GIT and Github Assignment

Name: Pooja Aswatha **Date:05/08/2024**

Introduction to Version Control System:

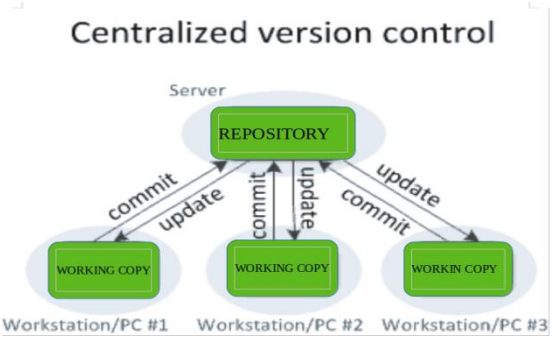
Version Control System: Version control systems are a category of software tools that helps in recording changes made to files by keeping a track of modifications done in the code. A version control system is a kind of software that helps the developer team to efficiently communicate and manage(track) all the changes that have been made to the source code along with the information like who made and what changes have been made. A separate branch is created for every contributor who made the changes and the changes aren't merged into the original source code unless all are analyzed as soon as the changes are green signaled they merged to the main source code. It not only keeps source code organized but also improves productivity by making the development process smooth.

Types of Version Control Systems:

- Local Version Control Systems
- Centralized Version Control Systems
- Distributed Version Control Systems
- **Local Version Control Systems:** It is one of the simplest forms and has a database that kept all the changes to files under revision control. RCS is one of the most common VCS tools. It keeps patch sets (differences between files) in a special format on disk. By adding up all the patches it can then re-create what any file looked like at any point in time.
- **Centralized Version Control Systems:** Centralized version control systems contain just one repository globally and every user need to commit for reflecting one's changes in the repository. It is possible for others to see your changes by updating.

Two things are required to make your changes visible to others which are:

I. You commit
II. They update

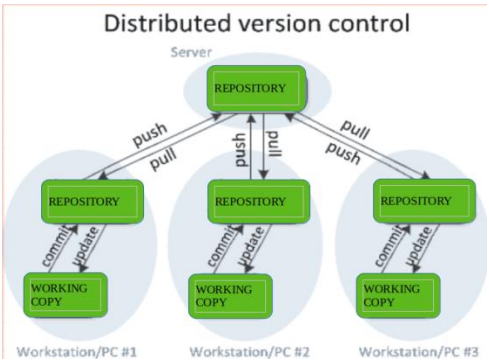


- **Distributed Version Control Systems:** Distributed version control systems contain multiple repositories. Each user has their own repository and working copy. Just committing your changes will not give others access to your changes. This is because commit will reflect those changes in your local repository and you need to push them in order to make them visible on the central repository. Similarly, When you update, you do not get others' changes unless you have first pulled those changes into your repository.

To make your changes visible to others, 4 things are required:

I. You commit
II. You push
III. They pull
IV. They update

The most popular distributed version control systems are **Git**,



Git (Distributed Version Control System):

it is a version control system that allows developers to track changes made to their code, collaborate with others, and manage different versions of their projects. It's a powerful tool that helps developers work together on software projects, ensuring that changes are properly recorded, and previous versions can be recovered if needed.

Git was created by Linus Torvalds in 2005 and has since become the most widely used version control system in the world. It's used by developers, engineers, and even writers to manage and collaborate on projects.

Some key features of Git include:

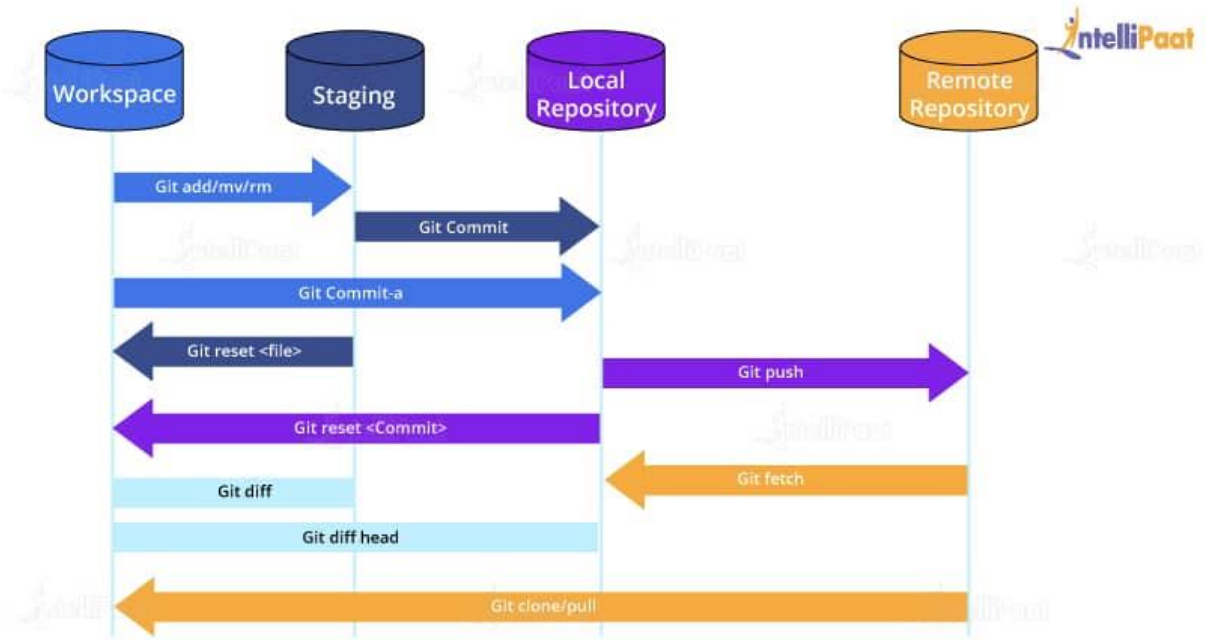
1. **Version control:** Git tracks changes made to code, allowing developers to see who made changes, when, and why.

2. **Branching and merging:** Git enables developers to create separate branches for new features or fixes, and then merge them into the main codebase.

3. **Distributed development:** Git allows multiple developers to work on the same project simultaneously, without conflicts.

4. **Open-source:** Git is free and open-source, making it accessible to everyone.

Git Architecture :



The three layers are:

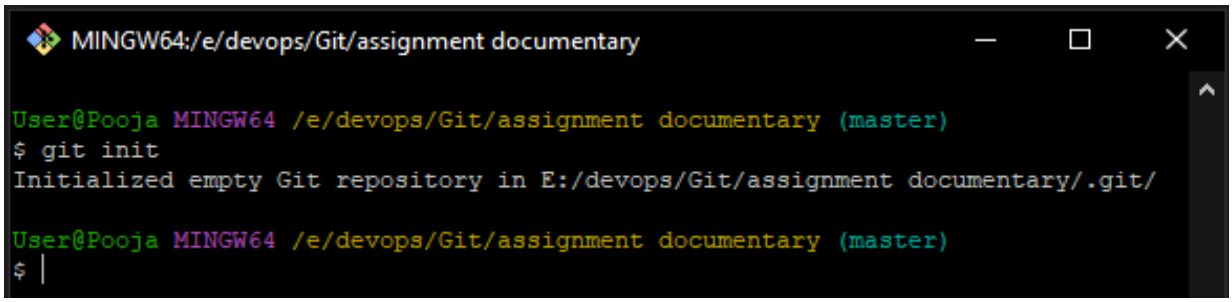
- **Working directory**: This is created when a Git project is initialized onto your local machine and allows you to edit the source code copied.
- **Staging area**: Post the edits, the code is staged in the staging area by applying the command, **git add**. This displays a preview for the next stage. In case further modifications are made in the working directory, the snapshots for these two layers will be different. However, these can be synced by using the same 'git add' command.
- **Local repository**: If no further edits are required to be done, then you can go ahead and apply the **git commit** command. This replicates the latest snapshots in all three stages, making them in sync with each other.
- **Central repository**: on Git Push, files are moved to central repo.

Git Commads:

1. **Git init:**

Usage: **git init [repository name]**

We have to navigate to our project directory and type the command **git init** to initialize a Git repository for our local project folder. Git will create a hidden **.git** directory and use it to keep its files organized in other subdirectories.

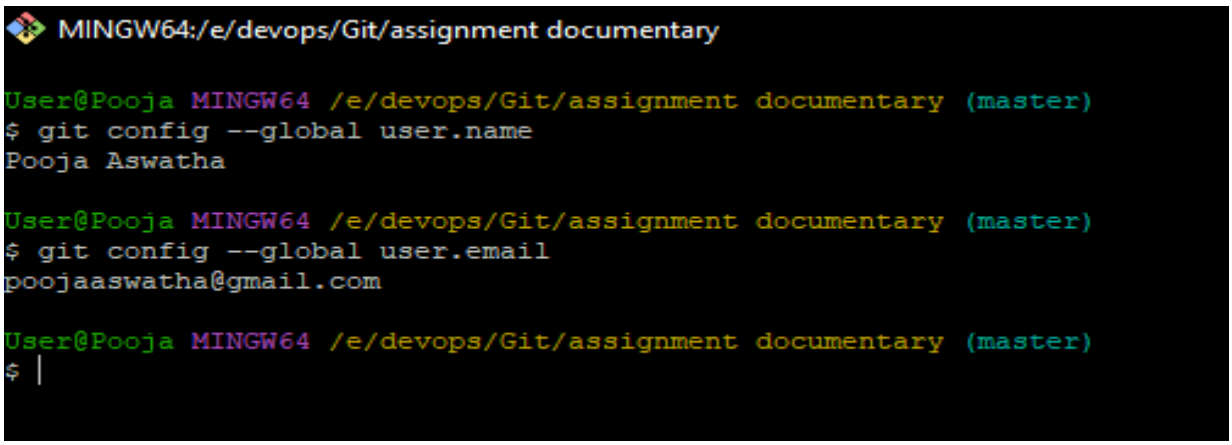


2. **git config**

Usage: git config –global user.name "[name]"

Usage: git config –global user.email "[email address]"

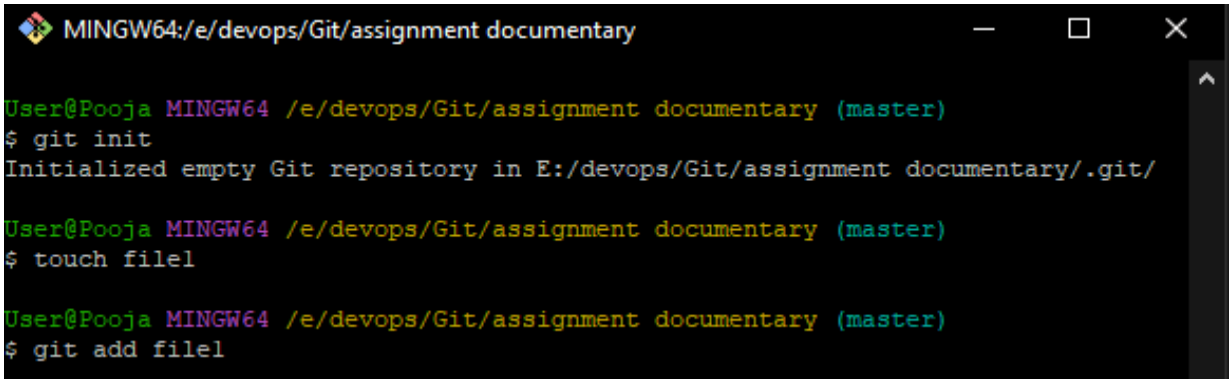
This command sets the author name and email address respectively to be used with your commits.



3. **git add**

Usage: (i) git add [file(s) name]

This will add the specified file(s) into the Git repository, the staging area, where they are already being tracked by Git and now ready to be committed.



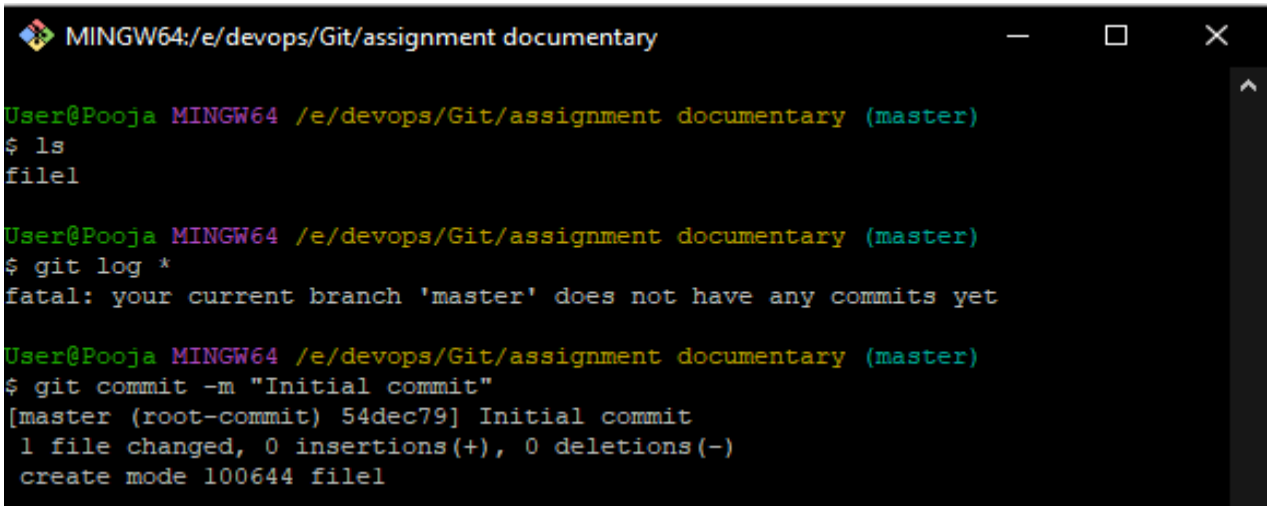
Usage: (ii) git add . or *

this will take all our files into the Git repository, i.e., into the staging area.

4. **git commit**

Usage: git commit -m “message”

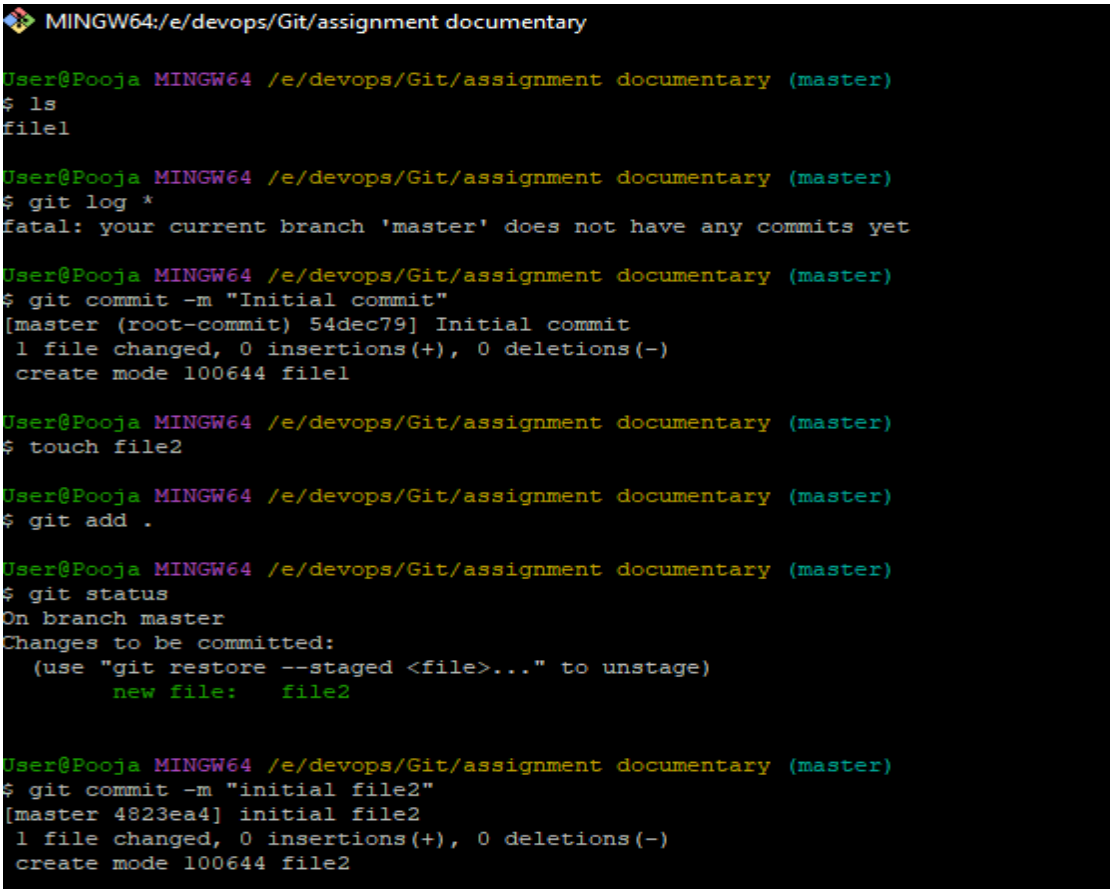
This command records or snapshots files permanently in the version history. All the files, which are there in the directory right now, are being saved in the Git file system.



5. **git status**

Usage: git status

This command will show the modified status of an existing file and the file addition status of a new file, if any, that has to be committed.



Branching:

Branching in Git is a powerful feature that allows developers to diverge from the main codebase (usually called the "master" or "main" branch) and work on new features, fixes, or experiments independently. Here's a brief overview:

Why Branching?

1. **Isolation:** Branches provide a safe space to work on new features without affecting the main codebase.

2. **Parallel Development:** Multiple developers can work on different branches simultaneously, without conflicts.

3. **Experimentation:** Branches allow for experimentation and testing of new ideas without risking the main codebase.

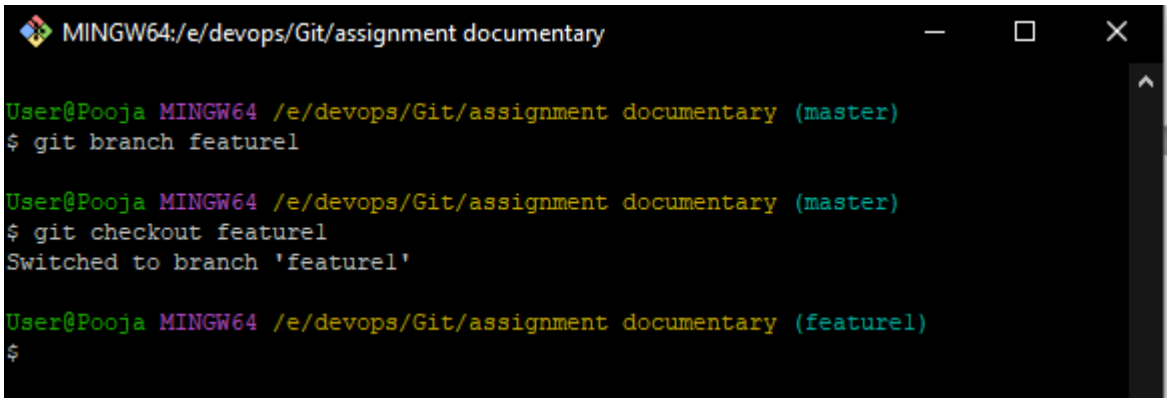
4. **Collaboration:** Branches make it easier for teams to collaborate on specific features or tasks.

Key Git Branching Commands

1. git branch <branch-name>: Create a new branch.

2. git checkout <branch-name>: Switch to a different branch.

4. git branch -d <branch-name>: Delete a branch (after merging).



Merging:

Merging in Git is the process of combining changes from two branches into a single branch. Here's a detailed overview:

Why Merge?

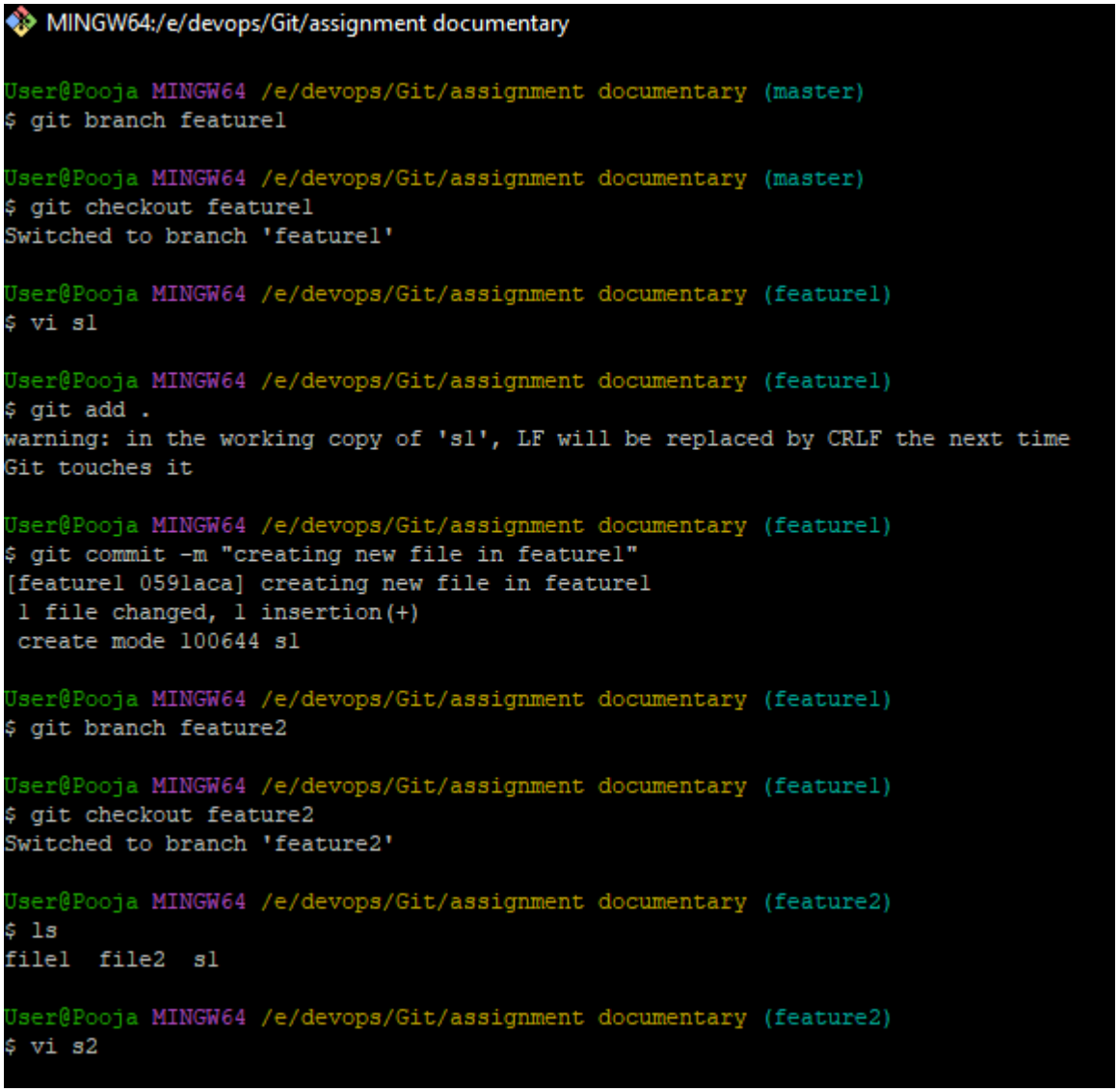
1. Integrate Changes: Merge brings together changes from different branches, creating a unified version.

2. Resolve Conflicts: Merge helps resolve conflicts between changes made in different branches.

3. Synchronize Branches: Merge ensures that branches are up-to-date with each other.

Command:

git merge <branch-name>: Merge changes from one branch into another.


MINGW64:/e/devops/Git/assignment documentary
User@Pooja MINGW64 /e/devops/Git/assignment documentary (master)
$ git branch feature1
User@Pooja MINGW64 /e/devops/Git/assignment documentary (master)
$ git checkout feature1
Switched to branch 'feature1'
User@Pooja MINGW64 /e/devops/Git/assignment documentary (feature1)
$ vi s1
User@Pooja MINGW64 /e/devops/Git/assignment documentary (feature1)
$ git add .
warning: in the working copy of 's1', LF will be replaced by CRLF the next time
Git touches it
User@Pooja MINGW64 /e/devops/Git/assignment documentary (feature1)
$ git commit -m "creating new file in feature1"
[feature1 059laca] creating new file in feature1
1 file changed, 1 insertion(+)
create mode 100644 s1
User@Pooja MINGW64 /e/devops/Git/assignment documentary (feature1)
$ git branch feature2
User@Pooja MINGW64 /e/devops/Git/assignment documentary (feature1)
$ git checkout feature2
Switched to branch 'feature2'
User@Pooja MINGW64 /e/devops/Git/assignment documentary (feature2)
$ ls
file1 file2 s1
User@Pooja MINGW64 /e/devops/Git/assignment documentary (feature2)
$ vi s2

```
MINGW64:/e/devops/Git/assignment documentary

User@Pooja MINGW64 /e/devops/Git/assignment documentary (feature2)
$ git add .
warning: in the working copy of 's2', LF will be replaced by CRLF the next time
Git touches it

User@Pooja MINGW64 /e/devops/Git/assignment documentary (feature2)
$ git commit -m "adding new file to feature2"
[feature2 4fb4a2e] adding new file to feature2
 1 file changed, 1 insertion(+)
 create mode 100644 s2

User@Pooja MINGW64 /e/devops/Git/assignment documentary (feature2)
$ ls
file1  file2  s1  s2

User@Pooja MINGW64 /e/devops/Git/assignment documentary (feature2)
$ cat s2
hello

User@Pooja MINGW64 /e/devops/Git/assignment documentary (feature2)
$ cat s1
hello welcome to git

User@Pooja MINGW64 /e/devops/Git/assignment documentary (feature2)
$ git status
On branch feature2
nothing to commit, working tree clean

User@Pooja MINGW64 /e/devops/Git/assignment documentary (feature2)
$ git log --oneline
4fb4a2e (HEAD -> feature2) adding new file to feature2
0591aca (feature1) creating new file in feature1
4823ea4 (master) initial file2
54dec79 Initial commit

User@Pooja MINGW64 /e/devops/Git/assignment documentary (feature2)
$ git checkout master
Switched to branch 'master'

User@Pooja MINGW64 /e/devops/Git/assignment documentary (master)
$ ls
file1  file2

User@Pooja MINGW64 /e/devops/Git/assignment documentary (master)
$ git merge feature2
```

```
User@Pooja MINGW64 /e/devops/Git/assignment documentary (master)
$ git merge feature2
Updating 4823ea4..4fb4a2e
Fast-forward
 s1 | 1 +
 s2 | 1 +
 2 files changed, 2 insertions(+)
 create mode 100644 s1
 create mode 100644 s2

User@Pooja MINGW64 /e/devops/Git/assignment documentary (master)
$ ls
file1  file2  s1  s2
```

Git Cherry-pick:

git cherry-pick is a command used in Git, a version control system, to apply changes from one commit to another. It allows you to select specific commits from one branch and apply them to another branch.

Here's a basic overview of how to use git cherry-pick:

1. git log: First, use git log to find the commit hash (a unique identifier) of the commit you want to cherry-pick.

2. git checkout: Then, switch to the branch where you want to apply the changes using git checkout <branch-name>.

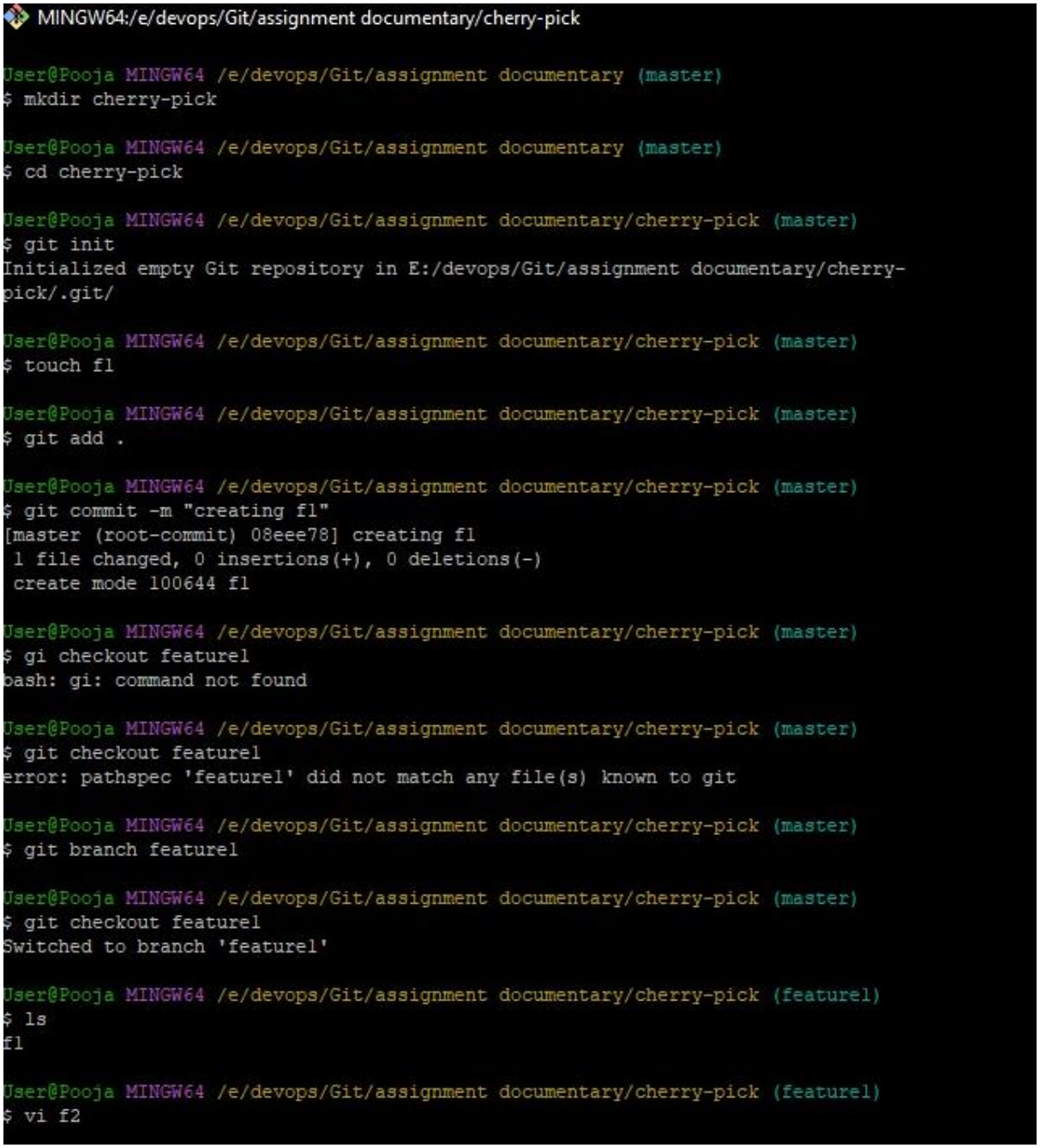
3. git cherry-pick <commit-hash>: Finally, use git cherry-pick <commit-hash> to apply the changes from the specified commit to the current branch.

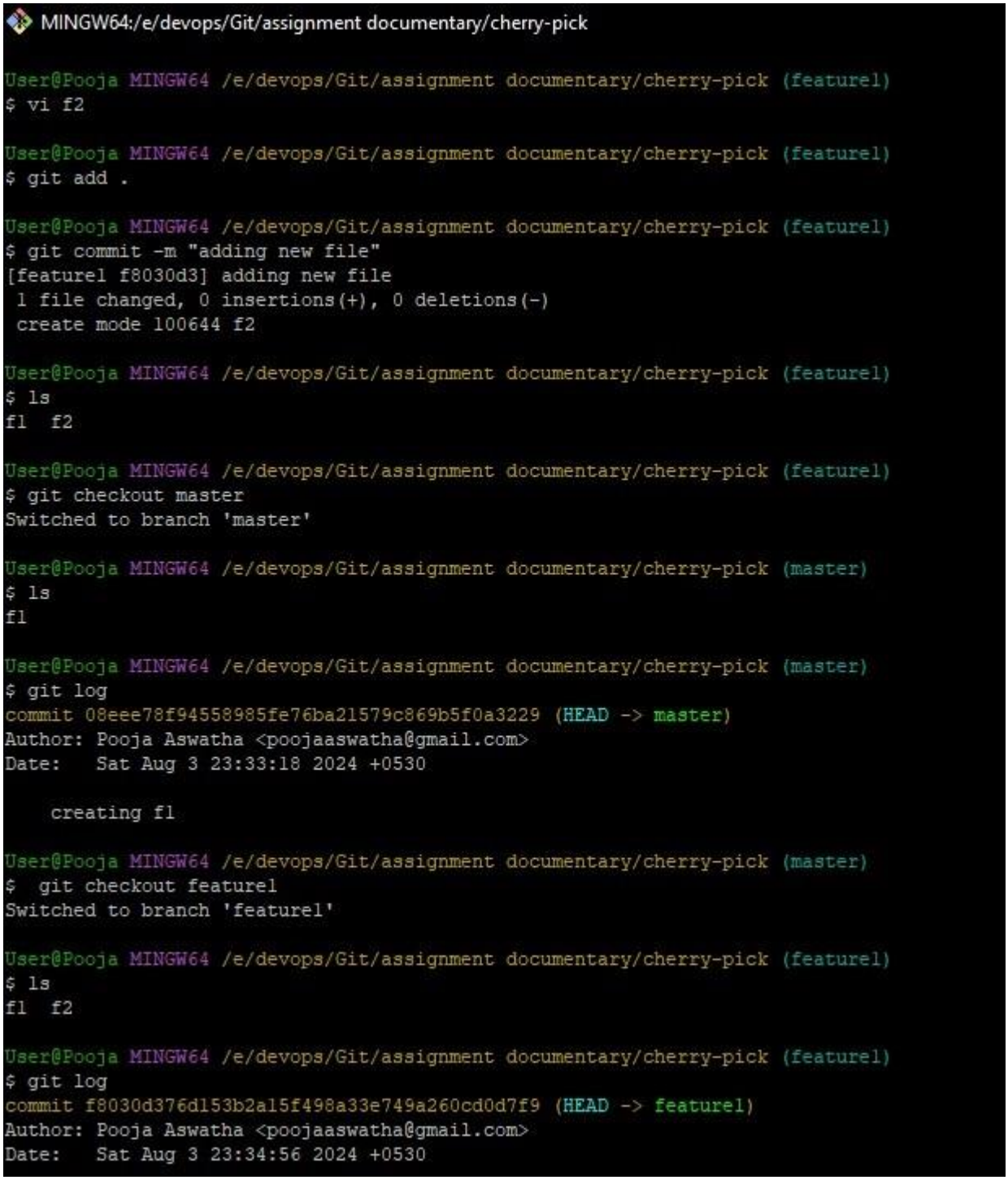
Some additional options you can use with git cherry-pick include:

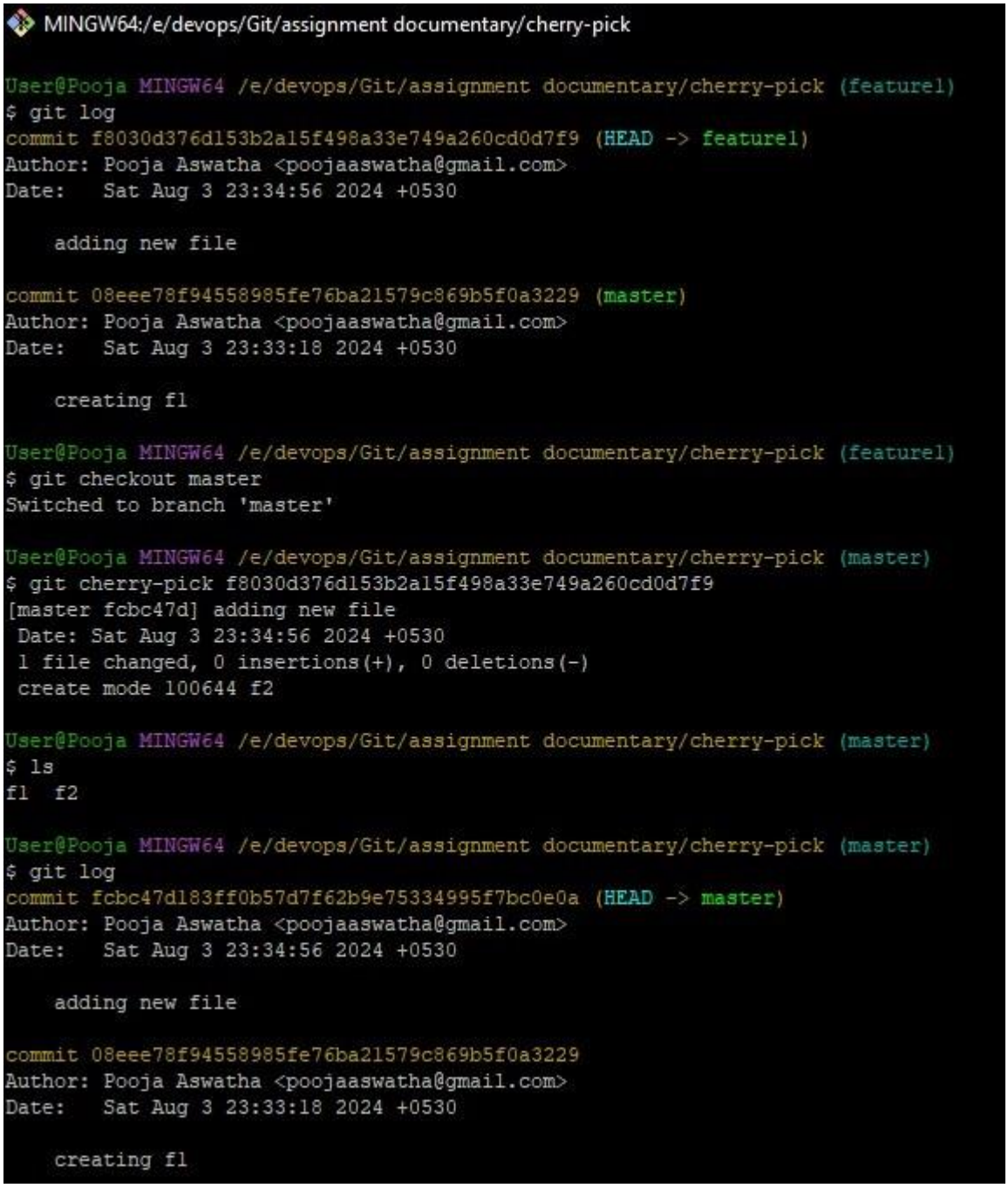
- -x: This option appends a line to the commit message to indicate that it's a cherry-pick.

- -e: This option allows you to edit the commit message before committing the changes.

- -n: This option doesn't commit the changes automatically, allowing you to review and modify them before committing.


MINGW64:/e/devops/Git/assignment documentary/cherry-pick
User@Pooja MINGW64 /e/devops/Git/assignment documentary (master)
$ mkdir cherry-pick
User@Pooja MINGW64 /e/devops/Git/assignment documentary (master)
$ cd cherry-pick
User@Pooja MINGW64 /e/devops/Git/assignment documentary/cherry-pick (master)
$ git init
Initialized empty Git repository in E:/devops/Git/assignment documentary/cherry-pick/.git/
User@Pooja MINGW64 /e/devops/Git/assignment documentary/cherry-pick (master)
$ touch f1
User@Pooja MINGW64 /e/devops/Git/assignment documentary/cherry-pick (master)
$ git add .
User@Pooja MINGW64 /e/devops/Git/assignment documentary/cherry-pick (master)
$ git commit -m "creating f1"
[master (root-commit) 08eee78] creating f1
1 file changed, 0 insertions(+), 0 deletions(-)
create mode 100644 f1
User@Pooja MINGW64 /e/devops/Git/assignment documentary/cherry-pick (master)
$ gi checkout feature1
bash: gi: command not found
User@Pooja MINGW64 /e/devops/Git/assignment documentary/cherry-pick (master)
$ git checkout feature1
error: pathspec 'feature1' did not match any file(s) known to git
User@Pooja MINGW64 /e/devops/Git/assignment documentary/cherry-pick (master)
$ git branch feature1
User@Pooja MINGW64 /e/devops/Git/assignment documentary/cherry-pick (master)
$ git checkout feature1
Switched to branch 'feature1'
User@Pooja MINGW64 /e/devops/Git/assignment documentary/cherry-pick (feature1)
$ ls
f1
User@Pooja MINGW64 /e/devops/Git/assignment documentary/cherry-pick (feature1)
$ vi f2


MINGW64:/e/devops/Git/assignment documentary/cherry-pick
User@Pooja MINGW64 /e/devops/Git/assignment documentary/cherry-pick (feature1)
$ vi f2
User@Pooja MINGW64 /e/devops/Git/assignment documentary/cherry-pick (feature1)
$ git add .
User@Pooja MINGW64 /e/devops/Git/assignment documentary/cherry-pick (feature1)
$ git commit -m "adding new file"
[feature1 f8030d3] adding new file
1 file changed, 0 insertions(+), 0 deletions(-)
create mode 100644 f2
User@Pooja MINGW64 /e/devops/Git/assignment documentary/cherry-pick (feature1)
$ ls
f1 f2
User@Pooja MINGW64 /e/devops/Git/assignment documentary/cherry-pick (feature1)
$ git checkout master
Switched to branch 'master'
User@Pooja MINGW64 /e/devops/Git/assignment documentary/cherry-pick (master)
$ ls
f1
User@Pooja MINGW64 /e/devops/Git/assignment documentary/cherry-pick (master)
$ git log
commit 08eee78f94558985fe76ba21579c869b5f0a3229 (HEAD -> master)
Author: Pooja Aswatha <poojaaswatha@gmail.com>
Date: Sat Aug 3 23:33:18 2024 +0530
creating f1
User@Pooja MINGW64 /e/devops/Git/assignment documentary/cherry-pick (master)
$ git checkout feature1
Switched to branch 'feature1'
User@Pooja MINGW64 /e/devops/Git/assignment documentary/cherry-pick (feature1)
$ ls
f1 f2
User@Pooja MINGW64 /e/devops/Git/assignment documentary/cherry-pick (feature1)
$ git log
commit f8030d376d153b2a15f498a33e749a260cd0d7f9 (HEAD -> feature1)
Author: Pooja Aswatha <poojaaswatha@gmail.com>
Date: Sat Aug 3 23:34:56 2024 +0530



Tagging:

Git tagging is a way to mark specific points in a Git repository's history as important. Here are some key aspects of Git tagging:

Why use tags?

- Mark release points (e.g., v1.0, v2.1)

- Create a snapshot of your code at a particular point in time

- Easily reference or revert to a specific version later

Types of tags

- **Lightweight tags:** Simple references to a specific commit

- **Annotated tags:** Store additional information like author, date, and message

Common Git tagging commands

- git tag: List all tags

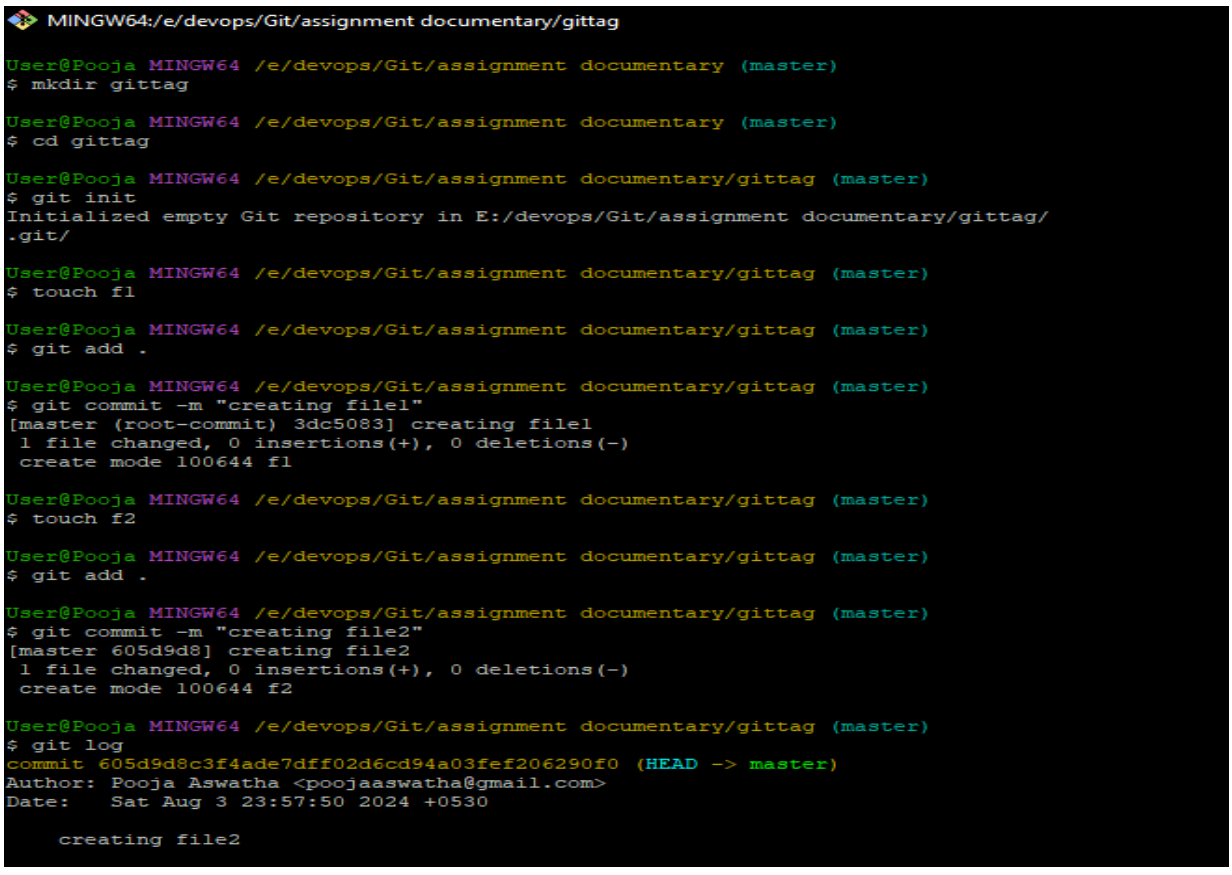
- git tag <tagname>: Create a lightweight tag

- git tag -a <tagname> -m "<message>": Create an annotated tag

- git show <tagname>: View tag details

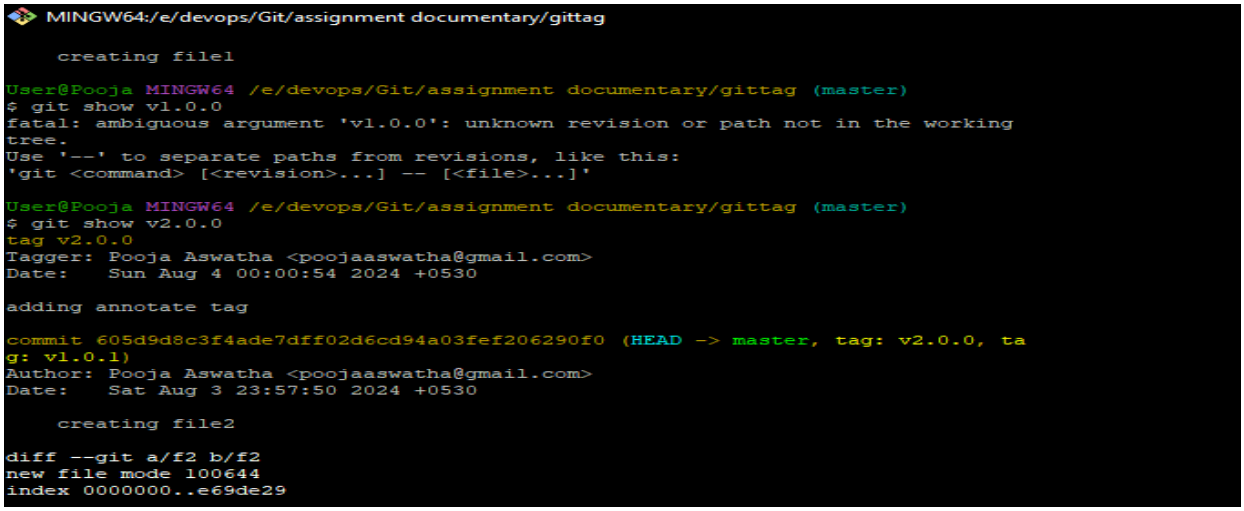
- git checkout <tagname>: Switch to a specific tag

- git tag -d <tagname>: Delete a tag



```
MINGW64:/e/devops/Git/assignment documentary/gittag
User@Pooja MINGW64 /e/devops/Git/assignment documentary/gittag (master)
$ git log
commit 605d9d8c3f4ade7dff02d6cd94a03fef206290f0 (HEAD -> master)
Author: Pooja Aswatha <poojaaswatha@gmail.com>
Date:   Sat Aug 3 23:57:50 2024 +0530

    creating file2

commit 3dc50837ae8dc976e26ec4a5aedla7356fa250f3
Author: Pooja Aswatha <poojaaswatha@gmail.com>
Date:   Sat Aug 3 23:57:24 2024 +0530

    creating file1

User@Pooja MINGW64 /e/devops/Git/assignment documentary/gittag (master)
$ git tag v1.0.1

User@Pooja MINGW64 /e/devops/Git/assignment documentary/gittag (master)
$ git tags
git: 'tags' is not a git command. See 'git --help'.

The most similar commands are
        stage
        tag

User@Pooja MINGW64 /e/devops/Git/assignment documentary/gittag (master)
$ git tag
v1.0.1

User@Pooja MINGW64 /e/devops/Git/assignment documentary/gittag (master)
$ git log
commit 605d9d8c3f4ade7dff02d6cd94a03fef206290f0 (HEAD -> master, tag: v1.0.1)
Author: Pooja Aswatha <poojaaswatha@gmail.com>
Date:   Sat Aug 3 23:57:50 2024 +0530

    creating file2

commit 3dc50837ae8dc976e26ec4a5aedla7356fa250f3
Author: Pooja Aswatha <poojaaswatha@gmail.com>
Date:   Sat Aug 3 23:57:24 2024 +0530

    creating file1
```

```
MINGW64:/e/devops/Git/assignment documentary/gittag

User@Pooja MINGW64 /e/devops/Git/assignment documentary/gittag (master)
$ git checkout v1.0.0
error: pathspec 'v1.0.0' did not match any file(s) known to git

User@Pooja MINGW64 /e/devops/Git/assignment documentary/gittag (master)
$ touch file3

User@Pooja MINGW64 /e/devops/Git/assignment documentary/gittag (master)
$ git add .

User@Pooja MINGW64 /e/devops/Git/assignment documentary/gittag (master)
$ git commit -a v2.0.0 -m "adding annotate tag"
fatal: paths 'v2.0.0 ...' with -a does not make sense

User@Pooja MINGW64 /e/devops/Git/assignment documentary/gittag (master)
$ git commit -a v2.0.0 -m "adding annotate tag"
fatal: paths 'v2.0.0 ...' with -a does not make sense

User@Pooja MINGW64 /e/devops/Git/assignment documentary/gittag (master)
$ git tag -a v2.0.0 -m "adding annotate tag"

User@Pooja MINGW64 /e/devops/Git/assignment documentary/gittag (master)
$ git tag
v1.0.1
v2.0.0

User@Pooja MINGW64 /e/devops/Git/assignment documentary/gittag (master)
$ git log
commit 605d9d8c3f4ade7dff02d6cd94a03fef206290f0 (HEAD -> master, tag: v2.0.0, ta
g: v1.0.1)
Author: Pooja Aswatha <poojaaswatha@gmail.com>
Date:   Sat Aug 3 23:57:50 2024 +0530

    creating file2

commit 3dc50837ae8dc976e26ec4a5aedla7356fa250f3
Author: Pooja Aswatha <poojaaswatha@gmail.com>
Date:   Sat Aug 3 23:57:24 2024 +0530

    creating file1

User@Pooja MINGW64 /e/devops/Git/assignment documentary/gittag (master)
$ git show v1.0.0
fatal: ambiguous argument 'v1.0.0': unknown revision or path not in the working
tree.
```



Difference between tag and branch:

Tag	Branch
i. Mark a specific point in the commit history ii. Immutable (cannot be changed once created) iii. Typically used to: - Mark release versions (e.g., v1.0, v2.1) - Create a snapshot of the code at a particular point in time - Easily reference or revert to a specific version later iv. Do not have a commit history of their own	**i.** Independent lines of development ii. Can be created, merged, and deleted iii. Typically used to: - Develop new features or fixes - Experiment with different approaches - Isolate changes from the main codebase (e.g., master branch) iv. Have their own commit history

Key differences:

- Immutability: Tags are fixed references, while branches can be changed and updated.
- Purpose: Tags mark specific points, while branches facilitate development and experimentation.
- Commit history: Tags don't have their own history, while branches do.

To illustrate the difference, consider a published book:

- A tag would represent a specific edition (e.g., "v1.0"), frozen in time.
- A branch would represent a manuscript being edited and updated (e.g., "draft-2"), which can change and evolve over time**.**

Git Revert

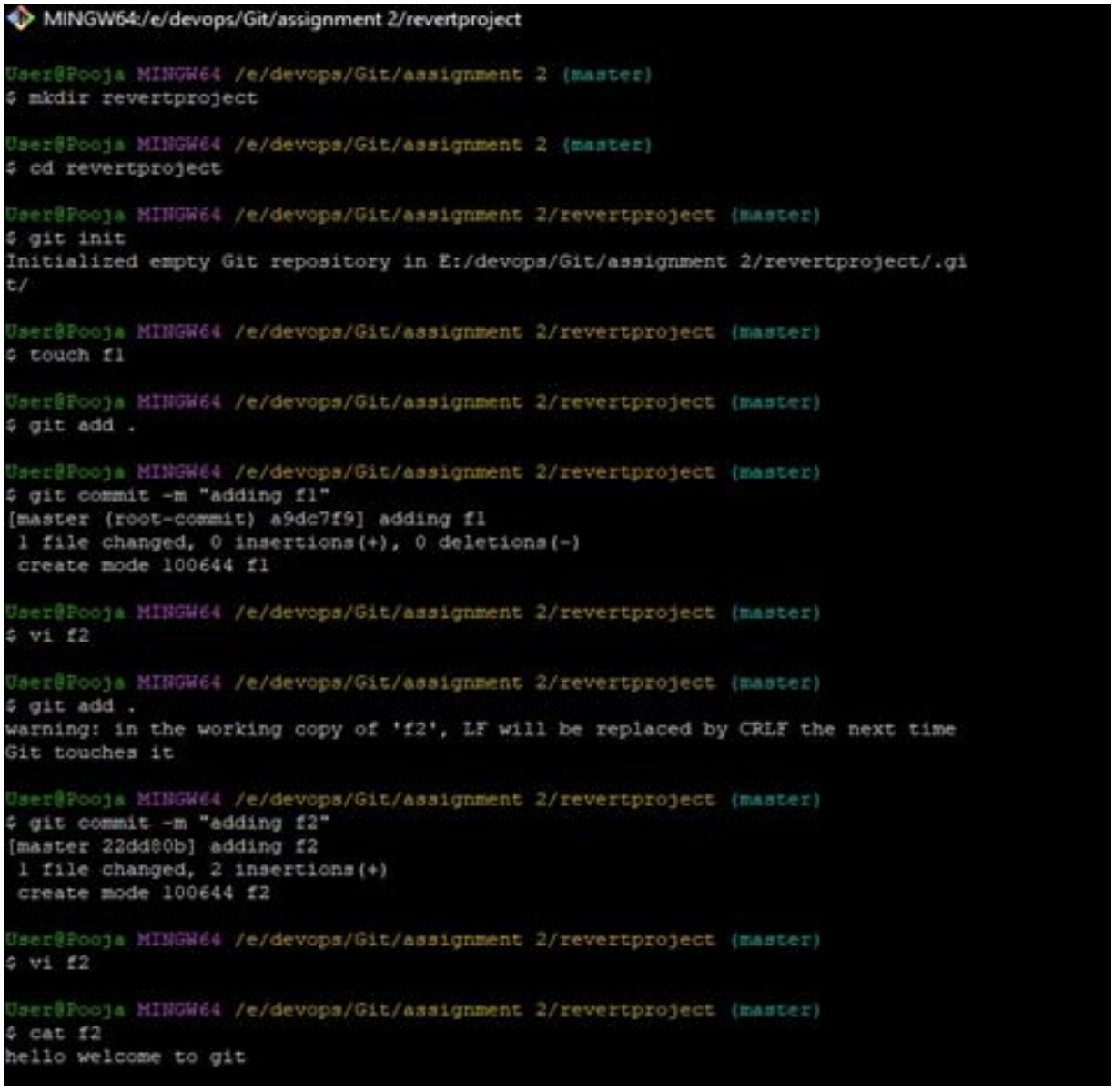
Git revert is a command used in Git to **undo changes made by a specific commit.** It creates a new commit that reverses the changes made by the original commit, effectively "reverting" the repository to a previous state.

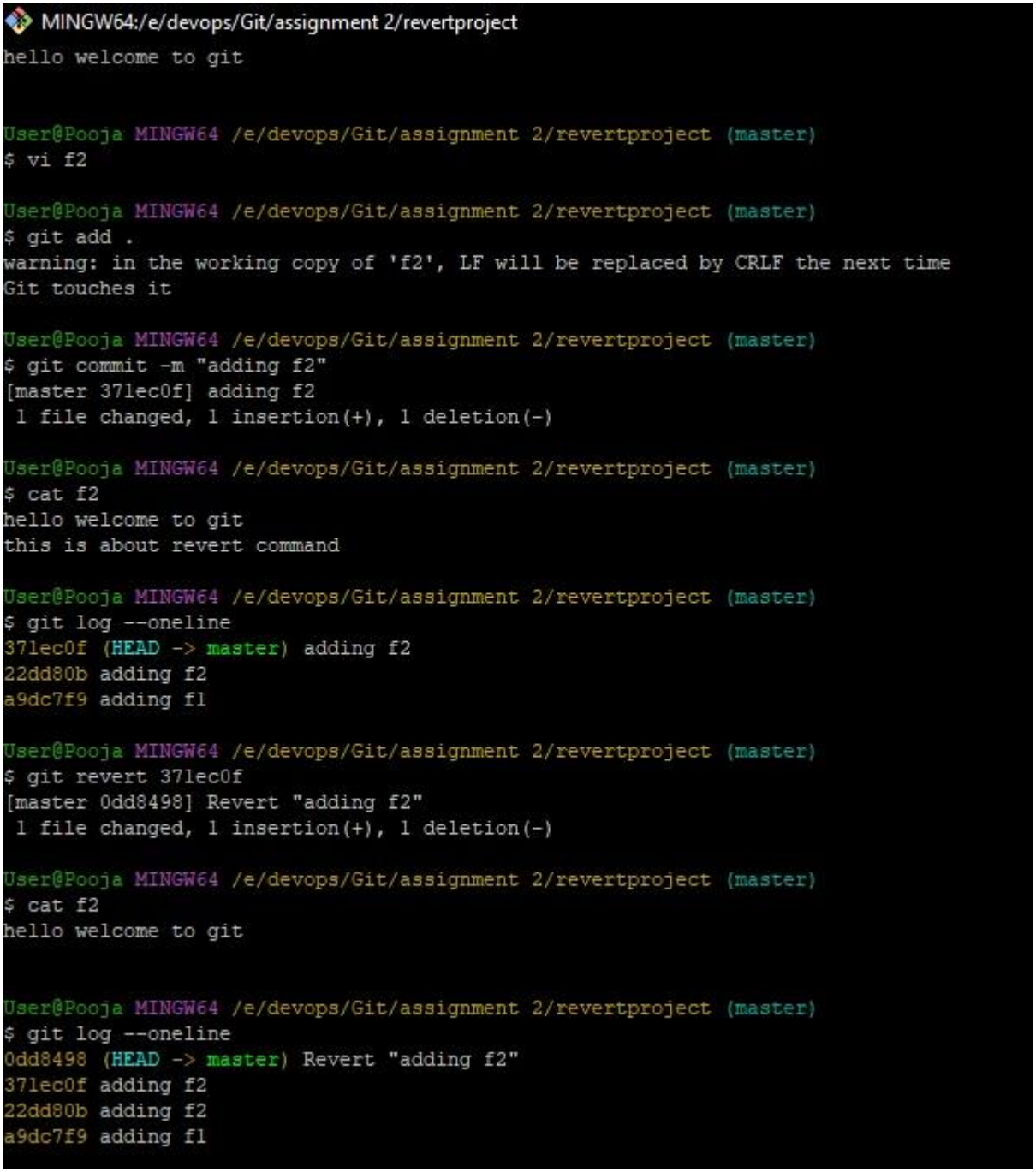
Here's a basic example of how to use git revert:

1. Find the commit hash of the commit you want to revert using git log.

2. Run git revert <commit_hash> to create a new commit that reverts the changes

Note: Git revert doesn't delete the original commit; it simply creates a new commit that reverses its changes. This allows you to maintain a record of all changes made to your repository.





.

Git Reset:

git reset is a command used in Git to reset your repository to a previous state by updating the index and working directory. It's used to undo changes, remove commits, or move the branch pointer to a different commit.

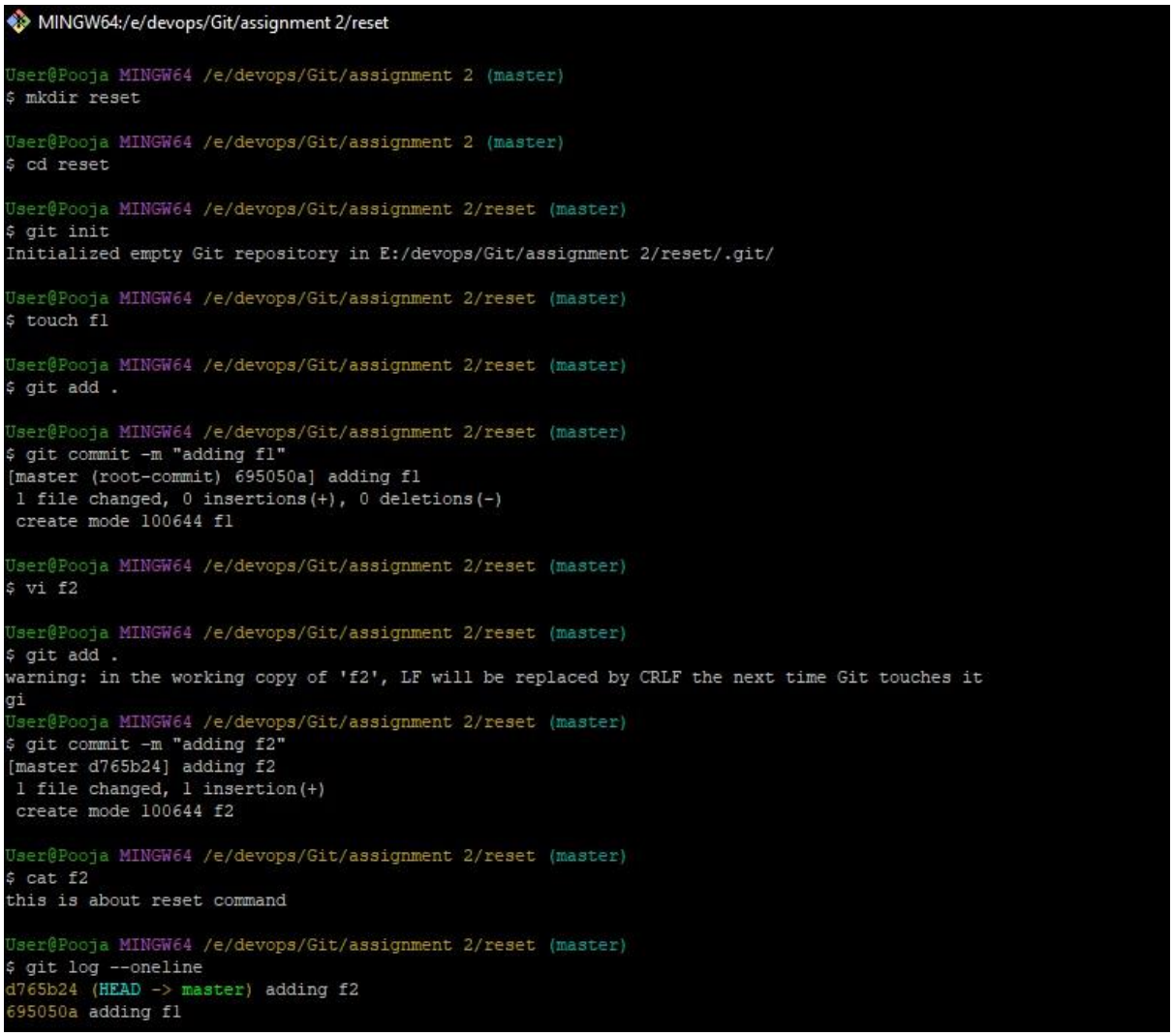
Here are some common uses of git reset:

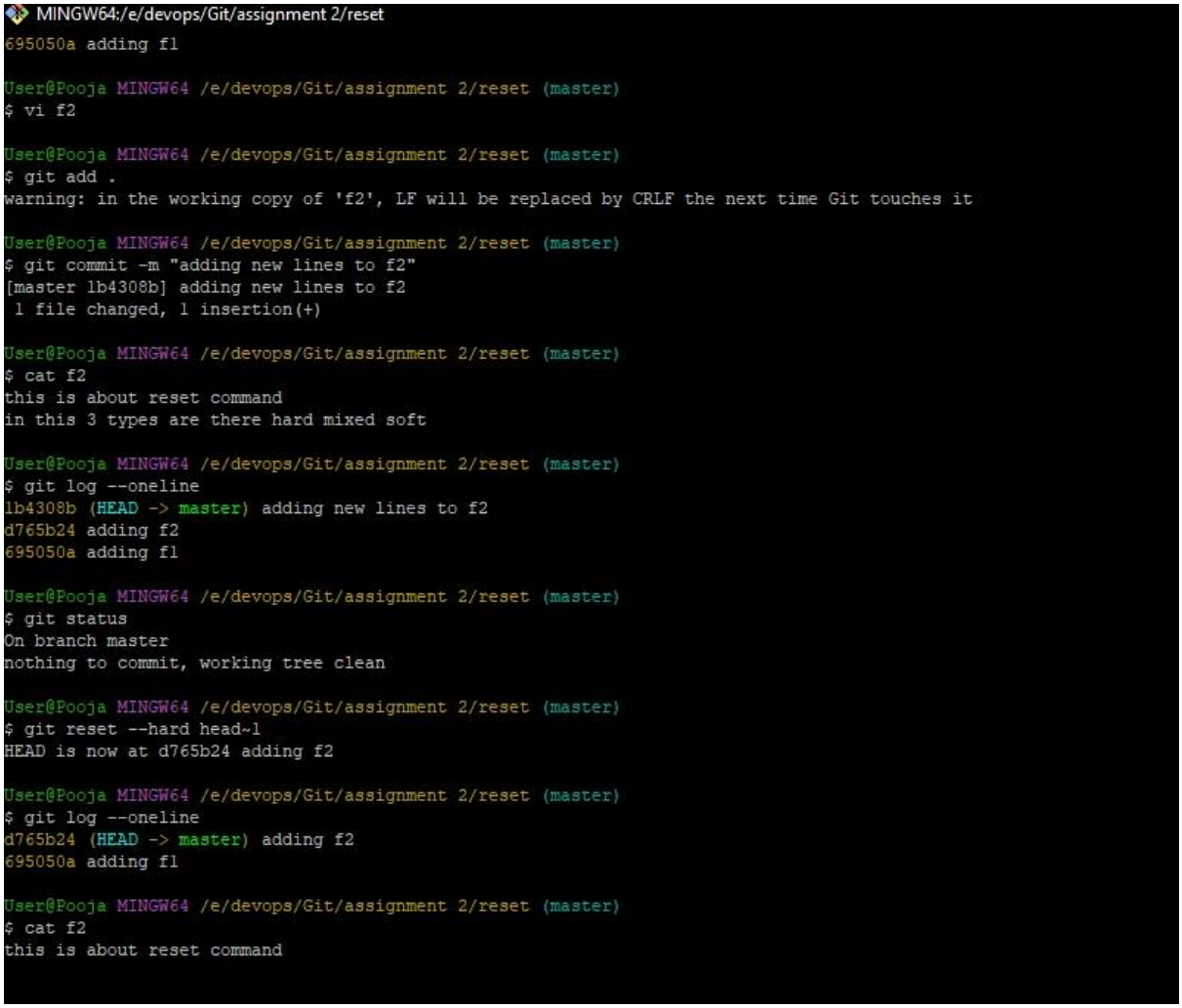
1. Un stage changes: git reset (or git reset --mixed) removes changes from the staging area, but leaves them in the working directory.

2. Discard changes: git reset --hard discards all changes in the working directory and staging area, resetting both to the last commit.

3. Move branch pointer: git reset --soft <commit> moves the branch pointer to a specific commit, without changing the index or working directory.

4. Remove commits: git reset --hard <commit> removes commits and resets the repository to a previous state.





Git Restore:

git restore is a Git command used to:

1. Discard changes: Revert changes made to a file or directory, restoring it to its state at the last commit.

2. Restore deleted files: Recover deleted files, bringing them back to the working directory.

Common use cases:

- Accidentally modified a file and want to revert to the original version.
- Deleted a file by mistake and want to recover it.
- Want to start fresh with a clean working directory.

Basic syntax:

- git restore <file>: Restore a single file.
- git restore .: Restore all changed files in the current directory.
- git restore --source=<commit> <file>: Restore a file from a specific commit.
- git rm –cached <file>: Unstage a file (remove it from the staging area).

Important notes:

- git restore is a safer alternative to git checkout for discarding changes, as it doesn't affect the branch or commit history.
- If you've already staged changes (with git add), you'll need to use git restore --staged to unstage them before discarding.

```
MINGW64:/e/devops/Git/assignment documentary/restore

User@Pooja MINGW64 /e/devops/Git/assignment documentary (master)
$ mkdir restore

User@Pooja MINGW64 /e/devops/Git/assignment documentary (master)
$ cd restore

User@Pooja MINGW64 /e/devops/Git/assignment documentary/restore (master)
$ git init
Initialized empty Git repository in E:/devops/Git/assignment documentary/restore
/.git/

User@Pooja MINGW64 /e/devops/Git/assignment documentary/restore (master)
$ touch f1

User@Pooja MINGW64 /e/devops/Git/assignment documentary/restore (master)
$ git add .

User@Pooja MINGW64 /e/devops/Git/assignment documentary/restore (master)
$ git status
On branch master

No commits yet

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)
        new file:   f1


User@Pooja MINGW64 /e/devops/Git/assignment documentary/restore (master)
$ git restore --staged f1
fatal: could not resolve HEAD

User@Pooja MINGW64 /e/devops/Git/assignment documentary/restore (master)
$ git status
On branch master

No commits yet

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)
        new file:   f1


User@Pooja MINGW64 /e/devops/Git/assignment documentary/restore (master)
$ rm f1
```

```
MINGW64:/e/devops/Git/assignment documentary/restore

User@Pooja MINGW64 /e/devops/Git/assignment documentary/restore (master)
$ git status
On branch master

No commits yet

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)
        new file:   fl


User@Pooja MINGW64 /e/devops/Git/assignment documentary/restore (master)
$ rm fl

User@Pooja MINGW64 /e/devops/Git/assignment documentary/restore (master)
$ ls

User@Pooja MINGW64 /e/devops/Git/assignment documentary/restore (master)
$ git log
fatal: your current branch 'master' does not have any commits yet

User@Pooja MINGW64 /e/devops/Git/assignment documentary/restore (master)
$ git restore fl

User@Pooja MINGW64 /e/devops/Git/assignment documentary/restore (master)
$ ls
fl

User@Pooja MINGW64 /e/devops/Git/assignment documentary/restore (master)
$ git log
fatal: your current branch 'master' does not have any commits yet

User@Pooja MINGW64 /e/devops/Git/assignment documentary/restore (master)
$ git status
On branch master

No commits yet

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)
        new file:   fl
```

```
User@Pooja MINGW64 /e/devops/Git/assignment documentary/restore (master)
$ git rm --cached fl
rm 'fl'

User@Pooja MINGW64 /e/devops/Git/assignment documentary/restore (master)
$ ls
fl

User@Pooja MINGW64 /e/devops/Git/assignment documentary/restore (master)
$ git status
On branch master

No commits yet

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

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

Git amend:

git commit --amend is a command used to modify the most recent commit. It allows you to:

1. Edit the commit message: Change the message of the last commit.

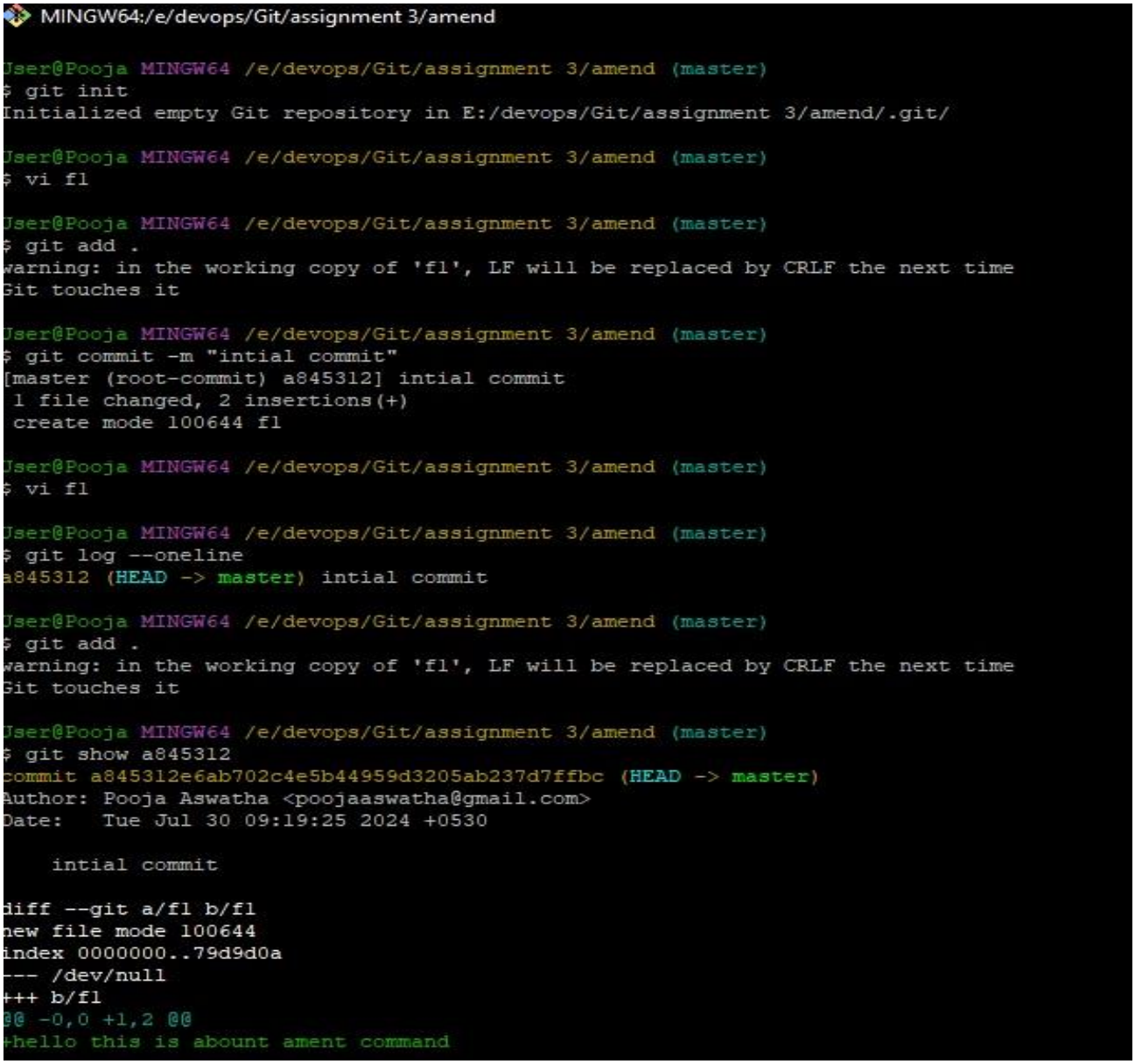
2. Add or remove files: Stage or unstage files to include or exclude them from the last commit.

When you run git commit --amend, Git:

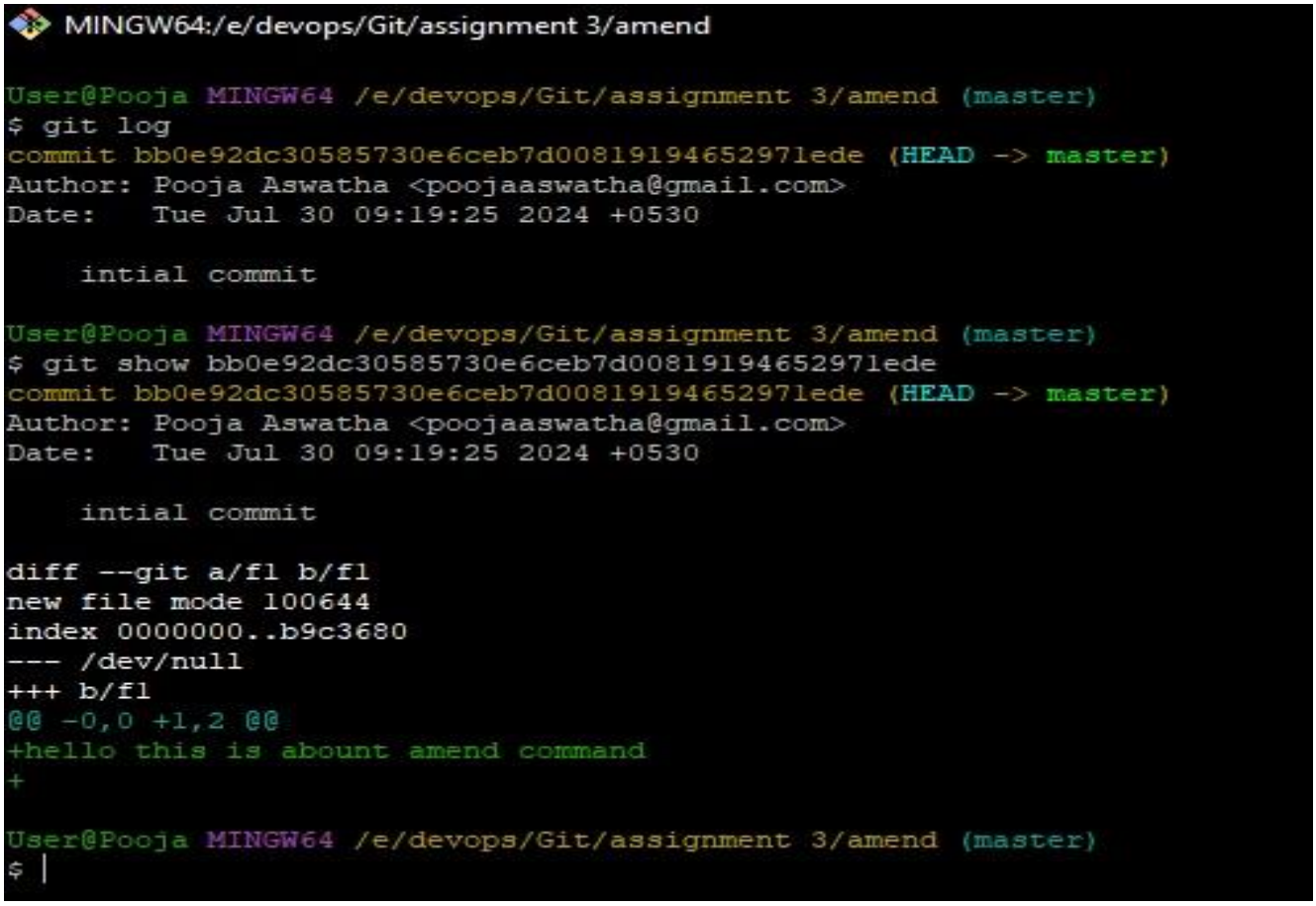
1. Creates a new commit with the updated changes.

2. Replaces the original commit with the new one.

3. Updates the branch pointer to point to the new commit.




MINGW64:/e/devops/Git/assignment 3/amend
+hello this is abount ament command
+
User@Pooja MINGW64 /e/devops/Git/assignment 3/amend (master)
$ git diff --cached
diff --git a/f1 b/f1
index 79d9d0a..b9c3680 100644
--- a/f1
+++ b/f1
@@ -1,2 +1,2 @@
-hello this is abount ament command
+hello this is abount amend command
User@Pooja MINGW64 /e/devops/Git/assignment 3/amend (master)
$ git commit --amend
[master bb0e92d] intial commit
Date: Tue Jul 30 09:19:25 2024 +0530
1 file changed, 2 insertions(+)
create mode 100644 f1
User@Pooja MINGW64 /e/devops/Git/assignment 3/amend (master)
$ git log --oneline
bb0e92d (HEAD -> master) intial commit
User@Pooja MINGW64 /e/devops/Git/assignment 3/amend (master)
$ git status
On branch master
nothing to commit, working tree clean
User@Pooja MINGW64 /e/devops/Git/assignment 3/amend (master)
$ git log
commit bb0e92dc30585730e6ceb7d00819194652971ede (HEAD -> master)
Author: Pooja Aswatha <poojaaswatha@gmail.com>
Date: Tue Jul 30 09:19:25 2024 +0530
intial commit
User@Pooja MINGW64 /e/devops/Git/assignment 3/amend (master)
$ git show bb0e92dc30585730e6ceb7d00819194652971ede
commit bb0e92dc30585730e6ceb7d00819194652971ede (HEAD -> master)
Author: Pooja Aswatha <poojaaswatha@gmail.com>
Date: Tue Jul 30 09:19:25 2024 +0530
intial commit



.gitignore:

.gitignore is a special file in Git repositories that tells Git which files or directories to ignore and not track. It's a way to exclude certain files or patterns from being committed to the repository.

Common uses for .gitignore:

1. Exclude sensitive information: Ignore files containing passwords, API keys, or other sensitive data.

2. Ignore build artifacts: Exclude compiled files, logs, or other generated files that don't need to be version-controlled.

3. Exclude operating system files: Ignore files created by your operating system, like .DS_Store or Thumbs.db.

4. Ignore editor or IDE settings: Exclude configuration files specific to your editor or IDE.

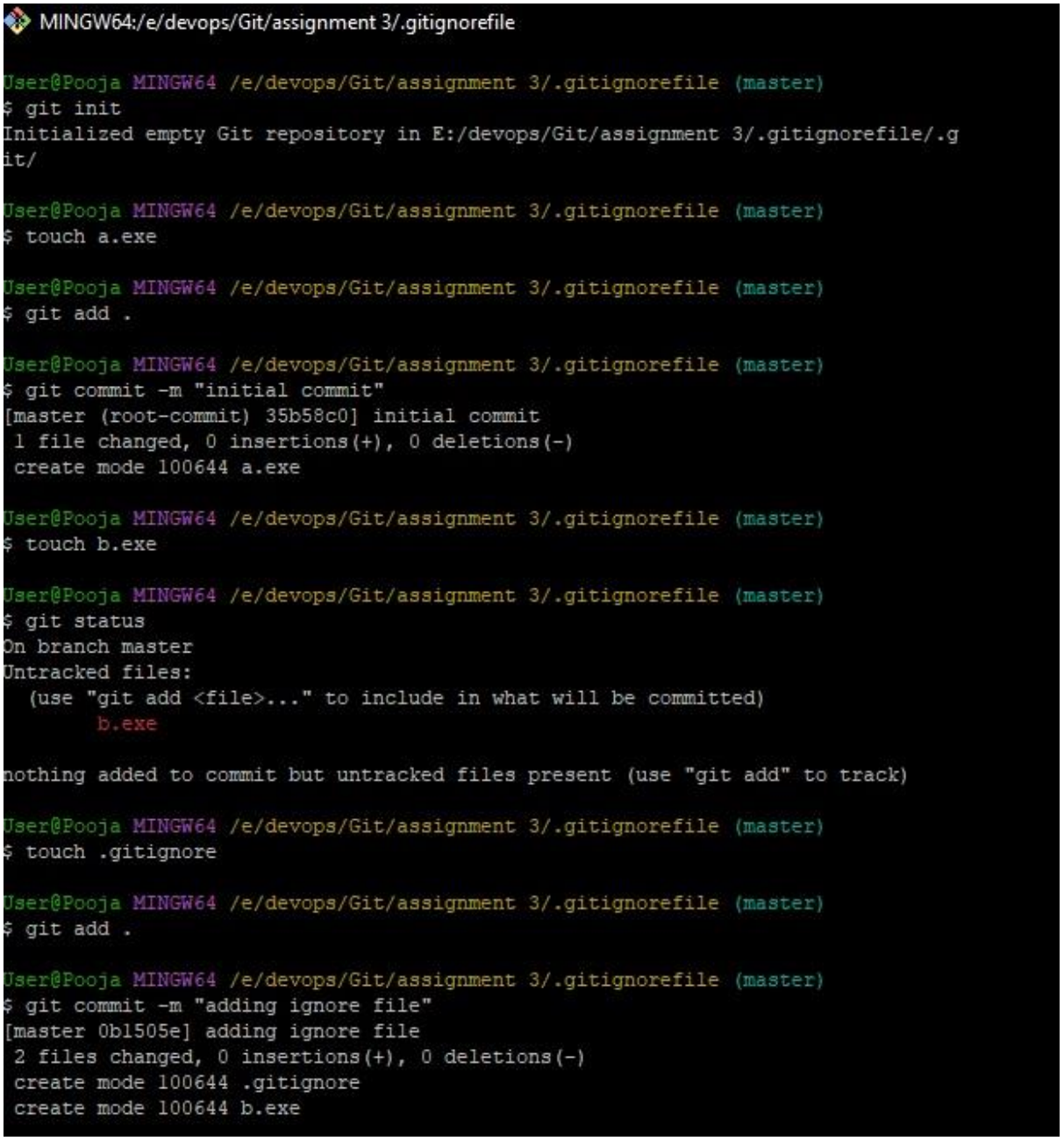
How to use .gitignore:

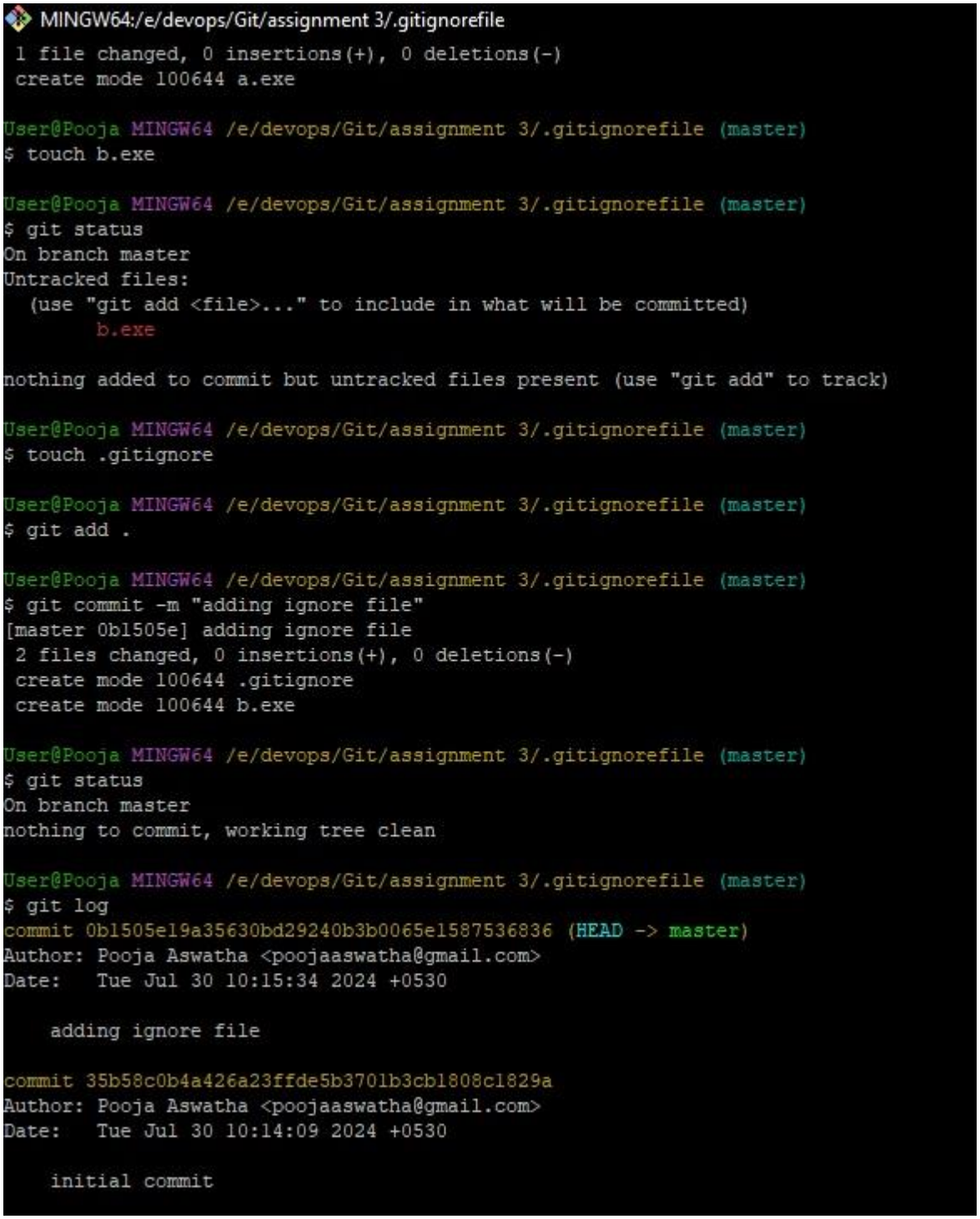
1. Create a .gitignore file in the root of your repository.

2. Add patterns or file names to ignore, one per line.

3. Use wildcards (*) to match multiple files or directories.

4. Use ! to negate a pattern and include a file or directory.


MINGW64:/e/devops/Git/assignment 3/.gitignorefile
User@Pooja MINGW64 /e/devops/Git/assignment 3/.gitignorefile (master)
$ git init
Initialized empty Git repository in E:/devops/Git/assignment 3/.gitignorefile/.git/
User@Pooja MINGW64 /e/devops/Git/assignment 3/.gitignorefile (master)
$ touch a.exe
User@Pooja MINGW64 /e/devops/Git/assignment 3/.gitignorefile (master)
$ git add .
User@Pooja MINGW64 /e/devops/Git/assignment 3/.gitignorefile (master)
$ git commit -m "initial commit"
[master (root-commit) 35b58c0] initial commit
1 file changed, 0 insertions(+), 0 deletions(-)
create mode 100644 a.exe
User@Pooja MINGW64 /e/devops/Git/assignment 3/.gitignorefile (master)
$ touch b.exe
User@Pooja MINGW64 /e/devops/Git/assignment 3/.gitignorefile (master)
$ git status
On branch master
Untracked files:
(use "git add <file>..." to include in what will be committed)
b.exe
nothing added to commit but untracked files present (use "git add" to track)
User@Pooja MINGW64 /e/devops/Git/assignment 3/.gitignorefile (master)
$ touch .gitignore
User@Pooja MINGW64 /e/devops/Git/assignment 3/.gitignorefile (master)
$ git add .
User@Pooja MINGW64 /e/devops/Git/assignment 3/.gitignorefile (master)
$ git commit -m "adding ignore file"
[master 0b1505e] adding ignore file
2 files changed, 0 insertions(+), 0 deletions(-)
create mode 100644 .gitignore
create mode 100644 b.exe


MINGW64:/e/devops/Git/assignment 3/.gitignorefile
1 file changed, 0 insertions(+), 0 deletions(-)
create mode 100644 a.exe
User@Pooja MINGW64 /e/devops/Git/assignment 3/.gitignorefile (master)
$ touch b.exe
User@Pooja MINGW64 /e/devops/Git/assignment 3/.gitignorefile (master)
$ git status
On branch master
Untracked files:
(use "git add <file>..." to include in what will be committed)
b.exe
nothing added to commit but untracked files present (use "git add" to track)
User@Pooja MINGW64 /e/devops/Git/assignment 3/.gitignorefile (master)
$ touch .gitignore
User@Pooja MINGW64 /e/devops/Git/assignment 3/.gitignorefile (master)
$ git add .
User@Pooja MINGW64 /e/devops/Git/assignment 3/.gitignorefile (master)
$ git commit -m "adding ignore file"
[master 0b1505e] adding ignore file
2 files changed, 0 insertions(+), 0 deletions(-)
create mode 100644 .gitignore
create mode 100644 b.exe
User@Pooja MINGW64 /e/devops/Git/assignment 3/.gitignorefile (master)
$ git status
On branch master
nothing to commit, working tree clean
User@Pooja MINGW64 /e/devops/Git/assignment 3/.gitignorefile (master)
$ git log
commit 0b1505e19a35630bd29240b3b0065e1587536836 (HEAD -> master)
Author: Pooja Aswatha <poojaaswatha@gmail.com>
Date: Tue Jul 30 10:15:34 2024 +0530
adding ignore file
commit 35b58c0b4a426a23ffde5b3701b3cb1808c1829a
Author: Pooja Aswatha <poojaaswatha@gmail.com>
Date: Tue Jul 30 10:14:09 2024 +0530
initial commit

Git Diff:

Git diff is a command used to show changes between commits, branches, or files in a Git repository. It displays the differences in a human-readable format, allowing you to review and understand the changes made.

Common uses of git diff:

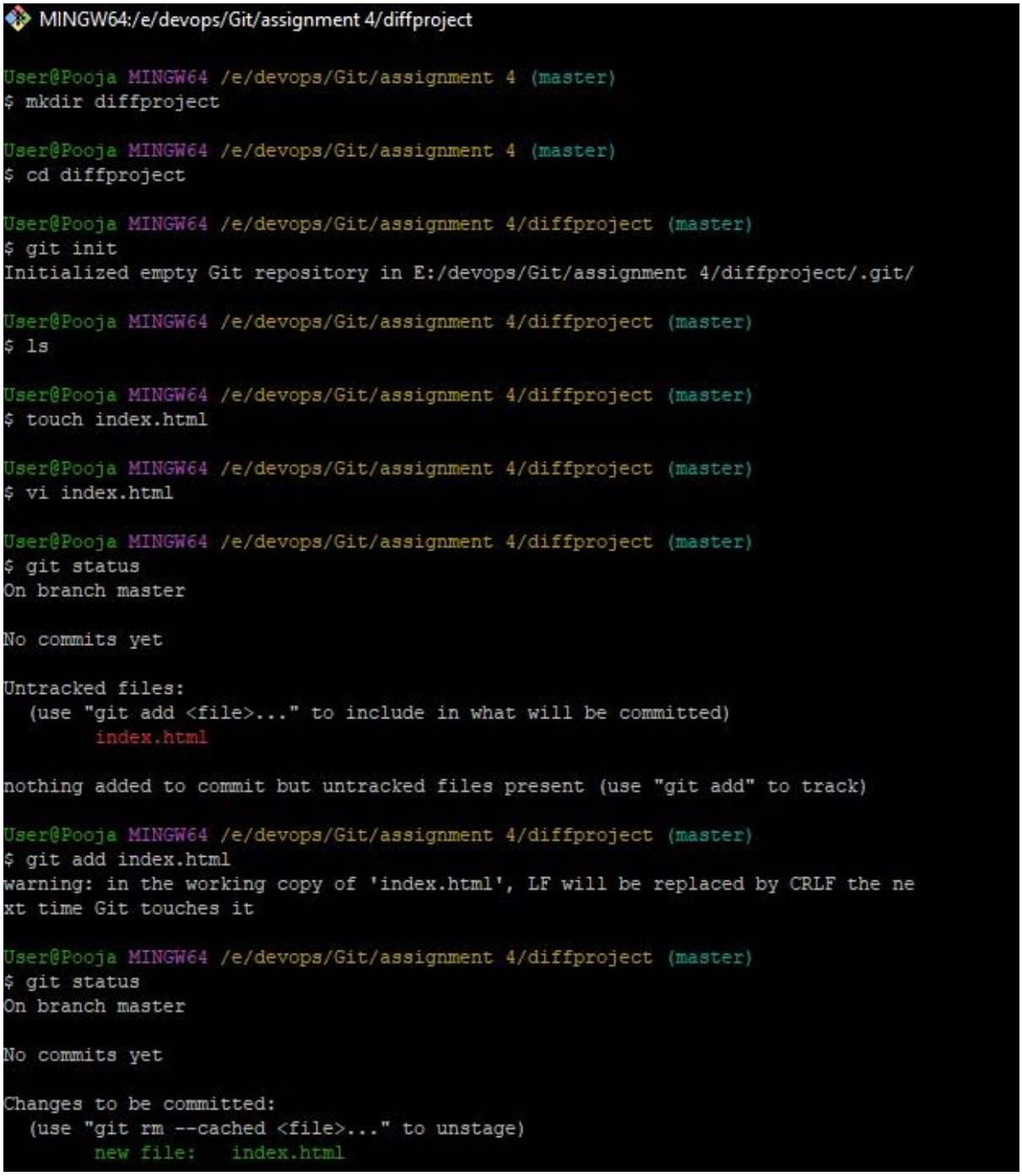
1. Compare changes between commits: git diff <commit1> <commit2> shows changes between two specific commits.

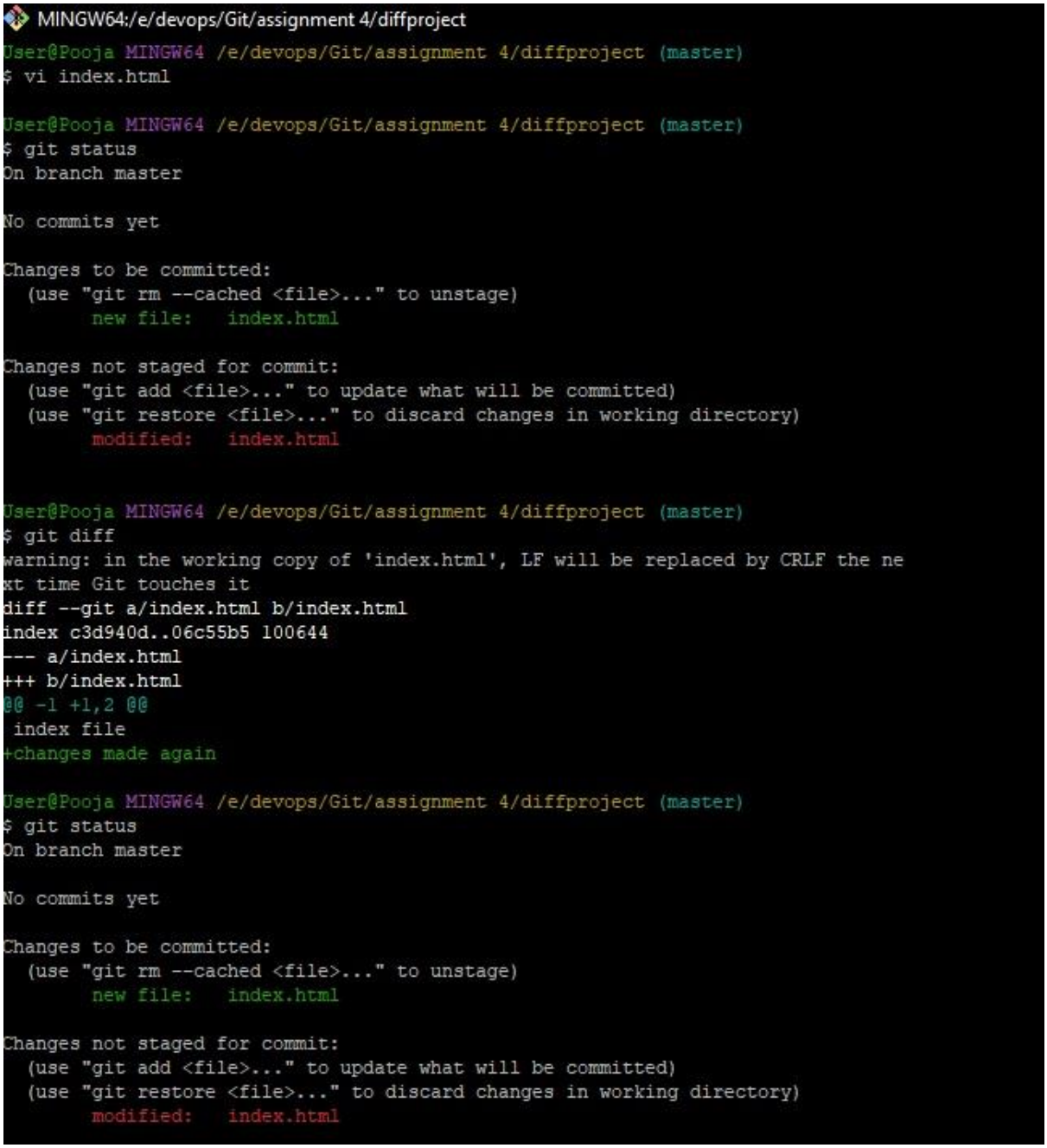
2. Compare changes between branches: git diff <branch1> <branch2> shows changes between two branches.

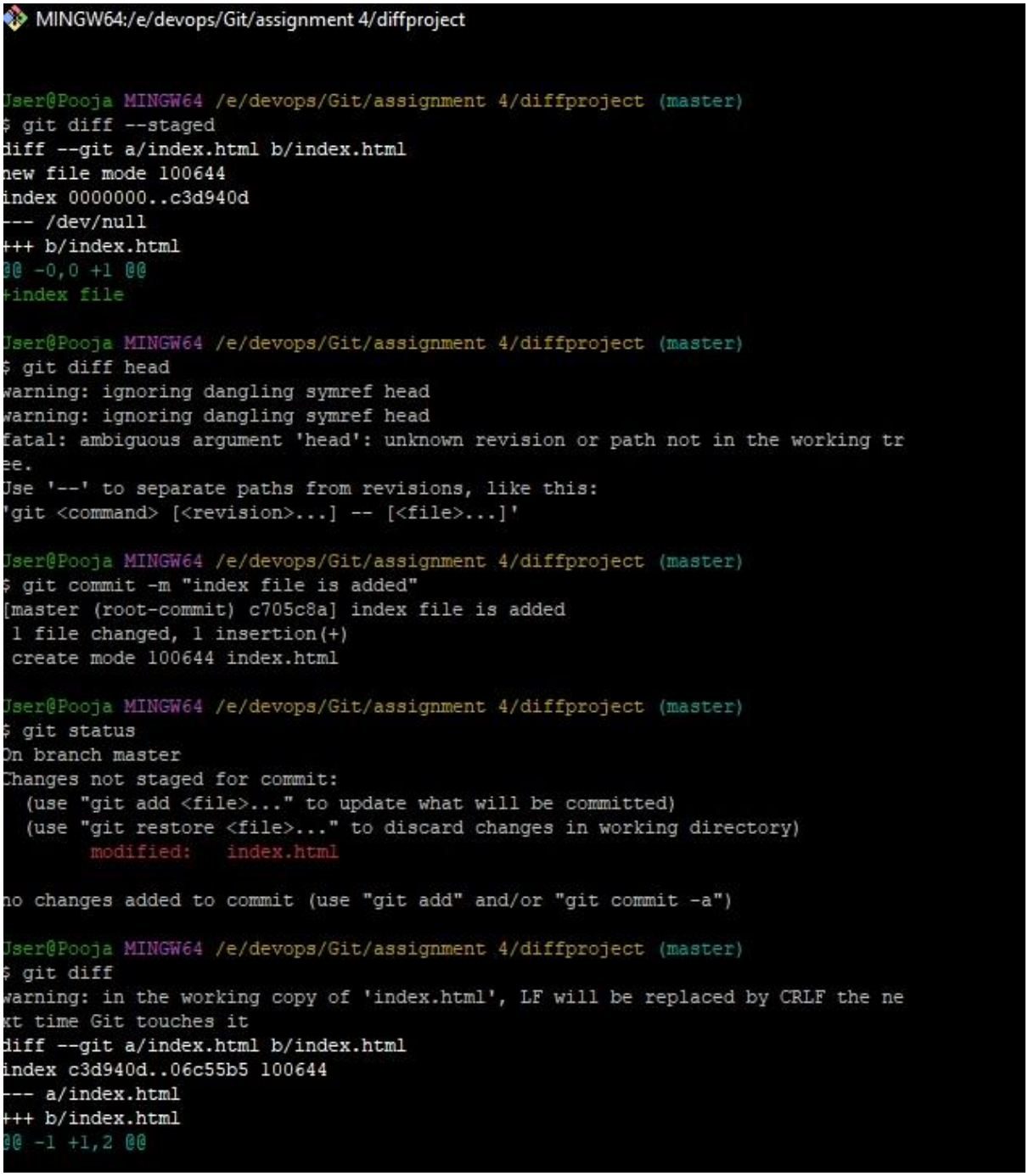
3. Compare changes in a file: git diff <file> shows changes made to a specific file.

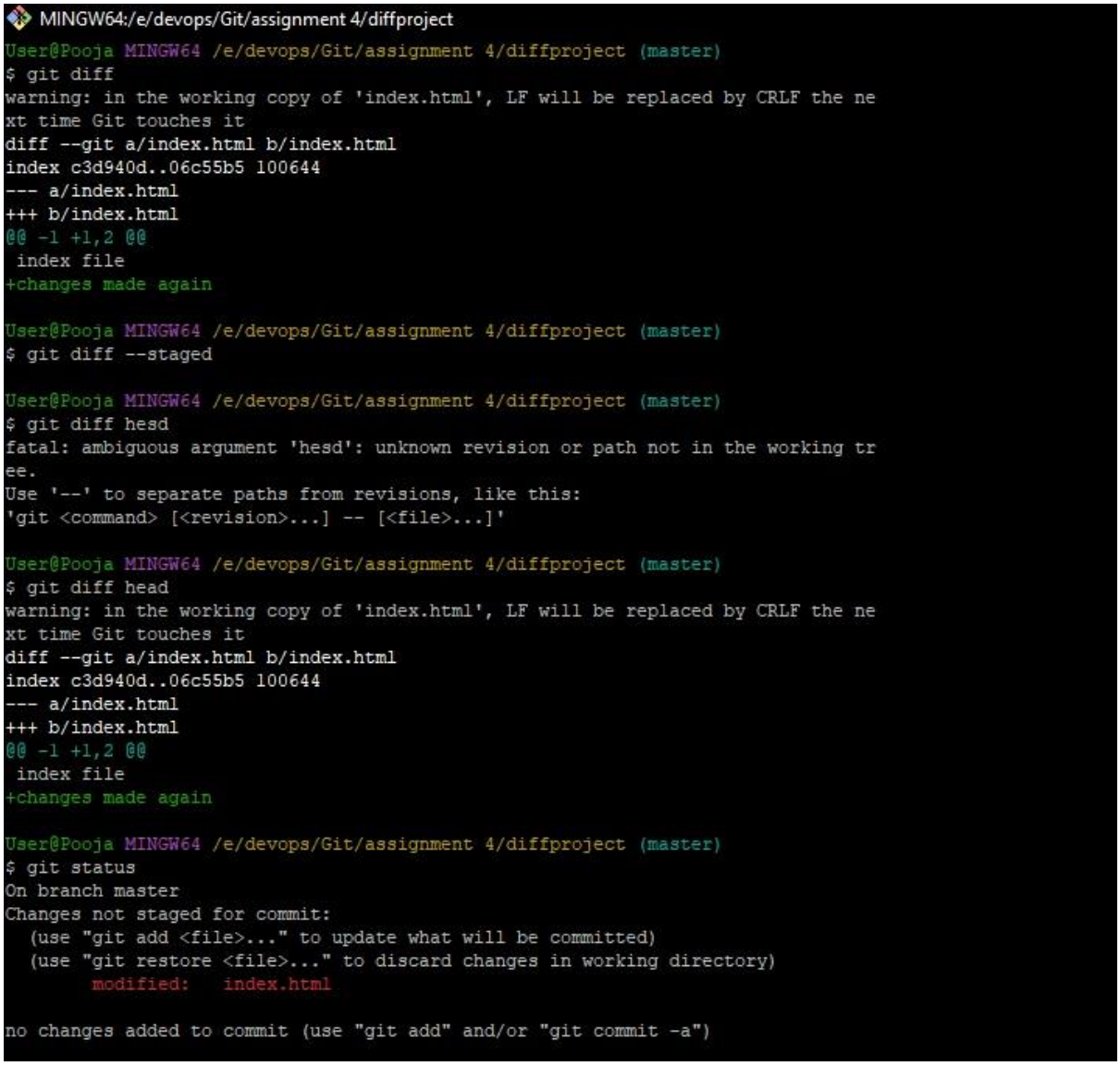
4. Compare staged and unstaged changes: git diff (without arguments) shows changes between the staging area and the working directory.

5. Compare changes since last commit: git diff HEAD shows changes made since the last commit.


MINGW64:/e/devops/Git/assignment 4/diffproject
User@Pooja MINGW64 /e/devops/Git/assignment 4 (master)
$ mkdir diffproject
User@Pooja MINGW64 /e/devops/Git/assignment 4 (master)
$ cd diffproject
User@Pooja MINGW64 /e/devops/Git/assignment 4/diffproject (master)
$ git init
Initialized empty Git repository in E:/devops/Git/assignment 4/diffproject/.git/
User@Pooja MINGW64 /e/devops/Git/assignment 4/diffproject (master)
$ ls
User@Pooja MINGW64 /e/devops/Git/assignment 4/diffproject (master)
$ touch index.html
User@Pooja MINGW64 /e/devops/Git/assignment 4/diffproject (master)
$ vi index.html
User@Pooja MINGW64 /e/devops/Git/assignment 4/diffproject (master)
$ git status
On branch master
No commits yet
Untracked files:
(use "git add <file>..." to include in what will be committed)
index.html
nothing added to commit but untracked files present (use "git add" to track)
User@Pooja MINGW64 /e/devops/Git/assignment 4/diffproject (master)
$ git add index.html
warning: in the working copy of 'index.html', LF will be replaced by CRLF the ne
xt time Git touches it
User@Pooja MINGW64 /e/devops/Git/assignment 4/diffproject (master)
$ git status
On branch master
No commits yet
Changes to be committed:
(use "git rm --cached <file>..." to unstage)
new file: index.html


MINGW64:/e/devops/Git/assignment 4/diffproject
User@Pooja MINGW64 /e/devops/Git/assignment 4/diffproject (master)
$ vi index.html
User@Pooja MINGW64 /e/devops/Git/assignment 4/diffproject (master)
$ git status
On branch master
No commits yet
Changes to be committed:
(use "git rm --cached <file>..." to unstage)
new file: index.html
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: index.html
User@Pooja MINGW64 /e/devops/Git/assignment 4/diffproject (master)
$ git diff
warning: in the working copy of 'index.html', LF will be replaced by CRLF the ne
xt time Git touches it
diff --git a/index.html b/index.html
index c3d940d..06c55b5 100644
--- a/index.html
+++ b/index.html
@@ -1 +1,2 @@
index file
+changes made again
User@Pooja MINGW64 /e/devops/Git/assignment 4/diffproject (master)
$ git status
On branch master
No commits yet
Changes to be committed:
(use "git rm --cached <file>..." to unstage)
new file: index.html
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: index.html


MINGW64:/e/devops/Git/assignment 4/diffproject
User@Pooja MINGW64 /e/devops/Git/assignment 4/diffproject (master)
$ git diff --staged
diff --git a/index.html b/index.html
new file mode 100644
index 0000000..c3d940d
--- /dev/null
+++ b/index.html
@@ -0,0 +1 @@
+index file
User@Pooja MINGW64 /e/devops/Git/assignment 4/diffproject (master)
$ git diff head
warning: ignoring dangling symref head
warning: ignoring dangling symref head
fatal: ambiguous argument 'head': unknown revision or path not in the working tr
ee.
Use '--' to separate paths from revisions, like this:
'git <command> [<revision>...] -- [<file>...]'
User@Pooja MINGW64 /e/devops/Git/assignment 4/diffproject (master)
$ git commit -m "index file is added"
[master (root-commit) c705c8a] index file is added
1 file changed, 1 insertion(+)
create mode 100644 index.html
User@Pooja MINGW64 /e/devops/Git/assignment 4/diffproject (master)
$ git status
On branch master
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: index.html
no changes added to commit (use "git add" and/or "git commit -a")
User@Pooja MINGW64 /e/devops/Git/assignment 4/diffproject (master)
$ git diff
warning: in the working copy of 'index.html', LF will be replaced by CRLF the ne
xt time Git touches it
diff --git a/index.html b/index.html
index c3d940d..06c55b5 100644
--- a/index.html
+++ b/index.html
@@ -1 +1,2 @@



Git Bisect:

Git bisect is a powerful command used to find the commit that introduced a bug or issue in your code. It uses a binary search algorithm to narrow down the possible commits until it finds the culprit.

Here's how to use git bisect:

1. Start the bisect process: git bisect start

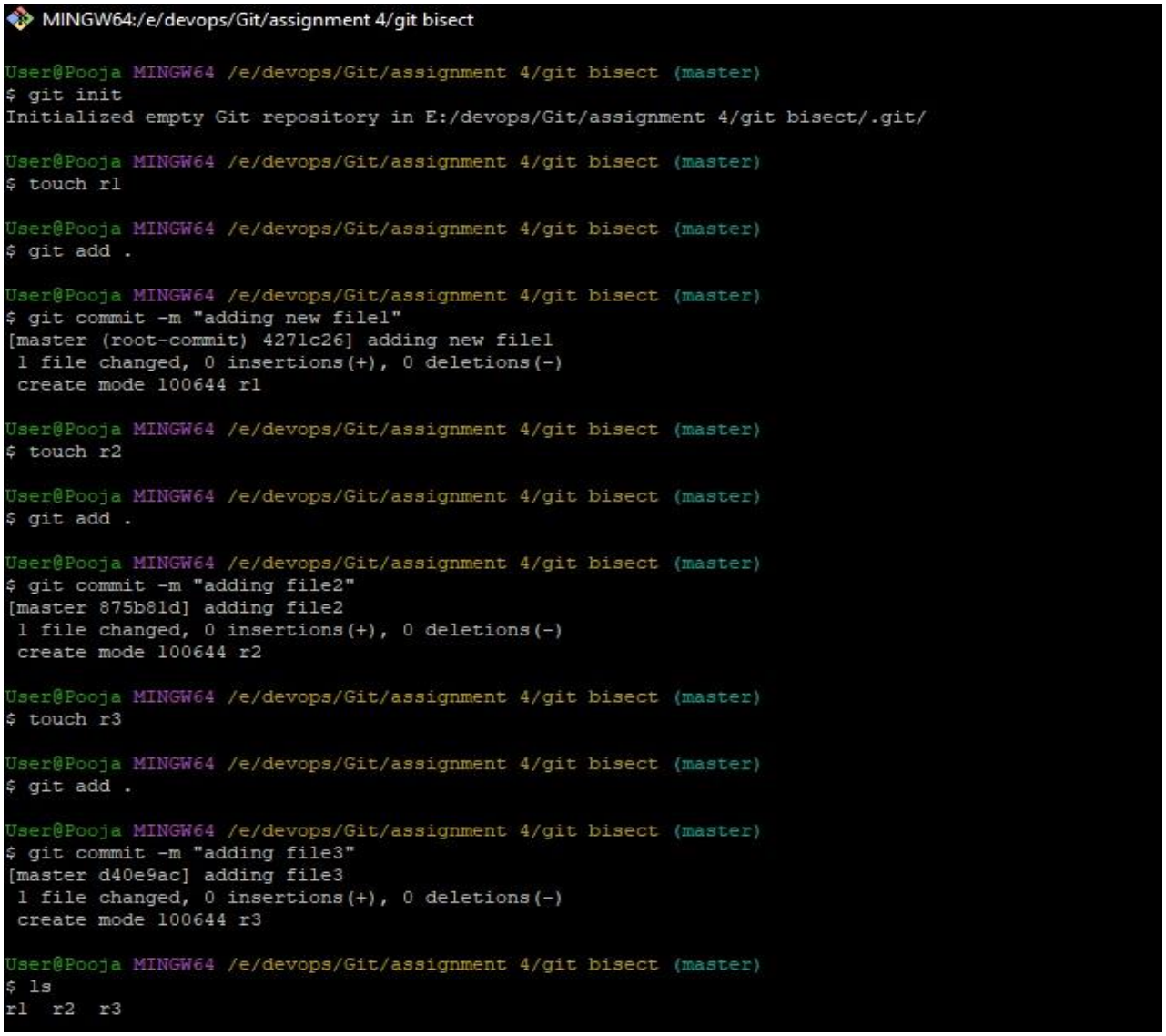
2. Mark the current commit as bad: git bisect bad

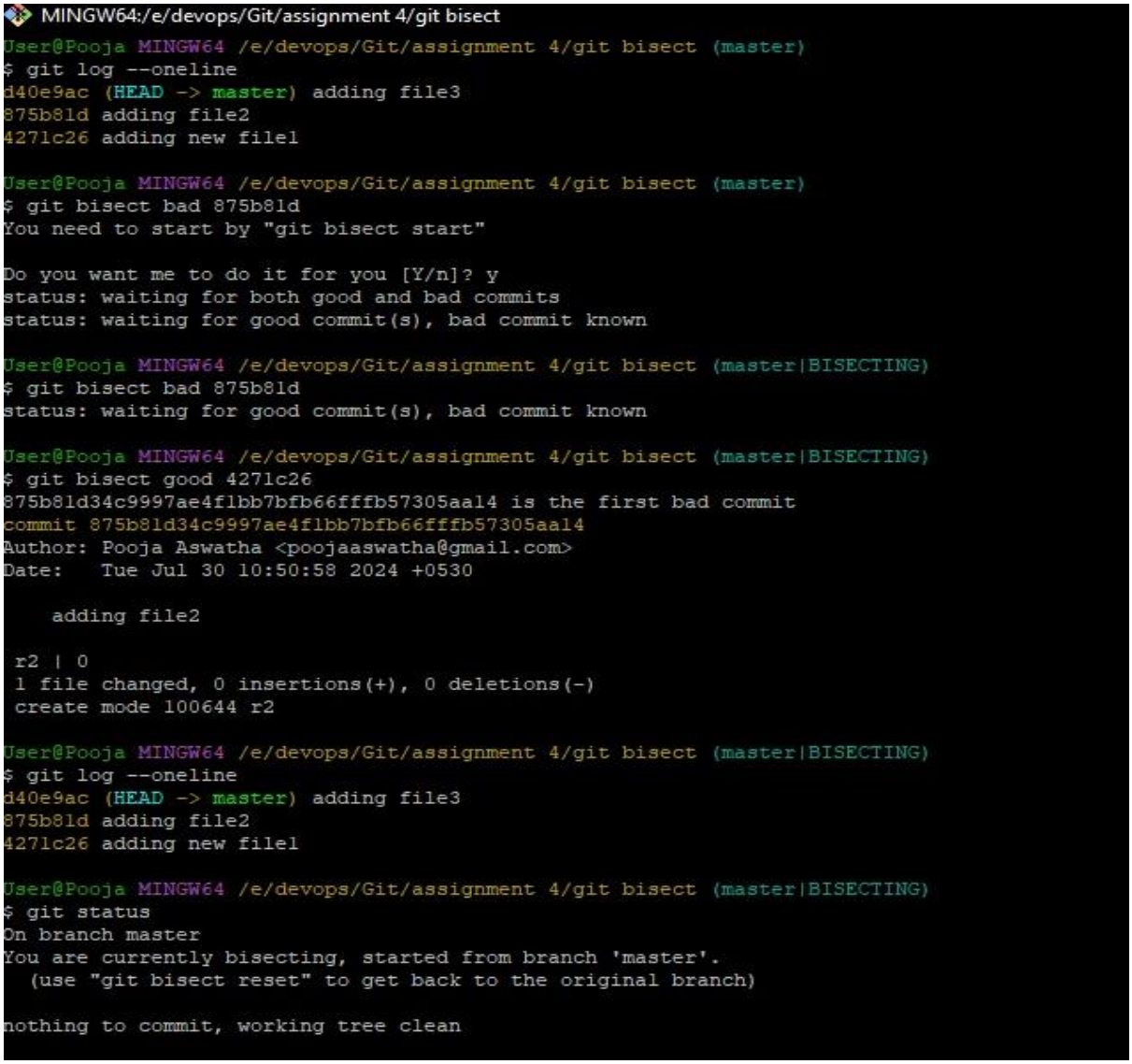
3. Mark a known good commit: git bisect good <commit>

4. Git will checkout a commit in the middle of the range.

5. Test the code and mark it as good or bad: git bisect good or git bisect bad

6. Repeat steps 4-5 until Git finds the bad commit.


MINGW64:/e/devops/Git/assignment 4/git bisect
User@Pooja MINGW64 /e/devops/Git/assignment 4/git bisect (master)
$ git init
Initialized empty Git repository in E:/devops/Git/assignment 4/git bisect/.git/
User@Pooja MINGW64 /e/devops/Git/assignment 4/git bisect (master)
$ touch r1
User@Pooja MINGW64 /e/devops/Git/assignment 4/git bisect (master)
$ git add .
User@Pooja MINGW64 /e/devops/Git/assignment 4/git bisect (master)
$ git commit -m "adding new file1"
[master (root-commit) 4271c26] adding new file1
1 file changed, 0 insertions(+), 0 deletions(-)
create mode 100644 r1
User@Pooja MINGW64 /e/devops/Git/assignment 4/git bisect (master)
$ touch r2
User@Pooja MINGW64 /e/devops/Git/assignment 4/git bisect (master)
$ git add .
User@Pooja MINGW64 /e/devops/Git/assignment 4/git bisect (master)
$ git commit -m "adding file2"
[master 875b81d] adding file2
1 file changed, 0 insertions(+), 0 deletions(-)
create mode 100644 r2
User@Pooja MINGW64 /e/devops/Git/assignment 4/git bisect (master)
$ touch r3
User@Pooja MINGW64 /e/devops/Git/assignment 4/git bisect (master)
$ git add .
User@Pooja MINGW64 /e/devops/Git/assignment 4/git bisect (master)
$ git commit -m "adding file3"
[master d40e9ac] adding file3
1 file changed, 0 insertions(+), 0 deletions(-)
create mode 100644 r3
User@Pooja MINGW64 /e/devops/Git/assignment 4/git bisect (master)
$ ls
r1 r2 r3



Git insta web:

git insta web is a command that allows you to instantly visualize your Git repository as a web-based Git repository browser. It's a quick way to explore your repository's history, commits, and files without leaving the command line.

Here's how to use git insta web:

1. Run git instaweb in your terminal.

2. Open a web browser and navigate to http://localhost:1234 (or the port number specified by Git).

3. You'll see a web-based interface showing your repository's history, commits, and files.

Git insta web uses the git web CGI script to generate the web interface. It's a simple and convenient way to:

- Explore your repository's history and commits.

- View file contents and changes.
- See commit messages and author information.

Amazon EMR:

Amazon EMR (Elastic MapReduce) is a cloud-based big data processing service offered by Amazon Web Services (AWS). It allows users to easily process and analyze large amounts of data using popular open-source frameworks like Apache Hadoop, Apache Spark, and Presto.

Key Features:

1. Scalability: Scale up or down to match your workload, without worrying about infrastructure management.

2. Flexibility: Choose from various processing frameworks, including Hadoop, Spark, and Presto.

3. Security: Leverage AWS security features, such as encryption and access controls.

4. Cost-effective: Pay only for the resources used, with options for spot pricing and reserved instances.

5. Integration: Seamlessly integrate with other AWS services, like S3, Glue, and Redshift.

Use cases:

1. Data processing: Process large datasets for analytics, machine learning, and data science workloads.

2. Data transformation: Transform and prepare data for analysis, reporting, or loading into data warehouses.

3. Machine learning: Train and deploy machine learning models using popular frameworks like TensorFlow and Scikit-learn.

4. Data warehousing: Use EMR to extract, transform, and load data into Amazon Redshift or other data warehouses.

Benefits:

1. Faster insights: Quickly process and analyze large datasets to gain insights and make data-driven decisions.

2. Increased productivity: Focus on data analysis and processing, without managing infrastructure.

3. Cost savings: Reduce costs by paying only for resources used and leveraging spot pricing.

Common applications:

1. Log analysis: Process and analyze log data for application monitoring and troubleshooting.

2. Clickstream analysis: Analyze user behavior and clickstream data for e-commerce and web applications.

3. Genomic analysis: Process and analyze large genomic datasets for research and healthcare applications.

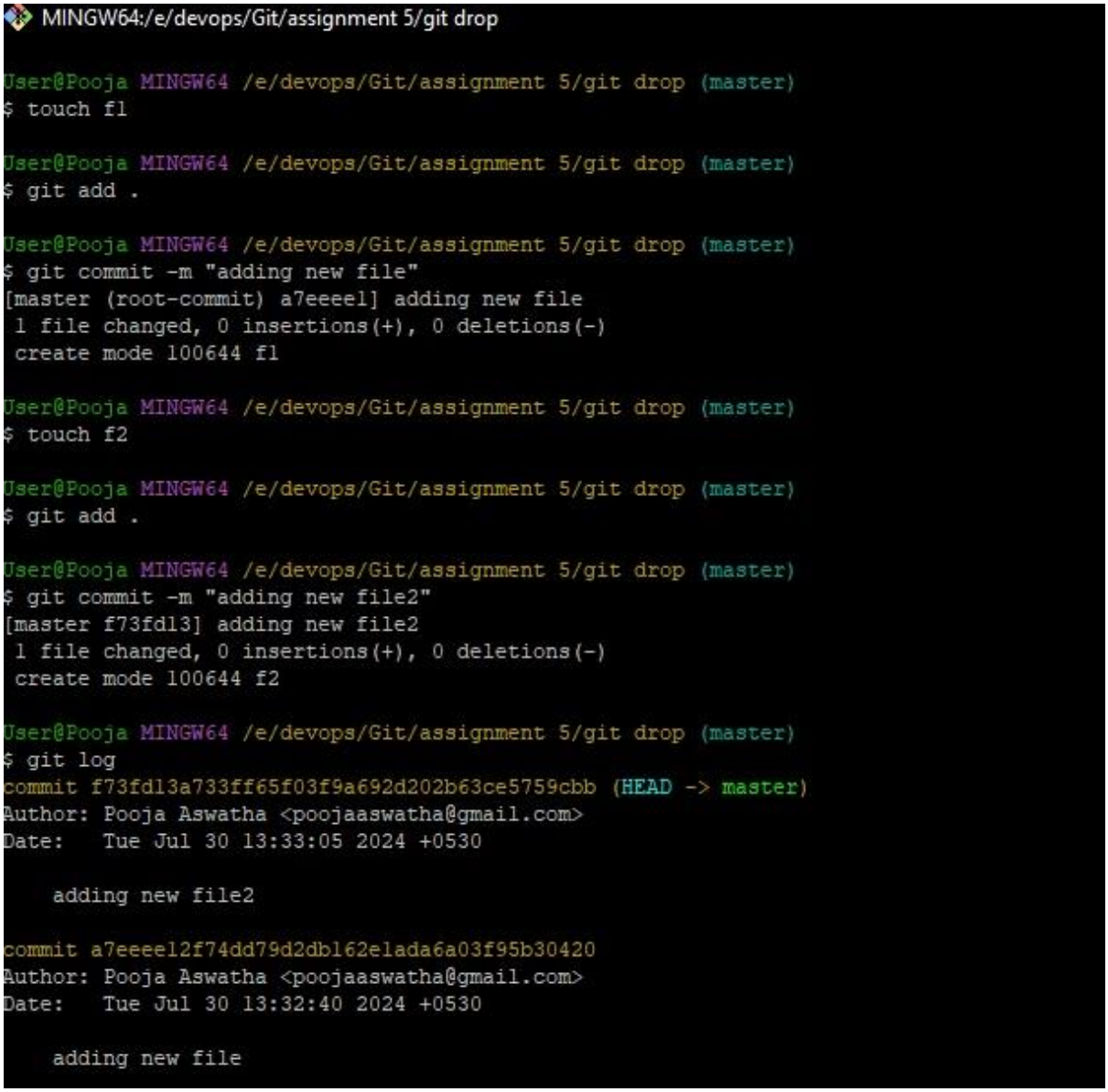
4. Financial analysis: Analyze financial data for risk management, portfolio optimization, and compliance.

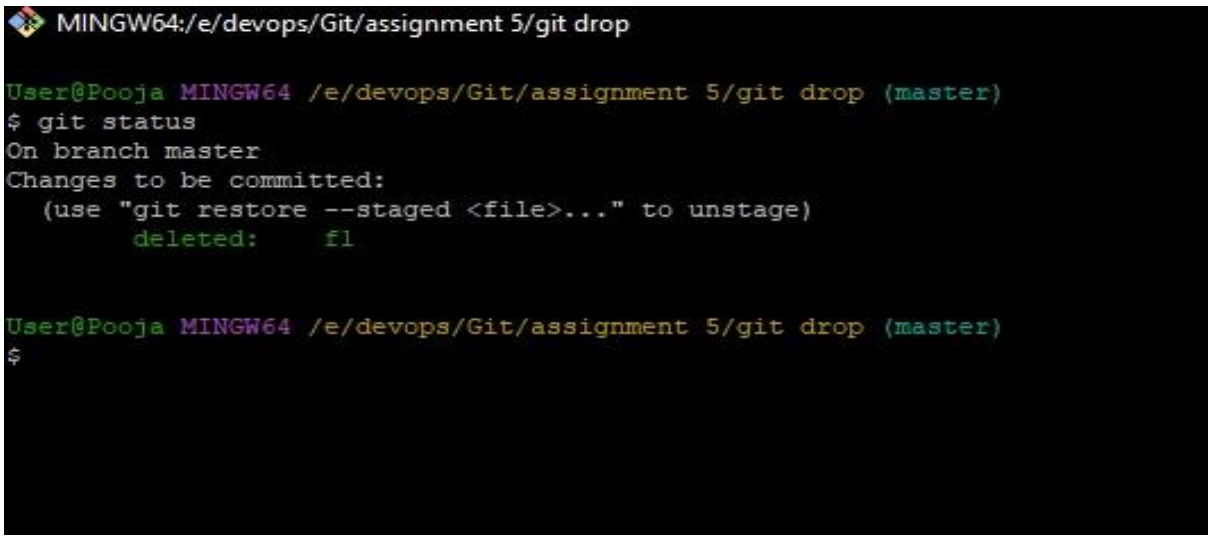
Git drop:

git drop is not a built-in Git command. However, you can use git reset or git restore to achieve similar results.

- git reset: Resets the current branch to a specific commit, discarding changes.
- git restore: Restores files to a previous state, discarding changes.

If you want to "drop" a commit, you can use git reset --hard to reset the branch to the previous commit, effectively deleting the most recent commit.


MINGW64:/e/devops/Git/assignment 5/git drop
User@Pooja MINGW64 /e/devops/Git/assignment 5/git drop (master)
$ touch f1
User@Pooja MINGW64 /e/devops/Git/assignment 5/git drop (master)
$ git add .
User@Pooja MINGW64 /e/devops/Git/assignment 5/git drop (master)
$ git commit -m "adding new file"
[master (root-commit) a7eeeel] adding new file
1 file changed, 0 insertions(+), 0 deletions(-)
create mode 100644 f1
User@Pooja MINGW64 /e/devops/Git/assignment 5/git drop (master)
$ touch f2
User@Pooja MINGW64 /e/devops/Git/assignment 5/git drop (master)
$ git add .
User@Pooja MINGW64 /e/devops/Git/assignment 5/git drop (master)
$ git commit -m "adding new file2"
[master f73fdl3] adding new file2
1 file changed, 0 insertions(+), 0 deletions(-)
create mode 100644 f2
User@Pooja MINGW64 /e/devops/Git/assignment 5/git drop (master)
$ git log
commit f73fdl3a733ff65f03f9a692d202b63ce5759cbb (HEAD -> master)
Author: Pooja Aswatha <poojaaswatha@gmail.com>
Date: Tue Jul 30 13:33:05 2024 +0530
adding new file2
commit a7eeee12f74dd79d2db162elada6a03f95b30420
Author: Pooja Aswatha <poojaaswatha@gmail.com>
Date: Tue Jul 30 13:32:40 2024 +0530
adding new file


MINGW64:/e/devops/Git/assignment 5/git drop
User@Pooja MINGW64 /e/devops/Git/assignment 5/git drop (master)
$ git status
On branch master
Changes to be committed:
(use "git restore --staged <file>..." to unstage)
deleted: f1
User@Pooja MINGW64 /e/devops/Git/assignment 5/git drop (master)
$

Git Rebase:

git rebase is a Git command that allows you to reapply your local commits on top of changes from another branch. It's a powerful tool for integrating changes from one branch into another, but it can be tricky to use.

Here's a brief overview of how it works:

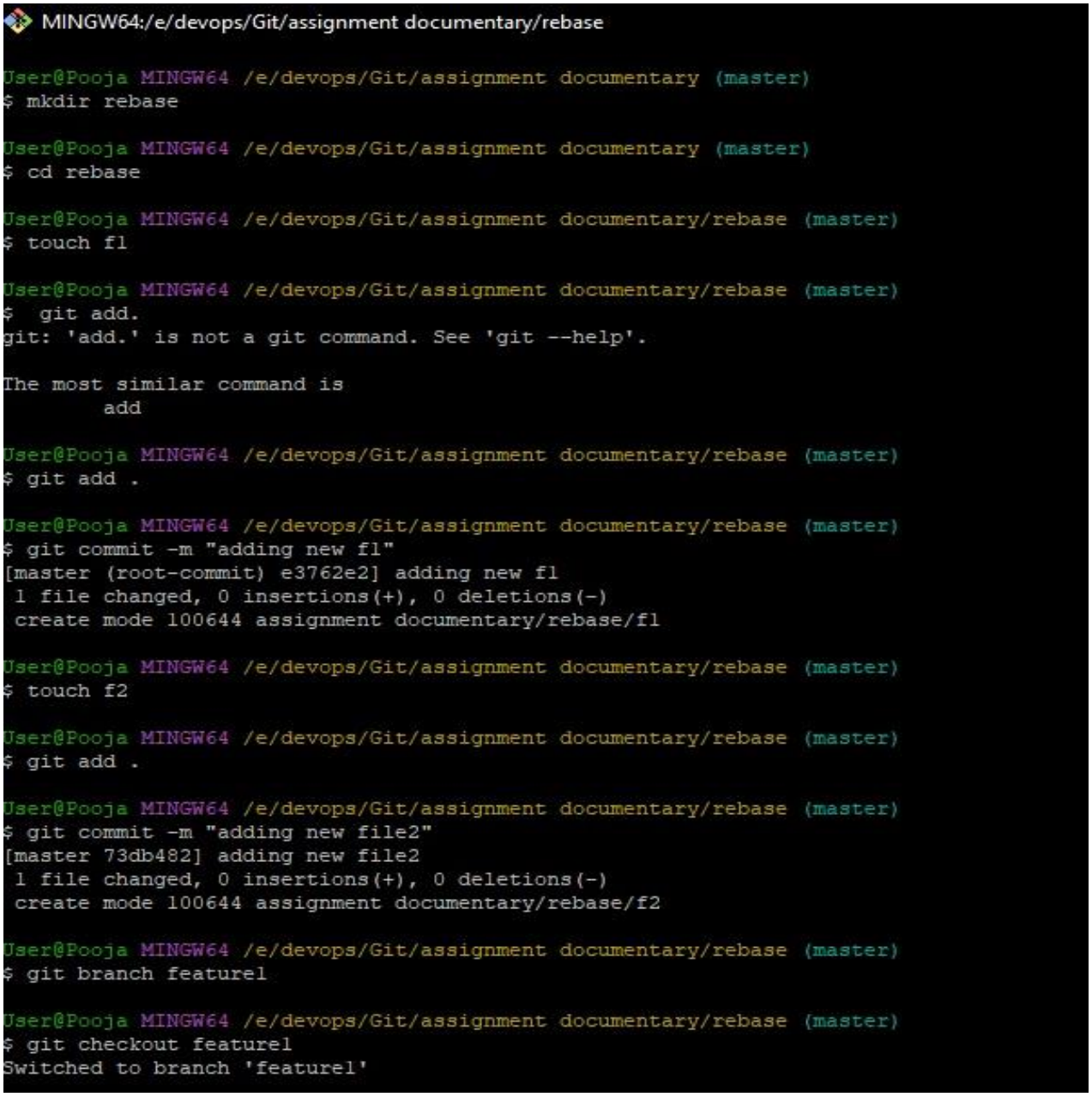
1. You start by checking out the branch you want to rebase (e.g., git checkout feature-branch).

2. You then run git rebase <base-branch>, where <base-branch> is the branch you want to rebase onto (e.g., git rebase main).

3. Git will then rewinds your local commits, apply the changes from the base branch, and reapply your local commits on top.

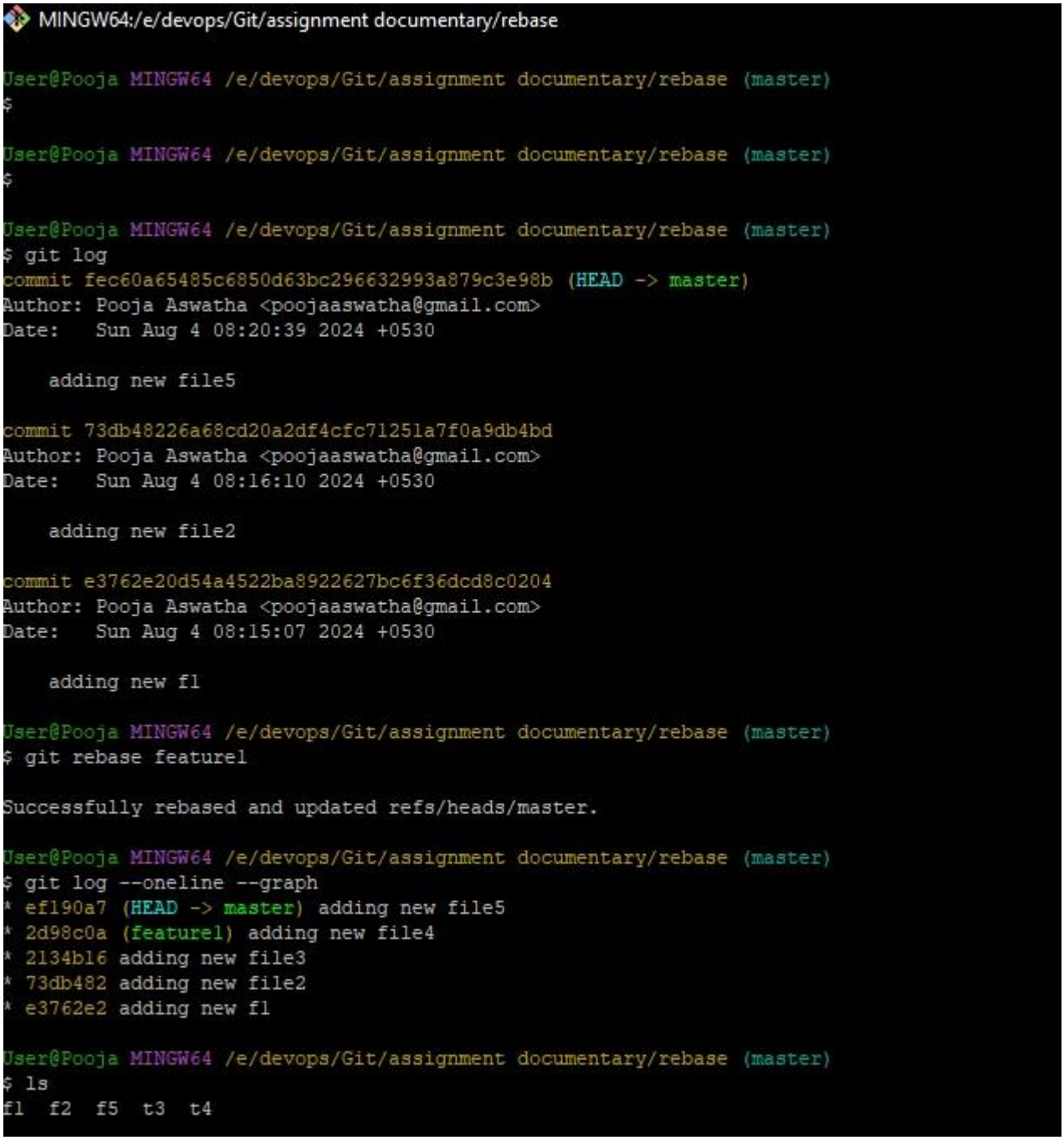
Some common use cases for git rebase include:

- Integrating changes from the main branch into a feature branch
- Squashing multiple commits into a single commit
- Editing or deleting previous commits


MINGW64:/e/devops/Git/assignment documentary/rebase
User@Pooja MINGW64 /e/devops/Git/assignment documentary (master)
$ mkdir rebase
User@Pooja MINGW64 /e/devops/Git/assignment documentary (master)
$ cd rebase
User@Pooja MINGW64 /e/devops/Git/assignment documentary/rebase (master)
$ touch f1
User@Pooja MINGW64 /e/devops/Git/assignment documentary/rebase (master)
$ git add.
git: 'add.' is not a git command. See 'git --help'.
The most similar command is
add
User@Pooja MINGW64 /e/devops/Git/assignment documentary/rebase (master)
$ git add .
User@Pooja MINGW64 /e/devops/Git/assignment documentary/rebase (master)
$ git commit -m "adding new f1"
[master (root-commit) e3762e2] adding new f1
1 file changed, 0 insertions(+), 0 deletions(-)
create mode 100644 assignment documentary/rebase/f1
User@Pooja MINGW64 /e/devops/Git/assignment documentary/rebase (master)
$ touch f2
User@Pooja MINGW64 /e/devops/Git/assignment documentary/rebase (master)
$ git add .
User@Pooja MINGW64 /e/devops/Git/assignment documentary/rebase (master)
$ git commit -m "adding new file2"
[master 73db482] adding new file2
1 file changed, 0 insertions(+), 0 deletions(-)
create mode 100644 assignment documentary/rebase/f2
User@Pooja MINGW64 /e/devops/Git/assignment documentary/rebase (master)
$ git branch feature1
User@Pooja MINGW64 /e/devops/Git/assignment documentary/rebase (master)
$ git checkout feature1
Switched to branch 'feature1'

```
MINGW64:/e/devops/Git/assignment documentary/rebase
$ git branch feature1

User@Pooja MINGW64 /e/devops/Git/assignment documentary/rebase (master)
$ git checkout feature1
Switched to branch 'feature1'

User@Pooja MINGW64 /e/devops/Git/assignment documentary/rebase (feature1)
$ touch t3

User@Pooja MINGW64 /e/devops/Git/assignment documentary/rebase (feature1)
$ git add .

User@Pooja MINGW64 /e/devops/Git/assignment documentary/rebase (feature1)
$ git commit -m "adding new file3"
[feature1 2134b16] adding new file3
 1 file changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 assignment documentary/rebase/t3

User@Pooja MINGW64 /e/devops/Git/assignment documentary/rebase (feature1)
$ touch t4

User@Pooja MINGW64 /e/devops/Git/assignment documentary/rebase (feature1)
$ git add .

User@Pooja MINGW64 /e/devops/Git/assignment documentary/rebase (feature1)
$ git commit -m "adding new file4"
[feature1 2d98c0a] adding new file4
 1 file changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 assignment documentary/rebase/t4

User@Pooja MINGW64 /e/devops/Git/assignment documentary/rebase (feature1)
$ ls
f1  f2  t3  t4

User@Pooja MINGW64 /e/devops/Git/assignment documentary/rebase (feature1)
$ git checkout master
Switched to branch 'master'

User@Pooja MINGW64 /e/devops/Git/assignment documentary/rebase (master)
$ touch f5

User@Pooja MINGW64 /e/devops/Git/assignment documentary/rebase (master)
$ git add .
```

```
MINGW64:/e/devops/Git/assignment documentary/rebase
$ git add .

User@Pooja MINGW64 /e/devops/Git/assignment documentary/rebase (master)
$ git commit -m "adding new file5"
[master fec60a6] adding new file5
 1 file changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 assignment documentary/rebase/f5

User@Pooja MINGW64 /e/devops/Git/assignment documentary/rebase (master)
$ ls
f1  f2  f5
```



Git rebase interactive:

git rebase -i (or git rebase --interactive) is a powerful tool that allows you to edit, squash, delete, or reorder commits in a branch. It's a great way to clean up your commit history before merging a feature branch into the main branch.

Here's how it works:

1. You start by checking out the branch you want to rebase (e.g., git checkout feature-branch).

2. You then run git rebase -i <base-branch>, where <base-branch> is the branch you want to rebase onto (e.g., git rebase -i main).

3. Git will open an interactive prompt in your default text editor, showing a list of commits to be rebased.

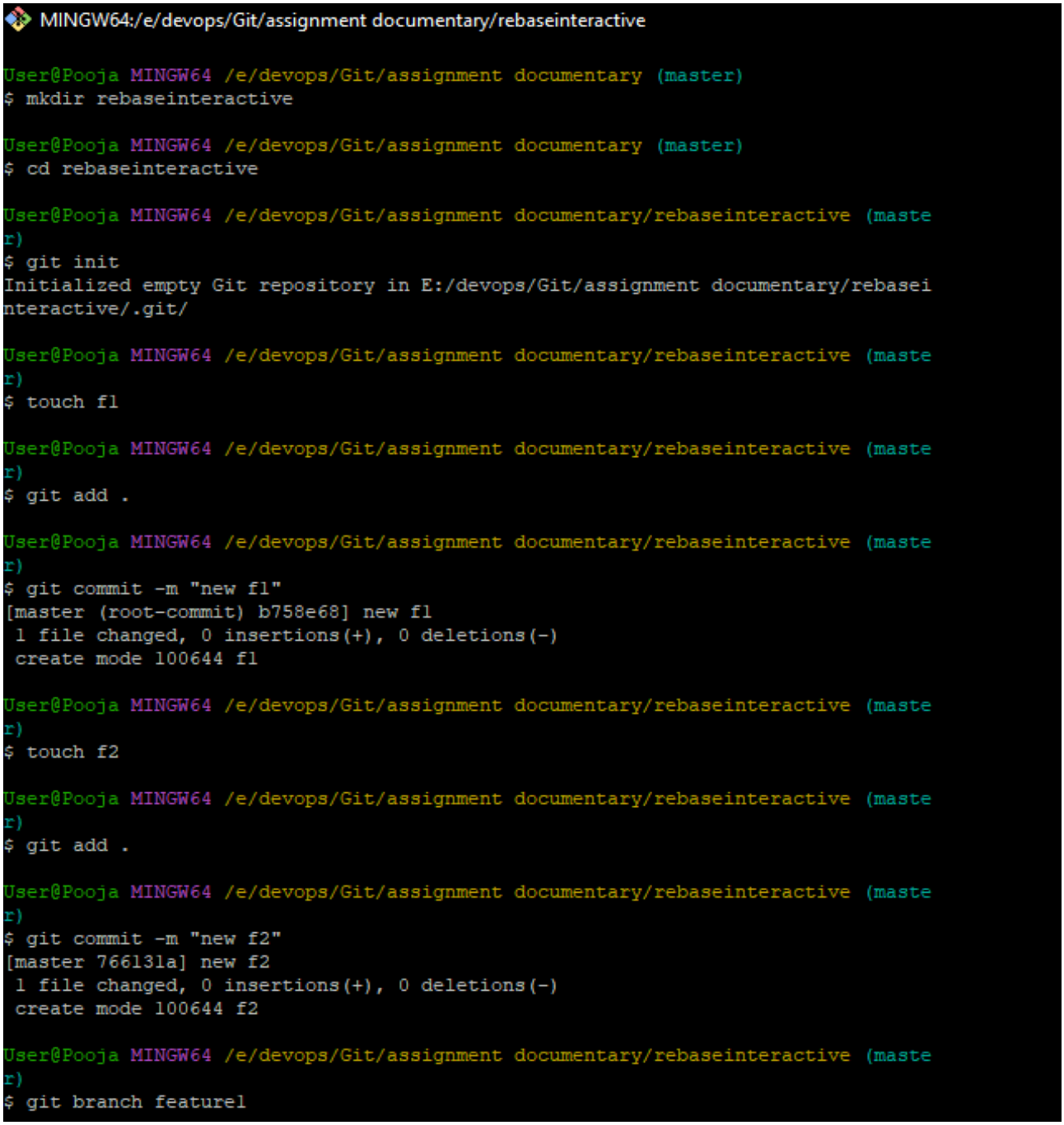
4. You can then edit the list by:

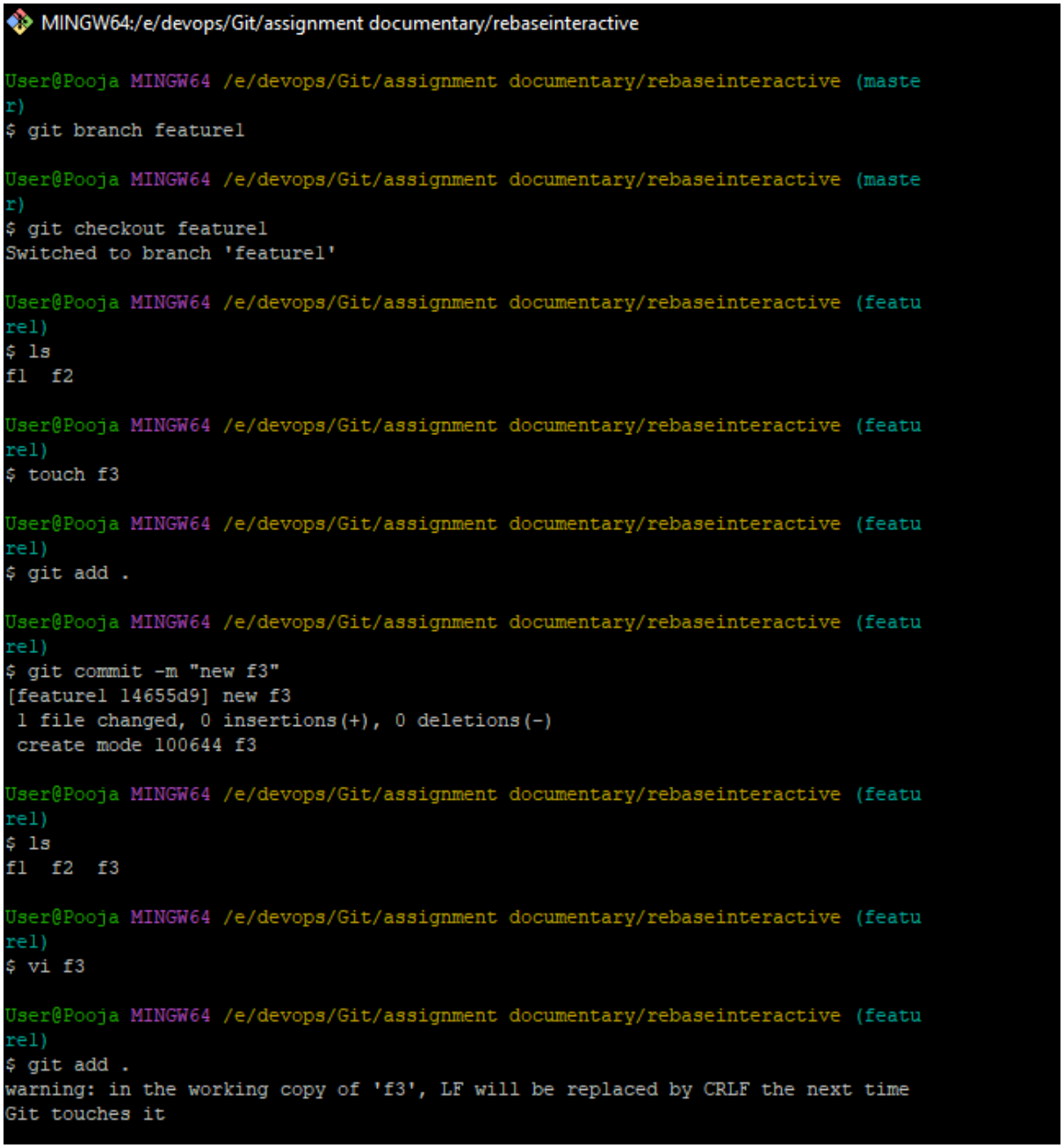
- Picking commits to keep (default)
- Squashing commits together (use s or squash)
- Editing commit messages (use e or edit)
- Deleting commits (use d or drop)
- Reordering commits (use r or reword)

5. Once you've made your changes, save and close the editor. Git will then reapply the commits according to your changes.

Some common use cases for git rebase -i include:

- Squashing multiple small commits into a single commit
- Editing commit messages for clarity or consistency
- Removing unnecessary commits
- Reordering commits to make sense chronologically


MINGW64:/e/devops/Git/assignment documentary/rebaseinteractive
User@Pooja MINGW64 /e/devops/Git/assignment documentary (master)
$ mkdir rebaseinteractive
User@Pooja MINGW64 /e/devops/Git/assignment documentary (master)
$ cd rebaseinteractive
User@Pooja MINGW64 /e/devops/Git/assignment documentary/rebaseinteractive (maste
r)
$ git init
Initialized empty Git repository in E:/devops/Git/assignment documentary/rebasei
nteractive/.git/
User@Pooja MINGW64 /e/devops/Git/assignment documentary/rebaseinteractive (maste
r)
$ touch f1
User@Pooja MINGW64 /e/devops/Git/assignment documentary/rebaseinteractive (maste
r)
$ git add .
User@Pooja MINGW64 /e/devops/Git/assignment documentary/rebaseinteractive (maste
r)
$ git commit -m "new f1"
[master (root-commit) b758e68] new f1
1 file changed, 0 insertions(+), 0 deletions(-)
create mode 100644 f1
User@Pooja MINGW64 /e/devops/Git/assignment documentary/rebaseinteractive (maste
r)
$ touch f2
User@Pooja MINGW64 /e/devops/Git/assignment documentary/rebaseinteractive (maste
r)
$ git add .
User@Pooja MINGW64 /e/devops/Git/assignment documentary/rebaseinteractive (maste
r)
$ git commit -m "new f2"
[master 76613la] new f2
1 file changed, 0 insertions(+), 0 deletions(-)
create mode 100644 f2
User@Pooja MINGW64 /e/devops/Git/assignment documentary/rebaseinteractive (maste
r)
$ git branch feature1


MINGW64:/e/devops/Git/assignment documentary/rebaseinteractive
User@Pooja MINGW64 /e/devops/Git/assignment documentary/rebaseinteractive (master)
$ git branch feature1
User@Pooja MINGW64 /e/devops/Git/assignment documentary/rebaseinteractive (master)
$ git checkout feature1
Switched to branch 'feature1'
User@Pooja MINGW64 /e/devops/Git/assignment documentary/rebaseinteractive (feature1)
$ ls
f1 f2
User@Pooja MINGW64 /e/devops/Git/assignment documentary/rebaseinteractive (feature1)
$ touch f3
User@Pooja MINGW64 /e/devops/Git/assignment documentary/rebaseinteractive (feature1)
$ git add .
User@Pooja MINGW64 /e/devops/Git/assignment documentary/rebaseinteractive (feature1)
$ git commit -m "new f3"
[feature1 14655d9] new f3
1 file changed, 0 insertions(+), 0 deletions(-)
create mode 100644 f3
User@Pooja MINGW64 /e/devops/Git/assignment documentary/rebaseinteractive (feature1)
$ ls
f1 f2 f3
User@Pooja MINGW64 /e/devops/Git/assignment documentary/rebaseinteractive (feature1)
$ vi f3
User@Pooja MINGW64 /e/devops/Git/assignment documentary/rebaseinteractive (feature1)
$ git add .
warning: in the working copy of 'f3', LF will be replaced by CRLF the next time
Git touches it

```
MINGW64:/e/devops/Git/assignment documentary/rebaseinteractive
User@Pooja MINGW64 /e/devops/Git/assignment documentary/rebaseinteractive (featu
rel)
$ git commit -m "adding new line"
[featurel b4ae6dl] adding new line
 l file changed, l insertion(+)

User@Pooja MINGW64 /e/devops/Git/assignment documentary/rebaseinteractive (featu
rel)
$ vi f3

User@Pooja MINGW64 /e/devops/Git/assignment documentary/rebaseinteractive (featu
rel)
$ git add .
warning: in the working copy of 'f3', LF will be replaced by CRLF the next time
Git touches it

User@Pooja MINGW64 /e/devops/Git/assignment documentary/rebaseinteractive (featu
rel)
$ git commit -m "adding new line"
[featurel a920d6f] adding new line
 l file changed, 2 insertions(+)

User@Pooja MINGW64 /e/devops/Git/assignment documentary/rebaseinteractive (featu
rel)
$ git log --oneline
a920d6f (HEAD -> featurel) adding new line
b4ae6dl adding new line
14655d9 new f3
766131a (master) new f2
b758e68 new fl
```

```
MINGW64:/e/devops/Git/assignment documentary/rebaseinteractive


User@Pooja MINGW64 /e/devops/Git/assignment documentary/rebaseinteractive (featu
rel)
$ git checkout master
Switched to branch 'master'

User@Pooja MINGW64 /e/devops/Git/assignment documentary/rebaseinteractive (maste
r)
$ git rebase -i featurel
Successfully rebased and updated refs/heads/master.

User@Pooja MINGW64 /e/devops/Git/assignment documentary/rebaseinteractive (maste
r)
$ git log
commit a920d6f12be291fe1b4b43ccc267787f29466059 (HEAD -> master, featurel)
Author: Pooja Aswatha <poojaaswatha@gmail.com>
Date:   Sun Aug 4 12:50:00 2024 +0530

    adding new line

commit b4ae6d11b6ef21dea2399e61af451790ebfa92ec
Author: Pooja Aswatha <poojaaswatha@gmail.com>
Date:   Sun Aug 4 12:49:01 2024 +0530

    adding new line

commit 14655d9459aef09595c0236154146953b275ecf7
Author: Pooja Aswatha <poojaaswatha@gmail.com>
Date:   Sun Aug 4 12:47:56 2024 +0530

    new f3

commit 766131a94e76a8ac0b640185686ald0d5dec4d33
Author: Pooja Aswatha <poojaaswatha@gmail.com>
Date:   Sun Aug 4 12:46:57 2024 +0530

    new f2
```

```
User@Pooja MINGW64 /e/devops/Git/assignment documentary/rebaseinteractive (maste
r)
$ git rebase -i head~3
error: cannot 'squash' without a previous commit
error: cannot 'squash' without a previous commit
error: cannot 'squash' without a previous commit
You can fix this with 'git rebase --edit-todo' and then run 'git rebase --contin
ue'.
Or you can abort the rebase with 'git rebase --abort'.

User@Pooja MINGW64 /e/devops/Git/assignment documentary/rebaseinteractive (maste
r|REBASE)
$ git log --oneline
766131a (HEAD) new f2
b758e68 new f1
```

Git Stash:

git stash is a Git command that allows you to temporarily save changes you've made to your working directory, so you can switch branches or work on something else without losing your progress.

Here's how it works:

1. You've made some changes to your code, but you're not ready to commit them yet.

2. You run git stash to save those changes away.

3. Git creates a new stash entry, which is essentially a snapshot of your changes.

4. Your working directory is then reset to the last commit, so you can switch branches or work on something else.

5. When you're ready to revisit your stashed changes, you can run git stash apply to reapply them to your working directory.

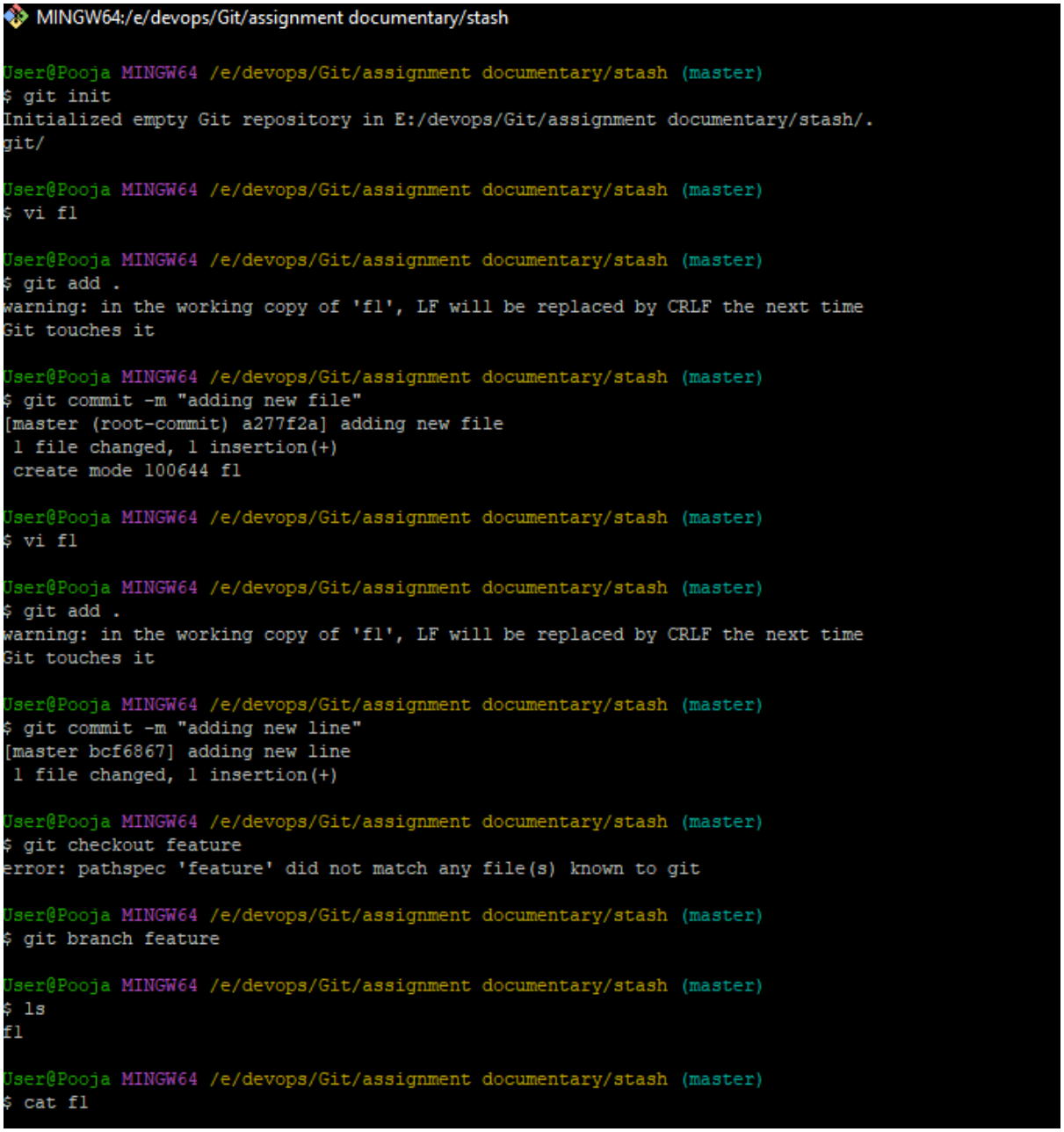
6. If you want to remove the stash entry, you can run git stash drop.

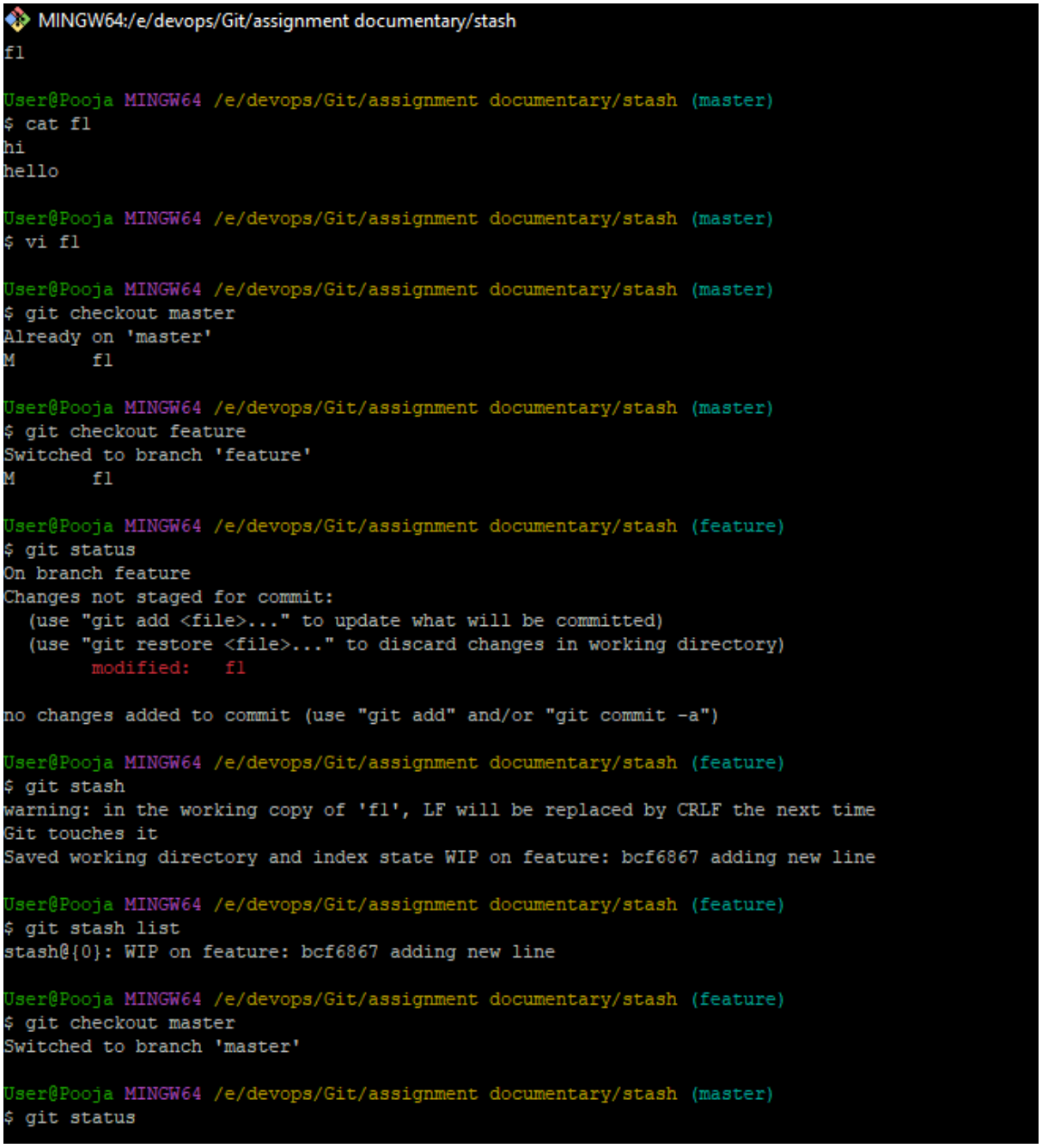
Some common use cases for git stash include:

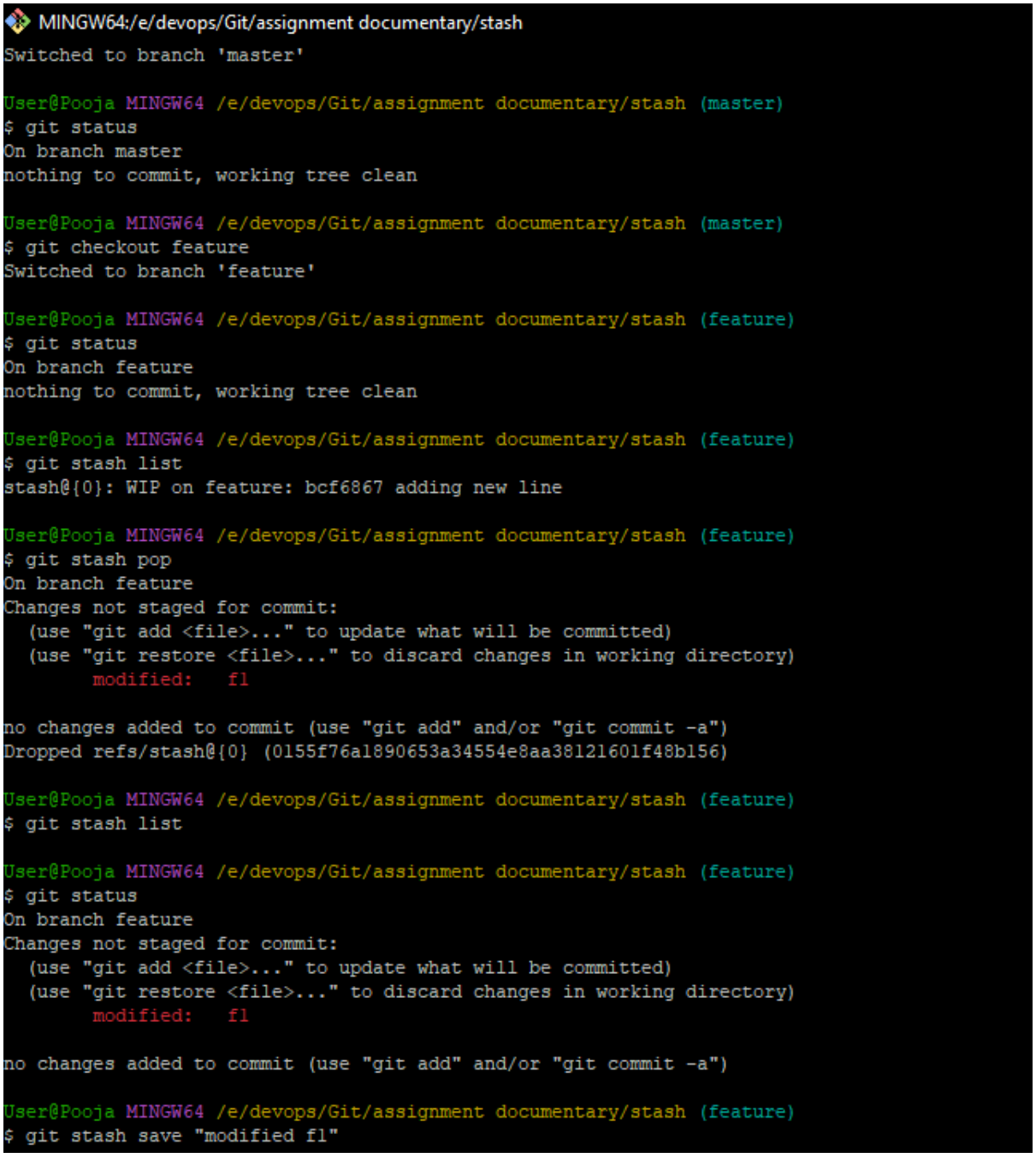
- Temporarily setting aside changes to work on a higher-priority task
- Switching branches without losing your progress
- Testing changes on a different branch without committing them
- Cleaning up your working directory before a commit

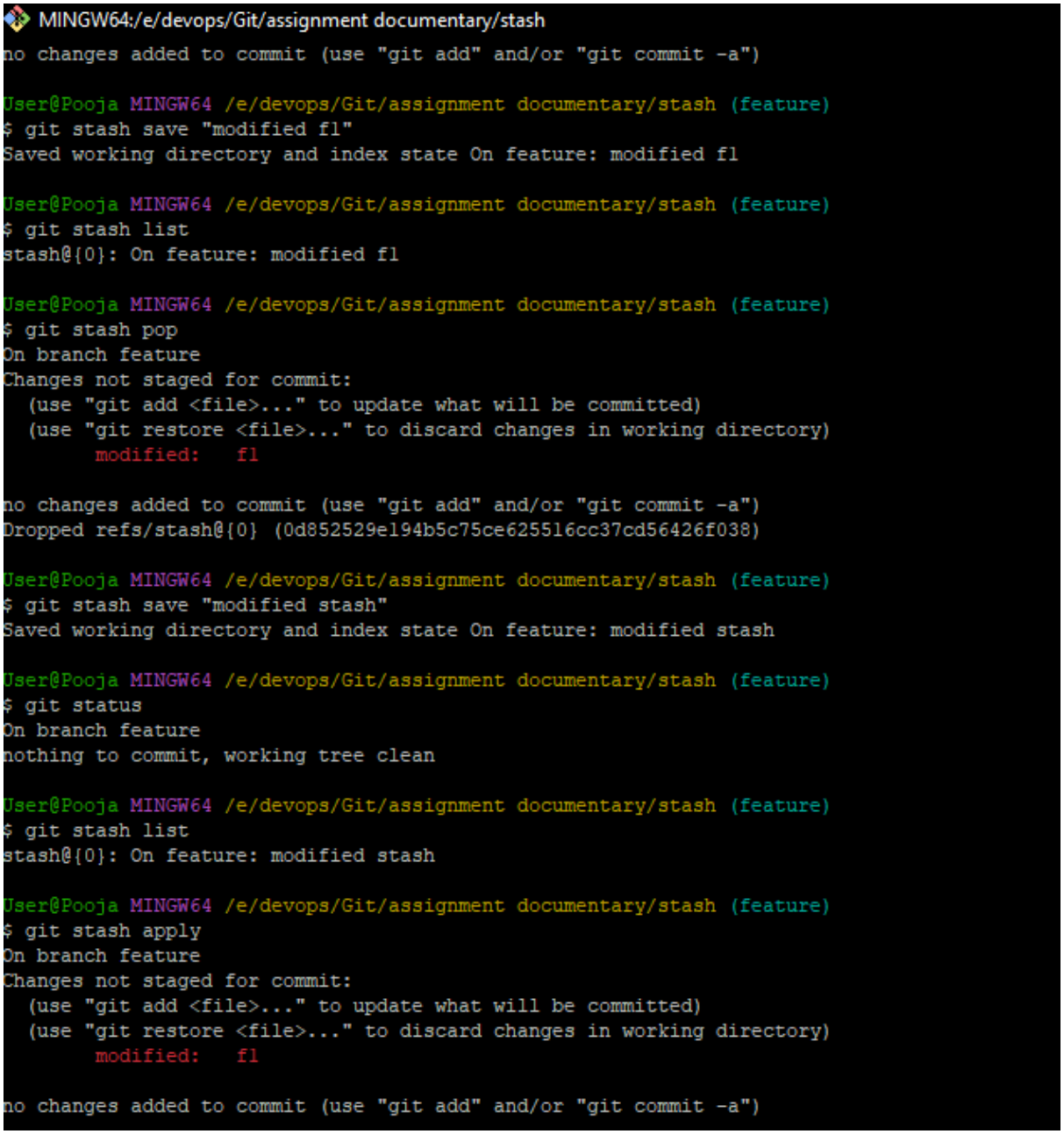
You can also use git stash with additional options, such as:

- git stash list: Shows a list of all your stash entries
- git stash show: Shows the changes in the latest stash entry
- git stash apply <stash-entry>: Applies a specific stash entry
- git stash pop: Applies the latest stash entry and removes it


MINGW64:/e/devops/Git/assignment documentary/stash
User@Pooja MINGW64 /e/devops/Git/assignment documentary/stash (master)
$ git init
Initialized empty Git repository in E:/devops/Git/assignment documentary/stash/.
git/
User@Pooja MINGW64 /e/devops/Git/assignment documentary/stash (master)
$ vi f1
User@Pooja MINGW64 /e/devops/Git/assignment documentary/stash (master)
$ git add .
warning: in the working copy of 'f1', LF will be replaced by CRLF the next time
Git touches it
User@Pooja MINGW64 /e/devops/Git/assignment documentary/stash (master)
$ git commit -m "adding new file"
[master (root-commit) a277f2a] adding new file
1 file changed, 1 insertion(+)
create mode 100644 f1
User@Pooja MINGW64 /e/devops/Git/assignment documentary/stash (master)
$ vi f1
User@Pooja MINGW64 /e/devops/Git/assignment documentary/stash (master)
$ git add .
warning: in the working copy of 'f1', LF will be replaced by CRLF the next time
Git touches it
User@Pooja MINGW64 /e/devops/Git/assignment documentary/stash (master)
$ git commit -m "adding new line"
[master bcf6867] adding new line
1 file changed, 1 insertion(+)
User@Pooja MINGW64 /e/devops/Git/assignment documentary/stash (master)
$ git checkout feature
error: pathspec 'feature' did not match any file(s) known to git
User@Pooja MINGW64 /e/devops/Git/assignment documentary/stash (master)
$ git branch feature
User@Pooja MINGW64 /e/devops/Git/assignment documentary/stash (master)
$ ls
f1
User@Pooja MINGW64 /e/devops/Git/assignment documentary/stash (master)
$ cat f1


MINGW64:/e/devops/Git/assignment documentary/stash
f1
User@Pooja MINGW64 /e/devops/Git/assignment documentary/stash (master)
$ cat f1
hi
hello
User@Pooja MINGW64 /e/devops/Git/assignment documentary/stash (master)
$ vi f1
User@Pooja MINGW64 /e/devops/Git/assignment documentary/stash (master)
$ git checkout master
Already on 'master'
M f1
User@Pooja MINGW64 /e/devops/Git/assignment documentary/stash (master)
$ git checkout feature
Switched to branch 'feature'
M f1
User@Pooja MINGW64 /e/devops/Git/assignment documentary/stash (feature)
$ git status
On branch feature
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: f1
no changes added to commit (use "git add" and/or "git commit -a")
User@Pooja MINGW64 /e/devops/Git/assignment documentary/stash (feature)
$ git stash
warning: in the working copy of 'f1', LF will be replaced by CRLF the next time
Git touches it
Saved working directory and index state WIP on feature: bcf6867 adding new line
User@Pooja MINGW64 /e/devops/Git/assignment documentary/stash (feature)
$ git stash list
stash@{0}: WIP on feature: bcf6867 adding new line
User@Pooja MINGW64 /e/devops/Git/assignment documentary/stash (feature)
$ git checkout master
Switched to branch 'master'
User@Pooja MINGW64 /e/devops/Git/assignment documentary/stash (master)
$ git status


MINGW64:/e/devops/Git/assignment documentary/stash
Switched to branch 'master'
User@Pooja MINGW64 /e/devops/Git/assignment documentary/stash (master)
$ git status
On branch master
nothing to commit, working tree clean
User@Pooja MINGW64 /e/devops/Git/assignment documentary/stash (master)
$ git checkout feature
Switched to branch 'feature'
User@Pooja MINGW64 /e/devops/Git/assignment documentary/stash (feature)
$ git status
On branch feature
nothing to commit, working tree clean
User@Pooja MINGW64 /e/devops/Git/assignment documentary/stash (feature)
$ git stash list
stash@{0}: WIP on feature: bcf6867 adding new line
User@Pooja MINGW64 /e/devops/Git/assignment documentary/stash (feature)
$ git stash pop
On branch feature
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: f1
no changes added to commit (use "git add" and/or "git commit -a")
Dropped refs/stash@{0} (0155f76a1890653a34554e8aa38121601f48b156)
User@Pooja MINGW64 /e/devops/Git/assignment documentary/stash (feature)
$ git stash list
User@Pooja MINGW64 /e/devops/Git/assignment documentary/stash (feature)
$ git status
On branch feature
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: f1
no changes added to commit (use "git add" and/or "git commit -a")
User@Pooja MINGW64 /e/devops/Git/assignment documentary/stash (feature)
$ git stash save "modified f1"


MINGW64:/e/devops/Git/assignment documentary/stash
no changes added to commit (use "git add" and/or "git commit -a")
User@Pooja MINGW64 /e/devops/Git/assignment documentary/stash (feature)
$ git stash save "modified f1"
Saved working directory and index state On feature: modified f1
User@Pooja MINGW64 /e/devops/Git/assignment documentary/stash (feature)
$ git stash list
stash@{0}: On feature: modified f1
User@Pooja MINGW64 /e/devops/Git/assignment documentary/stash (feature)
$ git stash pop
On branch feature
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: f1
no changes added to commit (use "git add" and/or "git commit -a")
Dropped refs/stash@{0} (0d852529e194b5c75ce625516cc37cd56426f038)
User@Pooja MINGW64 /e/devops/Git/assignment documentary/stash (feature)
$ git stash save "modified stash"
Saved working directory and index state On feature: modified stash
User@Pooja MINGW64 /e/devops/Git/assignment documentary/stash (feature)
$ git status
On branch feature
nothing to commit, working tree clean
User@Pooja MINGW64 /e/devops/Git/assignment documentary/stash (feature)
$ git stash list
stash@{0}: On feature: modified stash
User@Pooja MINGW64 /e/devops/Git/assignment documentary/stash (feature)
$ git stash apply
On branch feature
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: f1
no changes added to commit (use "git add" and/or "git commit -a")

```
User@Pooja MINGW64 /e/devops/Git/assignment documentary/stash (feature)
$ git stash apply
On branch feature
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:   f1

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

User@Pooja MINGW64 /e/devops/Git/assignment documentary/stash (feature)
$ git log
commit bcf68671ffb6ffa55d526e43f77a18404038de6f (HEAD -> feature, master)
Author: Pooja Aswatha <poojaaswatha@gmail.com>
Date:   Sun Aug 4 13:34:13 2024 +0530

    adding new line

commit a277f2a095168493faa5b064edlafe16eb19d141
Author: Pooja Aswatha <poojaaswatha@gmail.com>
Date:   Sun Aug 4 13:33:31 2024 +0530

    adding new file
```

Central Repo: Git Hub

GitHub is a web-based platform for version control and collaboration on software development projects. It allows developers to:

1. Host and manage repositories (repos) for their projects

2. Collaborate with others on the same project

3. Track changes and updates through commits and pull requests

4. Share and discover open-source software

Key features of GitHub include:

1. Repositories: Store and manage your project's code, documentation, and assets

2. Branches: Create separate lines of development for features, bug fixes, or experiments

3. Pull requests: Review and merge changes from one branch to another

4. Issues: Track bugs, enhancements, and tasks related to your project

5. Forking: Create a copy of someone else's repository to contribute or modify

6. GitHub Actions: Automate workflows and CI/CD pipelines

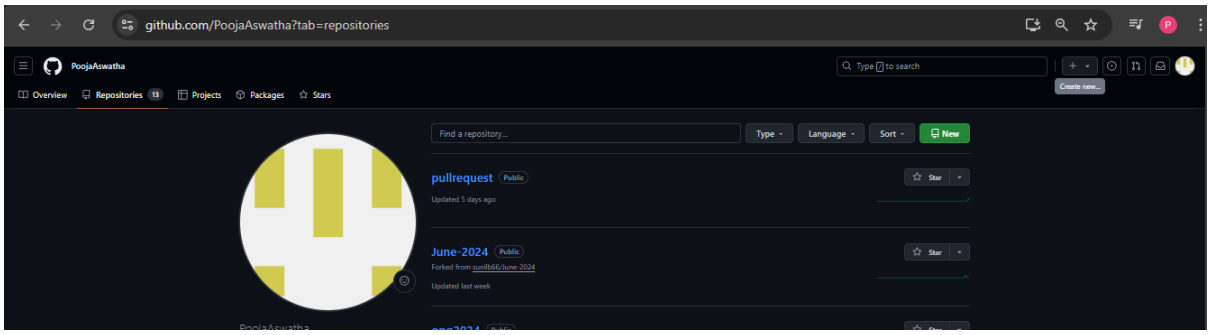
7. GitHub Pages: Host static websites directly from your repository

GitHub Account:

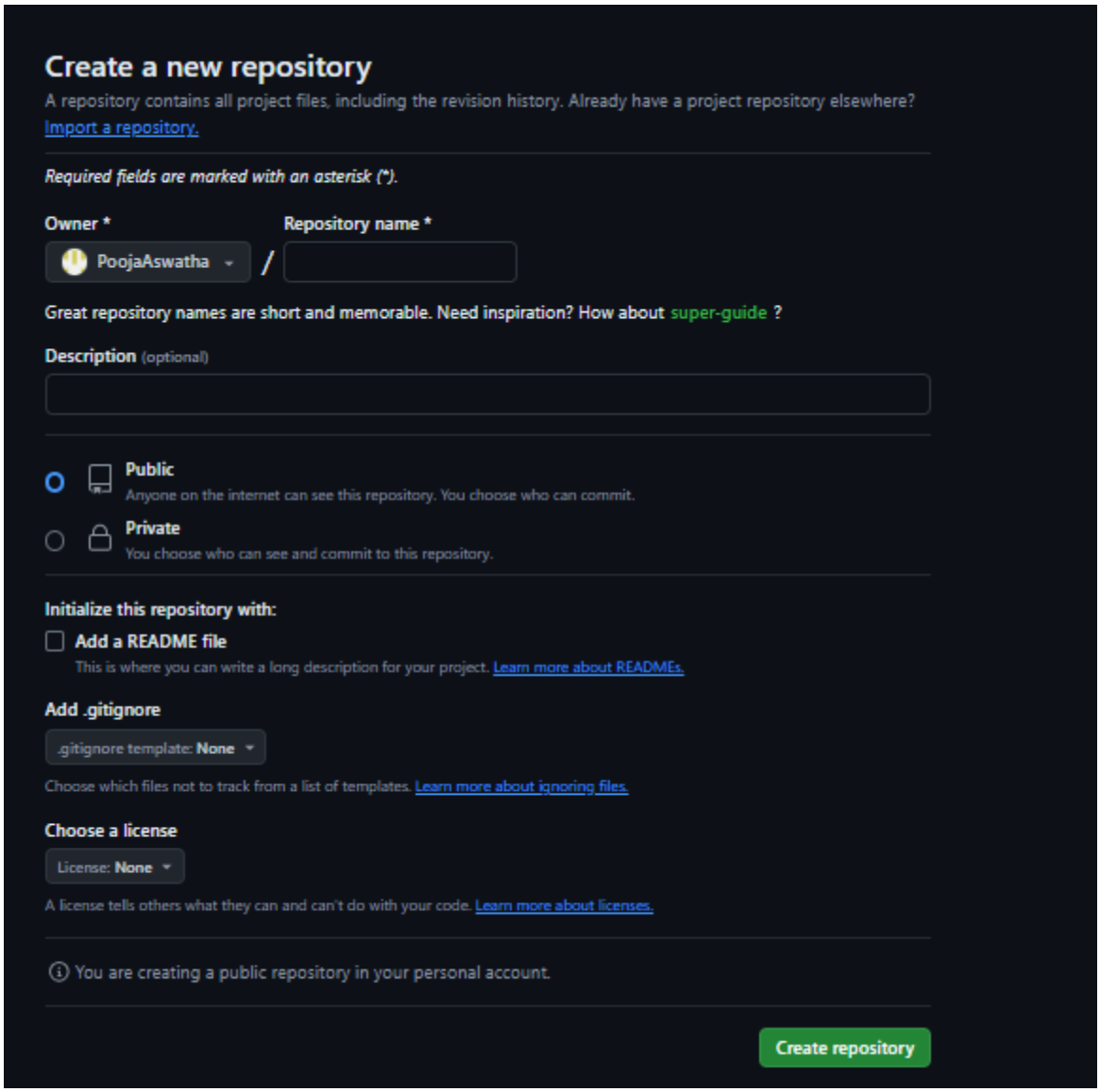
Go to GitHub and sign up for an account.

Create a Repository on GitHub

Now that you have made a GitHub account, sign in, and create a new Repo:



And fill in the relevant details:



Git clone:

The git clone command is used to create a copy of a specific repository or branch within a repository.

Git is a distributed version control system. Maximize the advantages of a full repository on your own machine by cloning, it allows you to download the entire repository, including all its history and branches.

Basic syntax:

git clone <repository-url>

Where <repository-url> is the URL of the remote repository you want to clone.

Some common options used with git clone include:

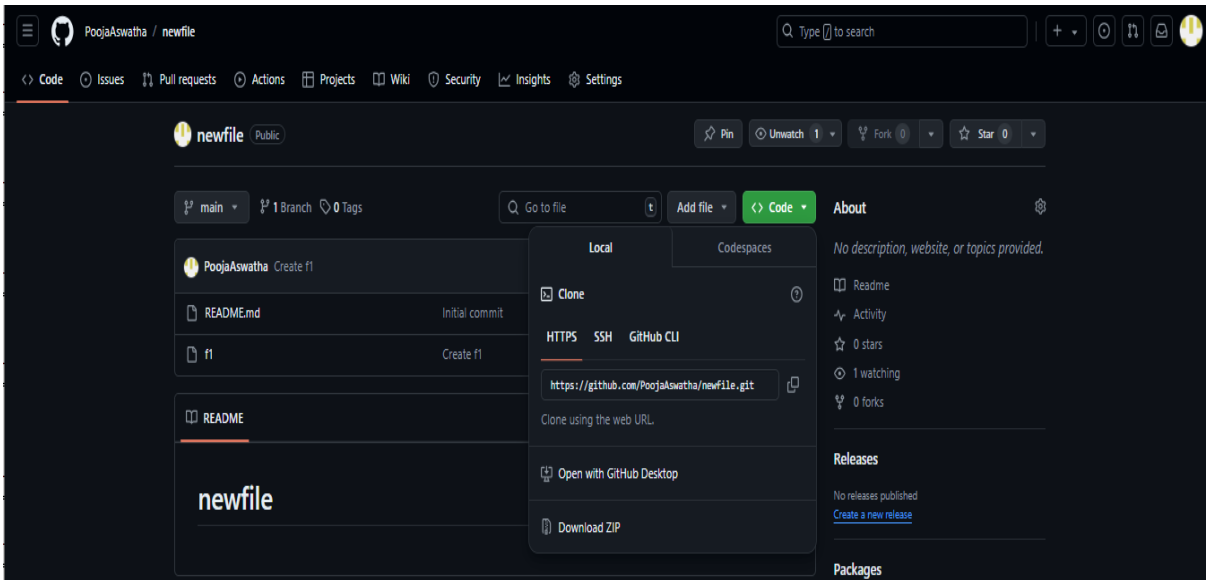
- -b <branch>: Clone a specific branch instead of the default branch (usually main or master)
- depth <number>: Clone only the last <number> commits, instead of the entire history
- recursive: Clone submodules (separate repositories nested inside the main repository)
- <local-directory>: Clone the repository into a specific local directory, instead of the default directory with the same name as the repository

Git clone only needs to be run once for each repository. After that, you can use other Git commands like git pull to update your local copy with changes from the remote repository.

Procedure for using git clone:

Step 1: Obtain the repository URL

- Get the URL of the repository you want to clone from GitHub



Step 2: Open a terminal or command prompt

- Open a terminal on your computer (e.g., Command Prompt on Windows, Terminal on Mac/Linux).
- Navigate to the directory where you want to clone the repository.

Step 3: Run the git clone command

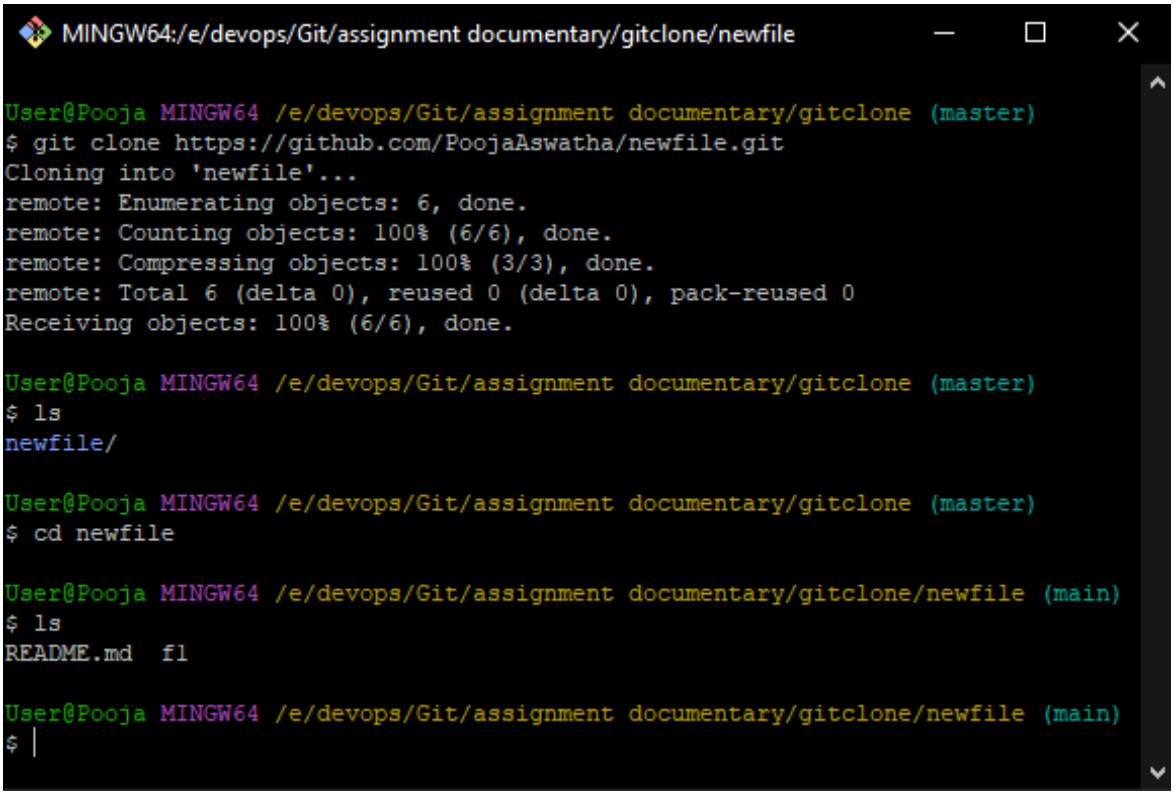
- Type git clone followed by the repository URL: `git clone (link unavailable).
- Press Enter to execute the command.

Step 4: Wait for the cloning process to complete

- Git will download the entire repository, including all its history and branches.
- Depending on the size of the repository, this may take a few seconds or several minutes.

Step 5: Verify the clone

- Once the cloning process is complete, verify that the repository has been cloned correctly by checking the directory contents.
- You should see the repository files and subdirectories.



Git push:

git push is a Git command that uploads your local changes to a remote repository. Here's the basic syntax:

git push <remote-name> <branch-name>

Where:

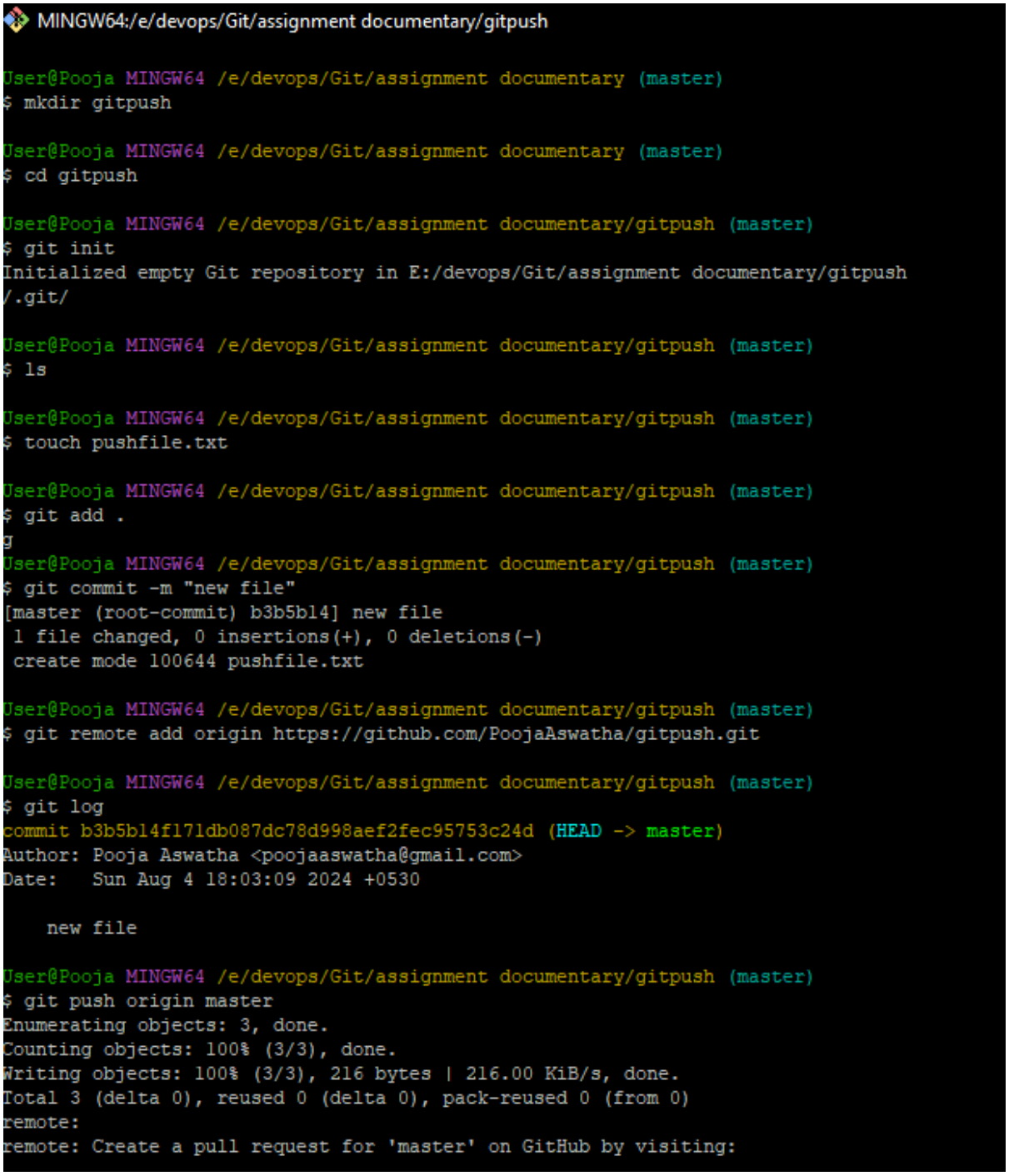
- <remote-name> is the name of the remote repository (e.g., origin)
- <branch-name> is the name of the branch you want to push changes to (e.g., main)

Some common options used with git push include:

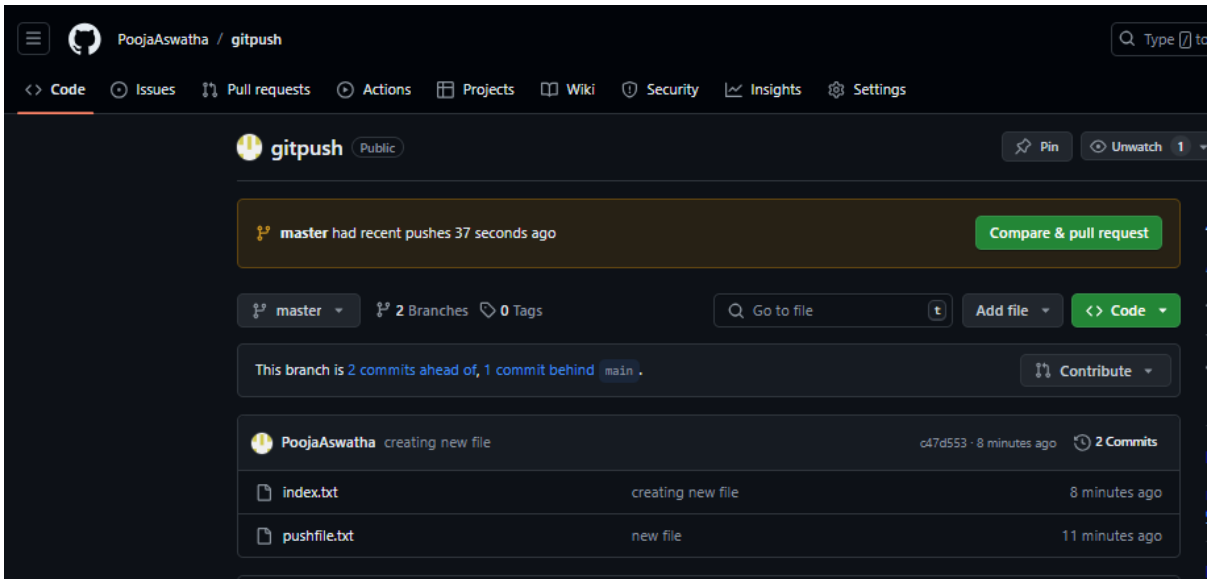
- -u or --set-upstream: Sets the upstream tracking information for the branch
- -f or --force: Forces the push, overwriting any changes on the remote repository
- --tags: Pushes tags to the remote repository
- --all: Pushes all branches to the remote repository

Benefits of using git push include:

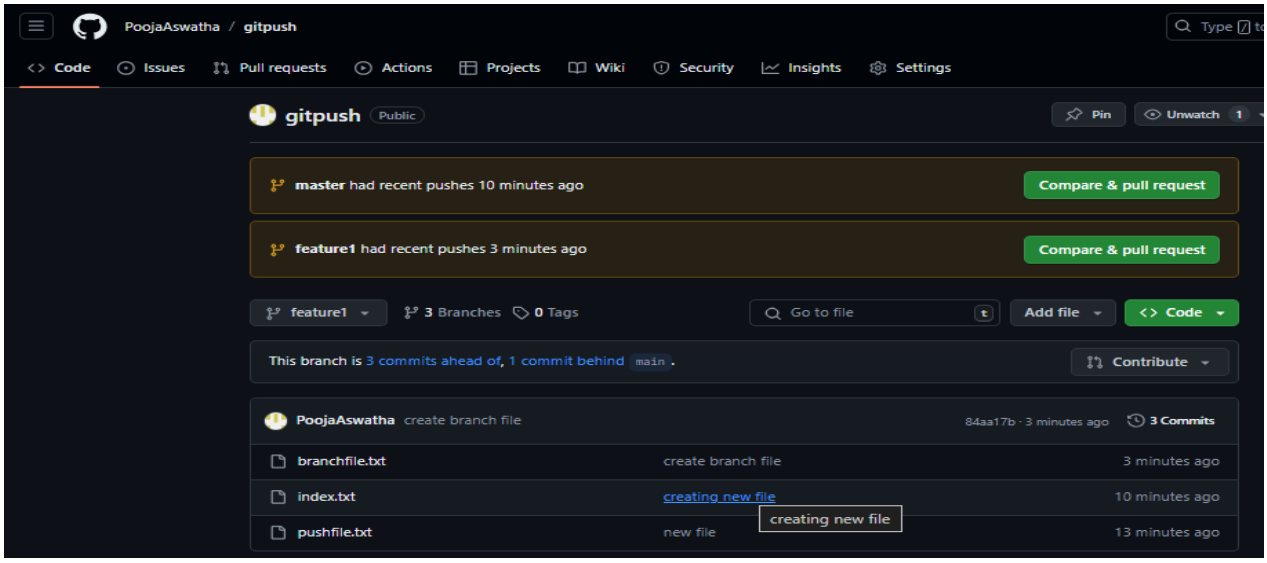
- Shares your changes with others working on the project
- Backs up your local changes to a remote repository
- Supports collaboration by allowing others to review and build upon your changes


MINGW64:/e/devops/Git/assignment documentary/gitpush
User@Pooja MINGW64 /e/devops/Git/assignment documentary (master)
$ mkdir gitpush
User@Pooja MINGW64 /e/devops/Git/assignment documentary (master)
$ cd gitpush
User@Pooja MINGW64 /e/devops/Git/assignment documentary/gitpush (master)
$ git init
Initialized empty Git repository in E:/devops/Git/assignment documentary/gitpush
/.git/
User@Pooja MINGW64 /e/devops/Git/assignment documentary/gitpush (master)
$ ls
User@Pooja MINGW64 /e/devops/Git/assignment documentary/gitpush (master)
$ touch pushfile.txt
User@Pooja MINGW64 /e/devops/Git/assignment documentary/gitpush (master)
$ git add .
g
User@Pooja MINGW64 /e/devops/Git/assignment documentary/gitpush (master)
$ git commit -m "new file"
[master (root-commit) b3b5b14] new file
1 file changed, 0 insertions(+), 0 deletions(-)
create mode 100644 pushfile.txt
User@Pooja MINGW64 /e/devops/Git/assignment documentary/gitpush (master)
$ git remote add origin https://github.com/PoojaAswatha/gitpush.git
User@Pooja MINGW64 /e/devops/Git/assignment documentary/gitpush (master)
$ git log
commit b3b5b14f171db087dc78d998aef2fec95753c24d (HEAD -> master)
Author: Pooja Aswatha <poojaaswatha@gmail.com>
Date: Sun Aug 4 18:03:09 2024 +0530
new file
User@Pooja MINGW64 /e/devops/Git/assignment documentary/gitpush (master)
$ git push origin master
Enumerating objects: 3, done.
Counting objects: 100% (3/3), done.
Writing objects: 100% (3/3), 216 bytes | 216.00 KiB/s, done.
Total 3 (delta 0), reused 0 (delta 0), pack-reused 0 (from 0)
remote:
remote: Create a pull request for 'master' on GitHub by visiting:

```
MINGW64:/e/devops/Git/assignment documentary/gitpush
User@Pooja MINGW64 /e/devops/Git/assignment documentary/gitpush (master)
$ git log
commit b3b5b14f171db087dc78d998aef2fec95753c24d (HEAD -> master)
Author: Pooja Aswatha <poojaaswatha@gmail.com>
Date:   Sun Aug 4 18:03:09 2024 +0530

    new file

User@Pooja MINGW64 /e/devops/Git/assignment documentary/gitpush (master)
$ git push origin master
Enumerating objects: 3, done.
Counting objects: 100% (3/3), done.
Writing objects: 100% (3/3), 216 bytes | 216.00 KiB/s, done.
Total 3 (delta 0), reused 0 (delta 0), pack-reused 0 (from 0)
remote:
remote: Create a pull request for 'master' on GitHub by visiting:
remote:      https://github.com/PoojaAswatha/gitpush/pull/new/master
remote:
To https://github.com/PoojaAswatha/gitpush.git
 * [new branch]      master -> master

User@Pooja MINGW64 /e/devops/Git/assignment documentary/gitpush (master)
$ touch index.txt

User@Pooja MINGW64 /e/devops/Git/assignment documentary/gitpush (master)
$ git add .
g
User@Pooja MINGW64 /e/devops/Git/assignment documentary/gitpush (master)
$ git commit -m "creating new file"
[master c47d553] creating new file
 1 file changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 index.txt

User@Pooja MINGW64 /e/devops/Git/assignment documentary/gitpush (master)
$ git push origin master
Enumerating objects: 3, done.
Counting objects: 100% (3/3), done.
Delta compression using up to 4 threads
Compressing objects: 100% (2/2), done.
Writing objects: 100% (2/2), 254 bytes | 254.00 KiB/s, done.
Total 2 (delta 0), reused 0 (delta 0), pack-reused 0 (from 0)
To https://github.com/PoojaAswatha/gitpush.git
   b3b5b14..c47d553  master -> master
```



```
MINGW64:/e/devops/Git/assignment documentary/gitpush
 create mode 100644 index.txt

User@Pooja MINGW64 /e/devops/Git/assignment documentary/gitpush (master)
$ git push origin master
Enumerating objects: 3, done.
Counting objects: 100% (3/3), done.
Delta compression using up to 4 threads
Compressing objects: 100% (2/2), done.
Writing objects: 100% (2/2), 254 bytes | 254.00 KiB/s, done.
Total 2 (delta 0), reused 0 (delta 0), pack-reused 0 (from 0)
To https://github.com/PoojaAswatha/gitpush.git
   b3b5b14..c47d553  master -> master

User@Pooja MINGW64 /e/devops/Git/assignment documentary/gitpush (master)
$ git checkout -b feature1
Switched to a new branch 'feature1'

User@Pooja MINGW64 /e/devops/Git/assignment documentary/gitpush (feature1)
$ touch branchfile.txt

User@Pooja MINGW64 /e/devops/Git/assignment documentary/gitpush (feature1)
$ git add .
g
User@Pooja MINGW64 /e/devops/Git/assignment documentary/gitpush (feature1)
$ git commit -m "create branch file"
[feature1 84aa17b] create branch file
 1 file changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 branchfile.txt

User@Pooja MINGW64 /e/devops/Git/assignment documentary/gitpush (feature1)
$ git push origin feature1
Enumerating objects: 3, done.
Counting objects: 100% (3/3), done.
Delta compression using up to 4 threads
Compressing objects: 100% (2/2), done.
Writing objects: 100% (2/2), 261 bytes | 261.00 KiB/s, done.
Total 2 (delta 0), reused 0 (delta 0), pack-reused 0 (from 0)
remote:
remote: Create a pull request for 'feature1' on GitHub by visiting:
remote:      https://github.com/PoojaAswatha/gitpush/pull/new/feature1
remote:
To https://github.com/PoojaAswatha/gitpush.git
 * [new branch]      feature1 -> feature1
```



Git Fetch:

Git fetch is a Git command that downloads changes from a remote repository, but doesn't merge them into your local branch. Here's a breakdown of the command:

git fetch [options] [remote-name] [refspec]

- [options]: Optional flags that modify the behavior of the command
- [remote-name]: The name of the remote repository (e.g., origin)
- [refspec]: Specifies which branches or refs to fetch (e.g., main or refs/heads/main)

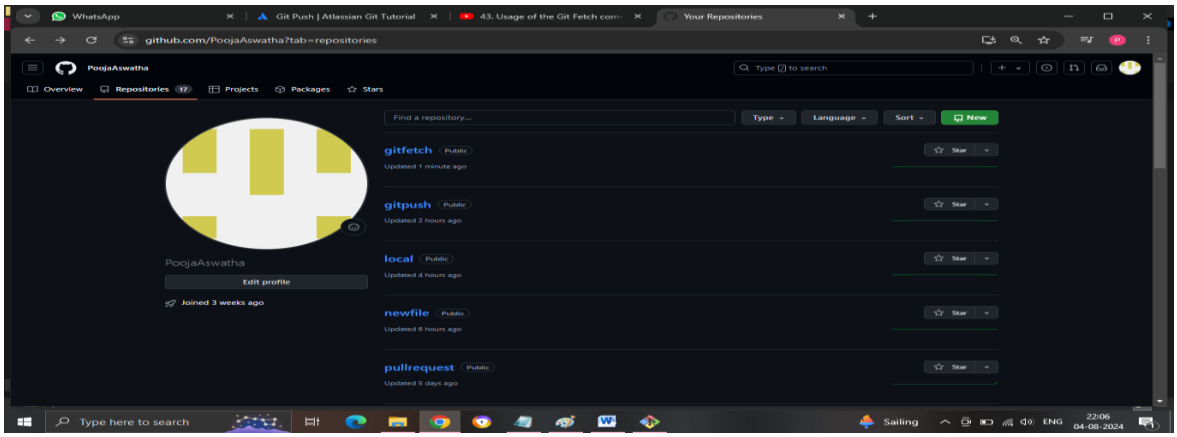
Common options:

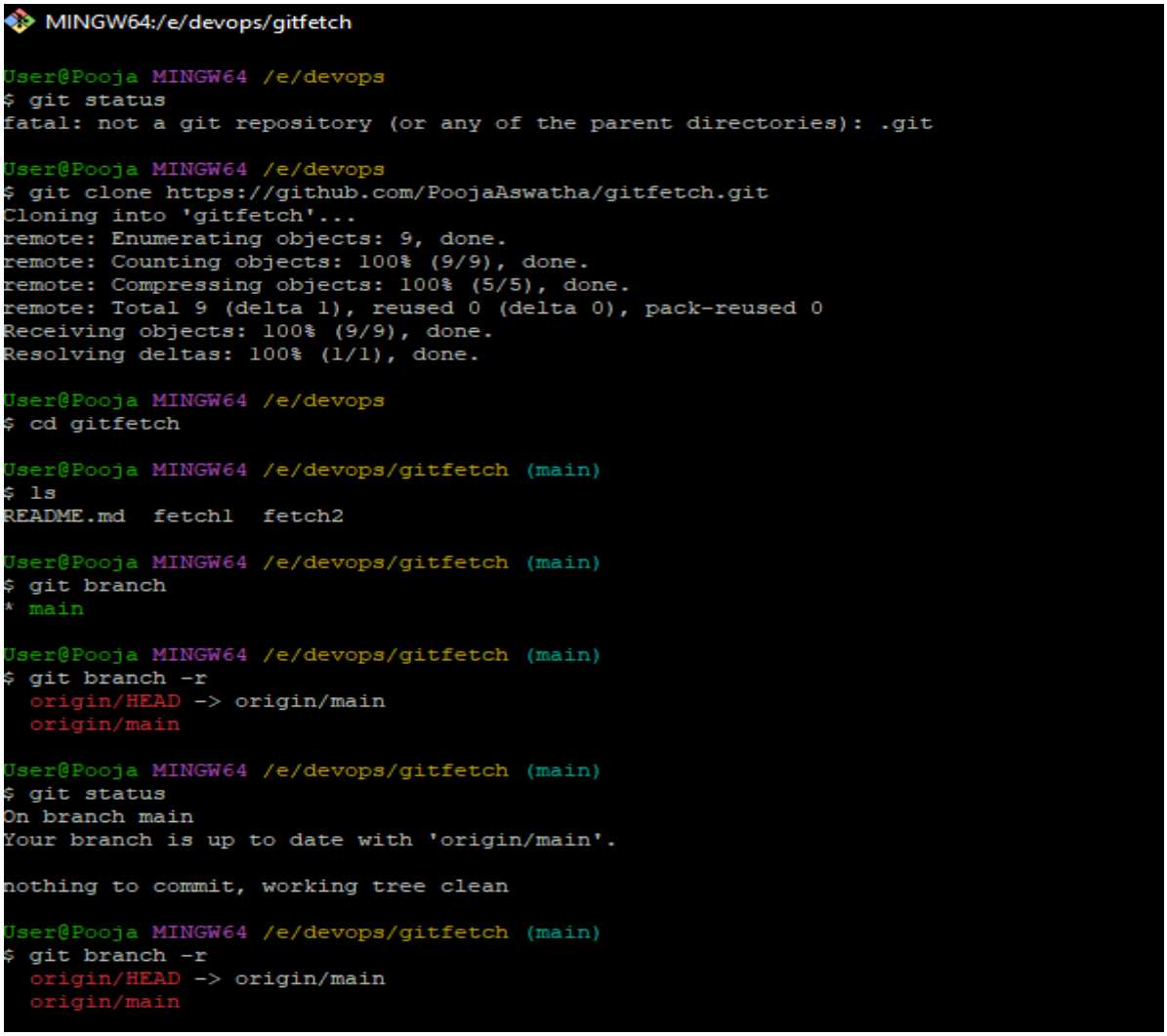
- --all: Fetches all branches from the remote repository
- --tags: Fetches tags from the remote repository
- -v or --verbose: Shows more detailed output during the fetch process

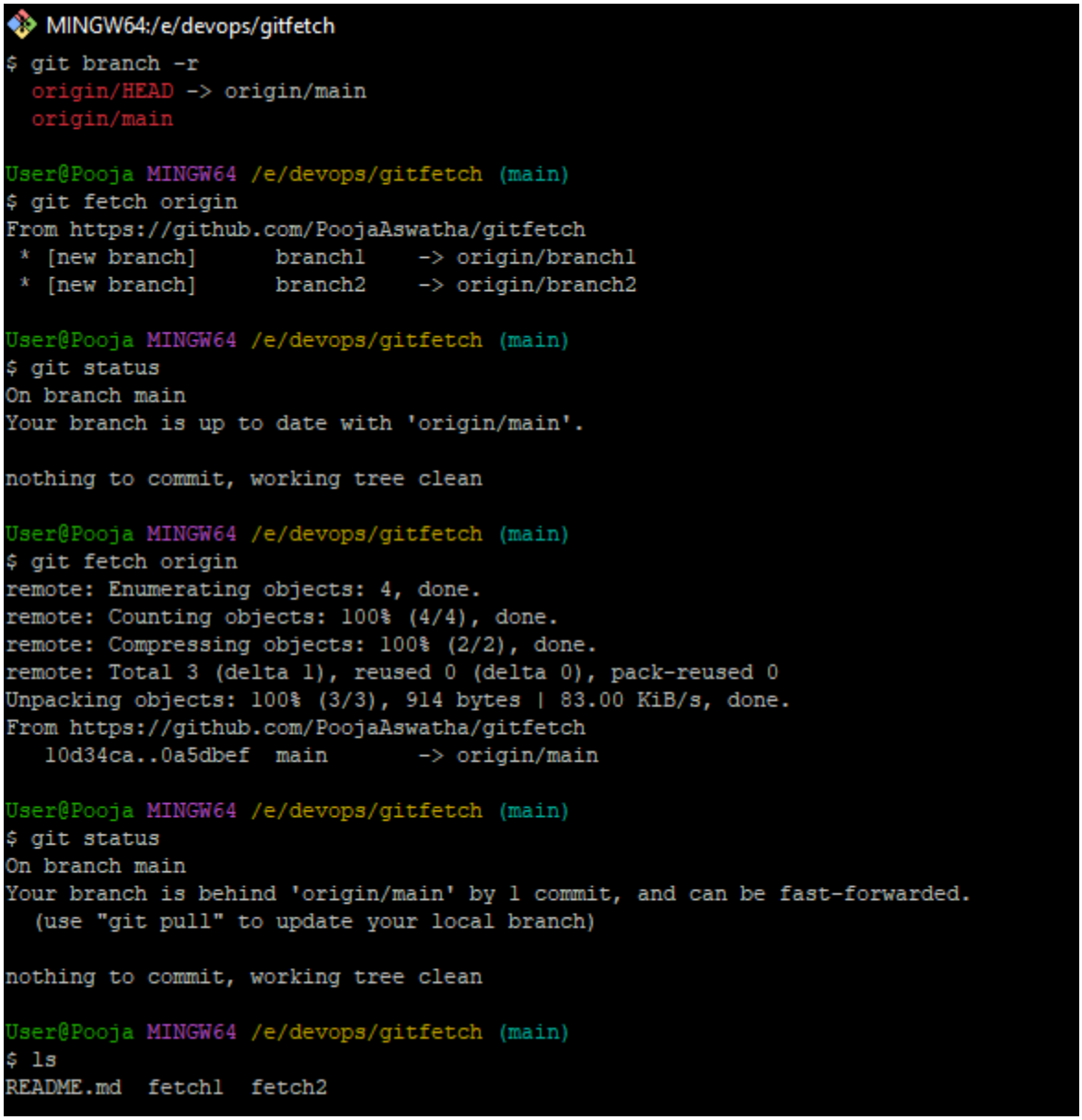
Example:

git fetch origin main

This command downloads changes from the origin remote repository's main branch, but doesn't merge them into your local main branch.




MINGW64:/e/devops/gitfetch
User@Pooja MINGW64 /e/devops
$ git status
fatal: not a git repository (or any of the parent directories): .git
User@Pooja MINGW64 /e/devops
$ git clone https://github.com/PoojaAswatha/gitfetch.git
Cloning into 'gitfetch'...
remote: Enumerating objects: 9, done.
remote: Counting objects: 100% (9/9), done.
remote: Compressing objects: 100% (5/5), done.
remote: Total 9 (delta 1), reused 0 (delta 0), pack-reused 0
Receiving objects: 100% (9/9), done.
Resolving deltas: 100% (1/1), done.
User@Pooja MINGW64 /e/devops
$ cd gitfetch
User@Pooja MINGW64 /e/devops/gitfetch (main)
$ ls
README.md fetch1 fetch2
User@Pooja MINGW64 /e/devops/gitfetch (main)
$ git branch
* main
User@Pooja MINGW64 /e/devops/gitfetch (main)
$ git branch -r
origin/HEAD -> origin/main
origin/main
User@Pooja MINGW64 /e/devops/gitfetch (main)
$ git status
On branch main
Your branch is up to date with 'origin/main'.
nothing to commit, working tree clean
User@Pooja MINGW64 /e/devops/gitfetch (main)
$ git branch -r
origin/HEAD -> origin/main
origin/main


MINGW64:/e/devops/gitfetch
$ git branch -r
origin/HEAD -> origin/main
origin/main
User@Pooja MINGW64 /e/devops/gitfetch (main)
$ git fetch origin
From https://github.com/PoojaAswatha/gitfetch
* [new branch] branch1 -> origin/branch1
* [new branch] branch2 -> origin/branch2
User@Pooja MINGW64 /e/devops/gitfetch (main)
$ git status
On branch main
Your branch is up to date with 'origin/main'.
nothing to commit, working tree clean
User@Pooja MINGW64 /e/devops/gitfetch (main)
$ git fetch origin
remote: Enumerating objects: 4, done.
remote: Counting objects: 100% (4/4), done.
remote: Compressing objects: 100% (2/2), done.
remote: Total 3 (delta 1), reused 0 (delta 0), pack-reused 0
Unpacking objects: 100% (3/3), 914 bytes | 83.00 KiB/s, done.
From https://github.com/PoojaAswatha/gitfetch
10d34ca..0a5dbef main -> origin/main
User@Pooja MINGW64 /e/devops/gitfetch (main)
$ git status
On branch main
Your branch is behind 'origin/main' by 1 commit, and can be fast-forwarded.
(use "git pull" to update your local branch)
nothing to commit, working tree clean
User@Pooja MINGW64 /e/devops/gitfetch (main)
$ ls
README.md fetch1 fetch2

```
MINGW64:/e/devops/gitfetch
$ ls
README.md  fetch1  fetch2

User@Pooja MINGW64 /e/devops/gitfetch (main)
$ git checkout origin/main
Note: switching to 'origin/main'.

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 0a5dbef Create fetch3

User@Pooja MINGW64 /e/devops/gitfetch ((0a5dbef...))
$ ls
README.md  fetch1  fetch2  fetch3

User@Pooja MINGW64 /e/devops/gitfetch ((0a5dbef...))
$ git log --oneline
0a5dbef (HEAD, origin/main, origin/HEAD) Create fetch3
10d34ca (origin/branch2, origin/branch1, main) Create fetch2
e35d82b Create fetch1
385bdb6 Initial commit

User@Pooja MINGW64 /e/devops/gitfetch ((0a5dbef...))
$ git switch main
Previous HEAD position was 0a5dbef Create fetch3
Switched to branch 'main'
Your branch is behind 'origin/main' by 1 commit, and can be fast-forwarded.
  (use "git pull" to update your local branch)
```

```
User@Pooja MINGW64 /e/devops/gitfetch (main)
$ git switch branch1
Switched to a new branch 'branch1'
branch 'branch1' set up to track 'origin/branch1'.

User@Pooja MINGW64 /e/devops/gitfetch (branch1)
$ git status
On branch branch1
Your branch is up to date with 'origin/branch1'.

nothing to commit, working tree clean
```

Git pull:

git pull is a Git command that fetches changes from a remote repository and merges them into your local branch. Here's a breakdown of the command:

git pull [options] [remote-name] [branch-name]

- [options]: Optional flags that modify the behavior of the command
- [remote-name]: The name of the remote repository (e.g., origin)
- [branch-name]: The name of the branch to pull changes into (e.g., main)

Common options:

- --ff-only: Only merges if the fetch results in a fast-forward merge (no conflicts)
- --no-ff: Creates a merge commit even if the fetch results in a fast-forward merge
- --rebase: Rebases your local branch onto the fetched changes instead of merging
- -v or --verbose: Shows more detailed output during the pull process

Git pull = get fetch + git merge

Git pull conflicts:

git pull conflicts occur when changes from the remote repository can't be automatically merged into your local branch. This happens when both you and someone else have made changes to the same file or code.

Common causes of git pull conflicts:

1. Simultaneous edits: Multiple people editing the same file or code simultaneously.

2. Divergent branches: Local and remote branches have diverged, making it difficult to merge changes.

3. Deleted or renamed files: Files deleted or renamed in one branch but not the other.

Resolving git pull conflicts:

1. git status: Check the status of your repository to identify conflicting files.

2. git diff: Review the changes causing the conflict.

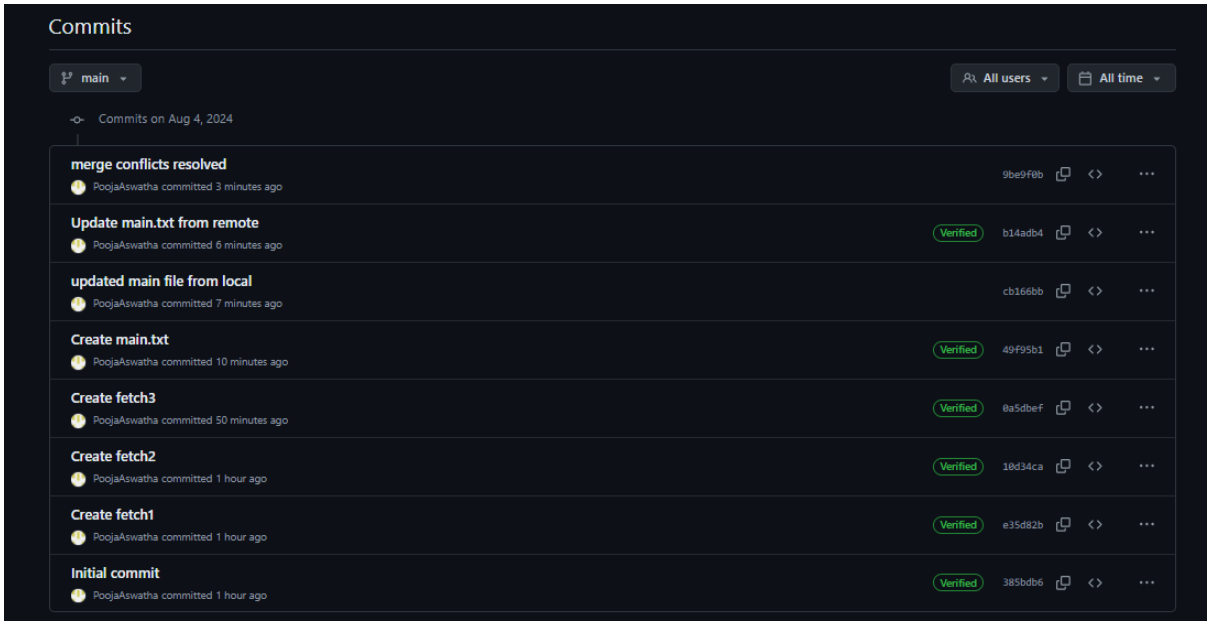
3. Manual edit: Open the conflicting file and manually merge the changes.

4. git add: Stage the resolved file.

5. git commit: Commit the resolved file with a meaningful message.

6. git pull --rebase: Rebase your local changes onto the updated remote branch.

7. git push: Push your resolved changes to the remote repository.


Commits
main
All users
All time
Commits on Aug 4, 2024
merge conflicts resolved
PoojaAswatha committed 3 minutes ago
9be9f0b
Update main.txt from remote
PoojaAswatha committed 6 minutes ago
Verified
b14adb4
updated main file from local
PoojaAswatha committed 7 minutes ago
cb166bb
Create main.txt
PoojaAswatha committed 10 minutes ago
Verified
49f95b1
Create fetch3
PoojaAswatha committed 50 minutes ago
Verified
0a5dbef
Create fetch2
PoojaAswatha committed 1 hour ago
Verified
10d34ca
Create fetch1
PoojaAswatha committed 1 hour ago
Verified
e35d82b
Initial commit
PoojaAswatha committed 1 hour ago
Verified
385bdb6

```
MINGW64:/e/devops/gitfetch

User@Pooja MINGW64 /e/devops/gitfetch (main)
$ ls
README.md  fetch1  fetch2

User@Pooja MINGW64 /e/devops/gitfetch (main)
$ git fetch
remote: Enumerating objects: 4, done.
remote: Counting objects: 100% (4/4), done.
remote: Compressing objects: 100% (2/2), done.
remote: Total 3 (delta 1), reused 0 (delta 0), pack-reused 0
Unpacking objects: 100% (3/3), 917 bytes | 91.00 KiB/s, done.
From https://github.com/PoojaAswatha/gitfetch
   0a5dbef..49f95b1  main       -> origin/main

User@Pooja MINGW64 /e/devops/gitfetch (main)
$ git status
On branch main
Your branch is behind 'origin/main' by 2 commits, and can be fast-forwarded.
  (use "git pull" to update your local branch)

nothing to commit, working tree clean

User@Pooja MINGW64 /e/devops/gitfetch (main)
$ git pull origin main
From https://github.com/PoojaAswatha/gitfetch
 * branch            main       -> FETCH_HEAD
Updating 10d34ca..49f95b1
Fast-forward
 fetch3   | 1 +
 main.txt | 1 +
 2 files changed, 2 insertions(+)
 create mode 100644 fetch3
 create mode 100644 main.txt

User@Pooja MINGW64 /e/devops/gitfetch (main)
$ ls
README.md  fetch1  fetch2  fetch3  main.txt

User@Pooja MINGW64 /e/devops/gitfetch (main)
$ git log --oneline
49f95b1 (HEAD -> main, origin/main, origin/HEAD) Create main.txt
0a5dbef Create fetch3
10d34ca (origin/branch2, origin/branch1, branch1) Create fetch2
e35d82b Create fetch1
385bdb6 Initial commit
```

```
MINGW64:/e/devops/gitfetch
385bdb6 Initial commit

User@Pooja MINGW64 /e/devops/gitfetch (main)
$ vi main.txt

User@Pooja MINGW64 /e/devops/gitfetch (main)
$ git add .
g
User@Pooja MINGW64 /e/devops/gitfetch (main)
$ git commit -m "updated main file from local"
[main cb166bb] updated main file from local
 1 file changed, 1 insertion(+)

User@Pooja MINGW64 /e/devops/gitfetch (main)
$ git status
On branch main
Your branch is ahead of 'origin/main' by 1 commit.
  (use "git push" to publish your local commits)

nothing to commit, working tree clean

User@Pooja MINGW64 /e/devops/gitfetch (main)
$ git fetch
remote: Enumerating objects: 5, done.
remote: Counting objects: 100% (5/5), done.
remote: Compressing objects: 100% (2/2), done.
remote: Total 3 (delta 1), reused 0 (delta 0), pack-reused 0
Unpacking objects: 100% (3/3), 929 bytes | 77.00 KiB/s, done.
From https://github.com/PoojaAswatha/gitfetch
   49f95b1..b14adb4  main       -> origin/main

User@Pooja MINGW64 /e/devops/gitfetch (main)
$ git pull origin main
From https://github.com/PoojaAswatha/gitfetch
 * branch            main       -> FETCH_HEAD
Auto-merging main.txt
CONFLICT (content): Merge conflict in main.txt
Automatic merge failed; fix conflicts and then commit the result.

User@Pooja MINGW64 /e/devops/gitfetch (main|MERGING)
$ git status
On branch main
Your branch and 'origin/main' have diverged,
and have 1 and 1 different commits each, respectively.
  (use "git pull" if you want to integrate the remote branch with yours)
```

```
MINGW64:/e/devops/gitfetch
User@Pooja MINGW64 /e/devops/gitfetch (main|MERGING)
$ git status
On branch main
Your branch and 'origin/main' have diverged,
and have 1 and 1 different commits each, respectively.
  (use "git pull" if you want to integrate the remote branch with yours)

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 modified:   main.txt

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

User@Pooja MINGW64 /e/devops/gitfetch (main|MERGING)
$ vi main.txt

User@Pooja MINGW64 /e/devops/gitfetch (main|MERGING)
$ git add .

User@Pooja MINGW64 /e/devops/gitfetch (main|MERGING)
$ git commit -m "merge conflicts resolved"
[main 9be9f0b] merge conflicts resolved

User@Pooja MINGW64 /e/devops/gitfetch (main)
$ git status
On branch main
Your branch is ahead of 'origin/main' by 2 commits.
  (use "git push" to publish your local commits)

nothing to commit, working tree clean

User@Pooja MINGW64 /e/devops/gitfetch (main)
$ git log --oneline
9be9f0b (HEAD -> main) merge conflicts resolved
b14adb4 (origin/main, origin/HEAD) Update main.txt from remote
cb166bb updated main file from local
49f95b1 Create main.txt
0a5dbef Create fetch3
10d34ca (origin/branch2, origin/branch1, branch1) Create fetch2
e35d82b Create fetch1
385bdb6 Initial commit
```

```
User@Pooja MINGW64 /e/devops/gitfetch (main)
$ git push origin main
Enumerating objects: 10, done.
Counting objects: 100% (10/10), done.
Delta compression using up to 4 threads
Compressing objects: 100% (5/5), done.
Writing objects: 100% (6/6), 604 bytes | 302.00 KiB/s, done.
Total 6 (delta 2), reused 0 (delta 0), pack-reused 0 (from 0)
remote: Resolving deltas: 100% (2/2), completed with 1 local object.
To https://github.com/PoojaAswatha/gitfetch.git
   b14adb4..9be9f0b  main -> main
```

Difference between Git Fetch and Git Pull

Git Fetch	Git Pull
Used to fetch all changes from the remote repository to the local repository without merging into the current working directory	Brings the copy of all the changes from a remote repository and merges them into the current working directory
Repository data is updated in the .git directory	The working directory is updated directly
Review of commits and changes can be done	Updates the changes to the local repository immediately.
No possibility of merge conflicts.	Merge conflicts are possible if the remote and the local repositories have done changes at the same place.
Command for Git fetch is git fetch<remote>	Command for Git Pull is git pull<remote><branch>
Git fetch basically imports the commits to local branches so as to keep up-to-date that what everybody is working on.	Git Pull basically brings the local branch up-to-date with the remote copy that will also updates the other remote tracking branches.

Pull request:

A pull request (PR) is a way to propose changes to a repository's codebase. It allows you to:

1. Notify team members of changes
2. Review code before merging
3. Discuss changes with team members
4. Ensure code quality and consistency

Pull request workflow:

1. Create a new branch for your changes
2. Commit and push your changes to the remote repository
3. Go to the repository's web interface (e.g., GitHub, GitLab)
4. Click "New pull request" or "Create pull request"
5. Select the branch you want to merge into (e.g., main)
6. Review and describe your changes

7. Submit the pull request

Pull request benefits:

1. Code review: Ensure code quality and consistency

2. Collaboration: Discuss changes with team members

3. Version control: Track changes and updates

4. Testing: Test changes before merging

5. Documentation: Provide context for changes

Pull request states:

1. Open: Waiting for review and approval

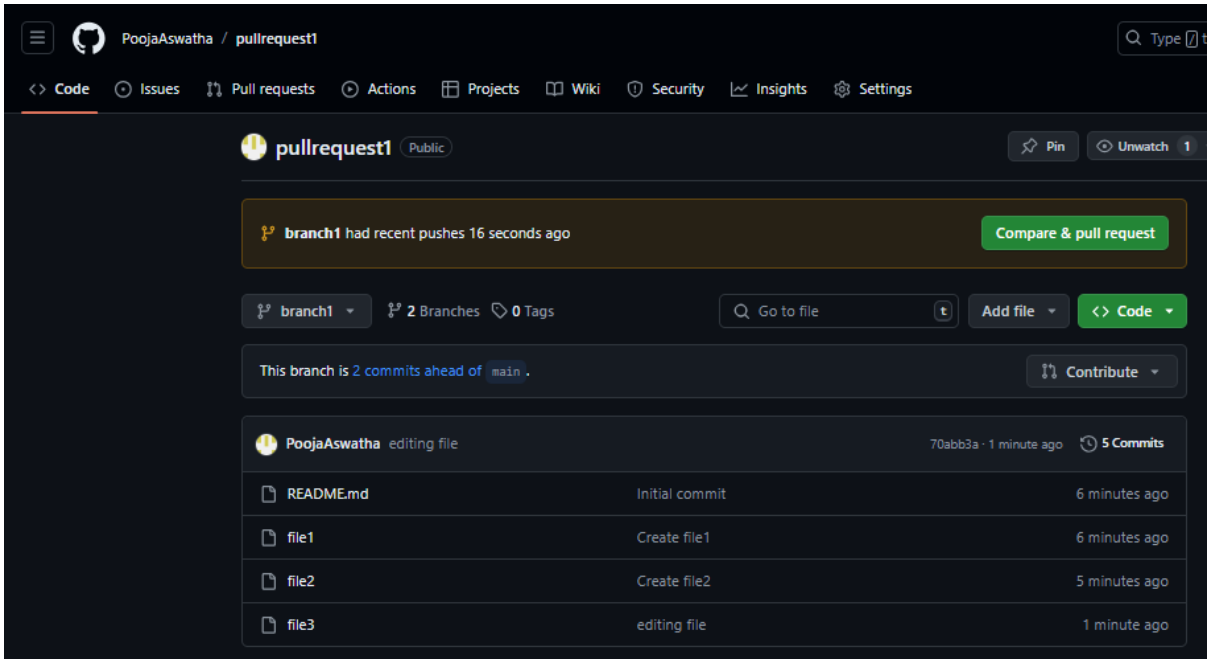
2. In progress: Being reviewed and discussed

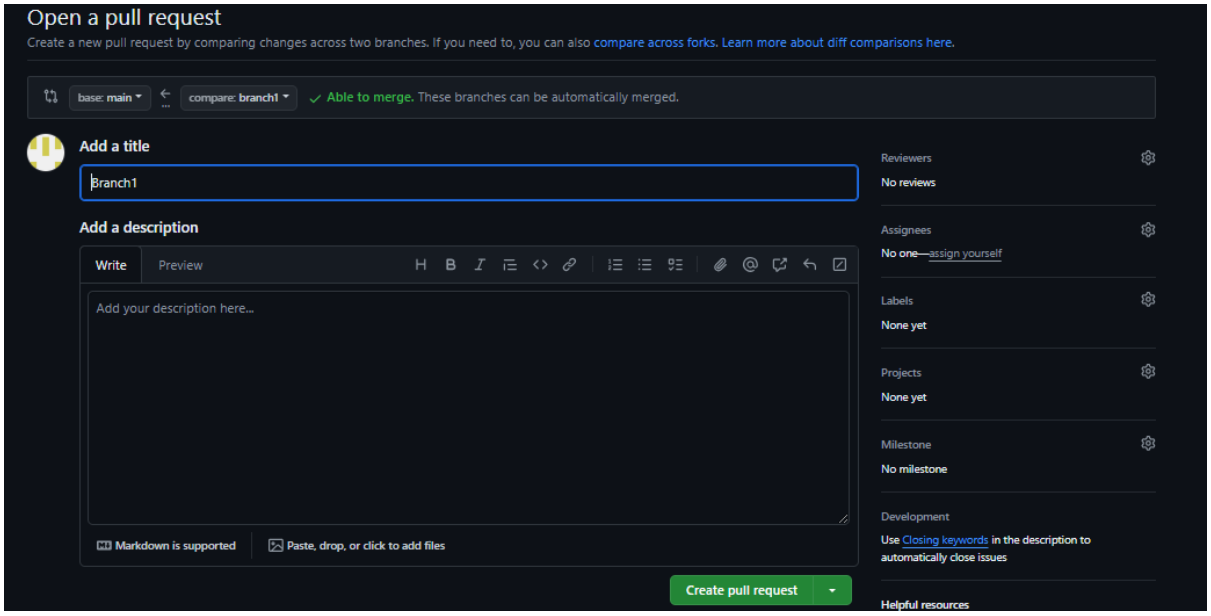
3. Approved: Ready to be merged

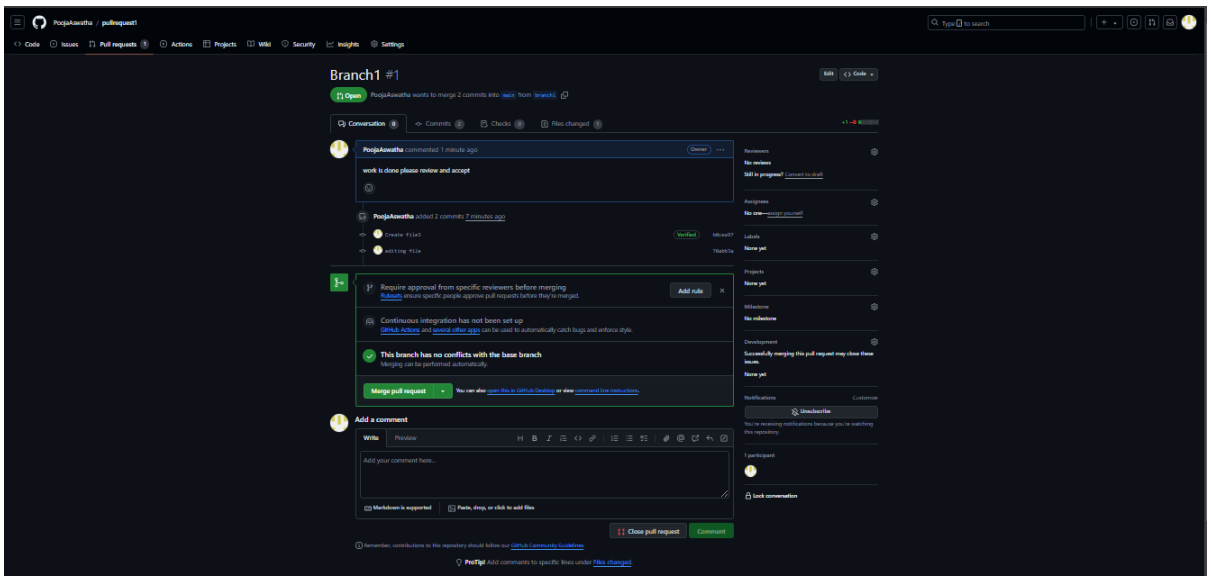
4. Merged: Changes have been merged into the target branch

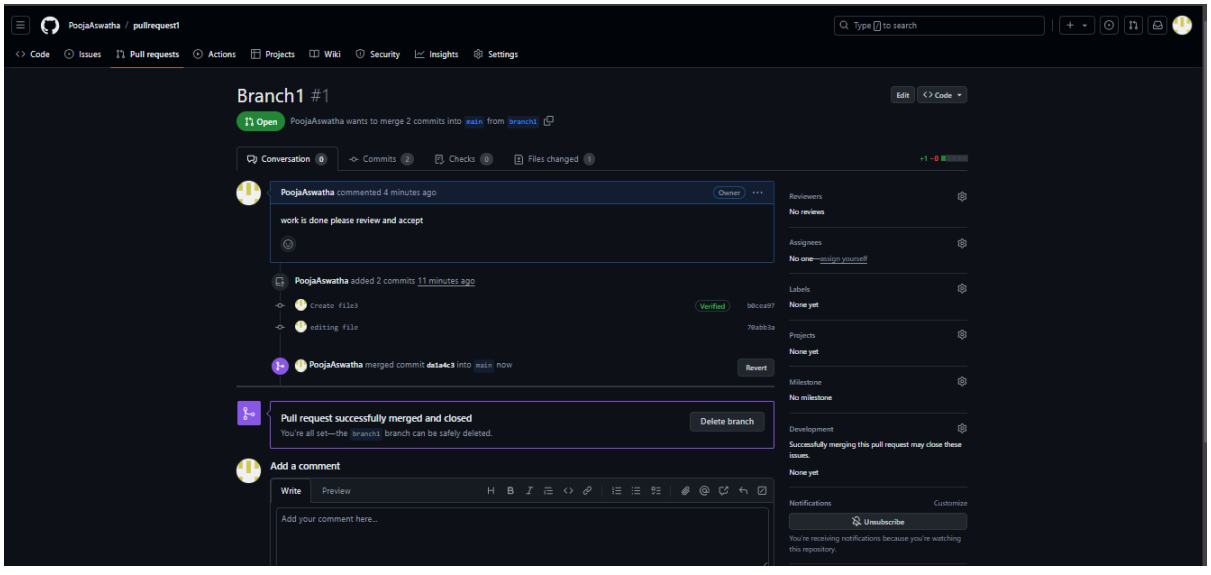
5. Closed: Pull request has been rejected or abandoned

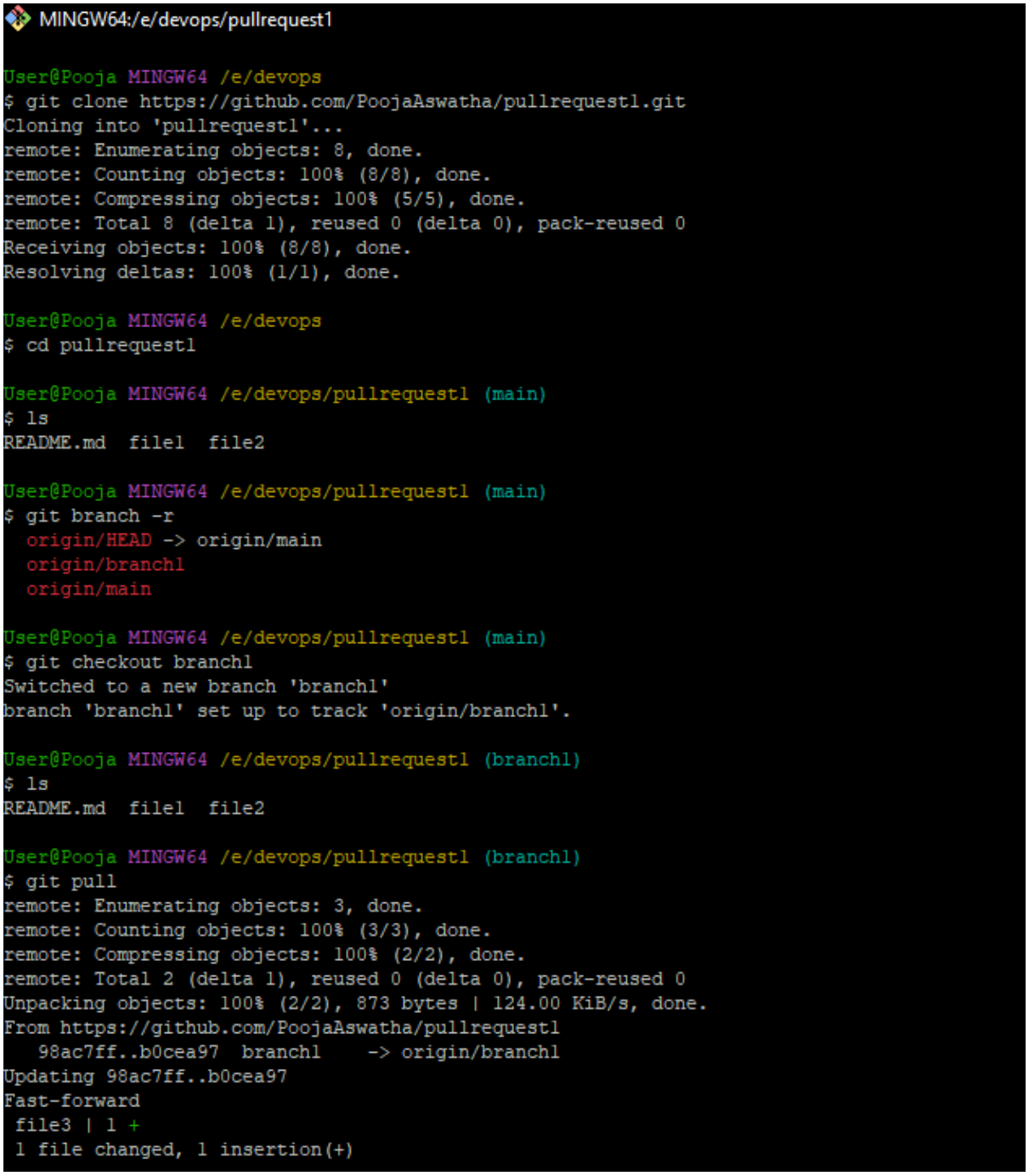
Pull requests are an essential tool for collaborative development, ensuring that changes are reviewed, tested, and approved before being merged into the main codebase.

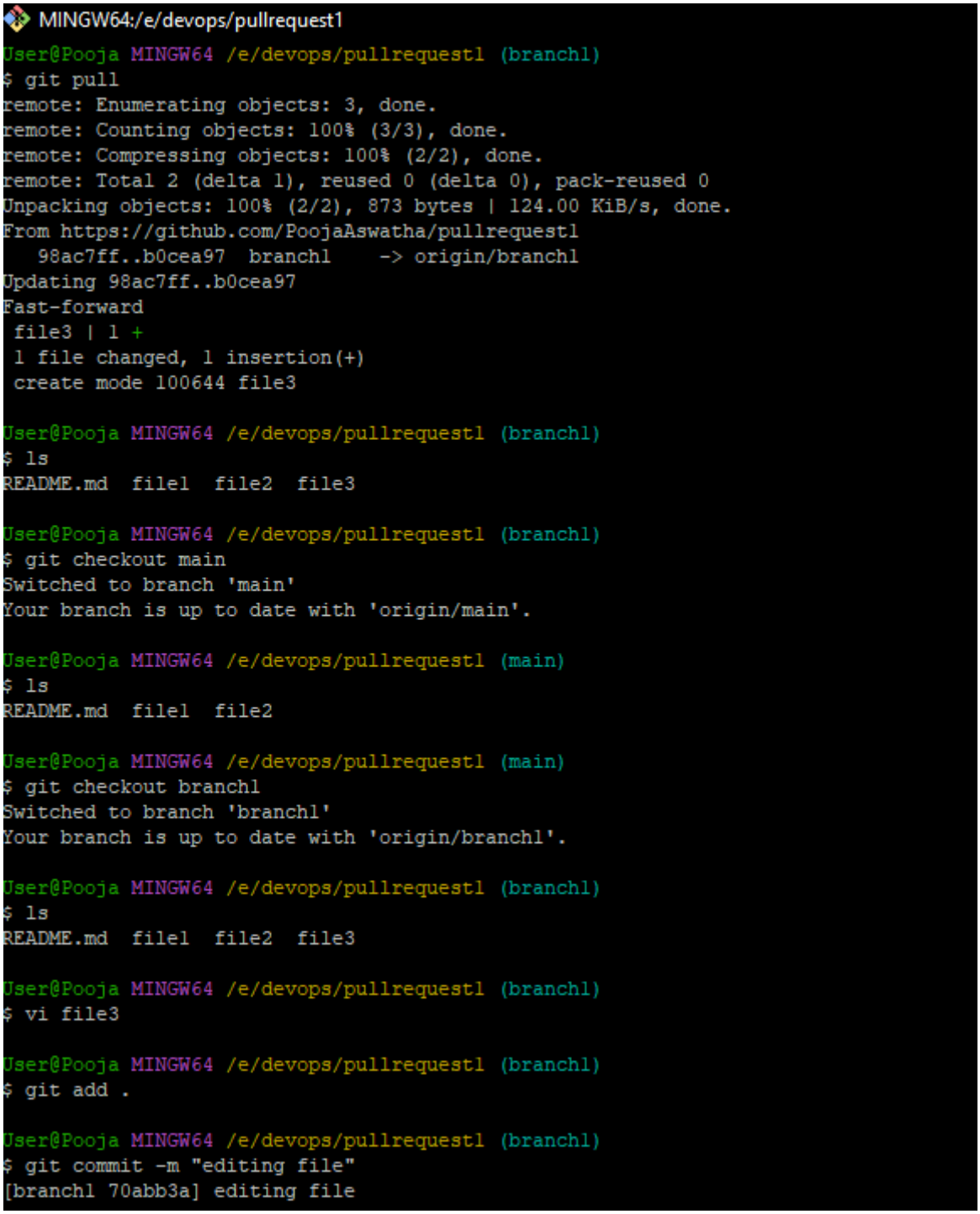


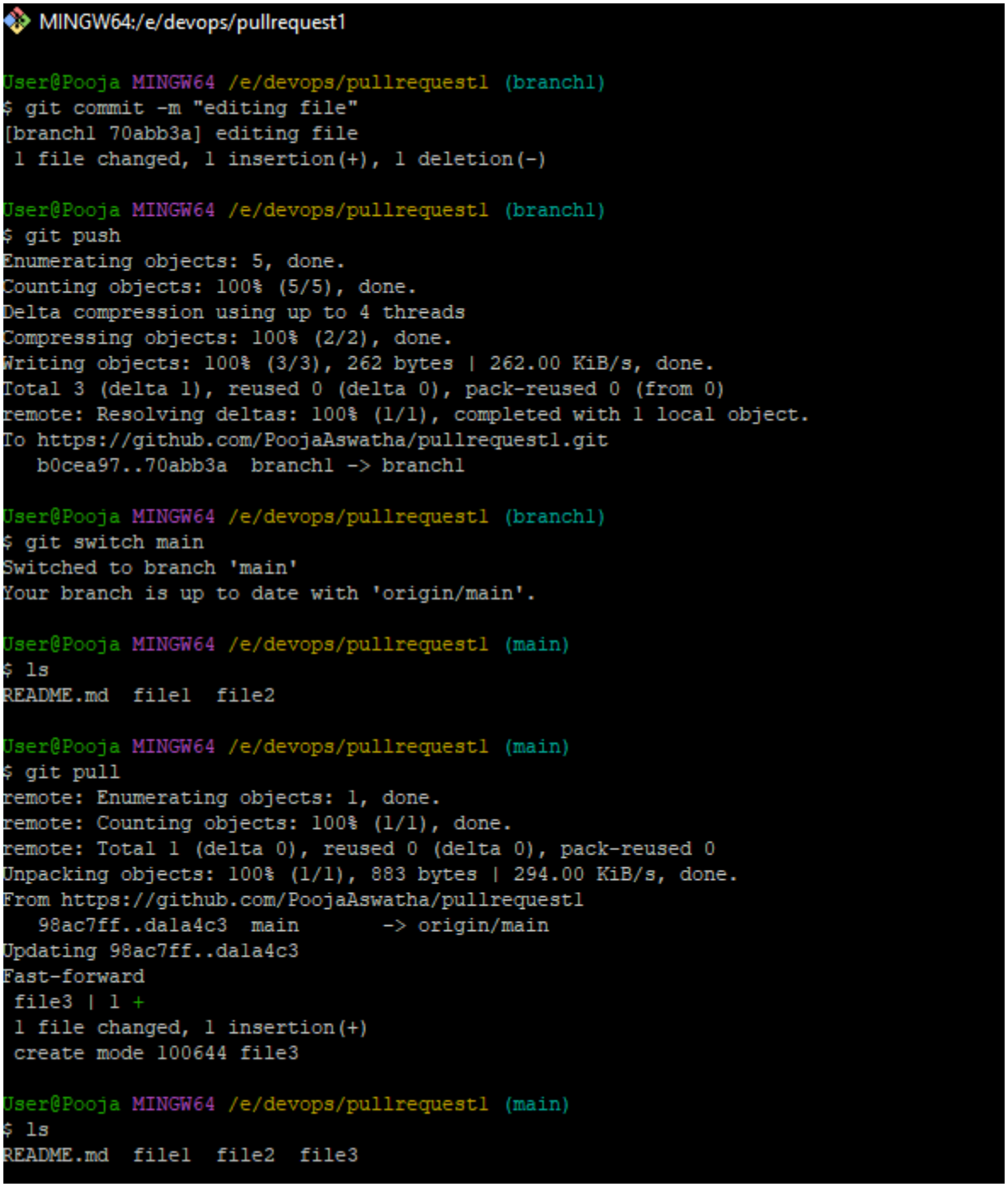

Open a pull request
Create a new pull request by comparing changes across two branches. If you need to, you can also compare across forks. Learn more about diff comparisons here.
base: main
compare: branch1
Able to merge. These branches can be automatically merged.
Add a title
Branch1
Add a description
Write
Preview
Add your description here...
Markdown is supported
Paste, drop, or click to add files
Create pull request
Reviewers
No reviews
Assignees
No one—assign yourself
Labels
None yet
Projects
None yet
Milestone
No milestone
Development
Use Closing keywords in the description to automatically close issues
Helpful resources


Branch1 #1
Open
PoojaAswatha wants to merge 2 commits into main from branch1
Conversation
Commits
Checks
Files changed
PoojaAswatha commented 1 minute ago
work is done please review and accept
PoojaAswatha added 2 commits 7 minutes ago
Require approval from specific reviewers before merging
Add rule
Continuous integration has not been set up
This branch has no conflicts with the base branch
Merge pull request
Add a comment
Write
Preview
Add your comment here...
Close pull request
Comment


PoojaAswatha / pullrequest1
Code
Issues
Pull requests
Actions
Projects
Wiki
Security
Insights
Settings
Branch1 #1
Open
PoojaAswatha wants to merge 2 commits into main from branch1
Edit
Code
Conversation
Commits
Checks
Files changed
PoojaAswatha commented 4 minutes ago
work is done please review and accept
PoojaAswatha added 2 commits 11 minutes ago
Create file3
editing file
Verified
PoojaAswatha merged commit da1a4c3 into main now
Revert
Pull request successfully merged and closed
You're all set—the branch1 branch can be safely deleted.
Delete branch
Add a comment
Write
Preview
Add your comment here...
Reviewers
No reviews
Assignees
No one—assign yourself
Labels
None yet
Projects
None yet
Milestone
No milestone
Development
Successfully merging this pull request may close these issues.
None yet
Notifications
Customize
Unsubscribe


MINGW64:/e/devops/pullrequest1
User@Pooja MINGW64 /e/devops
$ git clone https://github.com/PoojaAswatha/pullrequest1.git
Cloning into 'pullrequest1'...
remote: Enumerating objects: 8, done.
remote: Counting objects: 100% (8/8), done.
remote: Compressing objects: 100% (5/5), done.
remote: Total 8 (delta 1), reused 0 (delta 0), pack-reused 0
Receiving objects: 100% (8/8), done.
Resolving deltas: 100% (1/1), done.
User@Pooja MINGW64 /e/devops
$ cd pullrequest1
User@Pooja MINGW64 /e/devops/pullrequest1 (main)
$ ls
README.md file1 file2
User@Pooja MINGW64 /e/devops/pullrequest1 (main)
$ git branch -r
origin/HEAD -> origin/main
origin/branch1
origin/main
User@Pooja MINGW64 /e/devops/pullrequest1 (main)
$ git checkout branch1
Switched to a new branch 'branch1'
branch 'branch1' set up to track 'origin/branch1'.
User@Pooja MINGW64 /e/devops/pullrequest1 (branch1)
$ ls
README.md file1 file2
User@Pooja MINGW64 /e/devops/pullrequest1 (branch1)
$ git pull
remote: Enumerating objects: 3, done.
remote: Counting objects: 100% (3/3), done.
remote: Compressing objects: 100% (2/2), done.
remote: Total 2 (delta 1), reused 0 (delta 0), pack-reused 0
Unpacking objects: 100% (2/2), 873 bytes | 124.00 KiB/s, done.
From https://github.com/PoojaAswatha/pullrequest1
98ac7ff..b0cea97 branch1 -> origin/branch1
Updating 98ac7ff..b0cea97
Fast-forward
file3 | 1 +
1 file changed, 1 insertion(+)


MINGW64:/e/devops/pullrequest1
User@Pooja MINGW64 /e/devops/pullrequest1 (branch1)
$ git pull
remote: Enumerating objects: 3, done.
remote: Counting objects: 100% (3/3), done.
remote: Compressing objects: 100% (2/2), done.
remote: Total 2 (delta 1), reused 0 (delta 0), pack-reused 0
Unpacking objects: 100% (2/2), 873 bytes | 124.00 KiB/s, done.
From https://github.com/PoojaAswatha/pullrequest1
98ac7ff..b0cea97 branch1 -> origin/branch1
Updating 98ac7ff..b0cea97
Fast-forward
file3 | 1 +
1 file changed, 1 insertion(+)
create mode 100644 file3
User@Pooja MINGW64 /e/devops/pullrequest1 (branch1)
$ ls
README.md file1 file2 file3
User@Pooja MINGW64 /e/devops/pullrequest1 (branch1)
$ git checkout main
Switched to branch 'main'
Your branch is up to date with 'origin/main'.
User@Pooja MINGW64 /e/devops/pullrequest1 (main)
$ ls
README.md file1 file2
User@Pooja MINGW64 /e/devops/pullrequest1 (main)
$ git checkout branch1
Switched to branch 'branch1'
Your branch is up to date with 'origin/branch1'.
User@Pooja MINGW64 /e/devops/pullrequest1 (branch1)
$ ls
README.md file1 file2 file3
User@Pooja MINGW64 /e/devops/pullrequest1 (branch1)
$ vi file3
User@Pooja MINGW64 /e/devops/pullrequest1 (branch1)
$ git add .
User@Pooja MINGW64 /e/devops/pullrequest1 (branch1)
$ git commit -m "editing file"
[branch1 70abb3a] editing file



Git Fork:

git fork is a Git feature that allows you to create a copy of a repository, making it possible to:

1. Experiment with changes without affecting the original repository.

2. Contribute to open-source projects without direct write access.

3. Create a personalized version of a repository.

Forking a repository:

1. Go to the repository's web interface (e.g., GitHub, GitLab).

2. Click the "Fork" button.

3. Choose where to fork the repository (e.g., your personal account).

4. Wait for the forking process to complete.

Key aspects of forking:

1. The forked repository is a separate copy of the original.

2. Changes made to the forked repository don't affect the original.

3. You can sync your forked repository with the original using git pull and git merge.

4. You can submit pull requests from your forked repository to the original.

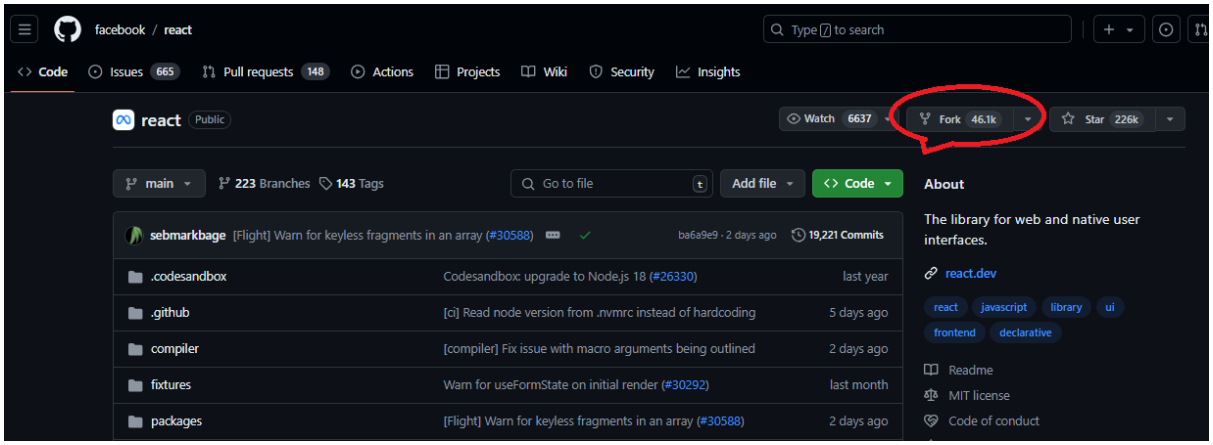
Common use cases for forking:

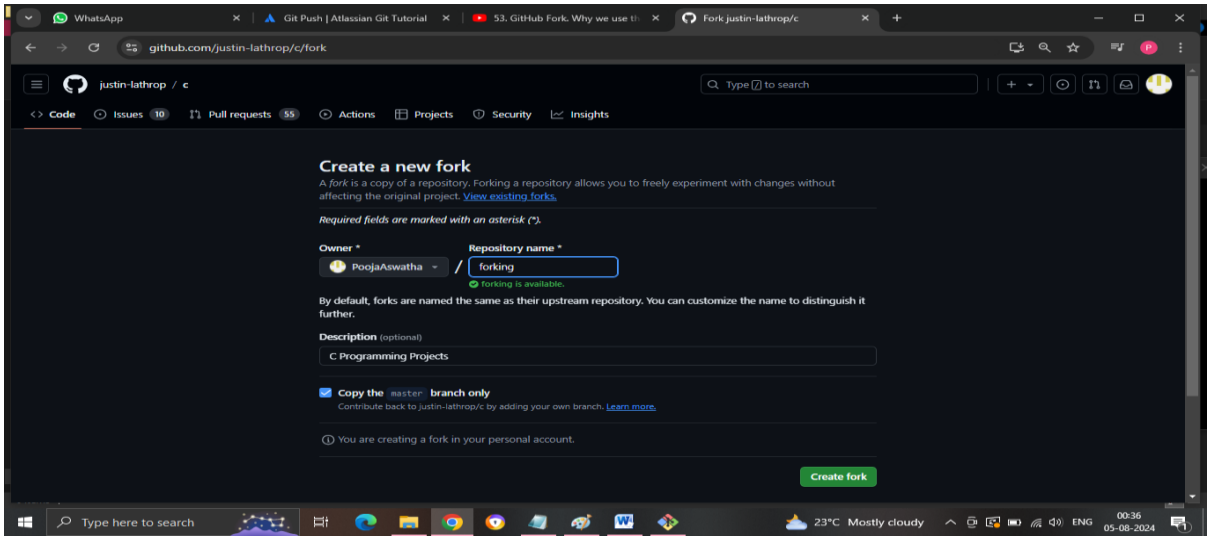
1. Contributing to open-source projects.

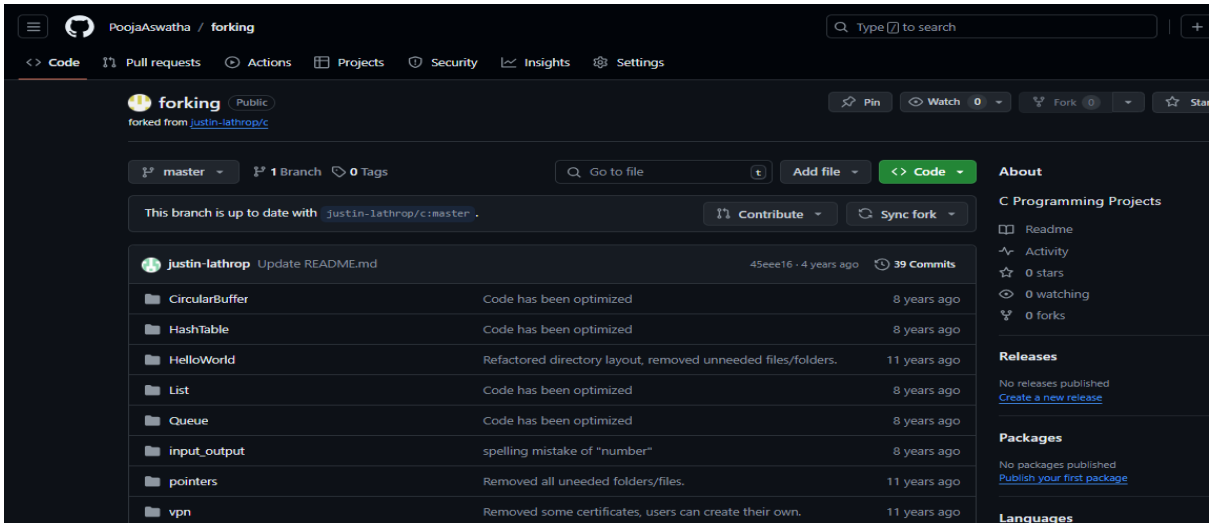
2. Creating a personalized version of a repository.

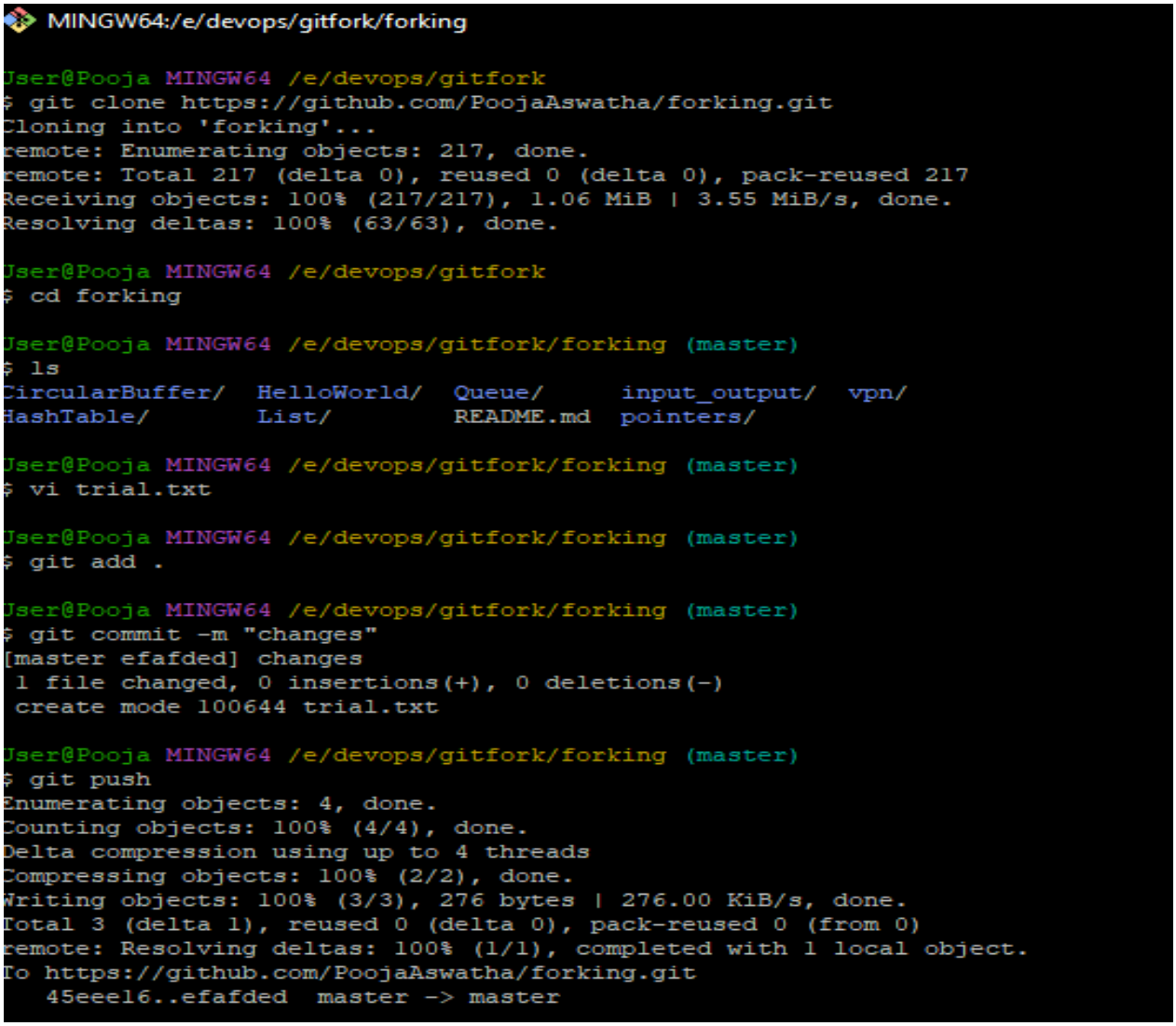
3. Experimenting with changes without affecting the original.

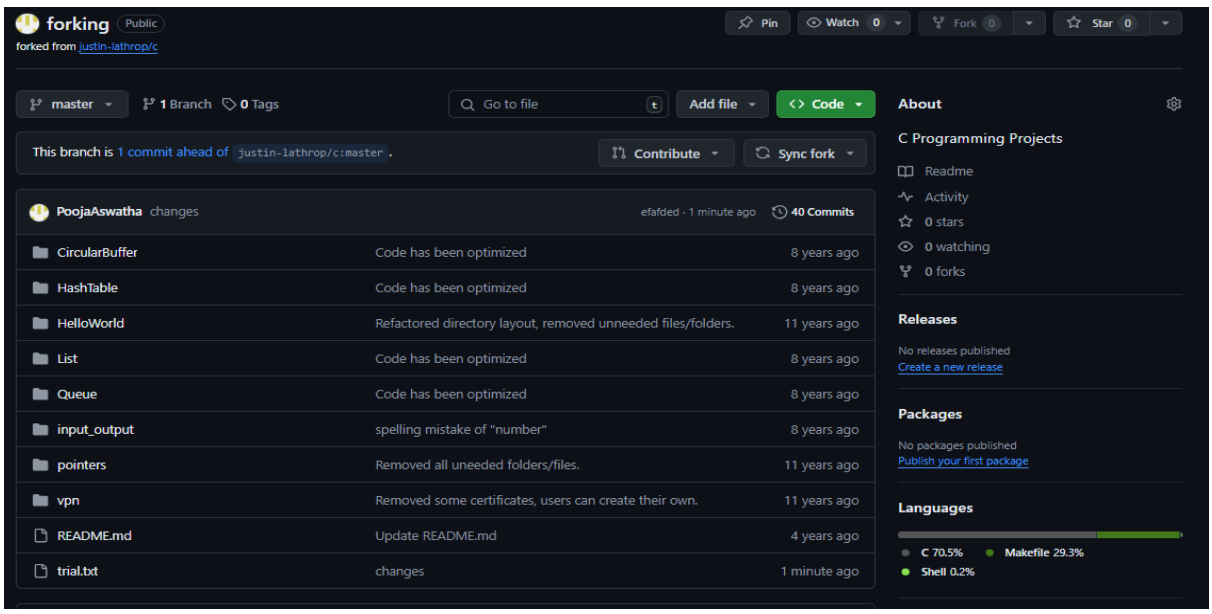
4. Learning from others by forking and studying their repositories.




WhatsApp
Git Push | Atlassian Git Tutorial
53. GitHub Fork. Why we use th
Fork justin-lathrop/c
github.com/justin-lathrop/c/fork
justin-lathrop / c
Type / to search
Code
Issues 10
Pull requests 55
Actions
Projects
Security
Insights
Create a new fork
A fork is a copy of a repository. Forking a repository allows you to freely experiment with changes without affecting the original project. View existing forks.
Required fields are marked with an asterisk (*).
Owner *
PoojaAswatha
Repository name *
forking
forking is available.
By default, forks are named the same as their upstream repository. You can customize the name to distinguish it further.
Description (optional)
C Programming Projects
Copy the master branch only
Contribute back to justin-lathrop/c by adding your own branch. Learn more.
You are creating a fork in your personal account.
Create fork
Type here to search
23°C Mostly cloudy
ENG
00:36
05-08-2024


PoojaAswatha / forking
Type / to search
Code
Pull requests
Actions
Projects
Security
Insights
Settings
forking
Public
forked from justin-lathrop/c
Pin
Watch 0
Fork 0
master
1 Branch
0 Tags
Go to file
t
Add file
Code
This branch is up to date with justin-lathrop/c:master.
Contribute
Sync fork
justin-lathrop Update README.md
45eee16 · 4 years ago
39 Commits
CircularBuffer Code has been optimized 8 years ago
HashTable Code has been optimized 8 years ago
HelloWorld Refactored directory layout, removed unneeded files/folders. 11 years ago
List Code has been optimized 8 years ago
Queue Code has been optimized 8 years ago
input_output spelling mistake of "number" 8 years ago
pointers Removed all uneeded folders/files. 11 years ago
vpn Removed some certificates, users can create their own. 11 years ago
About
C Programming Projects
Readme
Activity
0 stars
0 watching
0 forks
Releases
No releases published
Create a new release
Packages
No packages published
Publish your first package
Languages


MINGW64:/e/devops/gitfork/forking
User@Pooja MINGW64 /e/devops/gitfork
$ git clone https://github.com/PoojaAswatha/forking.git
Cloning into 'forking'...
remote: Enumerating objects: 217, done.
remote: Total 217 (delta 0), reused 0 (delta 0), pack-reused 217
Receiving objects: 100% (217/217), 1.06 MiB | 3.55 MiB/s, done.
Resolving deltas: 100% (63/63), done.
User@Pooja MINGW64 /e/devops/gitfork
$ cd forking
User@Pooja MINGW64 /e/devops/gitfork/forking (master)
$ ls
CircularBuffer/ HelloWorld/ Queue/ input_output/ vpn/
HashTable/ List/ README.md pointers/
User@Pooja MINGW64 /e/devops/gitfork/forking (master)
$ vi trial.txt
User@Pooja MINGW64 /e/devops/gitfork/forking (master)
$ git add .
User@Pooja MINGW64 /e/devops/gitfork/forking (master)
$ git commit -m "changes"
[master efafded] changes
1 file changed, 0 insertions(+), 0 deletions(-)
create mode 100644 trial.txt
User@Pooja MINGW64 /e/devops/gitfork/forking (master)
$ git push
Enumerating objects: 4, done.
Counting objects: 100% (4/4), done.
Delta compression using up to 4 threads
Compressing objects: 100% (2/2), done.
Writing objects: 100% (3/3), 276 bytes | 276.00 KiB/s, done.
Total 3 (delta 1), reused 0 (delta 0), pack-reused 0 (from 0)
remote: Resolving deltas: 100% (1/1), completed with 1 local object.
To https://github.com/PoojaAswatha/forking.git
45eeel6..efafded master -> master


forking
Public
forked from justin-lathrop/c
Pin
Watch 0
Fork 0
Star 0
master
1 Branch
0 Tags
Go to file
Add file
Code
This branch is 1 commit ahead of justin-lathrop/c:master.
Contribute
Sync fork
PoojaAswatha changes
efafded · 1 minute ago
40 Commits
CircularBuffer Code has been optimized 8 years ago
HashTable Code has been optimized 8 years ago
HelloWorld Refactored directory layout, removed unneeded files/folders. 11 years ago
List Code has been optimized 8 years ago
Queue Code has been optimized 8 years ago
input_output spelling mistake of "number" 8 years ago
pointers Removed all uneeded folders/files. 11 years ago
vpn Removed some certificates, users can create their own. 11 years ago
README.md Update README.md 4 years ago
trial.txt changes 1 minute ago
About
C Programming Projects
Readme
Activity
0 stars
0 watching
0 forks
Releases
No releases published
Create a new release
Packages
No packages published
Publish your first package
Languages
C 70.5%
Makefile 29.3%
Shell 0.2%

Git reflog:

git reflog is a Git command that displays a log of all reference updates made to the local repository. It shows a record of all changes made to branches, tags, and other references.

Common uses of git reflog:

1. View commit history: See a list of commits made to the repository.

2. Find lost commits: Recover commits that were accidentally deleted or overwritten.

3. Check reference updates: Verify changes made to branches, tags, and other references.

4. Debug issues: Investigate issues by examining the reference update history.

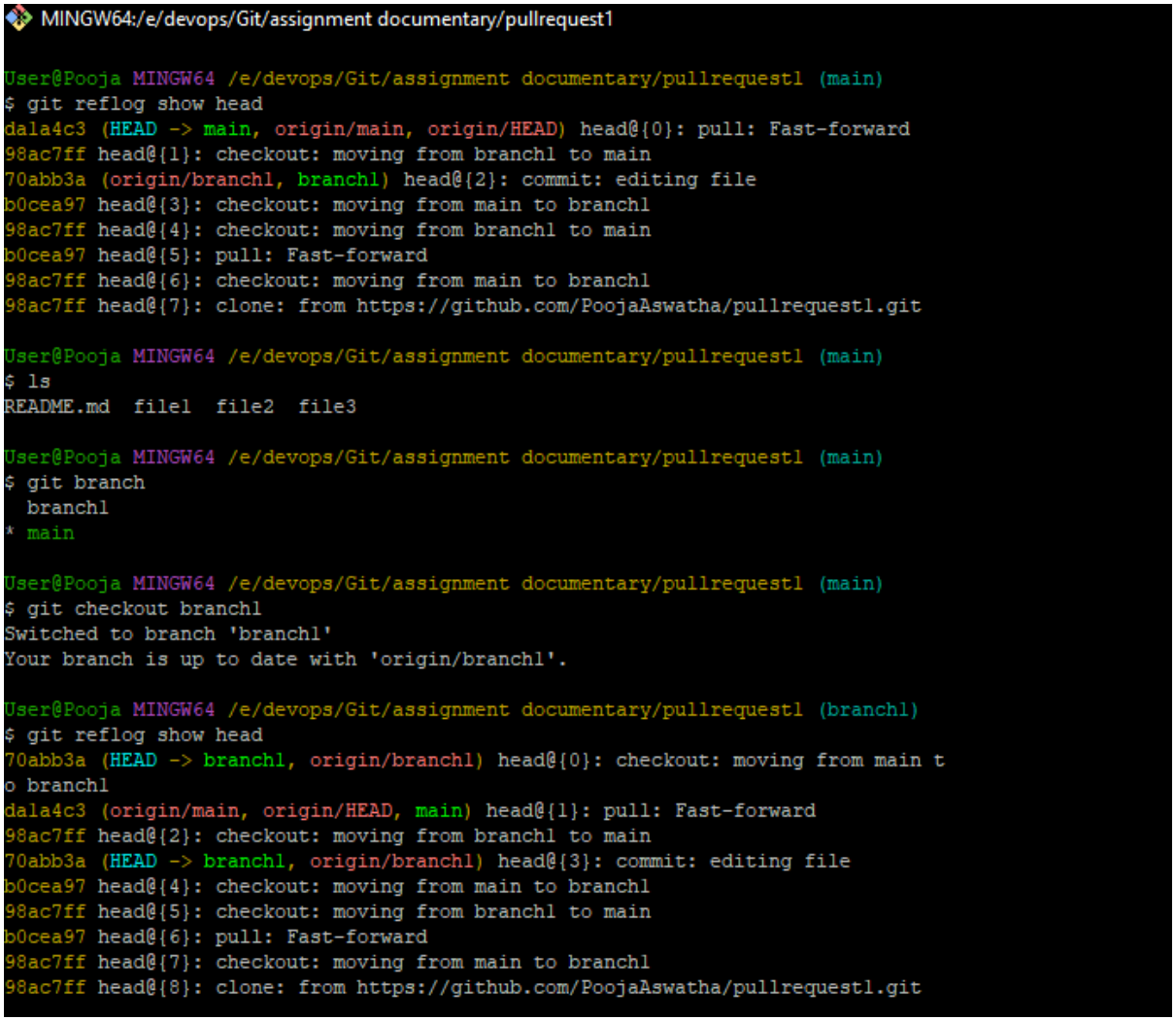
git reflog options:

1. --all: Show all references, including branches, tags, and others.

2. --date: Display dates in a specific format.

3. --pretty: Format the output using a specific format (e.g., --pretty=oneline).

4. --grep: Search for specific commits or references using a regular expression**.**


MINGW64:/e/devops/Git/assignment documentary/pullrequest1
User@Pooja MINGW64 /e/devops/Git/assignment documentary/pullrequest1 (main)
$ git reflog show head
dala4c3 (HEAD -> main, origin/main, origin/HEAD) head@{0}: pull: Fast-forward
98ac7ff head@{1}: checkout: moving from branch1 to main
70abb3a (origin/branch1, branch1) head@{2}: commit: editing file
b0cea97 head@{3}: checkout: moving from main to branch1
98ac7ff head@{4}: checkout: moving from branch1 to main
b0cea97 head@{5}: pull: Fast-forward
98ac7ff head@{6}: checkout: moving from main to branch1
98ac7ff head@{7}: clone: from https://github.com/PoojaAswatha/pullrequest1.git
User@Pooja MINGW64 /e/devops/Git/assignment documentary/pullrequest1 (main)
$ ls
README.md file1 file2 file3
User@Pooja MINGW64 /e/devops/Git/assignment documentary/pullrequest1 (main)
$ git branch
branch1
* main
User@Pooja MINGW64 /e/devops/Git/assignment documentary/pullrequest1 (main)
$ git checkout branch1
Switched to branch 'branch1'
Your branch is up to date with 'origin/branch1'.
User@Pooja MINGW64 /e/devops/Git/assignment documentary/pullrequest1 (branch1)
$ git reflog show head
70abb3a (HEAD -> branch1, origin/branch1) head@{0}: checkout: moving from main t
o branch1
dala4c3 (origin/main, origin/HEAD, main) head@{1}: pull: Fast-forward
98ac7ff head@{2}: checkout: moving from branch1 to main
70abb3a (HEAD -> branch1, origin/branch1) head@{3}: commit: editing file
b0cea97 head@{4}: checkout: moving from main to branch1
98ac7ff head@{5}: checkout: moving from branch1 to main
b0cea97 head@{6}: pull: Fast-forward
98ac7ff head@{7}: checkout: moving from main to branch1
98ac7ff head@{8}: clone: from https://github.com/PoojaAswatha/pullrequest1.git

---