

Recording Changes to the Repository –

Each file in a working directory can be in one of two states: tracked or untracked.
Tracked files are files that were in the last snapshot, as well as any newly staged files; they can be unmodified, modified, or staged. In short, tracked files are files that Git knows about.

Untracked files are everything else — any files in your working directory that were not in your last snapshot and are not in your staging area. When you first clone a repository, all of the files will be tracked and unmodified because Git just checked them out and you haven't edited anything.

As you edit files, Git sees them as modified, because you've changed them since your last commit. As you work, you selectively stage these modified files and then commit all those staged changes, and the cycle repeats.



Structure of a Git Repository

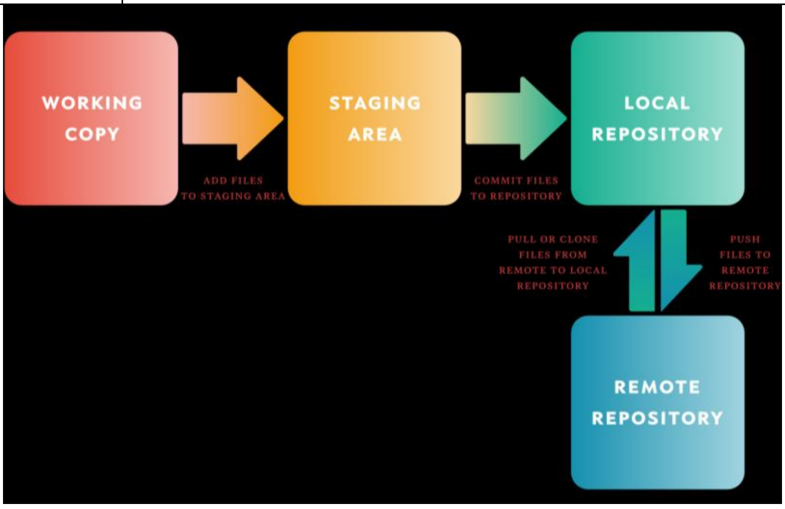
A Git repository contains several key components:

1. Working Directory –
    a. The local directory where you edit and manage files.
    b. Reflects the current state of your project.
    c. Example: Your local folder containing project files (index.html, main.js, etc.).
2. Staging Area (Index)
    a. A temporary area where changes are prepared (staged) before committing.
    b. Tracks which changes will be included in the next commit.
    c. Example: If you modify main.js, you use git add main.js to stage it.
3. Local Repository
    a. A hidden directory (.git) within your project folder that stores all Git-related data, including:
        i. Commit history.
        ii. Branches.
        iii. Tags.
        iv. Configuration settings.
    b. Example: The ".git" folder contains objects, references, and logs.

4. Remote Repository
    a. A version of the repository hosted on a remote server (e.g., GitHub, GitLab).
    b. Enables collaboration by syncing changes between local and remote repositories.
    c. Example: https://github.com/username/project.git.

Files and Directories Inside ".git"

#	Files and Directories	Details
1	objects/	Stores all content, including commits, trees, and blobs (files).
2	refs/	Contains references to branches and tags.
3	HEAD	Points to the current branch.
4	config	Stores repository configuration settings.
5	logs/	Tracks activities such as commits, checkouts, and merges.


WORKING COPY
ADD FILES TO STAGING AREA
STAGING AREA
COMMIT FILES TO REPOSITORY
LOCAL REPOSITORY
PULL OR CLONE FILES FROM REMOTE TO LOCAL REPOSITORY
PUSH FILES TO REMOTE REPOSITORY
REMOTE REPOSITORY

Inside a git repository, there are two data structures –

1. **the object store and**
2. **the index**.

All of this repository data is stored at the root of your working directory inside a hidden folder named ".git". You can read more about what's inside your ".git" folder.

Git Object Types

Object store lies at the heart of the git's data storage mechanism. It contains your original data files, all the log messages, author information, and other information required to rebuild any version or branch of the project.

Git places the following 4 types of objects in its object store which form the foundation of git's higher-level data structures:

#	Objects	Details
1	blobs	Represents each version of a file and is refer to a variable or file that may contain any data and whose underlying structure is ignored by the application. It contains the data of a file but no metadata or even the file's name.
2	trees	It represents a single level of directory data. It saves blob IDs, pathnames, and some metadata for all files in a directory. It may also recursively reference other

		(sub)tree objects, allowing it to construct a whole hierarchy of files and subdirectories.
3	commits	Each change made into the repository is represented by a commit object, which contains metadata such as the author, commit date, and log message. Each commit links to a tree object that records the state of the repository at the moment the commit was executed in a single full snapshot. The initial commit, also known as the root commit, has no parents and the following most of the commits have single parents.
4	tags	A tag object gives a given object, generally a commit, an arbitrary but presumably human-readable name such as Ver-1.0-Alpha.

Git index –
The index is a temporary and dynamic binary file that represents the entire directory structure of the repository at a specific moment. It essentially captures a snapshot of the project's overall structure at a given point in time. This state might correspond to a commit and its associated tree from the project's history, or it could represent a future state that you are currently working toward.

Basic Git Commands

Here are some commonly used Git commands –

1. git init – Initializes a new Git repository in the current directory. This command creates a ".git" folder to track changes.
   Example:
   ```
   mkdir my_project
   cd my_project
   git init
   ```
   This creates a Git repository in the my_project folder.
2. git clone – Creates a local copy of an existing remote repository.
   Example:
   ```
   git clone https://github.com/username/repository.git
   ```
   This clones the remote repository to your local machine.
3. git add – Stages changes (new files, modifications, or deletions) for the next commit.
   Example:
   ```
   git add file.txt
   git add .        # Stages all changes in the current directory
   ```
4. git commit – Records the staged changes in the local repository.
   Example:
   ```
   git commit -m "Add new feature"
   ```
   This creates a new commit with the message "Add new feature."
5. git push – Uploads local commits to a remote repository.
   Example:
   ```
   git push origin main
   ```
   This pushes the changes to the main branch of the remote repository.
6. git pull – Fetches and merges changes from a remote repository into the local repository.
   Example:
   ```
   git pull origin main
   ```
   This updates your local branch with changes from the remote main branch.

GitHub Operations Using Git

1. Forking a Repository – Forking creates a copy of someone else's repository under your GitHub account. This is useful for contributing to open-source projects.
   Steps:
   - Navigate to the repository on GitHub.
   - Click the Fork button.
   - The forked repository appears in your GitHub account.
2. Creating a Pull Request – A pull request is a way to propose changes to a repository. After making changes in your fork, you can submit a pull request to the original repository.
   Steps:
   - Push your changes to your forked repository.
   - Go to the original repository on GitHub.
   - Click Pull Requests > New Pull Request.
   - Select your fork and branch, then create the pull request.
3. Merging Branches – Combines changes from one branch into another, typically merging a feature branch into the main branch.
   Example:
   ```
   git checkout main
   git merge feature-branch
   ```
   This merges the changes from feature-branch into main.

Git for Version Control

1. Version Tracking – Git tracks every change made to the project, allowing you to view, revert, or branch from any version.
   Example:
   ```
   git log
   ```
   Displays the commit history.
2. Branching – Branches allow you to work on different features or fixes without affecting the main codebase.
   Example:
   ```
   git branch feature-branch
   git checkout feature-branch
   ```

3. Conflict Resolution – When merging branches, Git highlights conflicts if the same code is modified in multiple branches.
   Example: Edit the conflicting file, then:
   ```
   git add conflict-file.txt
   git commit -m "Resolve merge conflict"
   ```

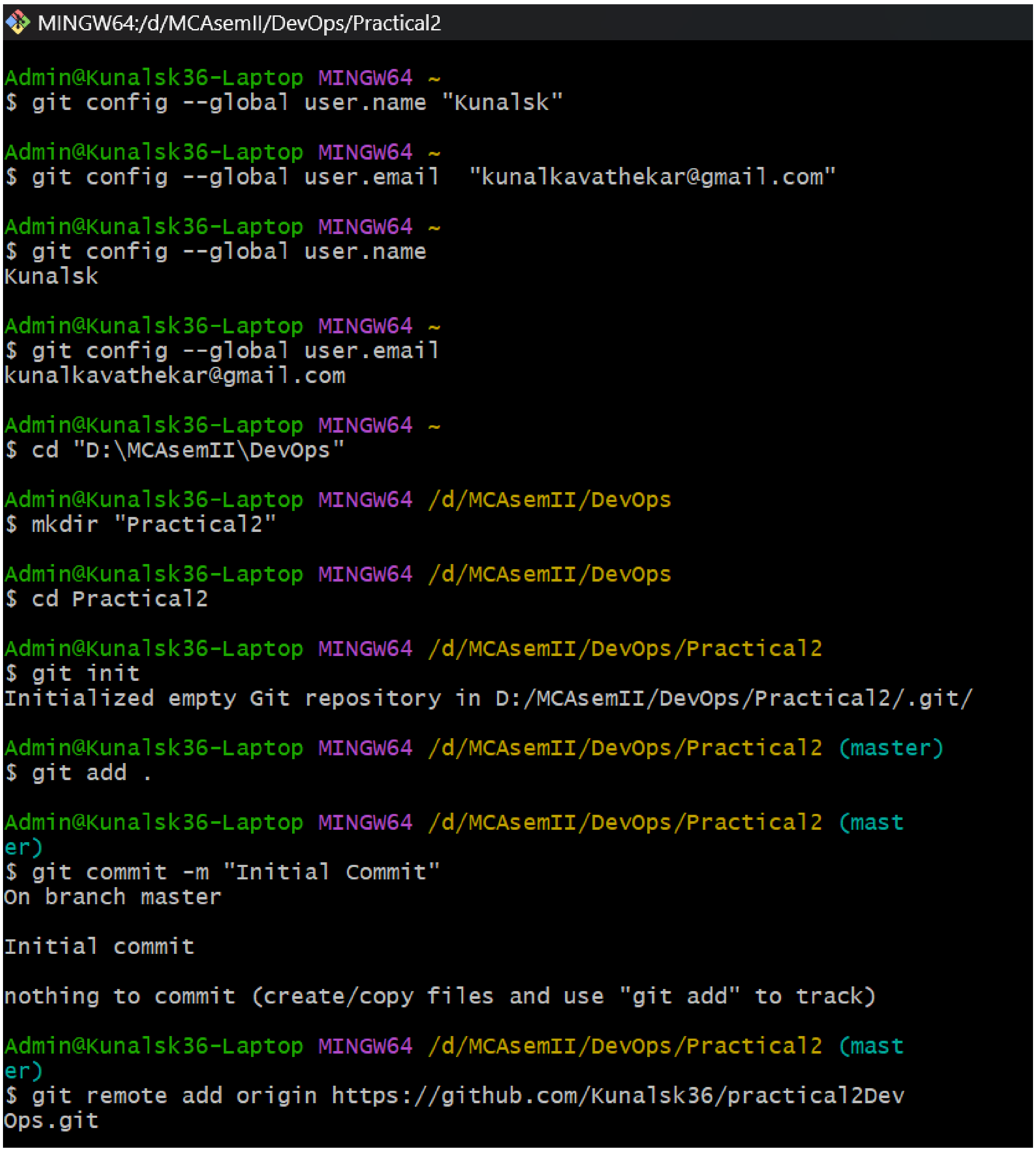
Exercise:

Consider your group is assigned to a project and this group needs to maintain the codebase using Git and GitHub by doing following tasks –

1. Repository Initialization

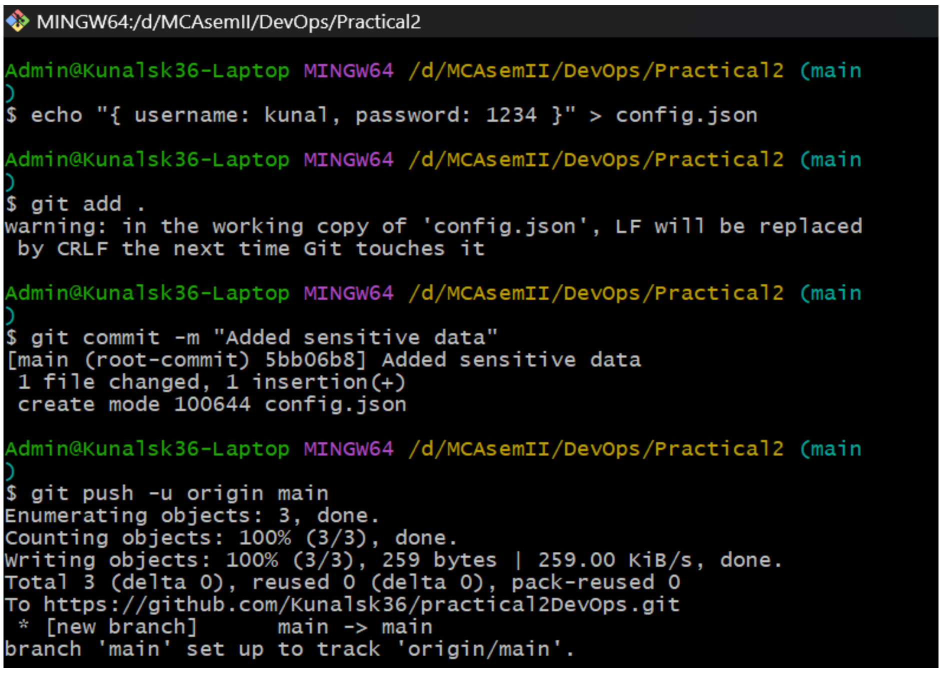
Problem: You're starting a new project. How do you set up Git to track changes in your project directory?

➔ By doing following steps we can track changes in our project directory:



MINGW64:/d/MCAsemII/DevOps/Practical2

```
Admin@Kunalsk36-Laptop MINGW64 ~
$ git config --global user.name "Kunalsk"

Admin@Kunalsk36-Laptop MINGW64 ~
$ git config --global user.email  "kunalkavathekar@gmail.com"

Admin@Kunalsk36-Laptop MINGW64 ~
$ git config --global user.name
Kunalsk

Admin@Kunalsk36-Laptop MINGW64 ~
$ git config --global user.email
kunalkavathekar@gmail.com

Admin@Kunalsk36-Laptop MINGW64 ~
$ cd "D:\MCAsemII\DevOps"

Admin@Kunalsk36-Laptop MINGW64 /d/MCAsemII/DevOps
$ mkdir "Practical2"

Admin@Kunalsk36-Laptop MINGW64 /d/MCAsemII/DevOps
$ cd Practical2

Admin@Kunalsk36-Laptop MINGW64 /d/MCAsemII/DevOps/Practical2
$ git init
Initialized empty Git repository in D:/MCAsemII/DevOps/Practical2/.git/

Admin@Kunalsk36-Laptop MINGW64 /d/MCAsemII/DevOps/Practical2 (master)
$ git add .

Admin@Kunalsk36-Laptop MINGW64 /d/MCAsemII/DevOps/Practical2 (mast
er)
$ git commit -m "Initial Commit"
On branch master

Initial commit

nothing to commit (create/copy files and use "git add" to track)

Admin@Kunalsk36-Laptop MINGW64 /d/MCAsemII/DevOps/Practical2 (mast
er)
$ git remote add origin https://github.com/Kunalsk36/practical2Dev
Ops.git
```

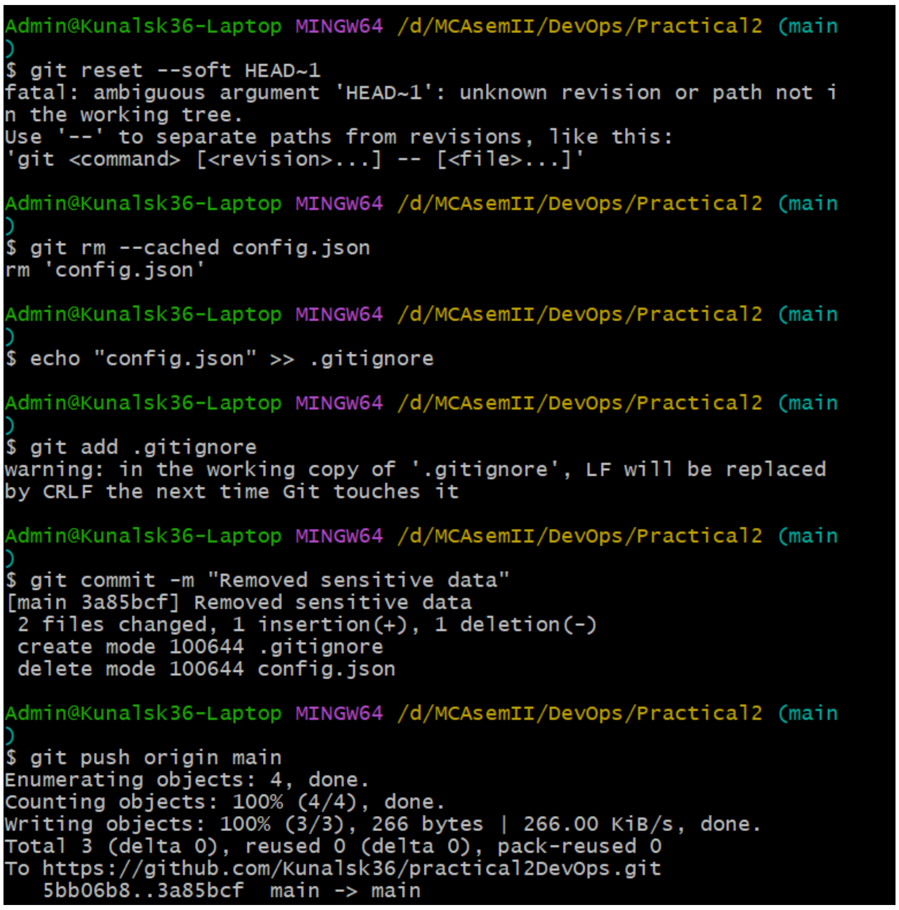


```
MINGW64:/d/MCAsemII/DevOps/Practical2

Admin@Kunalsk36-Laptop MINGW64 /d/MCAsemII/DevOps/Practical2 (main
)
$ echo "{ username: kunal, password: 1234 }" > config.json

Admin@Kunalsk36-Laptop MINGW64 /d/MCAsemII/DevOps/Practical2 (main
)
$ git add .
warning: in the working copy of 'config.json', LF will be replaced
 by CRLF the next time Git touches it

Admin@Kunalsk36-Laptop MINGW64 /d/MCAsemII/DevOps/Practical2 (main
)
$ git commit -m "Added sensitive data"
[main (root-commit) 5bb06b8] Added sensitive data
 1 file changed, 1 insertion(+)
 create mode 100644 config.json

Admin@Kunalsk36-Laptop MINGW64 /d/MCAsemII/DevOps/Practical2 (main
)
$ git push -u origin main
Enumerating objects: 3, done.
Counting objects: 100% (3/3), done.
Writing objects: 100% (3/3), 259 bytes | 259.00 KiB/s, done.
Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
To https://github.com/Kunalsk36/practical2DevOps.git
 * [new branch]      main -> main
branch 'main' set up to track 'origin/main'.
```

2. Undo a Mistaken Commit

Problem: You accidentally committed a file with sensitive data (e.g., `config.json`). How do you remove it from the commit history?

➔ We can remove a sensitive data file from the commit history by following steps:

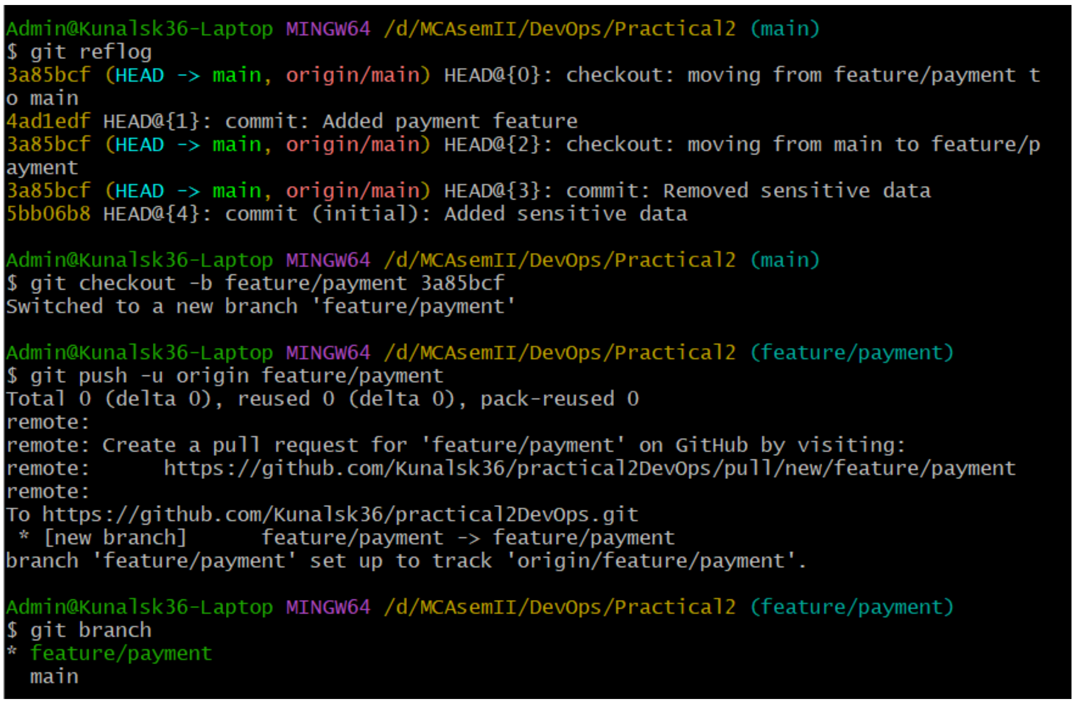


```
Admin@Kunalsk36-Laptop MINGW64 /d/MCAsemII/DevOps/Practical2 (main
)
$ git reset --soft HEAD~1
fatal: ambiguous argument 'HEAD~1': unknown revision or path not i
n the working tree.
Use '--' to separate paths from revisions, like this:
'git <command> [<revision>...] -- [<file>...]'

Admin@Kunalsk36-Laptop MINGW64 /d/MCAsemII/DevOps/Practical2 (main
)
$ git rm --cached config.json
rm 'config.json'

Admin@Kunalsk36-Laptop MINGW64 /d/MCAsemII/DevOps/Practical2 (main
)
$ echo "config.json" >> .gitignore

Admin@Kunalsk36-Laptop MINGW64 /d/MCAsemII/DevOps/Practical2 (main
)
$ git add .gitignore
warning: in the working copy of '.gitignore', LF will be replaced
by CRLF the next time Git touches it

Admin@Kunalsk36-Laptop MINGW64 /d/MCAsemII/DevOps/Practical2 (main
)
$ git commit -m "Removed sensitive data"
[main 3a85bcf] Removed sensitive data
 2 files changed, 1 insertion(+), 1 deletion(-)
 create mode 100644 .gitignore
 delete mode 100644 config.json

Admin@Kunalsk36-Laptop MINGW64 /d/MCAsemII/DevOps/Practical2 (main
)
$ git push origin main
Enumerating objects: 4, done.
Counting objects: 100% (4/4), done.
Writing objects: 100% (3/3), 266 bytes | 266.00 KiB/s, done.
Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
To https://github.com/Kunalsk36/practical2DevOps.git
   5bb06b8..3a85bcf  main -> main
```

3. Recover a Deleted Branch

Problem: A teammate deleted the `feature/payment` branch, but you need to restore it. How do you recover it?

➔ We can recover a deleted branch by following steps:



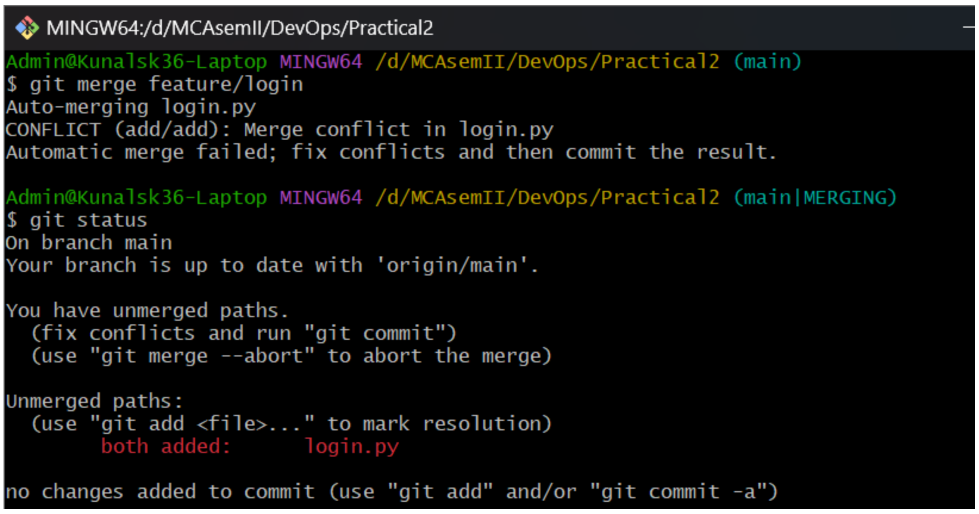
```
Admin@Kunalsk36-Laptop MINGW64 /d/MCAsemII/DevOps/Practical2 (main)
$ git reflog
3a85bcf (HEAD -> main, origin/main) HEAD@{0}: checkout: moving from feature/payment t
o main
4ad1edf HEAD@{1}: commit: Added payment feature
3a85bcf (HEAD -> main, origin/main) HEAD@{2}: checkout: moving from main to feature/p
ayment
3a85bcf (HEAD -> main, origin/main) HEAD@{3}: commit: Removed sensitive data
5bb06b8 HEAD@{4}: commit (initial): Added sensitive data

Admin@Kunalsk36-Laptop MINGW64 /d/MCAsemII/DevOps/Practical2 (main)
$ git checkout -b feature/payment 3a85bcf
Switched to a new branch 'feature/payment'

Admin@Kunalsk36-Laptop MINGW64 /d/MCAsemII/DevOps/Practical2 (feature/payment)
$ git push -u origin feature/payment
Total 0 (delta 0), reused 0 (delta 0), pack-reused 0
remote:
remote: Create a pull request for 'feature/payment' on GitHub by visiting:
remote:      https://github.com/Kunalsk36/practical2DevOps/pull/new/feature/payment
remote:
To https://github.com/Kunalsk36/practical2DevOps.git
 * [new branch]      feature/payment -> feature/payment
branch 'feature/payment' set up to track 'origin/feature/payment'.

Admin@Kunalsk36-Laptop MINGW64 /d/MCAsemII/DevOps/Practical2 (feature/payment)
$ git branch
* feature/payment
  main
```

4. Fix a Broken Merge

Problem: Merging `feature/login` into `main` caused conflicts in `login.py`. How do you resolve them?

➔ We can resolve the merging conflicts of 'login.py' by following steps:

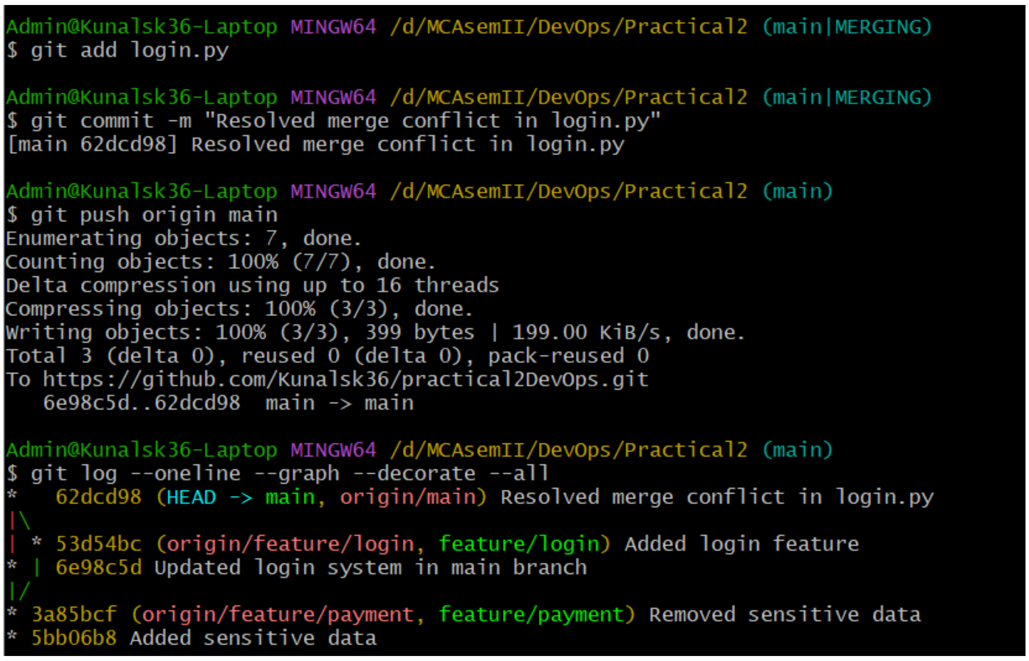
Step 1: Identify the merge conflicts, check which files have conflicts:



```
MINGW64:/d/MCAsemII/DevOps/Practical2
Admin@Kunalsk36-Laptop MINGW64 /d/MCAsemII/DevOps/Practical2 (main)
$ git merge feature/login
Auto-merging login.py
CONFLICT (add/add): Merge conflict in login.py
Automatic merge failed; fix conflicts and then commit the result.

Admin@Kunalsk36-Laptop MINGW64 /d/MCAsemII/DevOps/Practical2 (main|MERGING)
$ git status
On branch main
Your branch is up to date with 'origin/main'.

You have unmerged paths.
  (fix conflicts and run "git commit")
  (use "git merge --abort" to abort the merge)

Unmerged paths:
  (use "git add <file>..." to mark resolution)
        both added:      login.py

no changes added to commit (use "git add" and/or "git commit -a")
```

Step 2: Open the Conflicted File in a text editor then manually resolved the conflict, save the file and close it.

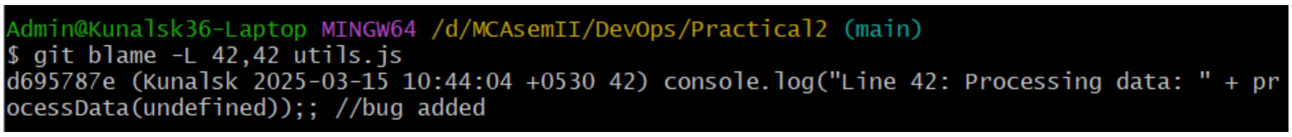
Step 3: Complete the Merge and verify it:



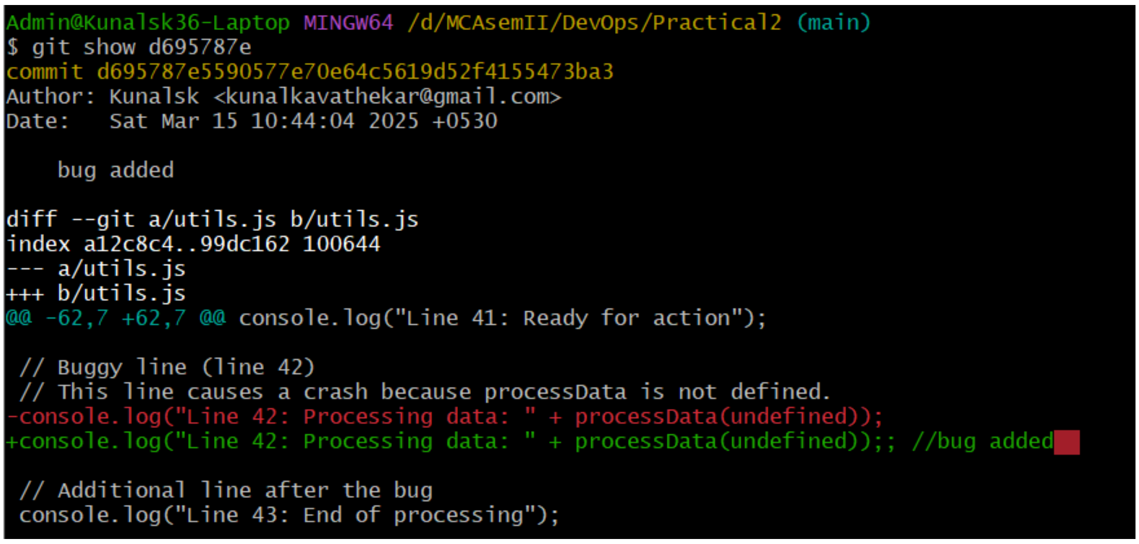
```
Admin@Kunalsk36-Laptop MINGW64 /d/MCAsemII/DevOps/Practical2 (main|MERGING)
$ git add login.py

Admin@Kunalsk36-Laptop MINGW64 /d/MCAsemII/DevOps/Practical2 (main|MERGING)
$ git commit -m "Resolved merge conflict in login.py"
[main 62dcd98] Resolved merge conflict in login.py

Admin@Kunalsk36-Laptop MINGW64 /d/MCAsemII/DevOps/Practical2 (main)
$ git push origin main
Enumerating objects: 7, done.
Counting objects: 100% (7/7), done.
Delta compression using up to 16 threads
Compressing objects: 100% (3/3), done.
Writing objects: 100% (3/3), 399 bytes | 199.00 KiB/s, done.
Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
To https://github.com/Kunalsk36/practical2DevOps.git
   6e98c5d..62dcd98  main -> main

Admin@Kunalsk36-Laptop MINGW64 /d/MCAsemII/DevOps/Practical2 (main)
$ git log --oneline --graph --decorate --all
*   62dcd98 (HEAD -> main, origin/main) Resolved merge conflict in login.py
|\
| * 53d54bc (origin/feature/login, feature/login) Added login feature
* | 6e98c5d Updated login system in main branch
|/
* 3a85bcf (origin/feature/payment, feature/payment) Removed sensitive data
* 5bb06b8 Added sensitive data
```

5. Track Down a Bug's Origin

Problem: Line 42 in `utils.js` is causing a crash. Who wrote that line, and in which commit?



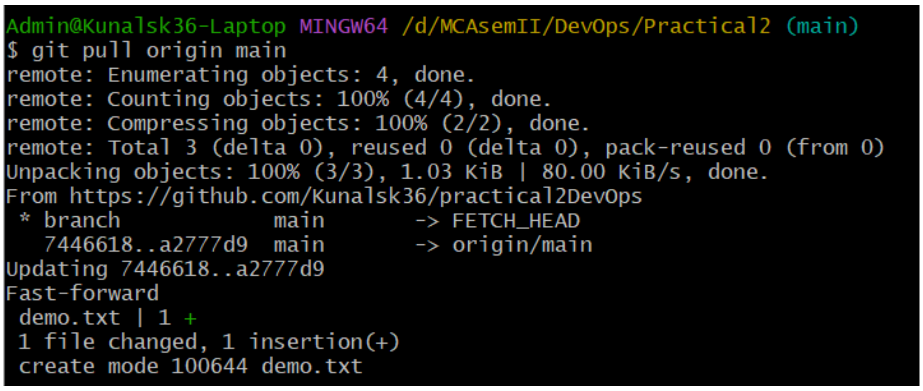
```
Admin@Kunalsk36-Laptop MINGW64 /d/MCAsemII/DevOps/Practical2 (main)
$ git blame -L 42,42 utils.js
d695787e (Kunalsk 2025-03-15 10:44:04 +0530 42) console.log("Line 42: Processing data: " + pr
ocessData(undefined));; //bug added
```



```
Admin@Kunalsk36-Laptop MINGW64 /d/MCAsemII/DevOps/Practical2 (main)
$ git show d695787e
commit d695787e5590577e70e64c5619d52f4155473ba3
Author: Kunalsk <kunalkavathekar@gmail.com>
Date:   Sat Mar 15 10:44:04 2025 +0530

    bug added

diff --git a/utils.js b/utils.js
index a12c8c4..99dc162 100644
--- a/utils.js
+++ b/utils.js
@@ -62,7 +62,7 @@ console.log("Line 41: Ready for action");

 // Buggy line (line 42)
 // This line causes a crash because processData is not defined.
-console.log("Line 42: Processing data: " + processData(undefined));
+console.log("Line 42: Processing data: " + processData(undefined));; //bug added

 // Additional line after the bug
 console.log("Line 43: End of processing");
```

6. Sync with a Remote Repository

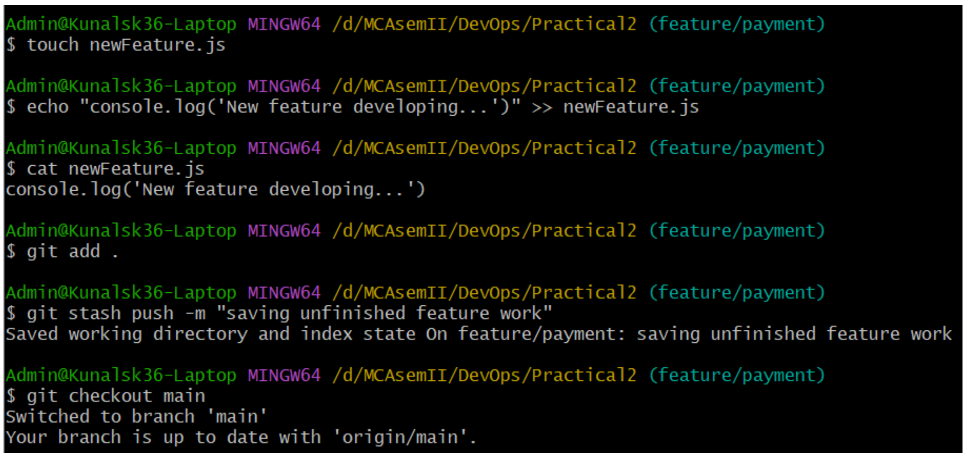
Problem: Your teammate pushed changes to GitHub, but your local repo is outdated. How do you update it?



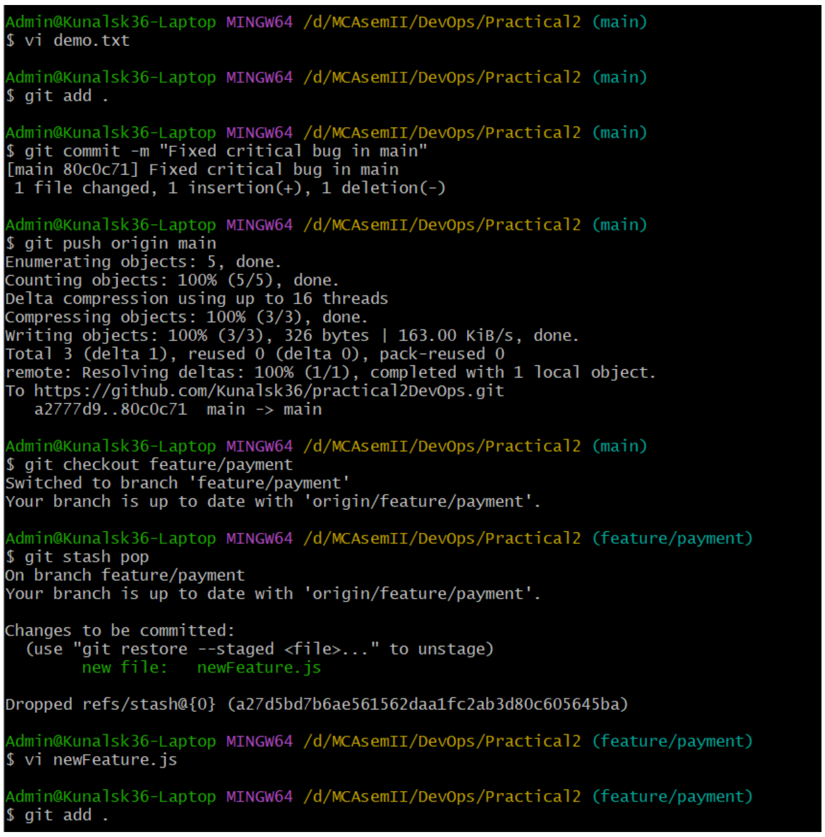
```
Admin@Kunalsk36-Laptop MINGW64 /d/MCAsemII/DevOps/Practical2 (main)
$ git pull origin main
remote: Enumerating objects: 4, done.
remote: Counting objects: 100% (4/4), done.
remote: Compressing objects: 100% (2/2), done.
remote: Total 3 (delta 0), reused 0 (delta 0), pack-reused 0 (from 0)
Unpacking objects: 100% (3/3), 1.03 KiB | 80.00 KiB/s, done.
From https://github.com/Kunalsk36/practical2DevOps
 * branch            main       -> FETCH_HEAD
   7446618..a2777d9  main       -> origin/main
Updating 7446618..a2777d9
Fast-forward
 demo.txt | 1 +
 1 file changed, 1 insertion(+)
 create mode 100644 demo.txt
```

7. Switch Context Mid-Task

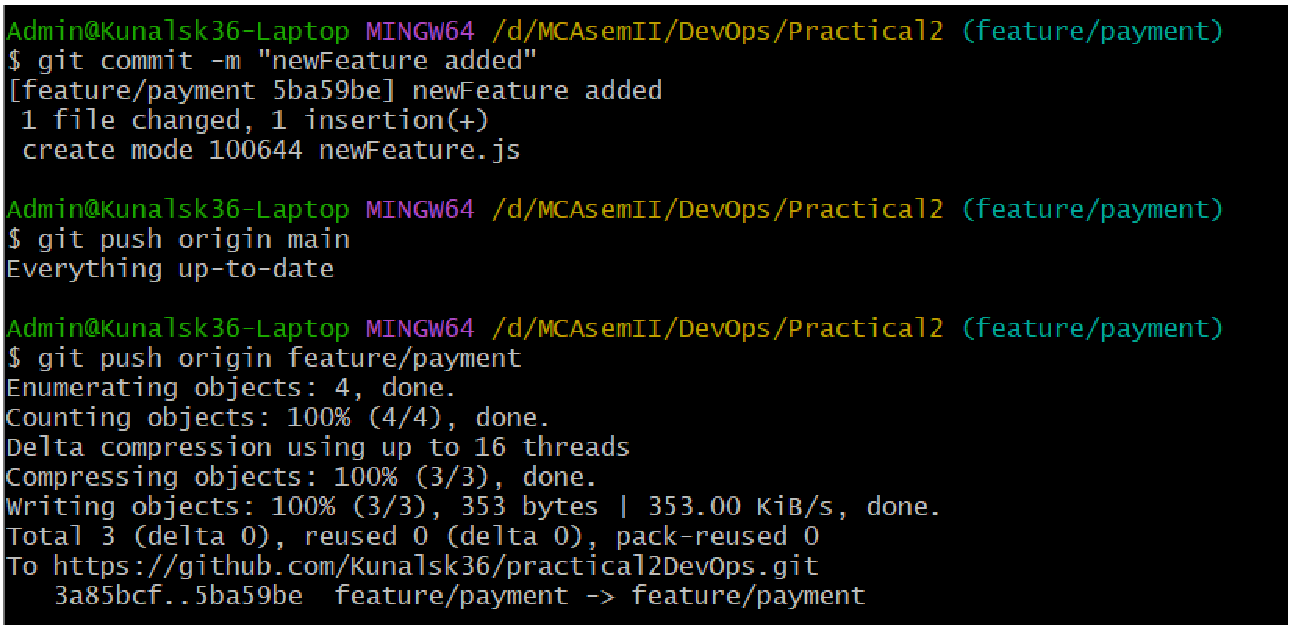
Problem: You're halfway through a feature when asked to fix a critical bug in `main`. How do you save your work?



```
Admin@Kunalsk36-Laptop MINGW64 /d/MCAsemII/DevOps/Practical2 (feature/payment)
$ touch newFeature.js

Admin@Kunalsk36-Laptop MINGW64 /d/MCAsemII/DevOps/Practical2 (feature/payment)
$ echo "console.log('New feature developing...')" >> newFeature.js

Admin@Kunalsk36-Laptop MINGW64 /d/MCAsemII/DevOps/Practical2 (feature/payment)
$ cat newFeature.js
console.log('New feature developing...')

Admin@Kunalsk36-Laptop MINGW64 /d/MCAsemII/DevOps/Practical2 (feature/payment)
$ git add .

Admin@Kunalsk36-Laptop MINGW64 /d/MCAsemII/DevOps/Practical2 (feature/payment)
$ git stash push -m "saving unfinished feature work"
Saved working directory and index state On feature/payment: saving unfinished feature work

Admin@Kunalsk36-Laptop MINGW64 /d/MCAsemII/DevOps/Practical2 (feature/payment)
$ git checkout main
Switched to branch 'main'
Your branch is up to date with 'origin/main'.
```

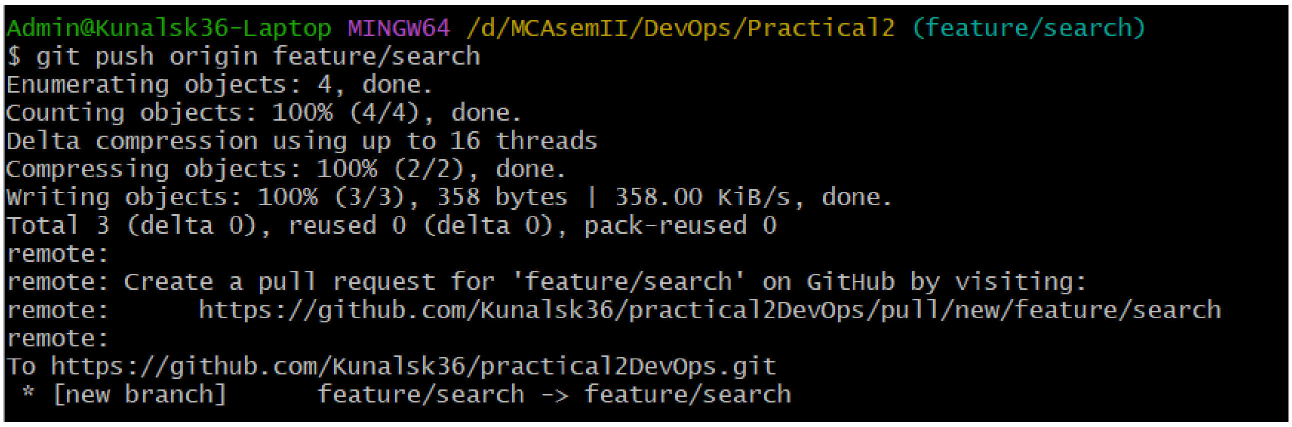


```
Admin@Kunalsk36-Laptop MINGW64 /d/MCAsemII/DevOps/Practical2 (main)
$ vi demo.txt

Admin@Kunalsk36-Laptop MINGW64 /d/MCAsemII/DevOps/Practical2 (main)
$ git add .

Admin@Kunalsk36-Laptop MINGW64 /d/MCAsemII/DevOps/Practical2 (main)
$ git commit -m "Fixed critical bug in main"
[main 80c0c71] Fixed critical bug in main
 1 file changed, 1 insertion(+), 1 deletion(-)

Admin@Kunalsk36-Laptop MINGW64 /d/MCAsemII/DevOps/Practical2 (main)
$ git push origin main
Enumerating objects: 5, done.
Counting objects: 100% (5/5), done.
Delta compression using up to 16 threads
Compressing objects: 100% (3/3), done.
Writing objects: 100% (3/3), 326 bytes | 163.00 KiB/s, done.
Total 3 (delta 1), reused 0 (delta 0), pack-reused 0
remote: Resolving deltas: 100% (1/1), completed with 1 local object.
To https://github.com/Kunalsk36/practical2DevOps.git
   a2777d9..80c0c71  main -> main

Admin@Kunalsk36-Laptop MINGW64 /d/MCAsemII/DevOps/Practical2 (main)
$ git checkout feature/payment
Switched to branch 'feature/payment'
Your branch is up to date with 'origin/feature/payment'.

Admin@Kunalsk36-Laptop MINGW64 /d/MCAsemII/DevOps/Practical2 (feature/payment)
$ git stash pop
On branch feature/payment
Your branch is up to date with 'origin/feature/payment'.

Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        new file:   newFeature.js

Dropped refs/stash@{0} (a27d5bd7b6ae561562daa1fc2ab3d80c605645ba)

Admin@Kunalsk36-Laptop MINGW64 /d/MCAsemII/DevOps/Practical2 (feature/payment)
$ vi newFeature.js

Admin@Kunalsk36-Laptop MINGW64 /d/MCAsemII/DevOps/Practical2 (feature/payment)
$ git add .
```


```
Admin@Kunalsk36-Laptop MINGW64 /d/MCAsemII/DevOps/Practical2 (feature/payment)
$ git commit -m "newFeature added"
[feature/payment 5ba59be] newFeature added
 1 file changed, 1 insertion(+)
 create mode 100644 newFeature.js

Admin@Kunalsk36-Laptop MINGW64 /d/MCAsemII/DevOps/Practical2 (feature/payment)
$ git push origin main
Everything up-to-date

Admin@Kunalsk36-Laptop MINGW64 /d/MCAsemII/DevOps/Practical2 (feature/payment)
$ git push origin feature/payment
Enumerating objects: 4, done.
Counting objects: 100% (4/4), done.
Delta compression using up to 16 threads
Compressing objects: 100% (3/3), done.
Writing objects: 100% (3/3), 353 bytes | 353.00 KiB/s, done.
Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
To https://github.com/Kunalsk36/practical2DevOps.git
   3a85bcf..5ba59be  feature/payment -> feature/payment
```

8. Publish a Local Branch

Problem: You created `feature/search` locally. How do you share it with your team on GitHub?


```
Admin@Kunalsk36-Laptop MINGW64 /d/MCAsemII/DevOps/Practical2 (feature/search)
$ git push origin feature/search
Enumerating objects: 4, done.
Counting objects: 100% (4/4), done.
Delta compression using up to 16 threads
Compressing objects: 100% (2/2), done.
Writing objects: 100% (3/3), 358 bytes | 358.00 KiB/s, done.
Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
remote:
remote: Create a pull request for 'feature/search' on GitHub by visiting:
remote:      https://github.com/Kunalsk36/practical2DevOps/pull/new/feature/search
remote:
To https://github.com/Kunalsk36/practical2DevOps.git
 * [new branch]      feature/search -> feature/search
```

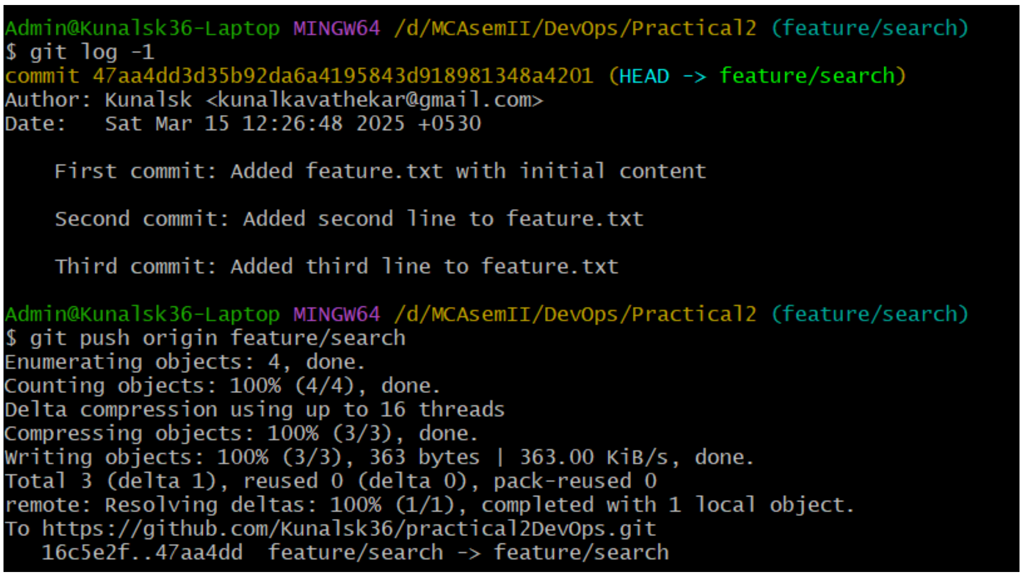
9. Rewrite Commit History

Problem: You made three messy commits for a feature. How do you combine them into one clean commit?

➔ Step 1: Run the following command to rebase the last three commits:
git rebase -i HEAD~3

Step 2: Mark Commits for Squashing
Change pick to squash (or s) for the second and third commits then Save and close.

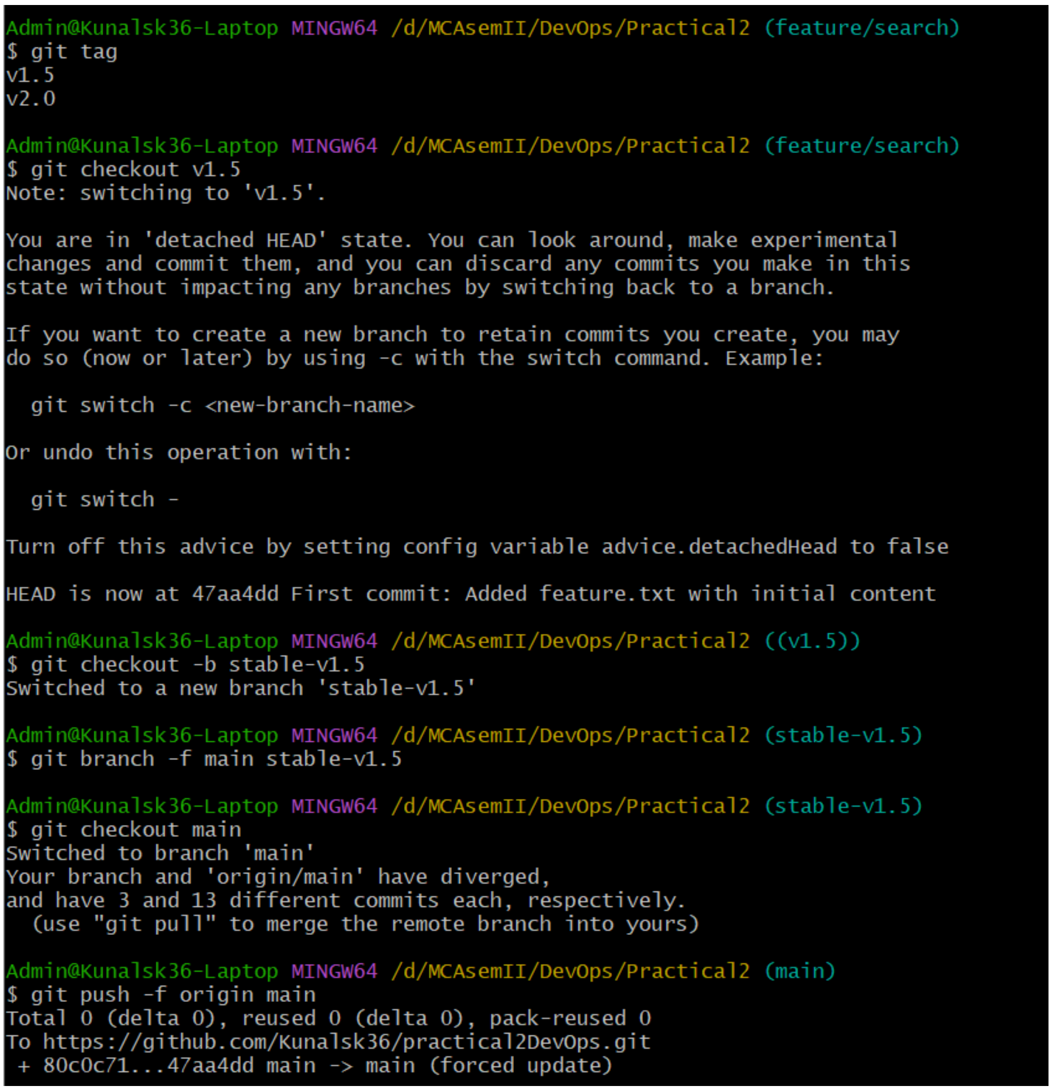
Step 3: Check commit and push commits to github



```
Admin@Kunalsk36-Laptop MINGW64 /d/MCAsemII/DevOps/Practical2 (feature/search)
$ git log -1
commit 47aa4dd3d35b92da6a4195843d918981348a4201 (HEAD -> feature/search)
Author: Kunalsk <kunalkavathekar@gmail.com>
Date:   Sat Mar 15 12:26:48 2025 +0530

    First commit: Added feature.txt with initial content

    Second commit: Added second line to feature.txt

    Third commit: Added third line to feature.txt

Admin@Kunalsk36-Laptop MINGW64 /d/MCAsemII/DevOps/Practical2 (feature/search)
$ git push origin feature/search
Enumerating objects: 4, done.
Counting objects: 100% (4/4), done.
Delta compression using up to 16 threads
Compressing objects: 100% (3/3), done.
Writing objects: 100% (3/3), 363 bytes | 363.00 KiB/s, done.
Total 3 (delta 1), reused 0 (delta 0), pack-reused 0
remote: Resolving deltas: 100% (1/1), completed with 1 local object.
To https://github.com/Kunalsk36/practical2DevOps.git
   16c5e2f..47aa4dd  feature/search -> feature/search
```

10. Roll Back to a Stable Version

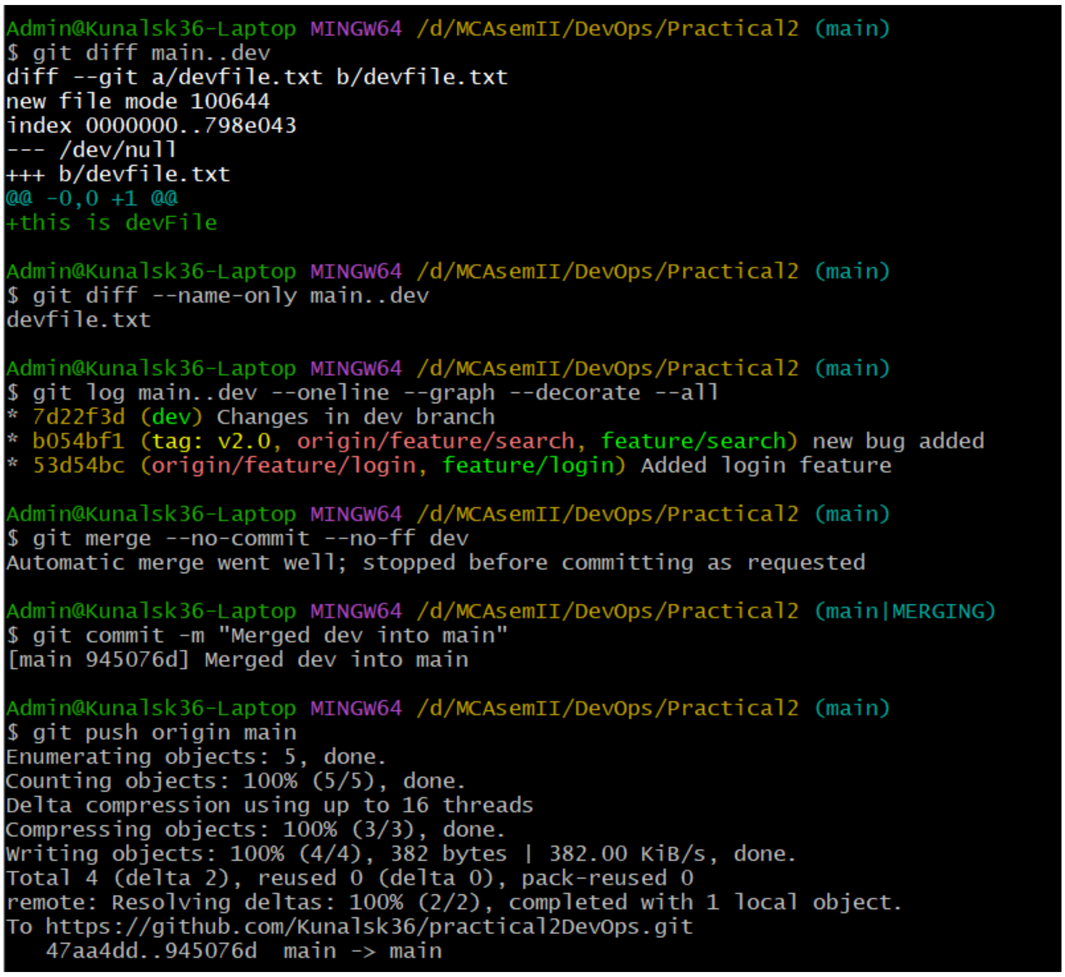
Problem: The latest release (`v2.0`) has a critical bug. How do you revert to `v1.5`?



```
Admin@Kunalsk36-Laptop MINGW64 /d/MCAsemII/DevOps/Practical2 (feature/search)
$ git tag
v1.5
v2.0

Admin@Kunalsk36-Laptop MINGW64 /d/MCAsemII/DevOps/Practical2 (feature/search)
$ git checkout v1.5
Note: switching to 'v1.5'.

You are in 'detached HEAD' state. You can look around, make experimental
changes and commit them, and you can discard any commits you make in this
state without impacting any branches by switching back to a branch.

If you want to create a new branch to retain commits you create, you may
do so (now or later) by using -c with the switch command. Example:

  git switch -c <new-branch-name>

Or undo this operation with:

  git switch -

Turn off this advice by setting config variable advice.detachedHead to false

HEAD is now at 47aa4dd First commit: Added feature.txt with initial content

Admin@Kunalsk36-Laptop MINGW64 /d/MCAsemII/DevOps/Practical2 ((v1.5))
$ git checkout -b stable-v1.5
Switched to a new branch 'stable-v1.5'

Admin@Kunalsk36-Laptop MINGW64 /d/MCAsemII/DevOps/Practical2 (stable-v1.5)
$ git branch -f main stable-v1.5

Admin@Kunalsk36-Laptop MINGW64 /d/MCAsemII/DevOps/Practical2 (stable-v1.5)
$ git checkout main
Switched to branch 'main'
Your branch and 'origin/main' have diverged,
and have 3 and 13 different commits each, respectively.
  (use "git pull" to merge the remote branch into yours)

Admin@Kunalsk36-Laptop MINGW64 /d/MCAsemII/DevOps/Practical2 (main)
$ git push -f origin main
Total 0 (delta 0), reused 0 (delta 0), pack-reused 0
To https://github.com/Kunalsk36/practical2DevOps.git
 + 80c0c71...47aa4dd main -> main (forced update)
```

11. Compare Branches

Problem: You need to see what's changed between `main` and `dev` before merging. How?

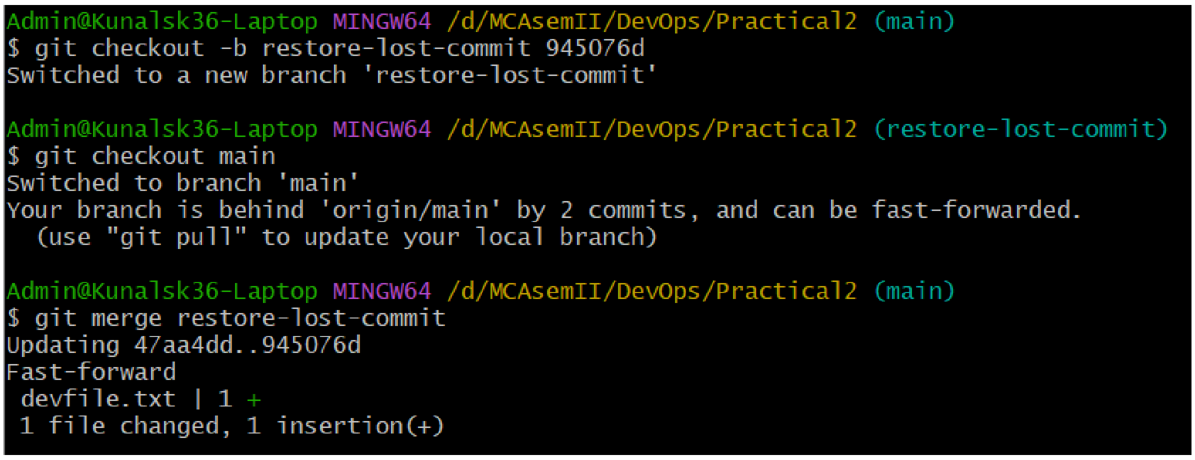


```
Admin@Kunalsk36-Laptop MINGW64 /d/MCAsemII/DevOps/Practical2 (main)
$ git diff main..dev
diff --git a/devfile.txt b/devfile.txt
new file mode 100644
index 0000000..798e043
--- /dev/null
+++ b/devfile.txt
@@ -0,0 +1 @@
+this is devFile

Admin@Kunalsk36-Laptop MINGW64 /d/MCAsemII/DevOps/Practical2 (main)
$ git diff --name-only main..dev
devfile.txt

Admin@Kunalsk36-Laptop MINGW64 /d/MCAsemII/DevOps/Practical2 (main)
$ git log main..dev --oneline --graph --decorate --all
* 7d22f3d (dev) Changes in dev branch
* b054bf1 (tag: v2.0, origin/feature/search, feature/search) new bug added
* 53d54bc (origin/feature/login, feature/login) Added login feature

Admin@Kunalsk36-Laptop MINGW64 /d/MCAsemII/DevOps/Practical2 (main)
$ git merge --no-commit --no-ff dev
Automatic merge went well; stopped before committing as requested

Admin@Kunalsk36-Laptop MINGW64 /d/MCAsemII/DevOps/Practical2 (main|MERGING)
$ git commit -m "Merged dev into main"
[main 945076d] Merged dev into main

Admin@Kunalsk36-Laptop MINGW64 /d/MCAsemII/DevOps/Practical2 (main)
$ git push origin main
Enumerating objects: 5, done.
Counting objects: 100% (5/5), done.
Delta compression using up to 16 threads
Compressing objects: 100% (3/3), done.
Writing objects: 100% (4/4), 382 bytes | 382.00 KiB/s, done.
Total 4 (delta 2), reused 0 (delta 0), pack-reused 0
remote: Resolving deltas: 100% (2/2), completed with 1 local object.
To https://github.com/Kunalsk36/practical2DevOps.git
   47aa4dd..945076d  main -> main
```

12. Reapply a Lost Commit

Problem: A teammate's commit `abc123` was accidentally deleted. How do you restore it?

➔ Step 1: Check Git Reflog for the Lost Commit

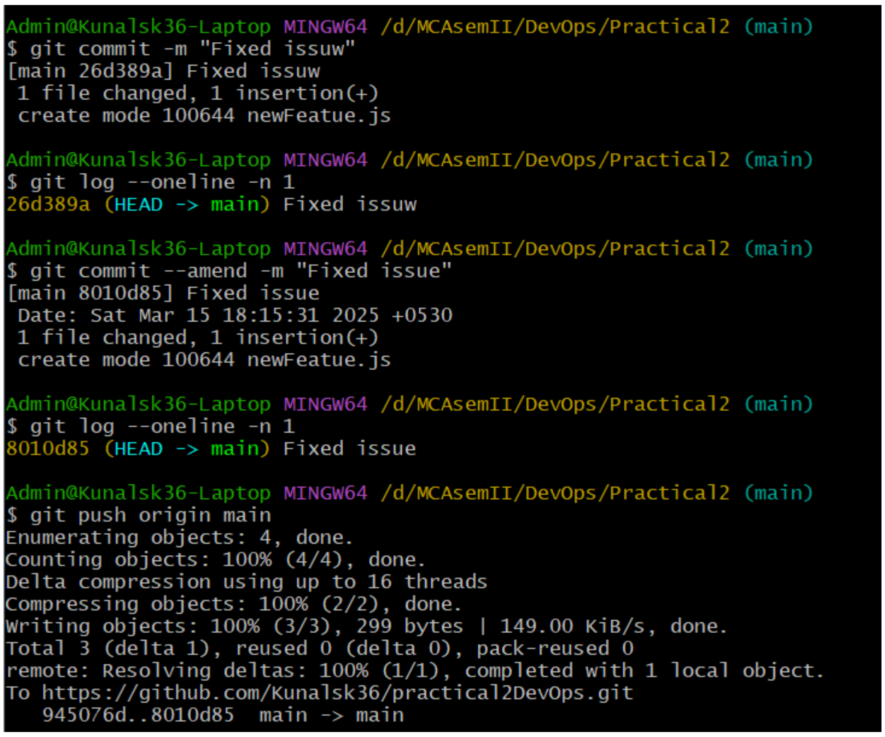


```
Admin@Kunalsk36-Laptop MINGW64 /d/MCAsemII/DevOps/Practical2 (main)
$ git checkout -b restore-lost-commit 945076d
Switched to a new branch 'restore-lost-commit'

Admin@Kunalsk36-Laptop MINGW64 /d/MCAsemII/DevOps/Practical2 (restore-lost-commit)
$ git checkout main
Switched to branch 'main'
Your branch is behind 'origin/main' by 2 commits, and can be fast-forwarded.
  (use "git pull" to update your local branch)

Admin@Kunalsk36-Laptop MINGW64 /d/MCAsemII/DevOps/Practical2 (main)
$ git merge restore-lost-commit
Updating 47aa4dd..945076d
Fast-forward
 devfile.txt | 1 +
 1 file changed, 1 insertion(+)
```

13. Fix a Commit Message

Problem: The last commit message says “Fixed issuw”. How do you correct the typo?

Commands Needed: `git commit --amend`.

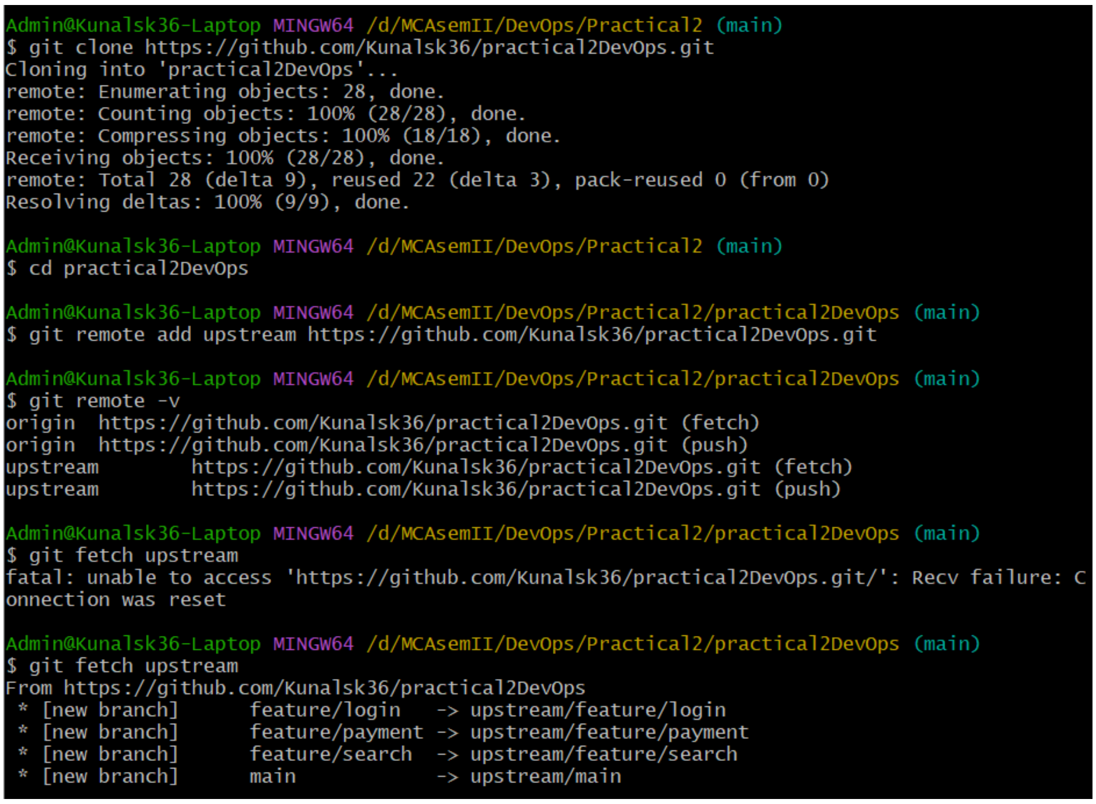


```
Admin@Kunalsk36-Laptop MINGW64 /d/MCAsemII/DevOps/Practical2 (main)
$ git commit -m "Fixed issuw"
[main 26d389a] Fixed issuw
 1 file changed, 1 insertion(+)
 create mode 100644 newFeatue.js

Admin@Kunalsk36-Laptop MINGW64 /d/MCAsemII/DevOps/Practical2 (main)
$ git log --oneline -n 1
26d389a (HEAD -> main) Fixed issuw

Admin@Kunalsk36-Laptop MINGW64 /d/MCAsemII/DevOps/Practical2 (main)
$ git commit --amend -m "Fixed issue"
[main 8010d85] Fixed issue
 Date: Sat Mar 15 18:15:31 2025 +0530
 1 file changed, 1 insertion(+)
 create mode 100644 newFeatue.js

Admin@Kunalsk36-Laptop MINGW64 /d/MCAsemII/DevOps/Practical2 (main)
$ git log --oneline -n 1
8010d85 (HEAD -> main) Fixed issue

Admin@Kunalsk36-Laptop MINGW64 /d/MCAsemII/DevOps/Practical2 (main)
$ git push origin main
Enumerating objects: 4, done.
Counting objects: 100% (4/4), done.
Delta compression using up to 16 threads
Compressing objects: 100% (2/2), done.
Writing objects: 100% (3/3), 299 bytes | 149.00 KiB/s, done.
Total 3 (delta 1), reused 0 (delta 0), pack-reused 0
remote: Resolving deltas: 100% (1/1), completed with 1 local object.
To https://github.com/Kunalsk36/practical2DevOps.git
   945076d..8010d85  main -> main
```

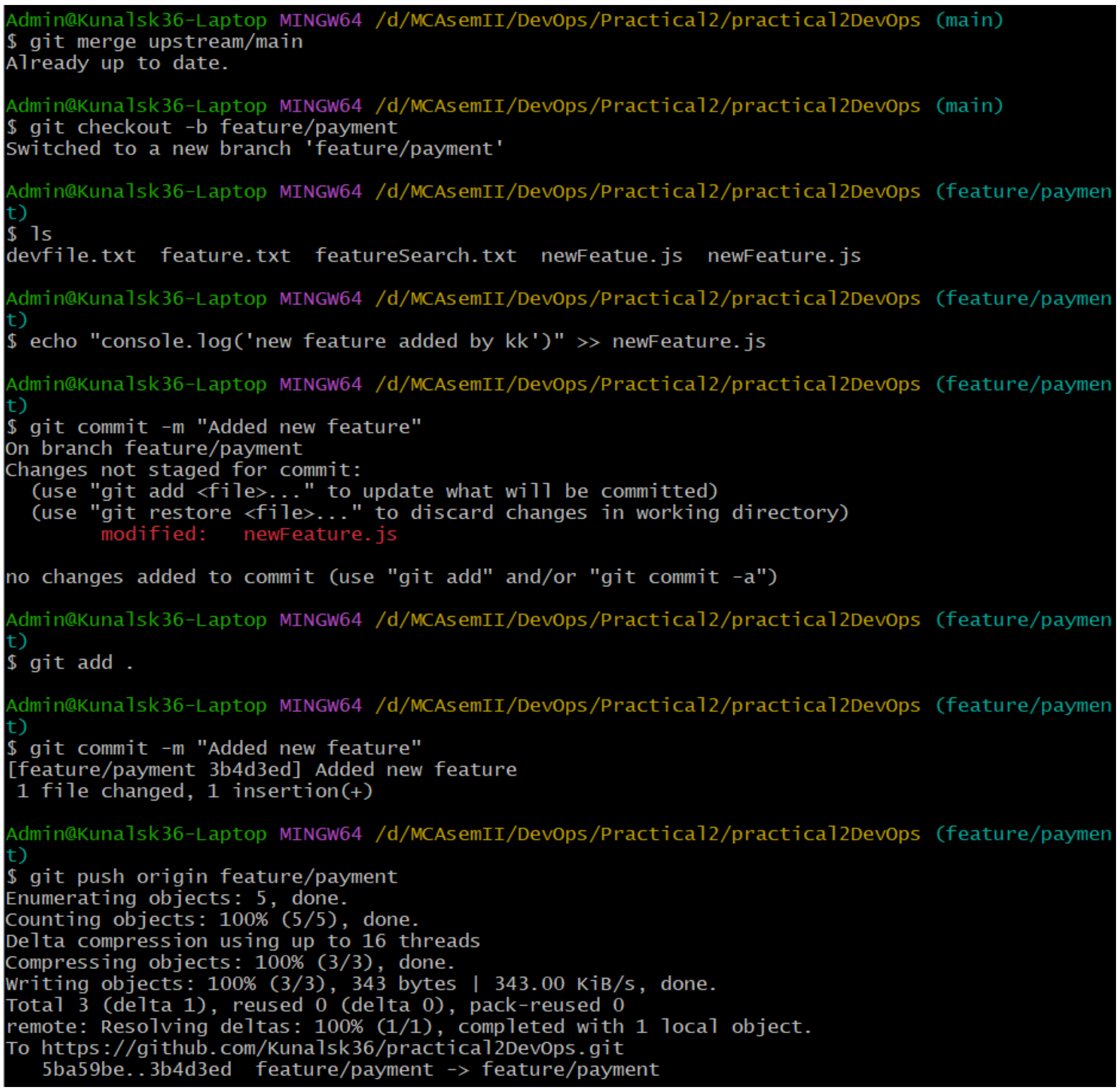
14. Collaborate via Fork

Problem: You want to contribute to an open-source project. How do you set up a fork?

Commands Needed: Fork on GitHub, `git clone`, `git remote add upstream`.



```
Admin@Kunalsk36-Laptop MINGW64 /d/MCAsemII/DevOps/Practical2 (main)
$ git clone https://github.com/Kunalsk36/practical2DevOps.git
Cloning into 'practical2DevOps'...
remote: Enumerating objects: 28, done.
remote: Counting objects: 100% (28/28), done.
remote: Compressing objects: 100% (18/18), done.
Receiving objects: 100% (28/28), done.
remote: Total 28 (delta 9), reused 22 (delta 3), pack-reused 0 (from 0)
Resolving deltas: 100% (9/9), done.

Admin@Kunalsk36-Laptop MINGW64 /d/MCAsemII/DevOps/Practical2 (main)
$ cd practical2DevOps

Admin@Kunalsk36-Laptop MINGW64 /d/MCAsemII/DevOps/Practical2/practical2DevOps (main)
$ git remote add upstream https://github.com/Kunalsk36/practical2DevOps.git

Admin@Kunalsk36-Laptop MINGW64 /d/MCAsemII/DevOps/Practical2/practical2DevOps (main)
$ git remote -v
origin  https://github.com/Kunalsk36/practical2DevOps.git (fetch)
origin  https://github.com/Kunalsk36/practical2DevOps.git (push)
upstream        https://github.com/Kunalsk36/practical2DevOps.git (fetch)
upstream        https://github.com/Kunalsk36/practical2DevOps.git (push)

Admin@Kunalsk36-Laptop MINGW64 /d/MCAsemII/DevOps/Practical2/practical2DevOps (main)
$ git fetch upstream
fatal: unable to access 'https://github.com/Kunalsk36/practical2DevOps.git/': Recv failure: C
onnection was reset

Admin@Kunalsk36-Laptop MINGW64 /d/MCAsemII/DevOps/Practical2/practical2DevOps (main)
$ git fetch upstream
From https://github.com/Kunalsk36/practical2DevOps
 * [new branch]      feature/login   -> upstream/feature/login
 * [new branch]      feature/payment -> upstream/feature/payment
 * [new branch]      feature/search  -> upstream/feature/search
 * [new branch]      main            -> upstream/main
```



```
Admin@Kunalsk36-Laptop MINGW64 /d/MCAsemII/DevOps/Practical2/practical2DevOps (main)
$ git merge upstream/main
Already up to date.

Admin@Kunalsk36-Laptop MINGW64 /d/MCAsemII/DevOps/Practical2/practical2DevOps (main)
$ git checkout -b feature/payment
Switched to a new branch 'feature/payment'

Admin@Kunalsk36-Laptop MINGW64 /d/MCAsemII/DevOps/Practical2/practical2DevOps (feature/paymen
t)
$ ls
devfile.txt  feature.txt  featureSearch.txt  newFeatue.js  newFeature.js

Admin@Kunalsk36-Laptop MINGW64 /d/MCAsemII/DevOps/Practical2/practical2DevOps (feature/paymen
t)
$ echo "console.log('new feature added by kk')" >> newFeature.js

Admin@Kunalsk36-Laptop MINGW64 /d/MCAsemII/DevOps/Practical2/practical2DevOps (feature/paymen
t)
$ git commit -m "Added new feature"
On branch feature/payment
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:   newFeature.js

no changes added to commit (use "git add" and/or "git commit -a")

Admin@Kunalsk36-Laptop MINGW64 /d/MCAsemII/DevOps/Practical2/practical2DevOps (feature/paymen
t)
$ git add .

Admin@Kunalsk36-Laptop MINGW64 /d/MCAsemII/DevOps/Practical2/practical2DevOps (feature/paymen
t)
$ git commit -m "Added new feature"
[feature/payment 3b4d3ed] Added new feature
 1 file changed, 1 insertion(+)

Admin@Kunalsk36-Laptop MINGW64 /d/MCAsemII/DevOps/Practical2/practical2DevOps (feature/paymen
t)
$ git push origin feature/payment
Enumerating objects: 5, done.
Counting objects: 100% (5/5), done.
Delta compression using up to 16 threads
Compressing objects: 100% (3/3), done.
Writing objects: 100% (3/3), 343 bytes | 343.00 KiB/s, done.
Total 3 (delta 1), reused 0 (delta 0), pack-reused 0
remote: Resolving deltas: 100% (1/1), completed with 1 local object.
To https://github.com/Kunalsk36/practical2DevOps.git
   5ba59be..3b4d3ed  feature/payment -> feature/payment
```

15. Rebase to Avoid Merge Commits

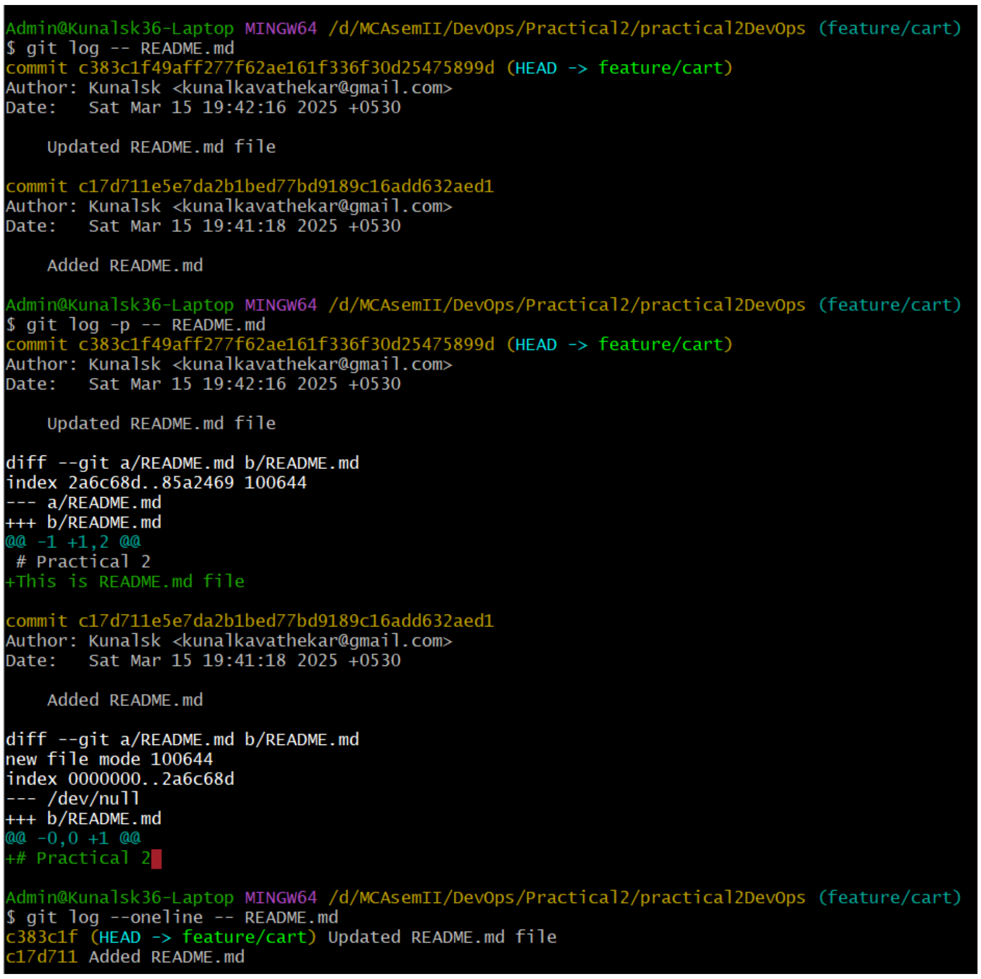
Problem: Your `feature/cart` branch is behind `main`. How do you update it without merge commits?

➔ Following commands needed to done:

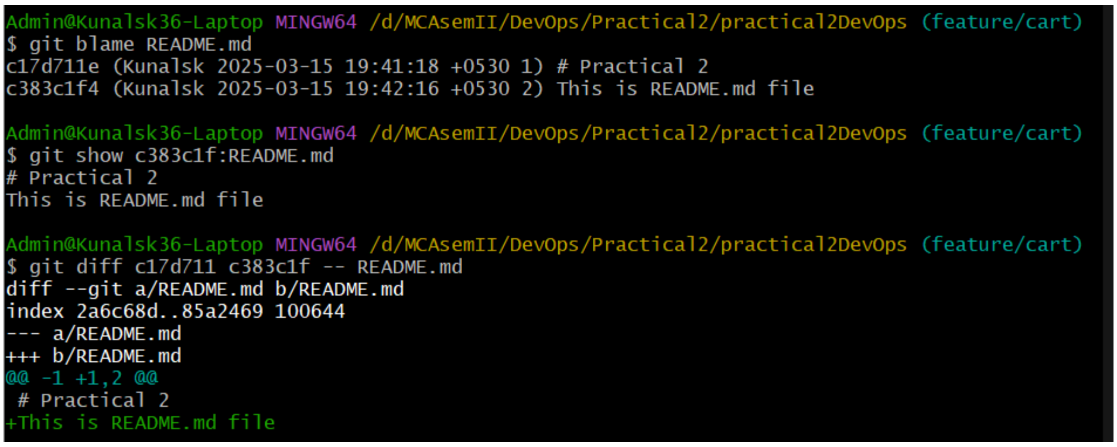
```
git checkout main
git pull origin main
git checkout feature/cart
git rebase main
git status  # Check for conflicts
git add <conflicted-file>
git rebase --continue  # Continue rebase if conflicts occur
git push origin feature/cart --force  # Only if already pushed before
```

16. Track File History

Problem: You need to see how `README.md` evolved over time. How do you view its history?



```
Admin@Kunalsk36-Laptop MINGW64 /d/MCAsemII/DevOps/Practical2/practical2DevOps (feature/cart)
$ git log -- README.md
commit c383c1f49aff277f62ae161f336f30d25475899d (HEAD -> feature/cart)
Author: Kunalsk <kunalkavathekar@gmail.com>
Date:   Sat Mar 15 19:42:16 2025 +0530

    Updated README.md file

commit c17d711e5e7da2b1bed77bd9189c16add632aed1
Author: Kunalsk <kunalkavathekar@gmail.com>
Date:   Sat Mar 15 19:41:18 2025 +0530

    Added README.md

Admin@Kunalsk36-Laptop MINGW64 /d/MCAsemII/DevOps/Practical2/practical2DevOps (feature/cart)
$ git log -p -- README.md
commit c383c1f49aff277f62ae161f336f30d25475899d (HEAD -> feature/cart)
Author: Kunalsk <kunalkavathekar@gmail.com>
Date:   Sat Mar 15 19:42:16 2025 +0530

    Updated README.md file

diff --git a/README.md b/README.md
index 2a6c68d..85a2469 100644
--- a/README.md
+++ b/README.md
@@ -1 +1,2 @@
 # Practical 2
+This is README.md file

commit c17d711e5e7da2b1bed77bd9189c16add632aed1
Author: Kunalsk <kunalkavathekar@gmail.com>
Date:   Sat Mar 15 19:41:18 2025 +0530

    Added README.md

diff --git a/README.md b/README.md
new file mode 100644
index 0000000..2a6c68d
--- /dev/null
+++ b/README.md
@@ -0,0 +1 @@
+# Practical 2

Admin@Kunalsk36-Laptop MINGW64 /d/MCAsemII/DevOps/Practical2/practical2DevOps (feature/cart)
$ git log --oneline -- README.md
c383c1f (HEAD -> feature/cart) Updated README.md file
c17d711 Added README.md
```

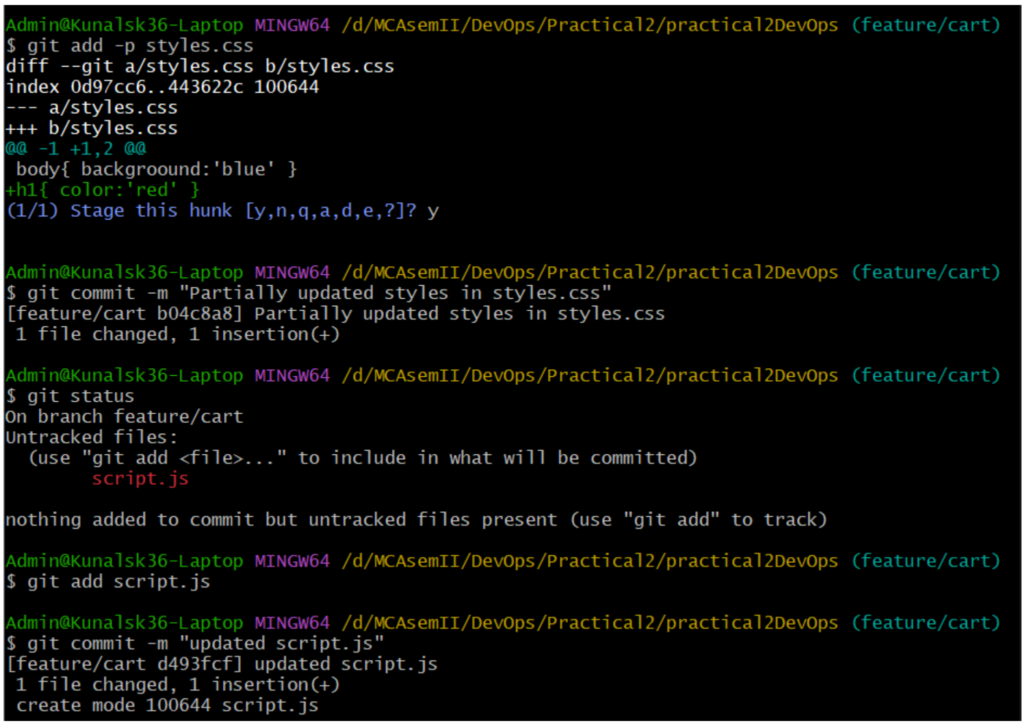


```
Admin@Kunalsk36-Laptop MINGW64 /d/MCAsemII/DevOps/Practical2/practical2DevOps (feature/cart)
$ git blame README.md
c17d711e (Kunalsk 2025-03-15 19:41:18 +0530 1) # Practical 2
c383c1f4 (Kunalsk 2025-03-15 19:42:16 +0530 2) This is README.md file

Admin@Kunalsk36-Laptop MINGW64 /d/MCAsemII/DevOps/Practical2/practical2DevOps (feature/cart)
$ git show c383c1f:README.md
# Practical 2
This is README.md file

Admin@Kunalsk36-Laptop MINGW64 /d/MCAsemII/DevOps/Practical2/practical2DevOps (feature/cart)
$ git diff c17d711 c383c1f -- README.md
diff --git a/README.md b/README.md
index 2a6c68d..85a2469 100644
--- a/README.md
+++ b/README.md
@@ -1 +1,2 @@
 # Practical 2
+This is README.md file
```

17. Partial File Staging

Problem: You modified `styles.css` and `script.js` but only want to commit `styles.css`. How?

Commands Needed: `git add -p`, `git commit`.

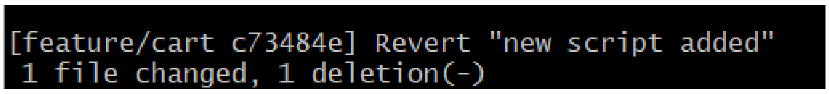


```
Admin@Kunalsk36-Laptop MINGW64 /d/MCAsemII/DevOps/Practical2/practical2DevOps (feature/cart)
$ git add -p styles.css
diff --git a/styles.css b/styles.css
index 0d97cc6..443622c 100644
--- a/styles.css
+++ b/styles.css
@@ -1 +1,2 @@
 body{ background:'blue' }
+h1{ color:'red' }
(1/1) Stage this hunk [y,n,q,a,d,e,?]? y


Admin@Kunalsk36-Laptop MINGW64 /d/MCAsemII/DevOps/Practical2/practical2DevOps (feature/cart)
$ git commit -m "Partially updated styles in styles.css"
[feature/cart b04c8a8] Partially updated styles in styles.css
 1 file changed, 1 insertion(+)

Admin@Kunalsk36-Laptop MINGW64 /d/MCAsemII/DevOps/Practical2/practical2DevOps (feature/cart)
$ git status
On branch feature/cart
Untracked files:
  (use "git add <file>..." to include in what will be committed)
        script.js

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

Admin@Kunalsk36-Laptop MINGW64 /d/MCAsemII/DevOps/Practical2/practical2DevOps (feature/cart)
$ git add script.js

Admin@Kunalsk36-Laptop MINGW64 /d/MCAsemII/DevOps/Practical2/practical2DevOps (feature/cart)
$ git commit -m "updated script.js"
[feature/cart d493fcf] updated script.js
 1 file changed, 1 insertion(+)
 create mode 100644 script.js
```

18. Revert a Public Commit

Problem: A commit pushed to `main` introduced a bug. How do you undo it without rewriting history?

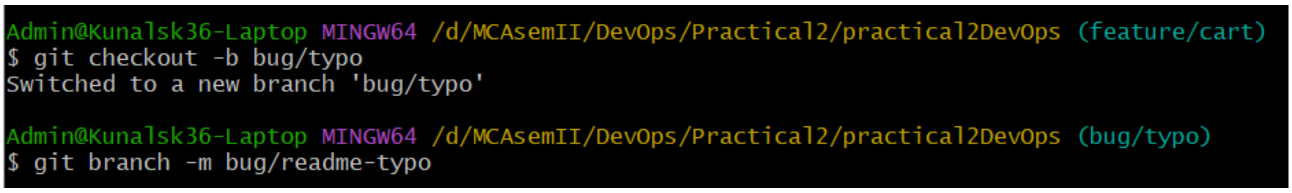
Commands Needed: `git revert <commit-hash>`.

- ➔ Step 1: Find the Buggy Commit's Hash using - git log --oneline --graph --decorate --all
- ➔ Step 2 : use `git revert <commit-hash>` command.
- ➔ 

```
[feature/cart c73484e] Revert "new script added"
 1 file changed, 1 deletion(-)
```

19. Rename a Branch

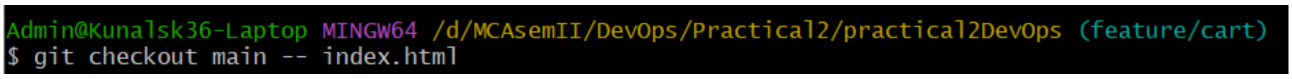
Problem: The branch `bug/typo` should be renamed to `bug/readme-typo`. How?



```
Admin@Kunalsk36-Laptop MINGW64 /d/MCAsemII/DevOps/Practical2/practical2DevOps (feature/cart)
$ git checkout -b bug/typo
Switched to a new branch 'bug/typo'

Admin@Kunalsk36-Laptop MINGW64 /d/MCAsemII/DevOps/Practical2/practical2DevOps (bug/typo)
$ git branch -m bug/readme-typo
```

20. Checkout a Specific File

Problem: You need to restore `index.html` from the `main` branch without switching branches.

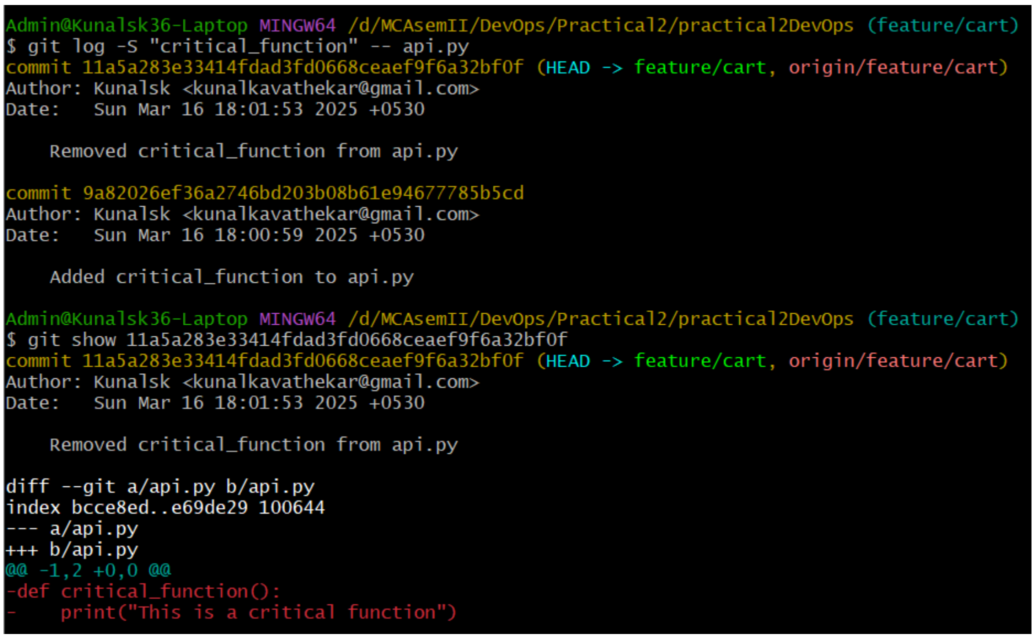


```
Admin@Kunalsk36-Laptop MINGW64 /d/MCAsemII/DevOps/Practical2/practical2DevOps (feature/cart)
$ git checkout main -- index.html
```

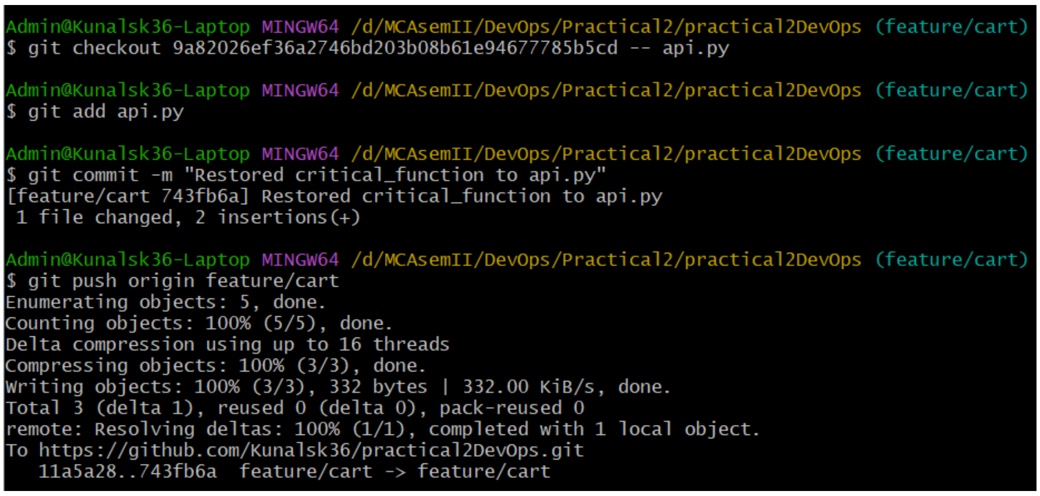
21. Find a Deleted Line

Problem: A critical function was removed from `api.py`. How do you find it when it was deleted?

Commands Needed: `git log -S "function_name"`.



```
Admin@Kunalsk36-Laptop MINGW64 /d/MCAsemII/DevOps/Practical2/practical2DevOps (feature/cart)
$ git log -S "critical_function" -- api.py
commit 11a5a283e33414fdad3fd0668ceaef9f6a32bf0f (HEAD -> feature/cart, origin/feature/cart)
Author: Kunalsk <kunalkavathekar@gmail.com>
Date:   Sun Mar 16 18:01:53 2025 +0530

    Removed critical_function from api.py

commit 9a82026ef36a2746bd203b08b61e94677785b5cd
Author: Kunalsk <kunalkavathekar@gmail.com>
Date:   Sun Mar 16 18:00:59 2025 +0530

    Added critical_function to api.py

Admin@Kunalsk36-Laptop MINGW64 /d/MCAsemII/DevOps/Practical2/practical2DevOps (feature/cart)
$ git show 11a5a283e33414fdad3fd0668ceaef9f6a32bf0f
commit 11a5a283e33414fdad3fd0668ceaef9f6a32bf0f (HEAD -> feature/cart, origin/feature/cart)
Author: Kunalsk <kunalkavathekar@gmail.com>
Date:   Sun Mar 16 18:01:53 2025 +0530

    Removed critical_function from api.py

diff --git a/api.py b/api.py
index bcce8ed..e69de29 100644
--- a/api.py
+++ b/api.py
@@ -1,2 +0,0 @@
-def critical_function():
-    print("This is a critical function")
```

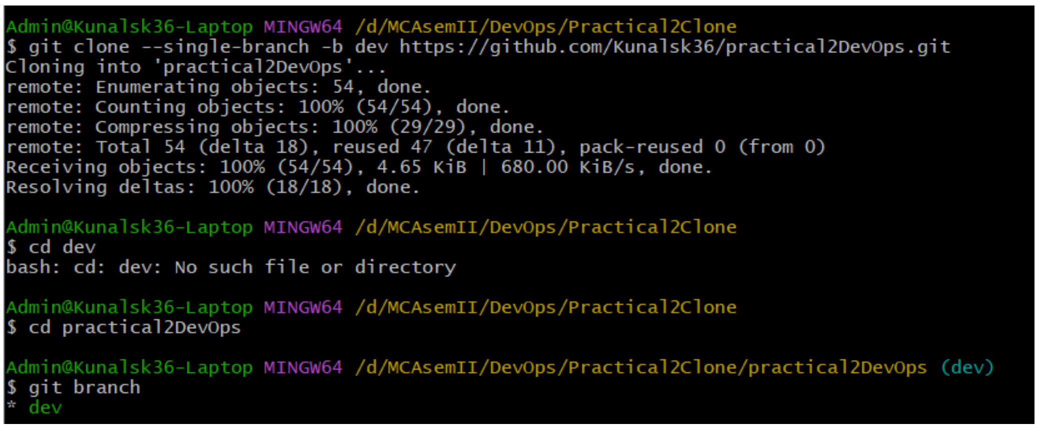


```
Admin@Kunalsk36-Laptop MINGW64 /d/MCAsemII/DevOps/Practical2/practical2DevOps (feature/cart)
$ git checkout 9a82026ef36a2746bd203b08b61e94677785b5cd -- api.py

Admin@Kunalsk36-Laptop MINGW64 /d/MCAsemII/DevOps/Practical2/practical2DevOps (feature/cart)
$ git add api.py

Admin@Kunalsk36-Laptop MINGW64 /d/MCAsemII/DevOps/Practical2/practical2DevOps (feature/cart)
$ git commit -m "Restored critical_function to api.py"
[feature/cart 743fb6a] Restored critical_function to api.py
 1 file changed, 2 insertions(+)

Admin@Kunalsk36-Laptop MINGW64 /d/MCAsemII/DevOps/Practical2/practical2DevOps (feature/cart)
$ git push origin feature/cart
Enumerating objects: 5, done.
Counting objects: 100% (5/5), done.
Delta compression using up to 16 threads
Compressing objects: 100% (3/3), done.
Writing objects: 100% (3/3), 332 bytes | 332.00 KiB/s, done.
Total 3 (delta 1), reused 0 (delta 0), pack-reused 0
remote: Resolving deltas: 100% (1/1), completed with 1 local object.
To https://github.com/Kunalsk36/practical2DevOps.git
   11a5a28..743fb6a  feature/cart -> feature/cart
```

22. Clone a Single Branch

Problem: The repository is huge, but you only need the `dev` branch. How do you clone it?

Commands Needed: `git clone --single-branch -b dev`.

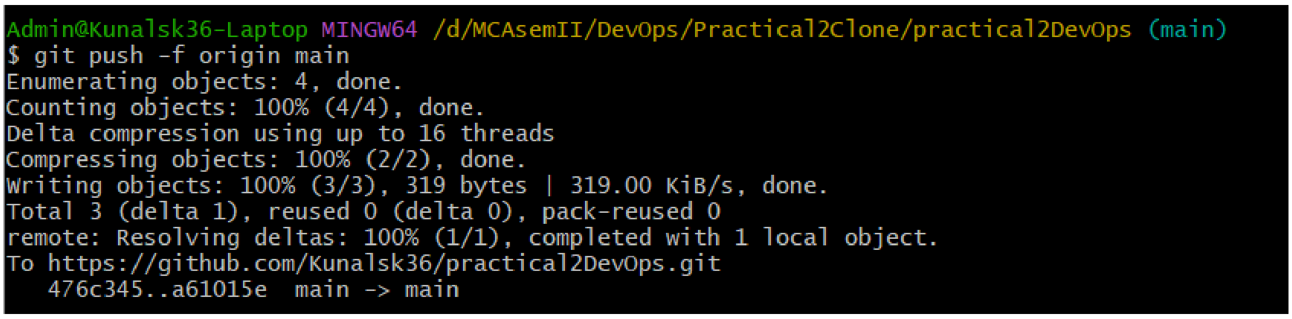


```
Admin@Kunalsk36-Laptop MINGW64 /d/MCAsemII/DevOps/Practical2Clone
$ git clone --single-branch -b dev https://github.com/Kunalsk36/practical2DevOps.git
Cloning into 'practical2DevOps'...
remote: Enumerating objects: 54, done.
remote: Counting objects: 100% (54/54), done.
remote: Compressing objects: 100% (29/29), done.
remote: Total 54 (delta 18), reused 47 (delta 11), pack-reused 0 (from 0)
Receiving objects: 100% (54/54), 4.65 KiB | 680.00 KiB/s, done.
Resolving deltas: 100% (18/18), done.

Admin@Kunalsk36-Laptop MINGW64 /d/MCAsemII/DevOps/Practical2Clone
$ cd dev
bash: cd: dev: No such file or directory

Admin@Kunalsk36-Laptop MINGW64 /d/MCAsemII/DevOps/Practical2Clone
$ cd practical2DevOps

Admin@Kunalsk36-Laptop MINGW64 /d/MCAsemII/DevOps/Practical2Clone/practical2DevOps (dev)
$ git branch
* dev
```

23. Override Remote Changes

Problem: You need to force-push your local `main` branch to overwrite a corrupted remote.

Commands Needed: `git push -f origin main`.



```
Admin@Kunalsk36-Laptop MINGW64 /d/MCAsemII/DevOps/Practical2Clone/practical2DevOps (main)
$ git push -f origin main
Enumerating objects: 4, done.
Counting objects: 100% (4/4), done.
Delta compression using up to 16 threads
Compressing objects: 100% (2/2), done.
Writing objects: 100% (3/3), 319 bytes | 319.00 KiB/s, done.
Total 3 (delta 1), reused 0 (delta 0), pack-reused 0
remote: Resolving deltas: 100% (1/1), completed with 1 local object.
To https://github.com/Kunalsk36/practical2DevOps.git
   476c345..a61015e  main -> main
```

24. Collaborate via Pull Request

Problem: You fixed a bug in a forked repo. How do you propose the change to the original project?

➔ Step 1. Clone the Forked Repository
  $ git clone https://github.com/your-username/forked-repo.git
  $ cd forked-repo

➔ Step 2. Add the Original Repository as Upstream
  $ git remote add upstream https://github.com/original-owner/original-repo.git
  $ git fetch upstream

➔ Step 3. Create a New Branch for the Fix
  git checkout -b bugfix-branch

➔ Step 4. Make and Commit the Fix
  $ git add .
  $ git commit -m "script.js file edited"

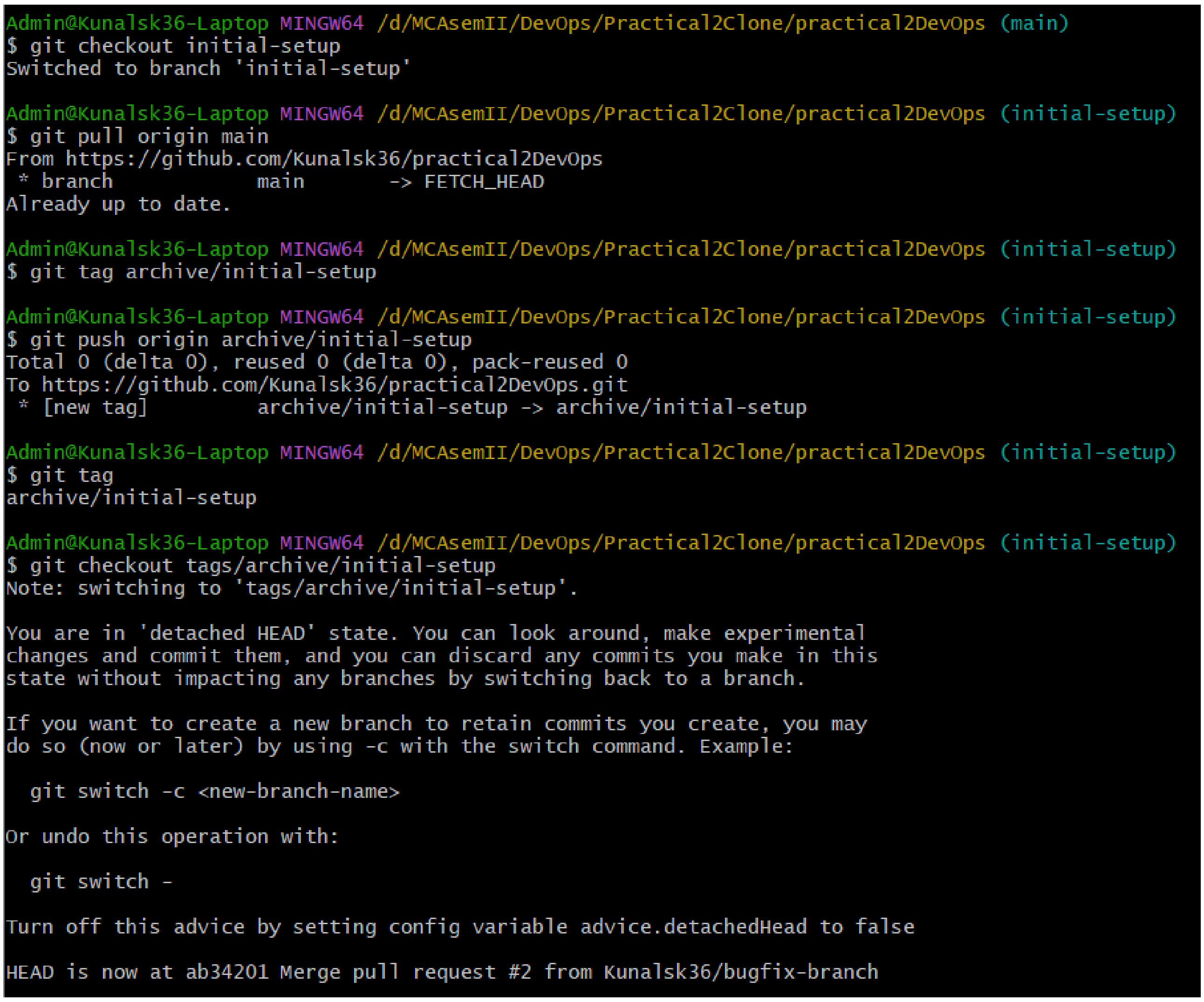
➔ Step 5. Push the Changes to Your Forked Repository
  git push origin bugfix-branch

➔ Step 6. Create a Pull Request (PR) on GitHub
  - Go to your forked repository on GitHub.
  - Click "Compare & pull request" next to your pushed branch.
  - Select the base repository (original repo) and base branch (e.g., main or develop).
  - Add a clear title and description for the PR.
  - Click "Create Pull Request".

➔ Step 7. Wait for Review & Merge
  - The original repo maintainers will review your PR.
  - They may request changes or merge it directly.

25. Archive a Repository

Problem: The `legacy-project` branch is no longer needed. How do you archive it?



```
Admin@Kunalsk36-Laptop MINGW64 /d/MCAsemII/DevOps/Practical2Clone/practical2DevOps (main)
$ git checkout initial-setup
Switched to branch 'initial-setup'

Admin@Kunalsk36-Laptop MINGW64 /d/MCAsemII/DevOps/Practical2Clone/practical2DevOps (initial-setup)
$ git pull origin main
From https://github.com/Kunalsk36/practical2DevOps
 * branch            main       -> FETCH_HEAD
Already up to date.

Admin@Kunalsk36-Laptop MINGW64 /d/MCAsemII/DevOps/Practical2Clone/practical2DevOps (initial-setup)
$ git tag archive/initial-setup

Admin@Kunalsk36-Laptop MINGW64 /d/MCAsemII/DevOps/Practical2Clone/practical2DevOps (initial-setup)
$ git push origin archive/initial-setup
Total 0 (delta 0), reused 0 (delta 0), pack-reused 0
To https://github.com/Kunalsk36/practical2DevOps.git
 * [new tag]         archive/initial-setup -> archive/initial-setup

Admin@Kunalsk36-Laptop MINGW64 /d/MCAsemII/DevOps/Practical2Clone/practical2DevOps (initial-setup)
$ git tag
archive/initial-setup

Admin@Kunalsk36-Laptop MINGW64 /d/MCAsemII/DevOps/Practical2Clone/practical2DevOps (initial-setup)
$ git checkout tags/archive/initial-setup
Note: switching to 'tags/archive/initial-setup'.

You are in 'detached HEAD' state. You can look around, make experimental
changes and commit them, and you can discard any commits you make in this
state without impacting any branches by switching back to a branch.

If you want to create a new branch to retain commits you create, you may
do so (now or later) by using -c with the switch command. Example:

  git switch -c <new-branch-name>

Or undo this operation with:

  git switch -

Turn off this advice by setting config variable advice.detachedHead to false

HEAD is now at ab34201 Merge pull request #2 from Kunalsk36/bugfix-branch
```