GIT

<u>GIT Architecture:</u>

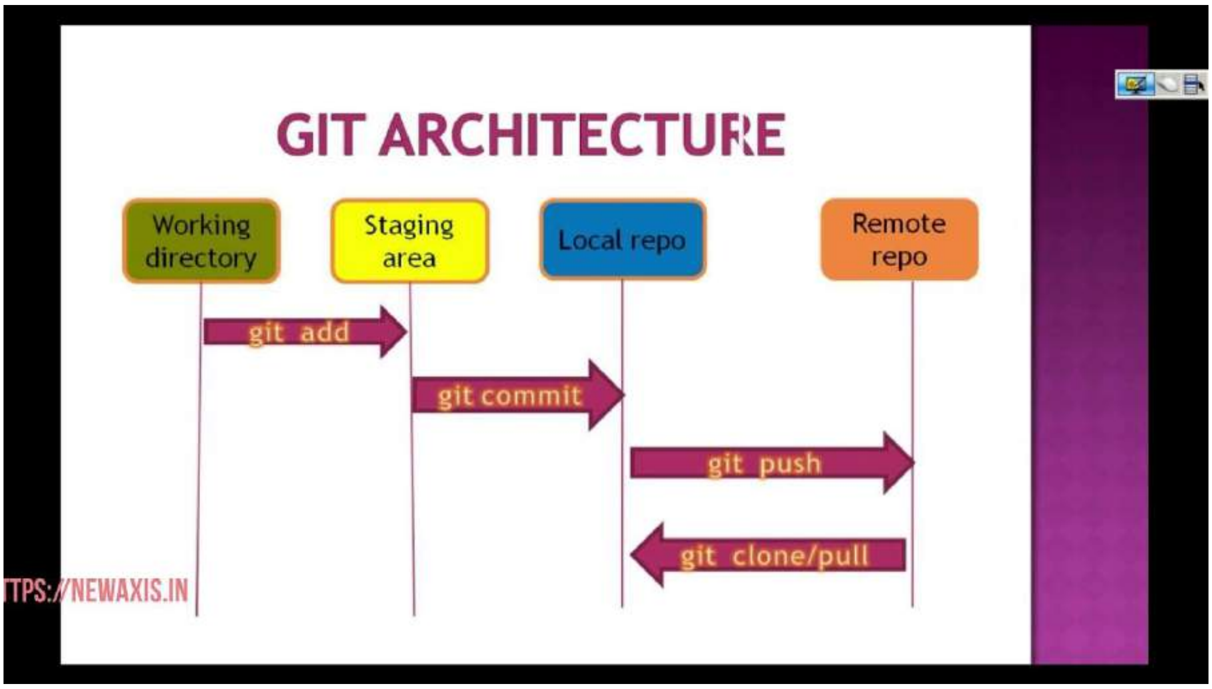
we have 4 stages

1.Workspace - it is a place where we edit , modify project related files
All the files in the workspace are visible to all directories.

2.Staging Area - On Git Add, files are moved from workspace to staging area where changes are saved

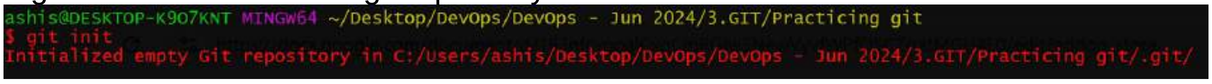
3.Local Repo - on Git Commit , files will be added to local/git repo & then we can track the file versions. Commit IDs are created here.

4.Central repo - On Git Push, files are moved to the central repo.



****

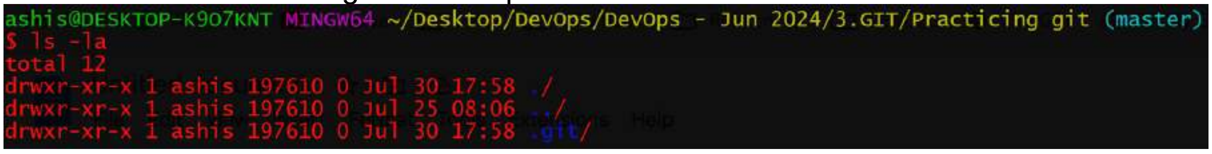
1.git init : To initialise the git repository



```
ashis@DESKTOP-K9O7KNT MINGW64 ~/Desktop/DevOps/DevOps - Jun 2024/3.GIT/Practicing git
$ git init
Initialized empty Git repository in C:/Users/ashis/Desktop/DevOps/DevOps - Jun 2024/3.GIT/Practicing git/.git/
```

****

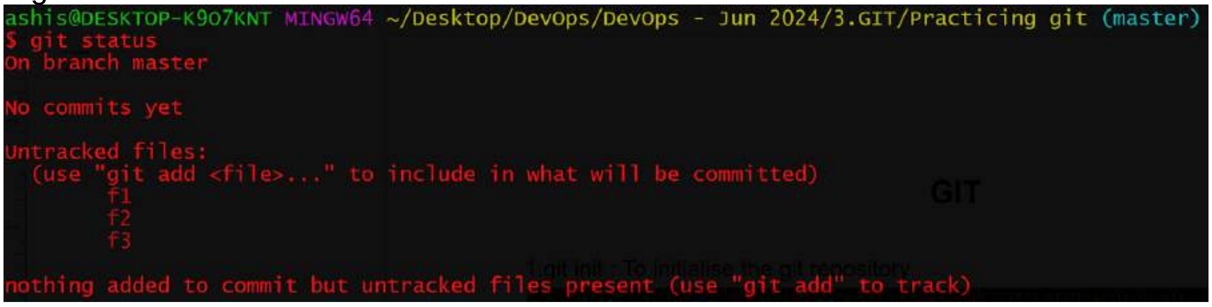
2.ls -la: To check if the .git folder is present



```
ashis@DESKTOP-K9O7KNT MINGW64 ~/Desktop/DevOps/DevOps - Jun 2024/3.GIT/Practicing git (master)
$ ls -la
total 12
drwxr-xr-x 1 ashis 197610 0 Jul 30 17:58 ./
drwxr-xr-x 1 ashis 197610 0 Jul 25 08:06 ../
drwxr-xr-x 1 ashis 197610 0 Jul 30 17:58 .git/
```

****

3.git status: To check the status of the files

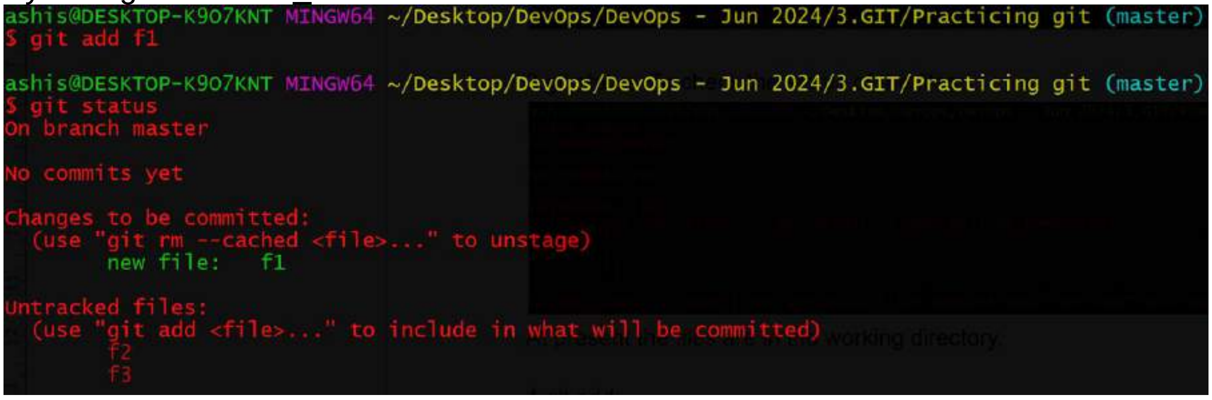


```
ashis@DESKTOP-K907KNT MINGW64 ~/Desktop/DevOps/DevOps - Jun 2024/3.GIT/Practicing git (master)
$ git status
On branch master

No commits yet

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        f1
        f2
        f3

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

At present the files are in the working directory.

****

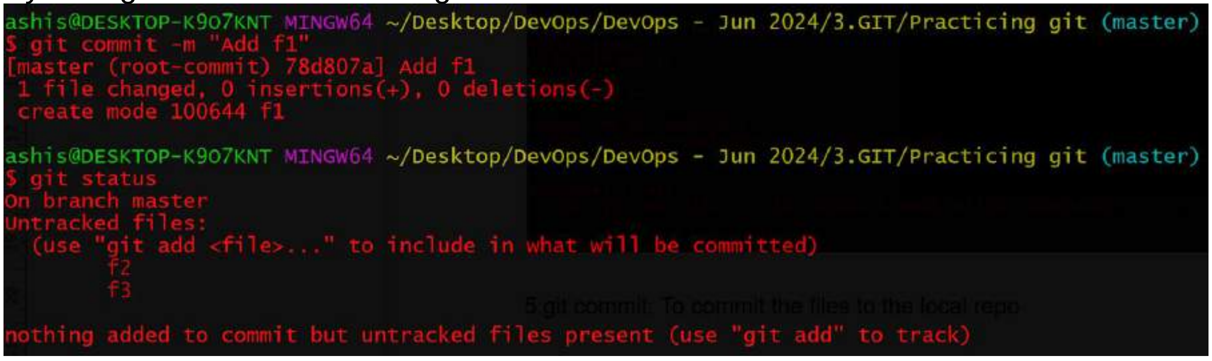
4.git add: To add the files to the staging area
Syntax: git add <file_name>



```
ashis@DESKTOP-K907KNT MINGW64 ~/Desktop/DevOps/DevOps - Jun 2024/3.GIT/Practicing git (master)
$ git add f1

ashis@DESKTOP-K907KNT MINGW64 ~/Desktop/DevOps/DevOps - Jun 2024/3.GIT/Practicing git (master)
$ git status
On branch master

No commits yet

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

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        f2
        f3
```

****

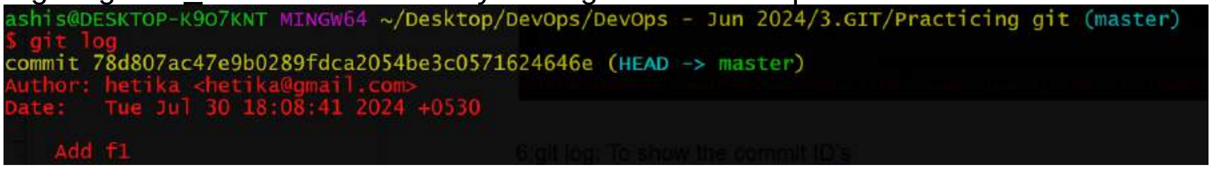
5.git commit: To commit the files to the local repo
Syntax: git commit -m <message>



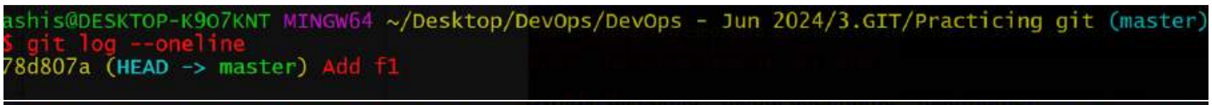
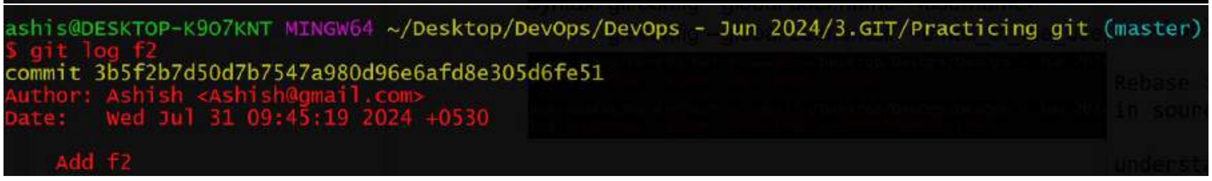
```
ashis@DESKTOP-K907KNT MINGW64 ~/Desktop/DevOps/DevOps - Jun 2024/3.GIT/Practicing git (master)
$ git commit -m "Add f1"
[master (root-commit) 78d807a] Add f1
 1 file changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 f1

ashis@DESKTOP-K907KNT MINGW64 ~/Desktop/DevOps/DevOps - Jun 2024/3.GIT/Practicing git (master)
$ git status
On branch master
Untracked files:
  (use "git add <file>..." to include in what will be committed)
        f2
        f3

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

****

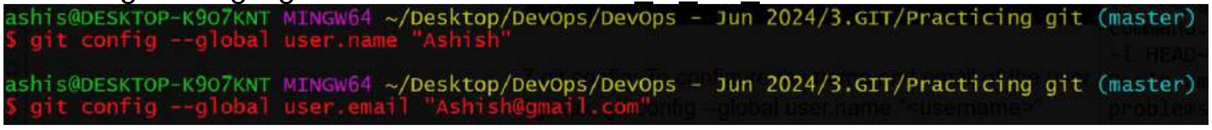
6:git log: To show the commit ID
   git log --oneline : To show commit ID in a single line
   git log <file_name> : To see only the logs related to a particular file



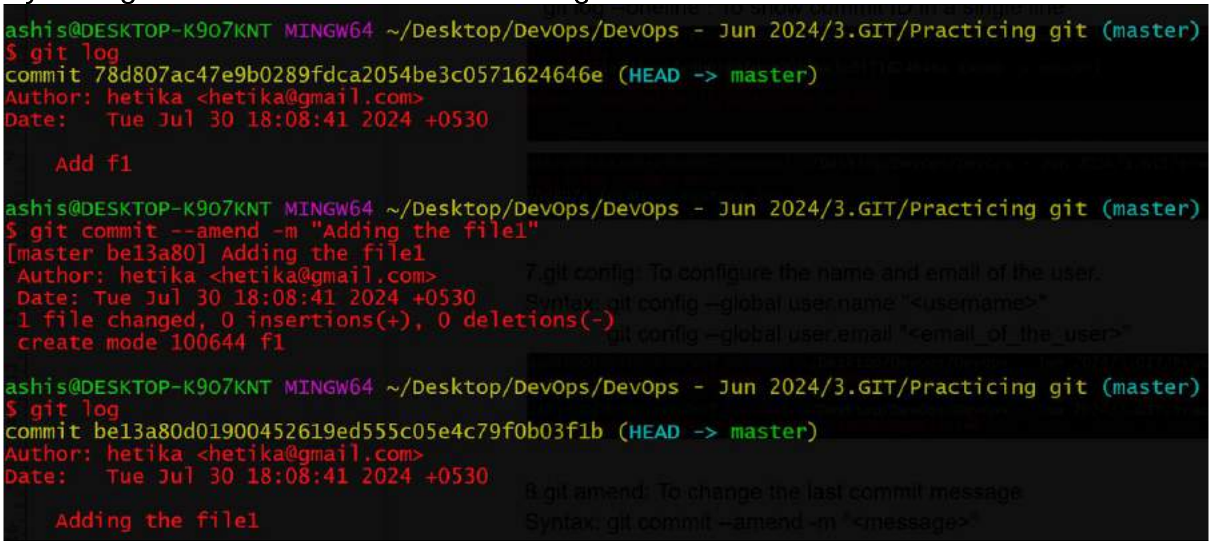
```
ashis@DESKTOP-K907KNT MINGW64 ~/Desktop/DevOps/DevOps - Jun 2024/3.GIT/Practicing git (master)
$ git log
commit 78d807ac47e9b0289fdca2054be3c0571624646e (HEAD -> master)
Author: hetika <hetika@gmail.com>
Date:   Tue Jul 30 18:08:41 2024 +0530

    Add f1
```


```
ashis@DESKTOP-K907KNT MINGW64 ~/Desktop/DevOps/DevOps - Jun 2024/3.GIT/Practicing git (master)
$ git log --oneline
78d807a (HEAD -> master) Add f1
```

```
ashis@DESKTOP-K907KNT MINGW64 ~/Desktop/DevOps/DevOps - Jun 2024/3.GIT/Practicing git (master)
$ git log f2
commit 3b5f2b7d50d7b7547a980d96e6afd8e305d6fe51
Author: Ashish <Ashish@gmail.com>
Date:   Wed Jul 31 09:45:19 2024 +0530

    Add f2
```
****

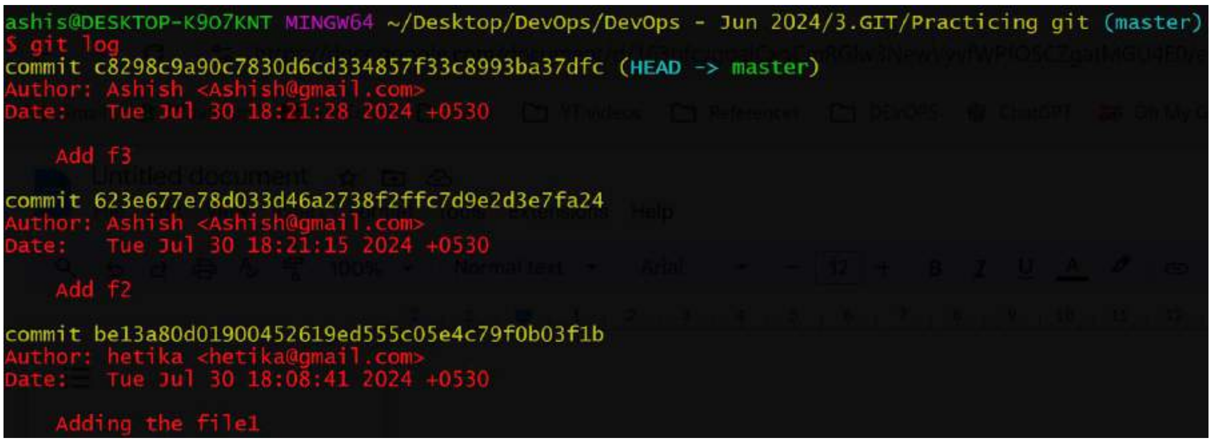
7.git config: To configure the name and email of the user.
Syntax: git config --global user.name "<username>"
            git config --global user.email "<email_of_the_user>"


```
ashis@DESKTOP-K907KNT MINGW64 ~/Desktop/DevOps/DevOps - Jun 2024/3.GIT/Practicing git (master)
$ git config --global user.name "Ashish"

ashis@DESKTOP-K907KNT MINGW64 ~/Desktop/DevOps/DevOps - Jun 2024/3.GIT/Practicing git (master)
$ git config --global user.email "Ashish@gmail.com"
```
****

8.git amend: To change the last commit message
Syntax: git commit --amend -m "<message>"


```
ashis@DESKTOP-K907KNT MINGW64 ~/Desktop/DevOps/DevOps - Jun 2024/3.GIT/Practicing git (master)
$ git log
commit 78d807ac47e9b0289fdca2054be3c0571624646e (HEAD -> master)
Author: hetika <hetika@gmail.com>
Date:   Tue Jul 30 18:08:41 2024 +0530

    Add f1

ashis@DESKTOP-K907KNT MINGW64 ~/Desktop/DevOps/DevOps - Jun 2024/3.GIT/Practicing git (master)
$ git commit --amend -m "Adding the file1"
[master be13a80] Adding the file1
 Author: hetika <hetika@gmail.com>
 Date: Tue Jul 30 18:08:41 2024 +0530
 1 file changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 f1

ashis@DESKTOP-K907KNT MINGW64 ~/Desktop/DevOps/DevOps - Jun 2024/3.GIT/Practicing git (master)
$ git log
commit be13a80d01900452619ed555c05e4c79f0b03f1b (HEAD -> master)
Author: hetika <hetika@gmail.com>
Date:   Tue Jul 30 18:08:41 2024 +0530

    Adding the file1
```
The first commit message was “Add f1” and then changed to “Adding the file1”
We can also see that the commit ID has also changed.
To change the most recent commit message, use the git commit --amend command.
To change older or multiple commit messages, use git rebase -i HEAD~N.
Don’t amend pushed commits as it may potentially cause a lot of problems to your colleagues.

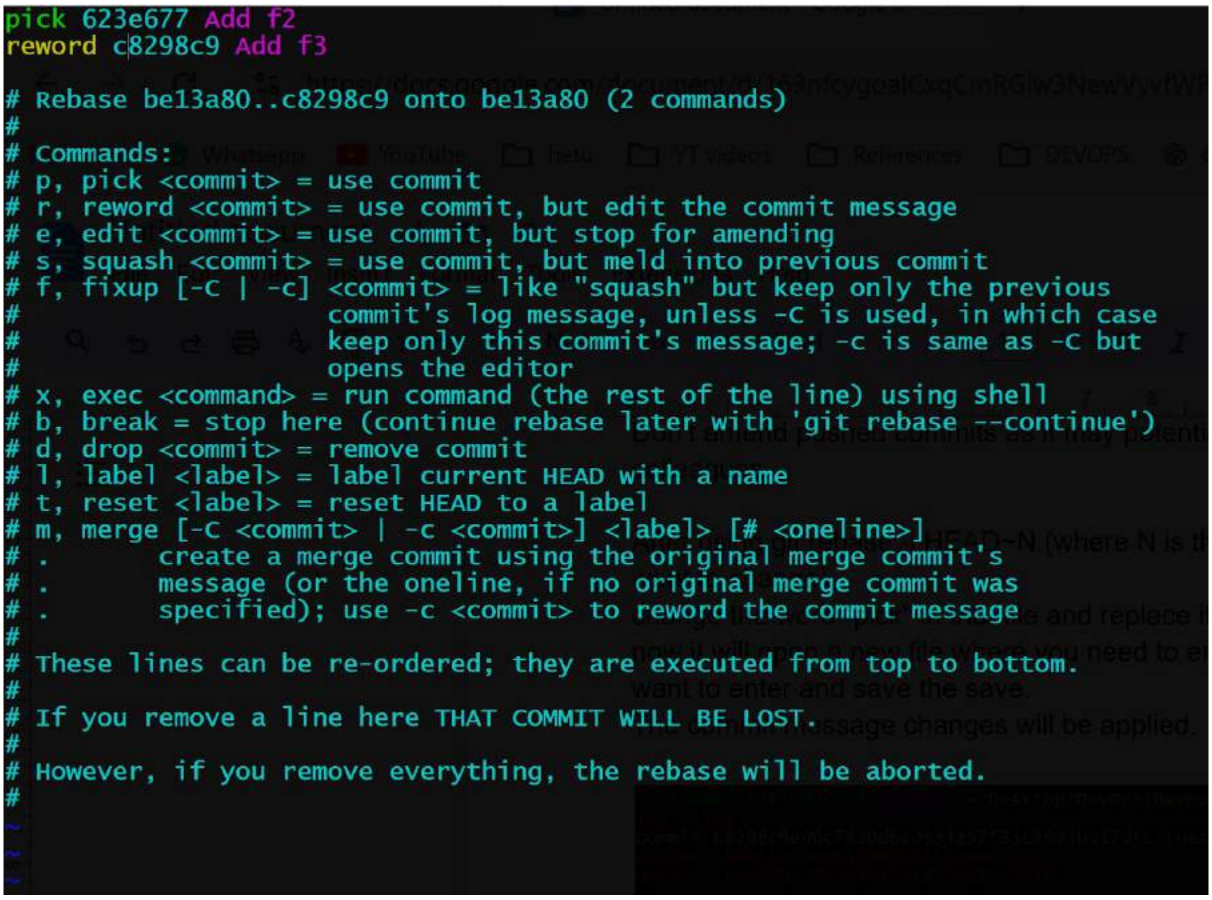
After using git rebase -i HEAD~N (where N is the number of the commit ID that you want to change),
change the word "pick" in that file and replace it with reword,
now it will open a new file where you need to enter the new commit msg that you want to enter and save the save.
The commit message changes will be applied.



```
ashis@DESKTOP-K9O7KNT MINGW64 ~/Desktop/DevOps/DevOps - Jun 2024/3.GIT/Practicing git (master)
$ git log
commit c8298c9a90c7830d6cd334857f33c8993ba37dfc (HEAD -> master)
Author: Ashish <Ashish@gmail.com>
Date:   Tue Jul 30 18:21:28 2024 +0530

    Add f3

commit 623e677e78d033d46a2738f2ffc7d9e2d3e7fa24
Author: Ashish <Ashish@gmail.com>
Date:   Tue Jul 30 18:21:15 2024 +0530

    Add f2

commit be13a80d01900452619ed555c05e4c79f0b03f1b
Author: hetika <hetika@gmail.com>
Date:   Tue Jul 30 18:08:41 2024 +0530

    Adding the file1
```

These are the commit ID before changing the commit message

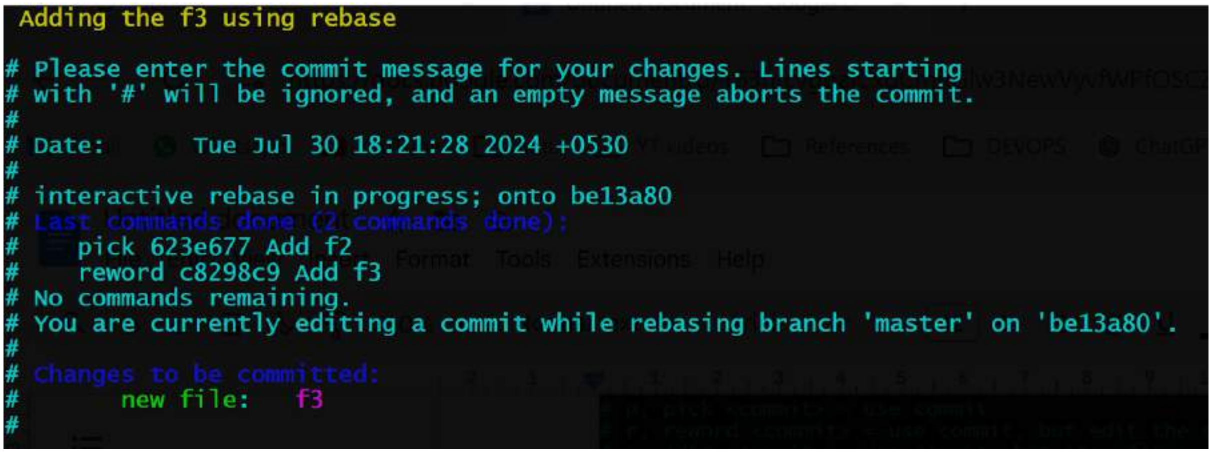
The text editor:



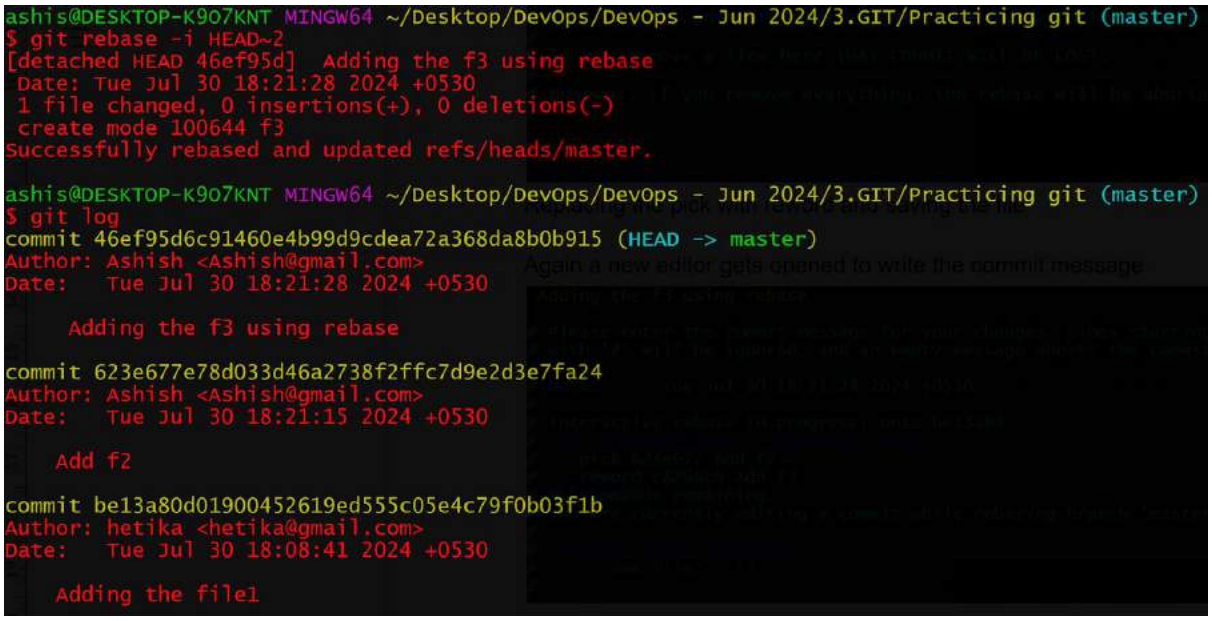
```
pick 623e677 Add f2
reword c8298c9 Add f3

# Rebase be13a80..c8298c9 onto be13a80 (2 commands)
#
# Commands:
# p, pick <commit> = use commit
# r, reword <commit> = use commit, but edit the commit message
# e, edit <commit> = use commit, but stop for amending
# s, squash <commit> = use commit, but meld into previous commit
# f, fixup [-C | -c] <commit> = like "squash" but keep only the previous
#                    commit's log message, unless -C is used, in which case
#                    keep only this commit's message; -c is same as -C but
#                    opens the editor
# x, exec <command> = run command (the rest of the line) using shell
# b, break = stop here (continue rebase later with 'git rebase --continue')
# d, drop <commit> = remove commit
# l, label <label> = label current HEAD with a name
# t, reset <label> = reset HEAD to a label
# m, merge [-C <commit> | -c <commit>] <label> [# <oneline>]
# .       create a merge commit using the original merge commit's
# .       message (or the oneline, if no original merge commit was
# .       specified); use -c <commit> to reword the commit message
#
# These lines can be re-ordered; they are executed from top to bottom.
#
# If you remove a line here THAT COMMIT WILL BE LOST.
#
# However, if you remove everything, the rebase will be aborted.
#
~
~
~
```

Replacing the pick with reword and saving the file

Again a new editor gets opened to write the commit message


```
Adding the f3 using rebase

# Please enter the commit message for your changes. Lines starting
# with '#' will be ignored, and an empty message aborts the commit.
#
# Date:      Tue Jul 30 18:21:28 2024 +0530
#
# interactive rebase in progress; onto be13a80
# Last commands done (2 commands done):
#    pick 623e677 Add f2
#    reword c8298c9 Add f3
# No commands remaining.
# You are currently editing a commit while rebasing branch 'master' on 'be13a80'.
#
# Changes to be committed:
#       new file:   f3
#
```

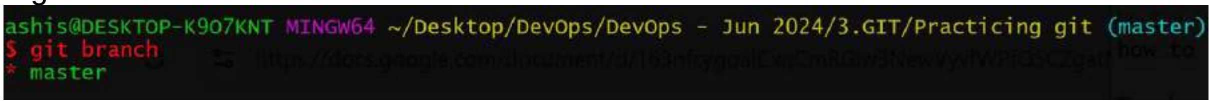
After changing the commit message, the git log looks like this with changed commit ID.


```
ashis@DESKTOP-K9O7KNT MINGW64 ~/Desktop/DevOps/DevOps - Jun 2024/3.GIT/Practicing git (master)
$ git rebase -i HEAD~2
[detached HEAD 46ef95d]  Adding the f3 using rebase
 Date: Tue Jul 30 18:21:28 2024 +0530
 1 file changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 f3
Successfully rebased and updated refs/heads/master.

ashis@DESKTOP-K9O7KNT MINGW64 ~/Desktop/DevOps/DevOps - Jun 2024/3.GIT/Practicing git (master)
$ git log
commit 46ef95d6c91460e4b99d9cdea72a368da8b0b915 (HEAD -> master)
Author: Ashish <Ashish@gmail.com>
Date:   Tue Jul 30 18:21:28 2024 +0530

     Adding the f3 using rebase

commit 623e677e78d033d46a2738f2ffc7d9e2d3e7fa24
Author: Ashish <Ashish@gmail.com>
Date:   Tue Jul 30 18:21:15 2024 +0530

    Add f2

commit be13a80d01900452619ed555c05e4c79f0b03f1b
Author: hetika <hetika@gmail.com>
Date:   Tue Jul 30 18:08:41 2024 +0530

    Adding the file1
```
****

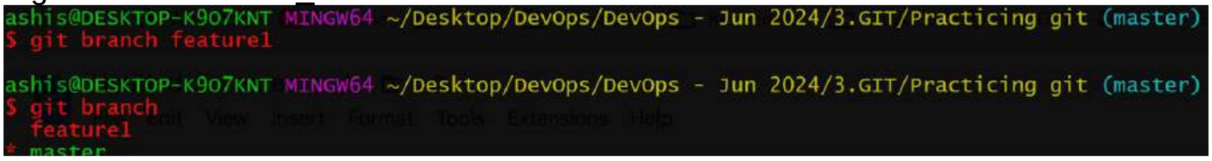
9.Branching: It is for parallel development, teams can work on same piece of code on different branches parallelly and later integrate by merging

- Why do we need branching? → to develop new features

a.git branch — list all branches

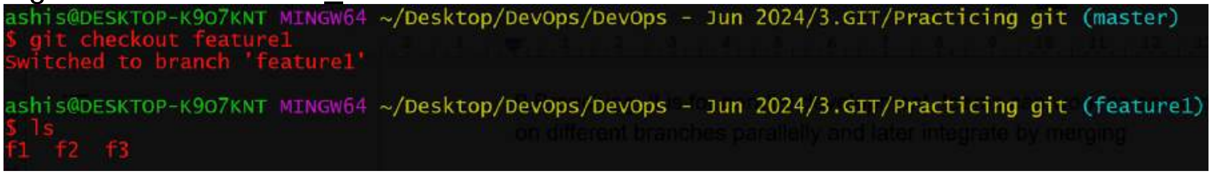

```
ashis@DESKTOP-K9O7KNT MINGW64 ~/Desktop/DevOps/DevOps - Jun 2024/3.GIT/Practicing git (master)
$ git branch
* master
```
Only master branch is present

b.git branch <branch_name> — create new branch


```
ashis@DESKTOP-K9O7KNT MINGW64 ~/Desktop/DevOps/DevOps - Jun 2024/3.GIT/Practicing git (master)
$ git branch feature1

ashis@DESKTOP-K9O7KNT MINGW64 ~/Desktop/DevOps/DevOps - Jun 2024/3.GIT/Practicing git (master)
$ git branch
  feature1
* master
```
It shows that we have 2 branches, master and feature1.

And we are working on the master branch. (* - represents the branch that we are working on)

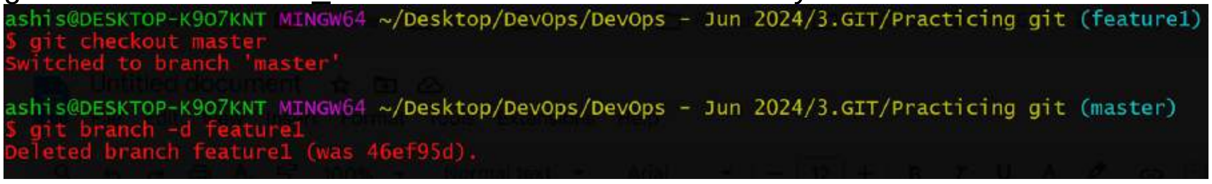
c.git checkout <branch_name>---- switch to branch



```
ashis@DESKTOP-K9O7KNT MINGW64 ~/Desktop/DevOps/DevOps - Jun 2024/3.GIT/Practicing git (master)
$ git checkout feature1
Switched to branch 'feature1'

ashis@DESKTOP-K9O7KNT MINGW64 ~/Desktop/DevOps/DevOps - Jun 2024/3.GIT/Practicing git (feature1)
$ ls
f1  f2  f3
```

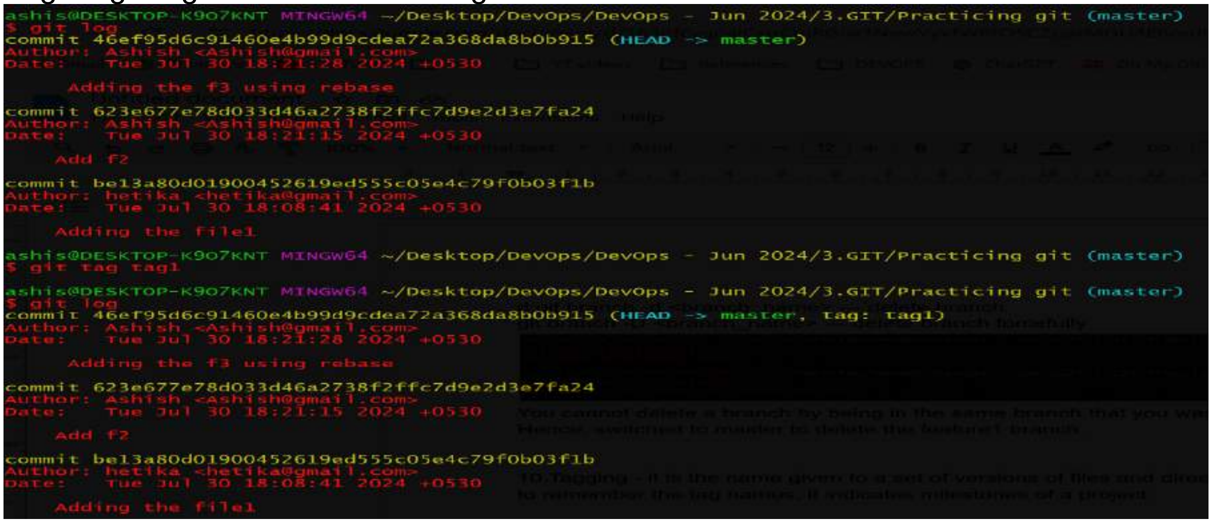
We can see that we have been switched to feature1 branch and the files in the master are copied to feature1 as the feature1 is checked out from master.

d.git branch -d <branch_name> --- delete branch
git branch -D <branch_name> --- delete branch forcefully



```
ashis@DESKTOP-K9O7KNT MINGW64 ~/Desktop/DevOps/DevOps - Jun 2024/3.GIT/Practicing git (feature1)
$ git checkout master
Switched to branch 'master'

ashis@DESKTOP-K9O7KNT MINGW64 ~/Desktop/DevOps/DevOps - Jun 2024/3.GIT/Practicing git (master)
$ git branch -d feature1
Deleted branch feature1 (was 46ef95d).
```

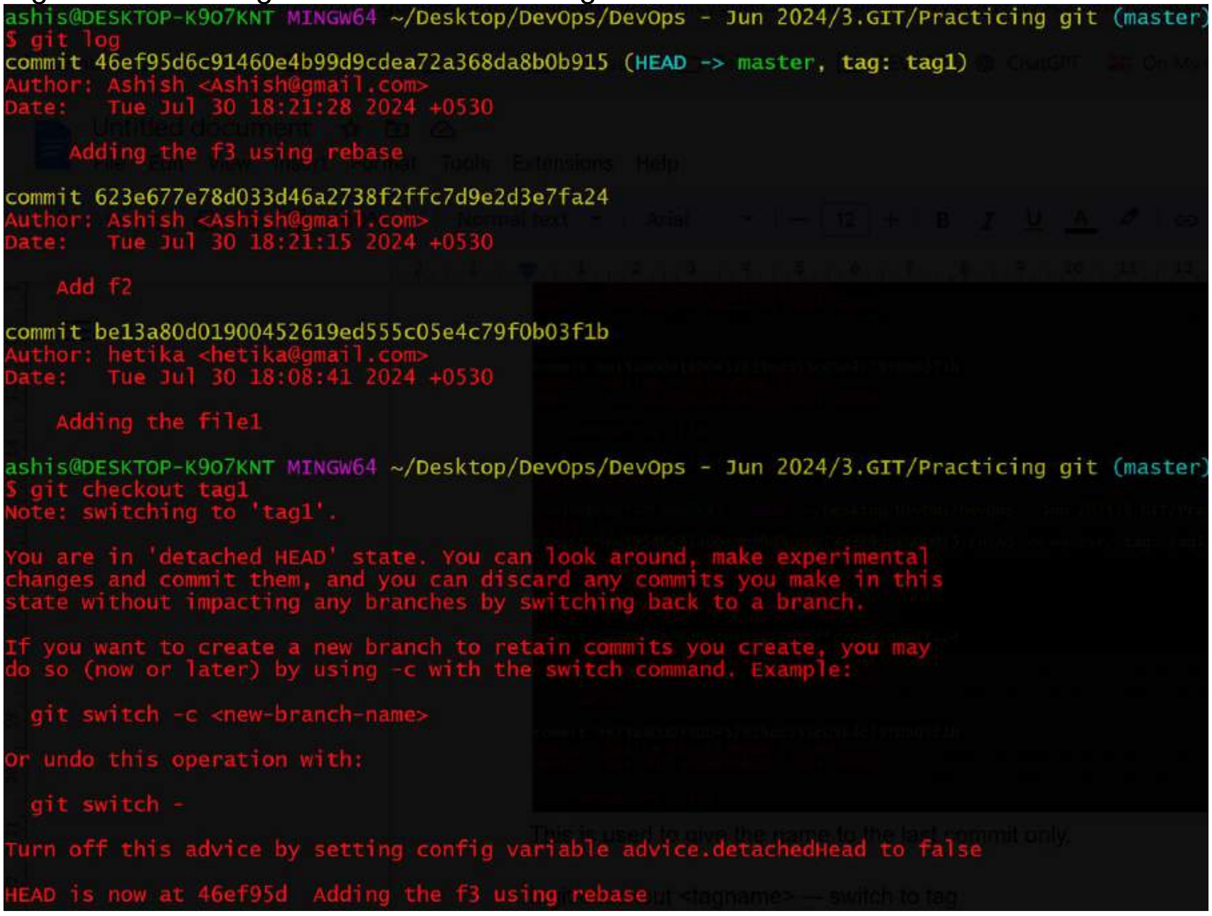
You cannot delete a branch by being in the same branch that you want to delete. Hence, switched to master to delete the feature1 branch.
****
10.Tagging - it is the name given to a set of versions of files and directories. it is easy to remember the tag names, it indicates milestones of a project.
a. git tag <tagname> -- create tag



```
ashis@DESKTOP-K9O7KNT MINGW64 ~/Desktop/DevOps/DevOps - Jun 2024/3.GIT/Practicing git (master)
$ git log
commit 46ef95d6c91460e4b99d9cdea72a368da8b0b915 (HEAD -> master)
Author: Ashish <Ashish@gmail.com>
Date:   Tue Jul 30 18:21:28 2024 +0530

    Adding the f3 using rebase

commit 623e677e78d033d46a2738f2ffc7d9e2d3e7fa24
Author: Ashish <Ashish@gmail.com>
Date:   Tue Jul 30 18:21:15 2024 +0530

    Add f2

commit be13a80d01900452619ed555c05e4c79f0b03f1b
Author: hetika <hetika@gmail.com>
Date:   Tue Jul 30 18:08:41 2024 +0530

    Adding the file1

ashis@DESKTOP-K9O7KNT MINGW64 ~/Desktop/DevOps/DevOps - Jun 2024/3.GIT/Practicing git (master)
$ git tag tag1

ashis@DESKTOP-K9O7KNT MINGW64 ~/Desktop/DevOps/DevOps - Jun 2024/3.GIT/Practicing git (master)
$ git log
commit 46ef95d6c91460e4b99d9cdea72a368da8b0b915 (HEAD -> master, tag: tag1)
Author: Ashish <Ashish@gmail.com>
Date:   Tue Jul 30 18:21:28 2024 +0530

    Adding the f3 using rebase

commit 623e677e78d033d46a2738f2ffc7d9e2d3e7fa24
Author: Ashish <Ashish@gmail.com>
Date:   Tue Jul 30 18:21:15 2024 +0530

    Add f2

commit be13a80d01900452619ed555c05e4c79f0b03f1b
Author: hetika <hetika@gmail.com>
Date:   Tue Jul 30 18:08:41 2024 +0530

    Adding the file1
```

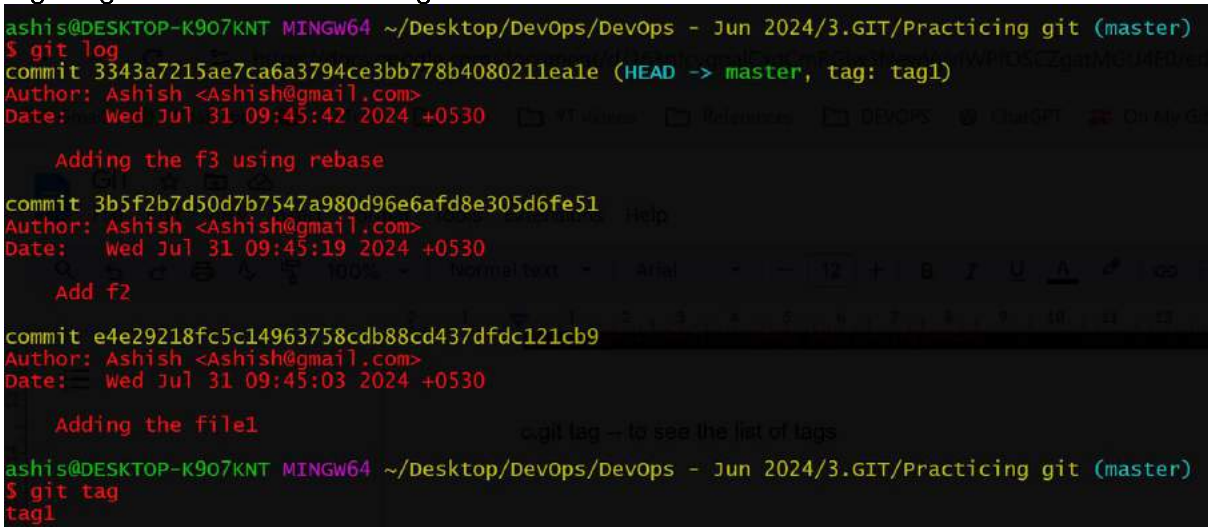
This is used to give the name to the last commit only.

b.git checkout <tagname> –- switch to tag



```
ashis@DESKTOP-K9O7KNT MINGW64 ~/Desktop/DevOps/DevOps - Jun 2024/3.GIT/Practicing git (master)
$ git log
commit 46ef95d6c91460e4b99d9cdea72a368da8b0b915 (HEAD -> master, tag: tag1)
Author: Ashish <Ashish@gmail.com>
Date:   Tue Jul 30 18:21:28 2024 +0530

     Adding the f3 using rebase

commit 623e677e78d033d46a2738f2ffc7d9e2d3e7fa24
Author: Ashish <Ashish@gmail.com>
Date:   Tue Jul 30 18:21:15 2024 +0530

    Add f2

commit be13a80d01900452619ed555c05e4c79f0b03f1b
Author: hetika <hetika@gmail.com>
Date:   Tue Jul 30 18:08:41 2024 +0530

    Adding the file1

ashis@DESKTOP-K9O7KNT MINGW64 ~/Desktop/DevOps/DevOps - Jun 2024/3.GIT/Practicing git (master)
$ git checkout tag1
Note: switching to 'tag1'.

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 46ef95d  Adding the f3 using rebase
```

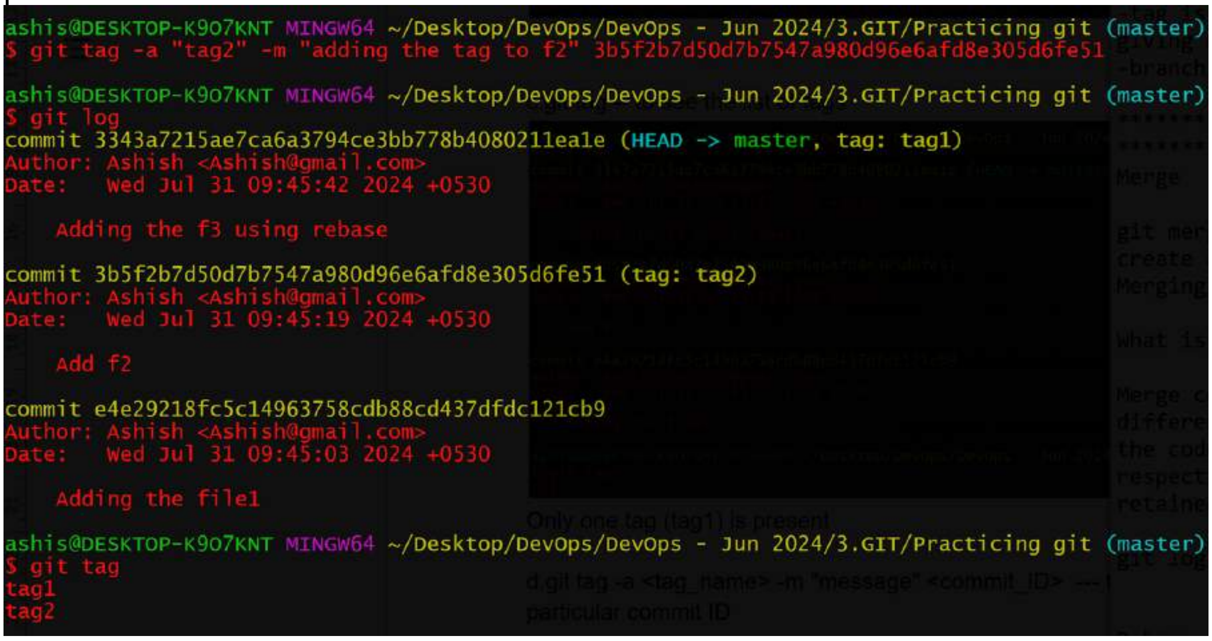
c.git tag -- to see the list of tags



```
ashis@DESKTOP-K9O7KNT MINGW64 ~/Desktop/DevOps/DevOps - Jun 2024/3.GIT/Practicing git (master)
$ git log
commit 3343a7215ae7ca6a3794ce3bb778b4080211ea1e (HEAD -> master, tag: tag1)
Author: Ashish <Ashish@gmail.com>
Date:   Wed Jul 31 09:45:42 2024 +0530

    Adding the f3 using rebase

commit 3b5f2b7d50d7b7547a980d96e6afd8e305d6fe51
Author: Ashish <Ashish@gmail.com>
Date:   Wed Jul 31 09:45:19 2024 +0530

    Add f2

commit e4e29218fc5c14963758cdb88cd437dfdc121cb9
Author: Ashish <Ashish@gmail.com>
Date:   Wed Jul 31 09:45:03 2024 +0530

    Adding the file1

ashis@DESKTOP-K9O7KNT MINGW64 ~/Desktop/DevOps/DevOps - Jun 2024/3.GIT/Practicing git (master)
$ git tag
tag1
```

Only one tag (tag1) is present

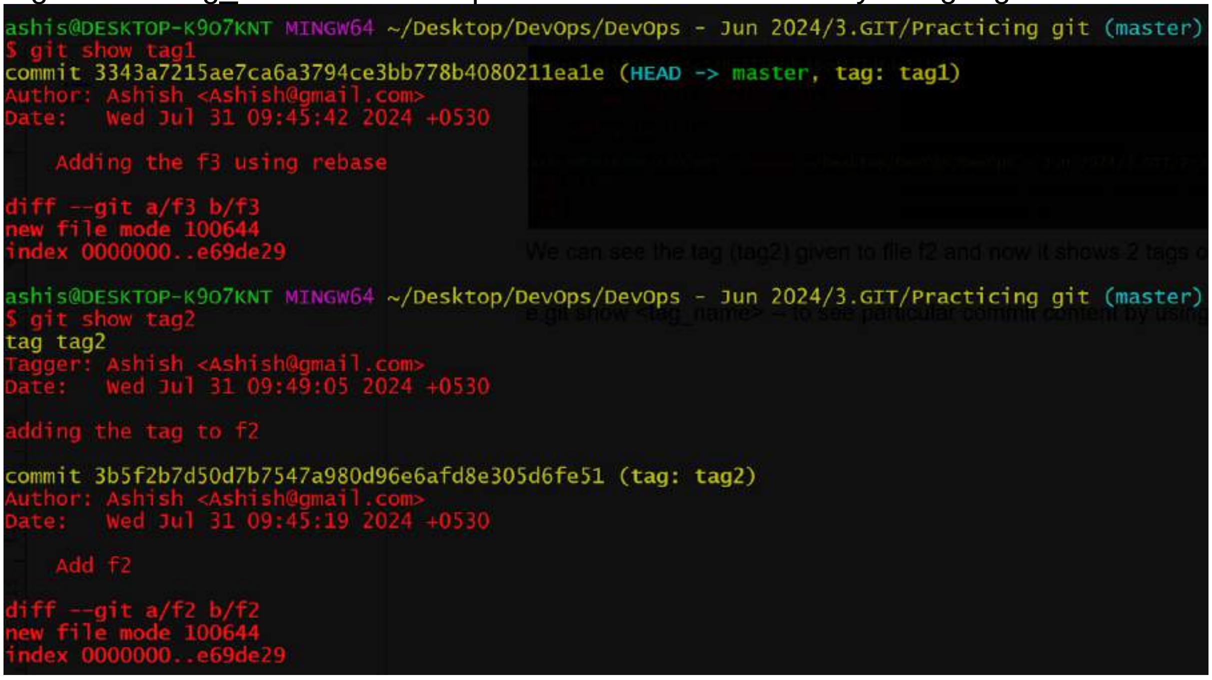
d.git tag -a <tag_name> -m "message" <commit_ID> --- to give a name to the particular commit ID



```
ashis@DESKTOP-K907KNT MINGW64 ~/Desktop/DevOps/DevOps - Jun 2024/3.GIT/Practicing git (master)
$ git tag -a "tag2" -m "adding the tag to f2" 3b5f2b7d50d7b7547a980d96e6afd8e305d6fe51

ashis@DESKTOP-K907KNT MINGW64 ~/Desktop/DevOps/DevOps - Jun 2024/3.GIT/Practicing git (master)
$ git log
commit 3343a7215ae7ca6a3794ce3bb778b4080211ea1e (HEAD -> master, tag: tag1)
Author: Ashish <Ashish@gmail.com>
Date:   Wed Jul 31 09:45:42 2024 +0530

    Adding the f3 using rebase

commit 3b5f2b7d50d7b7547a980d96e6afd8e305d6fe51 (tag: tag2)
Author: Ashish <Ashish@gmail.com>
Date:   Wed Jul 31 09:45:19 2024 +0530

    Add f2

commit e4e29218fc5c14963758cdb88cd437dfdc121cb9
Author: Ashish <Ashish@gmail.com>
Date:   Wed Jul 31 09:45:03 2024 +0530

    Adding the file1

ashis@DESKTOP-K907KNT MINGW64 ~/Desktop/DevOps/DevOps - Jun 2024/3.GIT/Practicing git (master)
$ git tag
tag1
tag2
```

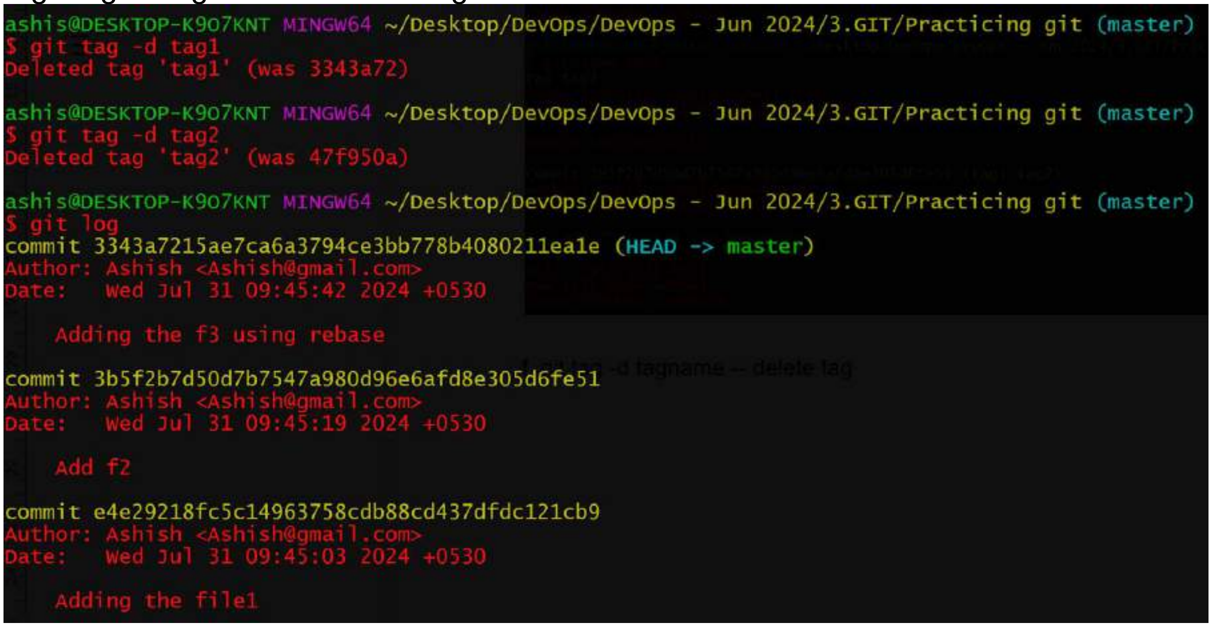
We can see the tag (tag2) given to file f2 and now it shows 2 tags on git tag.

e.git show <tag_name> -- to see particular commit content by using tag



```
ashis@DESKTOP-K907KNT MINGW64 ~/Desktop/DevOps/DevOps - Jun 2024/3.GIT/Practicing git (master)
$ git show tag1
commit 3343a7215ae7ca6a3794ce3bb778b4080211ea1e (HEAD -> master, tag: tag1)
Author: Ashish <Ashish@gmail.com>
Date:   Wed Jul 31 09:45:42 2024 +0530

    Adding the f3 using rebase

diff --git a/f3 b/f3
new file mode 100644
index 0000000..e69de29

ashis@DESKTOP-K907KNT MINGW64 ~/Desktop/DevOps/DevOps - Jun 2024/3.GIT/Practicing git (master)
$ git show tag2
tag tag2
Tagger: Ashish <Ashish@gmail.com>
Date:   Wed Jul 31 09:49:05 2024 +0530

adding the tag to f2

commit 3b5f2b7d50d7b7547a980d96e6afd8e305d6fe51 (tag: tag2)
Author: Ashish <Ashish@gmail.com>
Date:   Wed Jul 31 09:45:19 2024 +0530

    Add f2

diff --git a/f2 b/f2
new file mode 100644
index 0000000..e69de29
```

f. git tag -d tagname -- delete tag



```
ashis@DESKTOP-K9O7KNT MINGW64 ~/Desktop/DevOps/DevOps - Jun 2024/3.GIT/Practicing git (master)
$ git tag -d tag1
Deleted tag 'tag1' (was 3343a72)

ashis@DESKTOP-K9O7KNT MINGW64 ~/Desktop/DevOps/DevOps - Jun 2024/3.GIT/Practicing git (master)
$ git tag -d tag2
Deleted tag 'tag2' (was 47f950a)

ashis@DESKTOP-K9O7KNT MINGW64 ~/Desktop/DevOps/DevOps - Jun 2024/3.GIT/Practicing git (master)
$ git log
commit 3343a7215ae7ca6a3794ce3bb778b4080211ea1e (HEAD -> master)
Author: Ashish <Ashish@gmail.com>
Date:   Wed Jul 31 09:45:42 2024 +0530

    Adding the f3 using rebase

commit 3b5f2b7d50d7b7547a980d96e6afd8e305d6fe51
Author: Ashish <Ashish@gmail.com>
Date:   Wed Jul 31 09:45:19 2024 +0530

    Add f2

commit e4e29218fc5c14963758cdb88cd437dfdc121cb9
Author: Ashish <Ashish@gmail.com>
Date:   Wed Jul 31 09:45:03 2024 +0530

    Adding the file1
```

****

11. What is the difference between git branch and git tag.

Git Branch:

- Used to create a new line of development. It allows you to develop features, fix bugs, or experiment independently.
- Ideal for parallel development. For example, you can have multiple branches for different features, bug fixes, or releases.
- Dynamic and can move as new commits are added to the branch. Branches are pointers that advance with each new commit.
- Essential for collaborative development, allowing multiple developers to work on different branches and merge their work together.

Git Tag:

- Used to mark a specific commit as important. It does not create a new line of development and does not change the commit history.
- Ideal for marking releases or milestones in the project's history.
- Static and does not move. Once created, it permanently points to the same commit.
- Used for marking points in history for reference, not for collaboration or integration of changes.

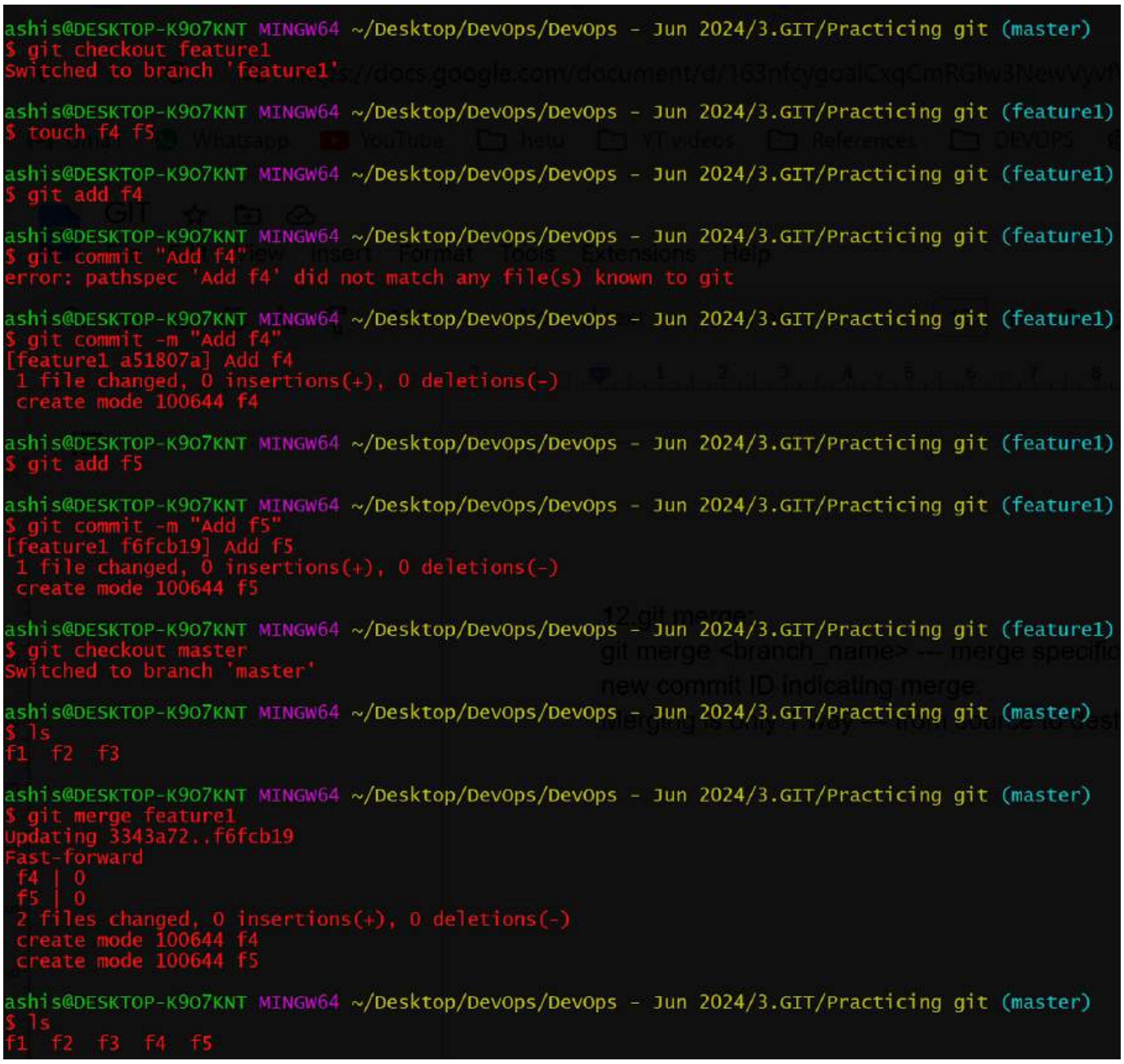
In summary, git tag is used for marking significant commits, while `git branch` is used for creating separate lines of development for commits.

****

12.git merge:

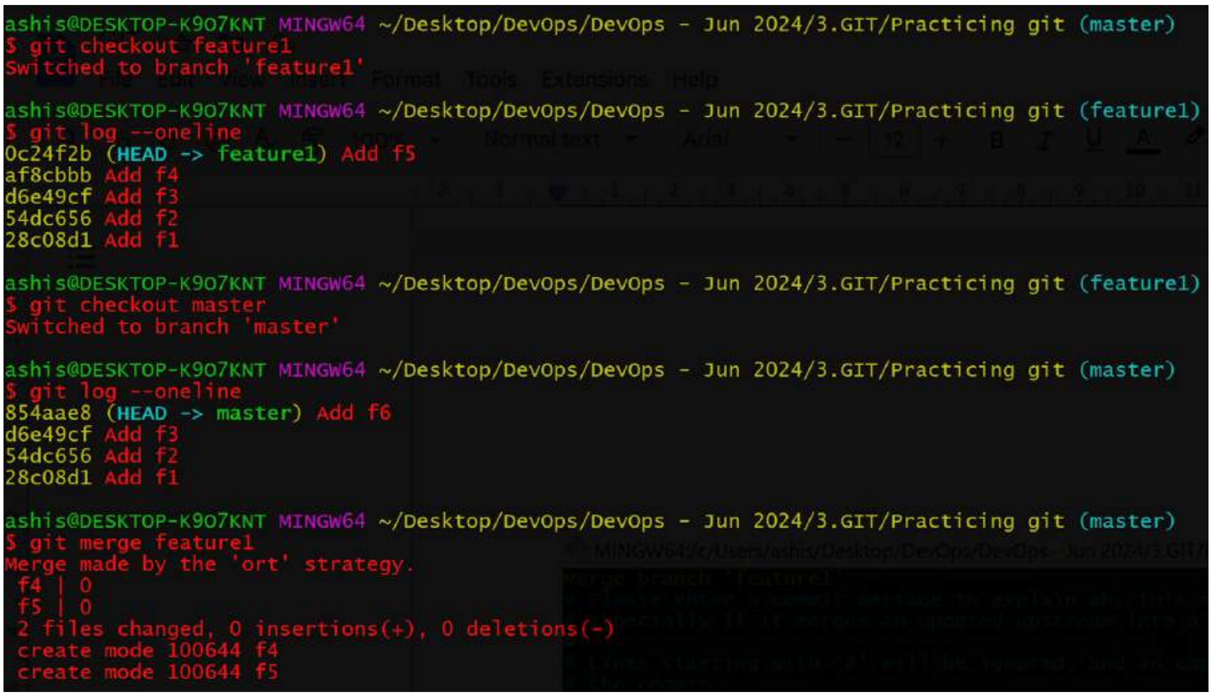
- git merge <branch_name> → merge specific branch to checked out branch.
- git branch --merged →lists the branches that have been merged into the current branch
- git branch --no merged → lists the branches that have not been merged

Creates a new commit ID indicating merge.
Merging is only 1 way --- from source to destination.



```
ashis@DESKTOP-K9O7KNT MINGW64 ~/Desktop/DevOps/DevOps - Jun 2024/3.GIT/Practicing git (master)
$ git checkout feature1
Switched to branch 'feature1'

ashis@DESKTOP-K9O7KNT MINGW64 ~/Desktop/DevOps/DevOps - Jun 2024/3.GIT/Practicing git (feature1)
$ touch f4 f5

ashis@DESKTOP-K9O7KNT MINGW64 ~/Desktop/DevOps/DevOps - Jun 2024/3.GIT/Practicing git (feature1)
$ git add f4

ashis@DESKTOP-K9O7KNT MINGW64 ~/Desktop/DevOps/DevOps - Jun 2024/3.GIT/Practicing git (feature1)
$ git commit "Add f4"
error: pathspec 'Add f4' did not match any file(s) known to git

ashis@DESKTOP-K9O7KNT MINGW64 ~/Desktop/DevOps/DevOps - Jun 2024/3.GIT/Practicing git (feature1)
$ git commit -m "Add f4"
[feature1 a51807a] Add f4
 1 file changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 f4

ashis@DESKTOP-K9O7KNT MINGW64 ~/Desktop/DevOps/DevOps - Jun 2024/3.GIT/Practicing git (feature1)
$ git add f5

ashis@DESKTOP-K9O7KNT MINGW64 ~/Desktop/DevOps/DevOps - Jun 2024/3.GIT/Practicing git (feature1)
$ git commit -m "Add f5"
[feature1 f6fcb19] Add f5
 1 file changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 f5

ashis@DESKTOP-K9O7KNT MINGW64 ~/Desktop/DevOps/DevOps - Jun 2024/3.GIT/Practicing git (feature1)
$ git checkout master
Switched to branch 'master'

ashis@DESKTOP-K9O7KNT MINGW64 ~/Desktop/DevOps/DevOps - Jun 2024/3.GIT/Practicing git (master)
$ ls
f1  f2  f3

ashis@DESKTOP-K9O7KNT MINGW64 ~/Desktop/DevOps/DevOps - Jun 2024/3.GIT/Practicing git (master)
$ git merge feature1
Updating 3343a72..f6fcb19
Fast-forward
 f4 | 0
 f5 | 0
 2 files changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 f4
 create mode 100644 f5

ashis@DESKTOP-K9O7KNT MINGW64 ~/Desktop/DevOps/DevOps - Jun 2024/3.GIT/Practicing git (master)
$ ls
f1  f2  f3  f4  f5
```

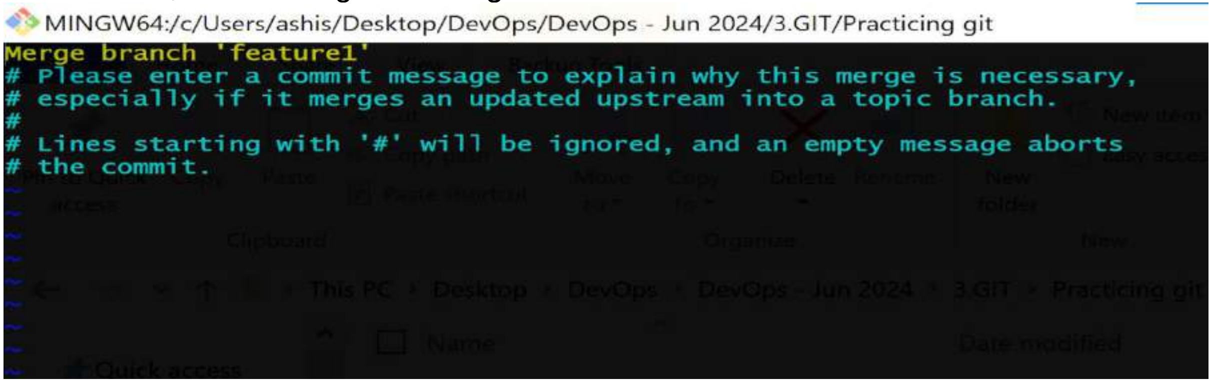
- We can see that we have files f1 f2 f3 on master.
- We created files f4 f5 on feature1 and tried to merge them to master.
- Now, files f1 f2 f3 f4 f5 all are on master.
- **NOTE:** If there are no changes made in the master after the feature1 was created (that means only the changes were made in the feature1), then there won't be any new commit ID for merge.



```
ashis@DESKTOP-K907KNT MINGW64 ~/Desktop/DevOps/DevOps - Jun 2024/3.GIT/Practicing git (master)
$ git checkout feature1
Switched to branch 'feature1'

ashis@DESKTOP-K907KNT MINGW64 ~/Desktop/DevOps/DevOps - Jun 2024/3.GIT/Practicing git (feature1)
$ git log --oneline
0c24f2b (HEAD -> feature1) Add f5
af8cbbb Add f4
d6e49cf Add f3
54dc656 Add f2
28c08d1 Add f1

ashis@DESKTOP-K907KNT MINGW64 ~/Desktop/DevOps/DevOps - Jun 2024/3.GIT/Practicing git (feature1)
$ git checkout master
Switched to branch 'master'

ashis@DESKTOP-K907KNT MINGW64 ~/Desktop/DevOps/DevOps - Jun 2024/3.GIT/Practicing git (master)
$ git log --oneline
854aae8 (HEAD -> master) Add f6
d6e49cf Add f3
54dc656 Add f2
28c08d1 Add f1

ashis@DESKTOP-K907KNT MINGW64 ~/Desktop/DevOps/DevOps - Jun 2024/3.GIT/Practicing git (master)
$ git merge feature1
Merge made by the 'ort' strategy.
 f4 | 0
 f5 | 0
 2 files changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 f4
 create mode 100644 f5
```

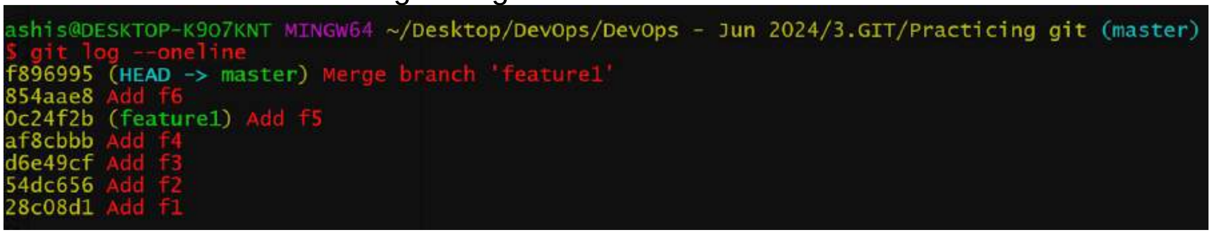
- We had f1 f2 f3 in master
- Created f4 f5 in feature1
- Came back to master and created f6
- Now I tried to merge feature1 with master branch.

  A text editor gets opened and we are supposed to give the message for the commit that will be created on git merge.
  Here, the message is "Merge branch 'feature1'".



```
MINGW64:/c/Users/ashis/Desktop/DevOps/DevOps - Jun 2024/3.GIT/Practicing git
Merge branch 'feature1'
# Please enter a commit message to explain why this merge is necessary,
# especially if it merges an updated upstream into a topic branch.
#
# Lines starting with '#' will be ignored, and an empty message aborts
# the commit.
~
~
~
~
~
~
~
~
```

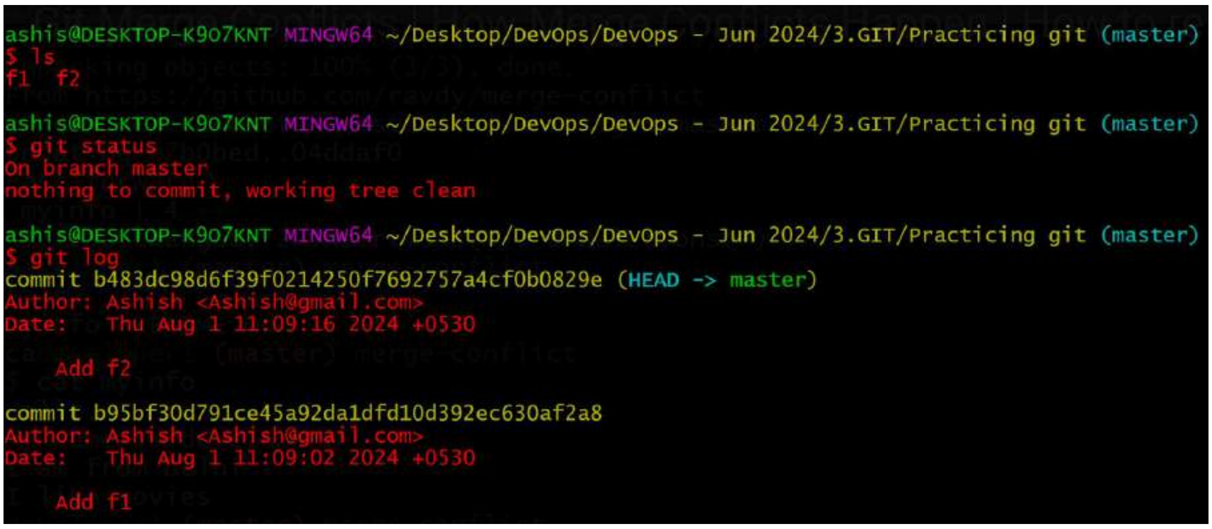
On git log, we can see that a new commit ID with the message that we gave in the editor has been created on git merge.



```
ashis@DESKTOP-K907KNT MINGW64 ~/Desktop/DevOps/DevOps - Jun 2024/3.GIT/Practicing git (master)
$ git log --oneline
f896995 (HEAD -> master) Merge branch 'feature1'
854aae8 Add f6
0c24f2b (feature1) Add f5
af8cbbb Add f4
d6e49cf Add f3
54dc656 Add f2
28c08d1 Add f1
```

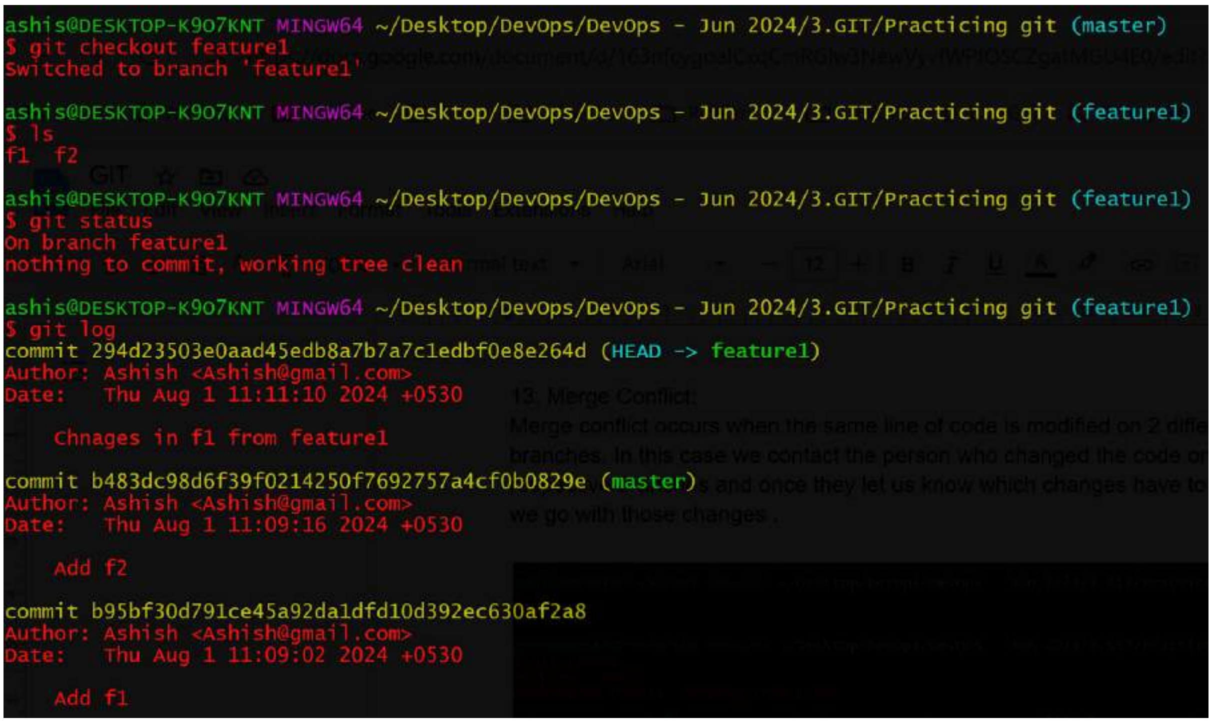
****

13. Merge Conflict:
Merge conflict occurs when the same line of code is modified on 2 different branches. In this case we contact the person who changed the code on respective branches and once they let us know which changes have to be retained , we go with those changes .



```
ashis@DESKTOP-K9O7KNT MINGW64 ~/Desktop/DevOps/DevOps - Jun 2024/3.GIT/Practicing git (master)
$ ls
f1  f2

ashis@DESKTOP-K9O7KNT MINGW64 ~/Desktop/DevOps/DevOps - Jun 2024/3.GIT/Practicing git (master)
$ git status
On branch master
nothing to commit, working tree clean

ashis@DESKTOP-K9O7KNT MINGW64 ~/Desktop/DevOps/DevOps - Jun 2024/3.GIT/Practicing git (master)
$ git log
commit b483dc98d6f39f0214250f7692757a4cf0b0829e (HEAD -> master)
Author: Ashish <Ashish@gmail.com>
Date:   Thu Aug 1 11:09:16 2024 +0530

    Add f2

commit b95bf30d791ce45a92da1dfd10d392ec630af2a8
Author: Ashish <Ashish@gmail.com>
Date:   Thu Aug 1 11:09:02 2024 +0530

    Add f1
```

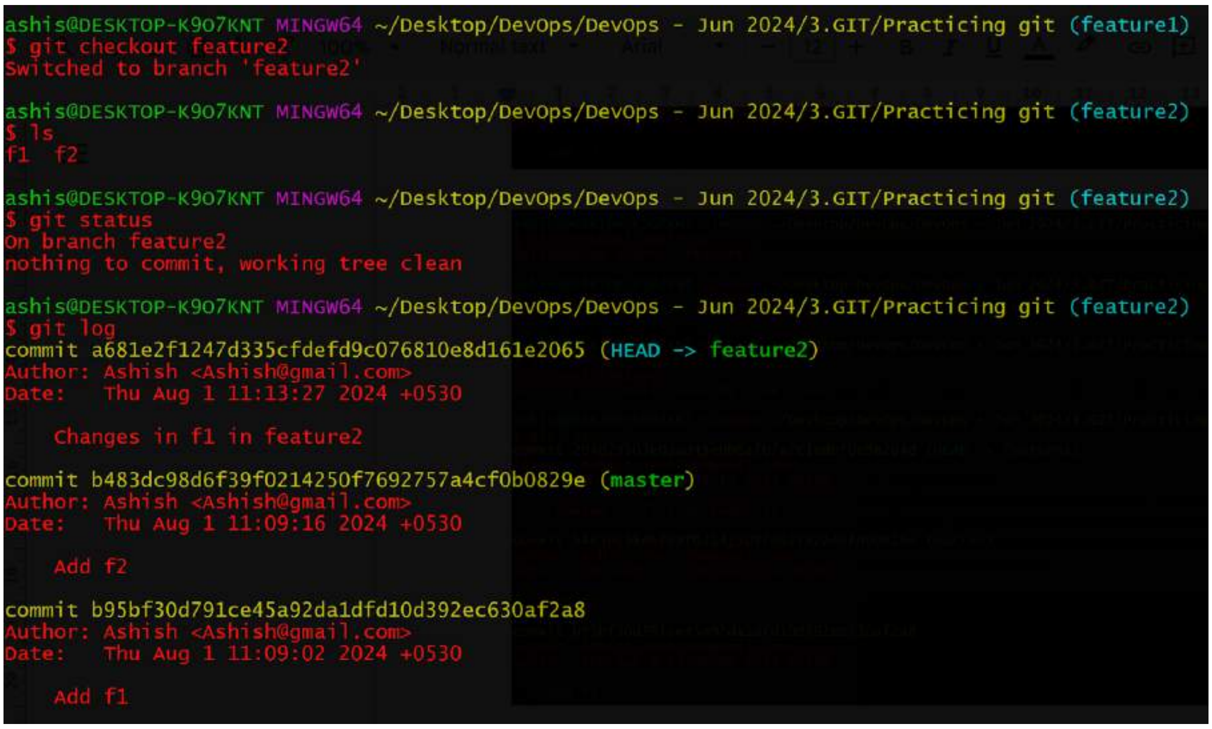
- f1 f2 files on master branch.
- Contents of f1:
  Hello,
  My name is ABC.
  I am from XYZ city.
  Thanks.
- f2 was committed without any content.



```
ashis@DESKTOP-K9O7KNT MINGW64 ~/Desktop/DevOps/DevOps - Jun 2024/3.GIT/Practicing git (master)
$ git checkout feature1
Switched to branch 'feature1'

ashis@DESKTOP-K9O7KNT MINGW64 ~/Desktop/DevOps/DevOps - Jun 2024/3.GIT/Practicing git (feature1)
$ ls
f1  f2

ashis@DESKTOP-K9O7KNT MINGW64 ~/Desktop/DevOps/DevOps - Jun 2024/3.GIT/Practicing git (feature1)
$ git status
On branch feature1
nothing to commit, working tree clean

ashis@DESKTOP-K9O7KNT MINGW64 ~/Desktop/DevOps/DevOps - Jun 2024/3.GIT/Practicing git (feature1)
$ git log
commit 294d23503e0aad45edb8a7b7a7c1edbf0e8e264d (HEAD -> feature1)
Author: Ashish <Ashish@gmail.com>
Date:   Thu Aug 1 11:11:10 2024 +0530

    Chnages in f1 from feature1

commit b483dc98d6f39f0214250f7692757a4cf0b0829e (master)
Author: Ashish <Ashish@gmail.com>
Date:   Thu Aug 1 11:09:16 2024 +0530

    Add f2

commit b95bf30d791ce45a92da1dfd10d392ec630af2a8
Author: Ashish <Ashish@gmail.com>
Date:   Thu Aug 1 11:09:02 2024 +0530

    Add f1
```

- Created a branch feature1 and made the changes in the f1 and committed it.
- Contents of f1 in feature1:
  Hello,

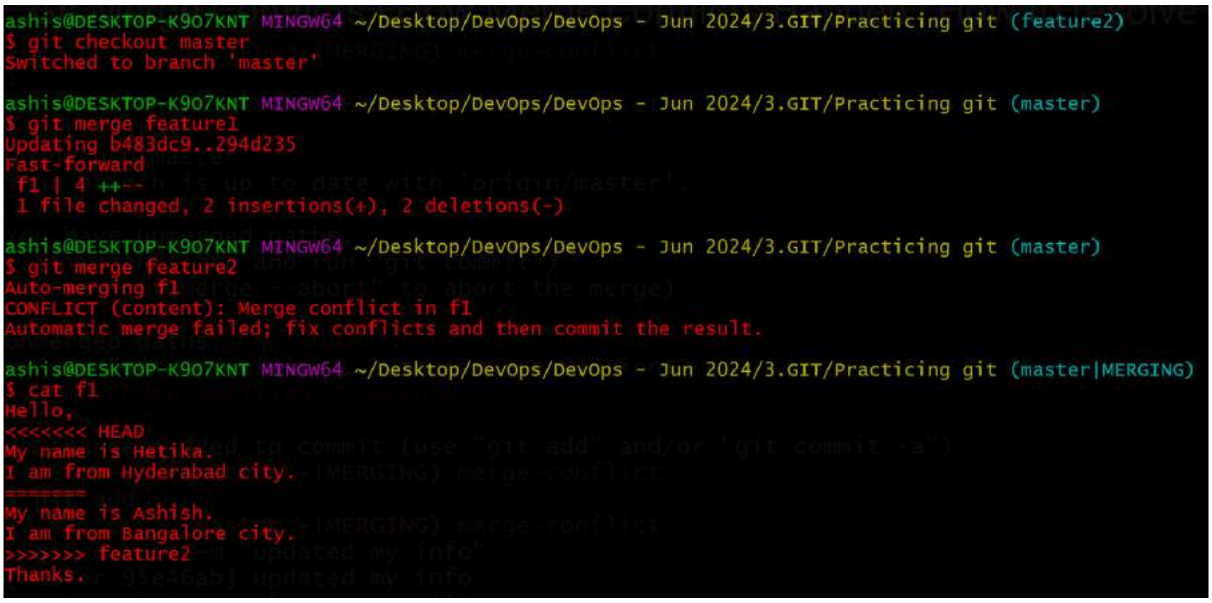
My name is Hetika.
I am from Hyderabad city.
Thanks.



```
ashis@DESKTOP-K9O7KNT MINGW64 ~/Desktop/DevOps/DevOps - Jun 2024/3.GIT/Practicing git (feature1)
$ git checkout feature2
Switched to branch 'feature2'

ashis@DESKTOP-K9O7KNT MINGW64 ~/Desktop/DevOps/DevOps - Jun 2024/3.GIT/Practicing git (feature2)
$ ls
f1  f2

ashis@DESKTOP-K9O7KNT MINGW64 ~/Desktop/DevOps/DevOps - Jun 2024/3.GIT/Practicing git (feature2)
$ git status
On branch feature2
nothing to commit, working tree clean

ashis@DESKTOP-K9O7KNT MINGW64 ~/Desktop/DevOps/DevOps - Jun 2024/3.GIT/Practicing git (feature2)
$ git log
commit a681e2f1247d335cfdefd9c076810e8d161e2065 (HEAD -> feature2)
Author: Ashish <Ashish@gmail.com>
Date:   Thu Aug 1 11:13:27 2024 +0530

    Changes in f1 in feature2

commit b483dc98d6f39f0214250f7692757a4cf0b0829e (master)
Author: Ashish <Ashish@gmail.com>
Date:   Thu Aug 1 11:09:16 2024 +0530

    Add f2

commit b95bf30d791ce45a92da1dfd10d392ec630af2a8
Author: Ashish <Ashish@gmail.com>
Date:   Thu Aug 1 11:09:02 2024 +0530

    Add f1
```

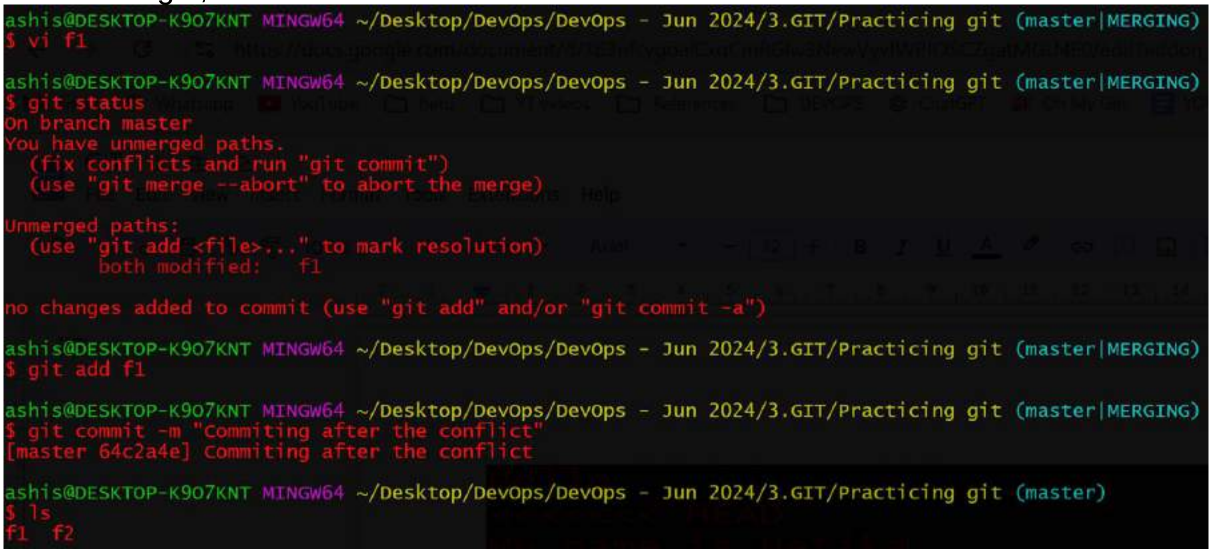
- Created a branch feature2 and made the changes in the f1 and committed it.
- Contents of f1 in feature2:
  Hello,
  My name is Ashish.
  I am from Bangalore city.
  Thanks.

- After making the changes in the f1 in both the branches, firstly merged the feature1 with master. It got merged automatically.
- Later, when I tried merging the branch feature2 with master, the conflict occurred.
- Resolved the conflict by opening the file f1 and retaining all the necessary changes in it and saving it.
- Then, I have to add and commit the file f1 to resolve the conflict.



```
ashis@DESKTOP-K9O7KNT MINGW64 ~/Desktop/DevOps/DevOps - Jun 2024/3.GIT/Practicing git (feature2)
$ git checkout master
Switched to branch 'master'

ashis@DESKTOP-K9O7KNT MINGW64 ~/Desktop/DevOps/DevOps - Jun 2024/3.GIT/Practicing git (master)
$ git merge feature1
Updating b483dc9..294d235
Fast-forward
 f1 | 4 ++--
 1 file changed, 2 insertions(+), 2 deletions(-)

ashis@DESKTOP-K9O7KNT MINGW64 ~/Desktop/DevOps/DevOps - Jun 2024/3.GIT/Practicing git (master)
$ git merge feature2
Auto-merging f1
CONFLICT (content): Merge conflict in f1
Automatic merge failed; fix conflicts and then commit the result.

ashis@DESKTOP-K9O7KNT MINGW64 ~/Desktop/DevOps/DevOps - Jun 2024/3.GIT/Practicing git (master|MERGING)
$ cat f1
Hello,
<<<<<<< HEAD
My name is Hetika.
I am from Hyderabad city.
=======
My name is Ashish.
I am from Bangalore city.
>>>>>>> feature2
Thanks.
```

After editing it,



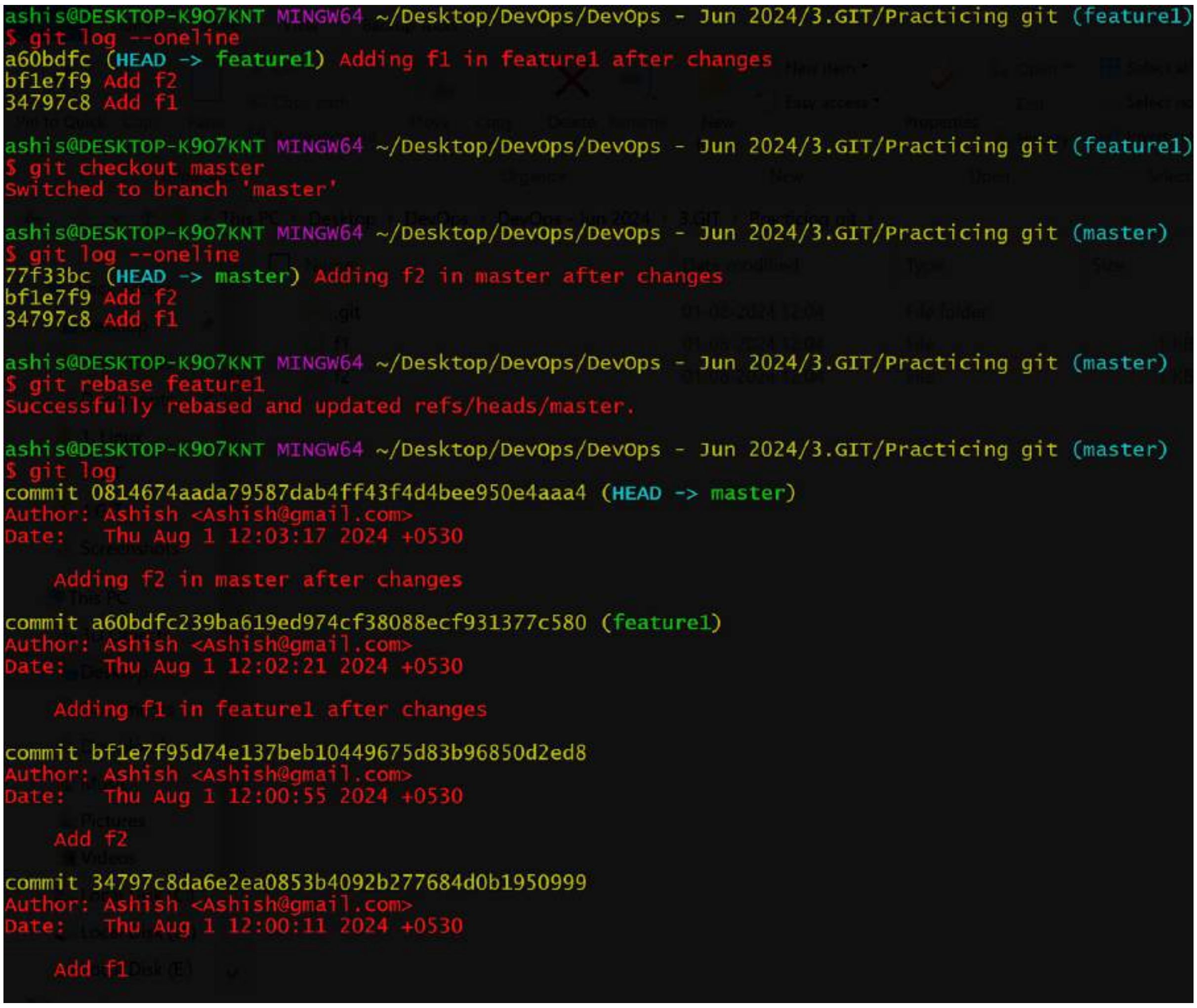
```
ashis@DESKTOP-K9O7KNT MINGW64 ~/Desktop/DevOps/DevOps - Jun 2024/3.GIT/Practicing git (master|MERGING)
$ vi f1

ashis@DESKTOP-K9O7KNT MINGW64 ~/Desktop/DevOps/DevOps - Jun 2024/3.GIT/Practicing git (master|MERGING)
$ git status
On branch master
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:   f1

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

ashis@DESKTOP-K9O7KNT MINGW64 ~/Desktop/DevOps/DevOps - Jun 2024/3.GIT/Practicing git (master|MERGING)
$ git add f1

ashis@DESKTOP-K9O7KNT MINGW64 ~/Desktop/DevOps/DevOps - Jun 2024/3.GIT/Practicing git (master|MERGING)
$ git commit -m "Commiting after the conflict"
[master 64c2a4e] Commiting after the conflict

ashis@DESKTOP-K9O7KNT MINGW64 ~/Desktop/DevOps/DevOps - Jun 2024/3.GIT/Practicing git (master)
$ ls
f1  f2
```

****

14.git rebase:

git rebase <branch_name> : rewrites the history by creating new commits for each commit in source branch since commit history is rewritten, it will be difficult to understand the conflict in some cases as commits are no longer reachable.

- Created f1 f2 in master and committed it
- Created feature1 and made changes in the f1 and committed it again.
- Moved to master and made changes in the f2 and committed it again.
- On git rebase, we can see that the feature1 has been merged with master.
- We can see that the commit ID has been changed after the rebase.



```
ashis@DESKTOP-K907KNT MINGW64 ~/Desktop/DevOps/DevOps - Jun 2024/3.GIT/Practicing git (feature1)
$ git log --oneline
a60bdfc (HEAD -> feature1) Adding f1 in feature1 after changes
bf1e7f9 Add f2
34797c8 Add f1

ashis@DESKTOP-K907KNT MINGW64 ~/Desktop/DevOps/DevOps - Jun 2024/3.GIT/Practicing git (feature1)
$ git checkout master
Switched to branch 'master'

ashis@DESKTOP-K907KNT MINGW64 ~/Desktop/DevOps/DevOps - Jun 2024/3.GIT/Practicing git (master)
$ git log --oneline
77f33bc (HEAD -> master) Adding f2 in master after changes
bf1e7f9 Add f2
34797c8 Add f1

ashis@DESKTOP-K907KNT MINGW64 ~/Desktop/DevOps/DevOps - Jun 2024/3.GIT/Practicing git (master)
$ git rebase feature1
Successfully rebased and updated refs/heads/master.

ashis@DESKTOP-K907KNT MINGW64 ~/Desktop/DevOps/DevOps - Jun 2024/3.GIT/Practicing git (master)
$ git log
commit 0814674aada79587dab4ff43f4d4bee950e4aaa4 (HEAD -> master)
Author: Ashish <Ashish@gmail.com>
Date:   Thu Aug 1 12:03:17 2024 +0530

    Adding f2 in master after changes

commit a60bdfc239ba619ed974cf38088ecf931377c580 (feature1)
Author: Ashish <Ashish@gmail.com>
Date:   Thu Aug 1 12:02:21 2024 +0530

    Adding f1 in feature1 after changes

commit bf1e7f95d74e137beb10449675d83b96850d2ed8
Author: Ashish <Ashish@gmail.com>
Date:   Thu Aug 1 12:00:55 2024 +0530

    Add f2

commit 34797c8da6e2ea0853b4092b277684d0b1950999
Author: Ashish <Ashish@gmail.com>
Date:   Thu Aug 1 12:00:11 2024 +0530

    Add f1
```

****

15.What is the difference between Merge and rebase?
Both merge and rebase perform the same operation of integrating branches, but the difference is how they do it.
Merge :

- Creates new commitID indicating merge.
- Merge conflict can be handled easily, as the commits are reachable

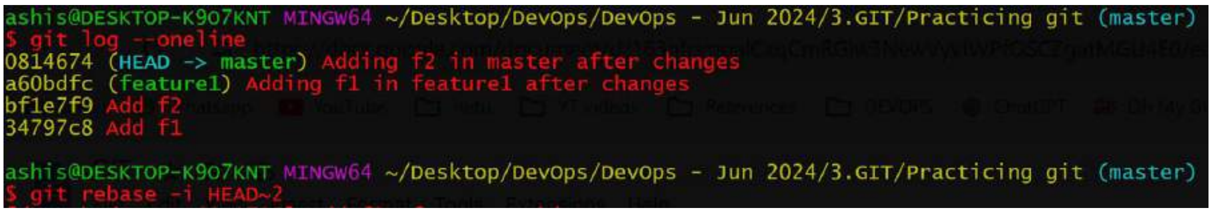
Rebase :

- Rewrites the history by creating new commits for each commit in the source branch.
- Since commit history is rewritten, it will be difficult to understand the conflict in some cases as commits are no longer reachable.

****

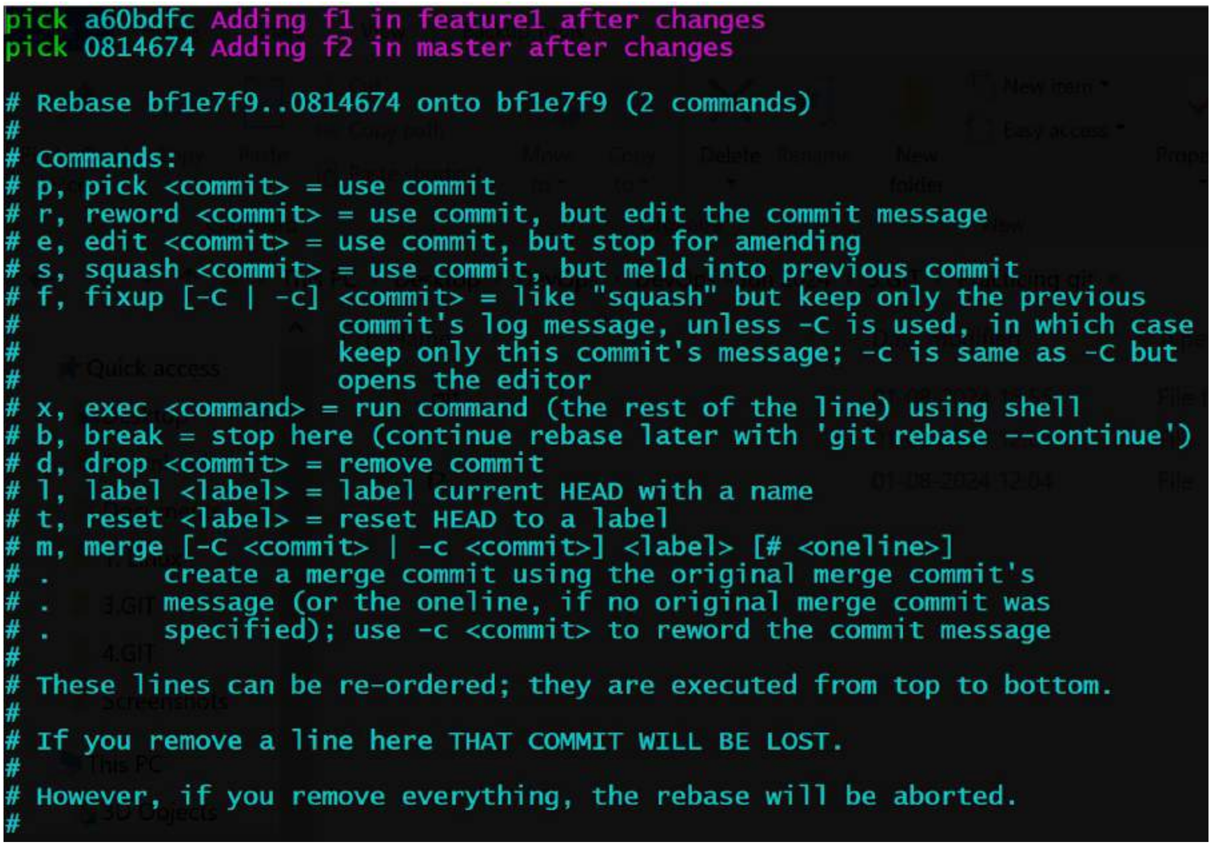
16.git squash : It is a technique to condense large number of commits to make into a small number of meaningful commits so that we can make git history clear

Syntax: git rebase -i HEAD~N



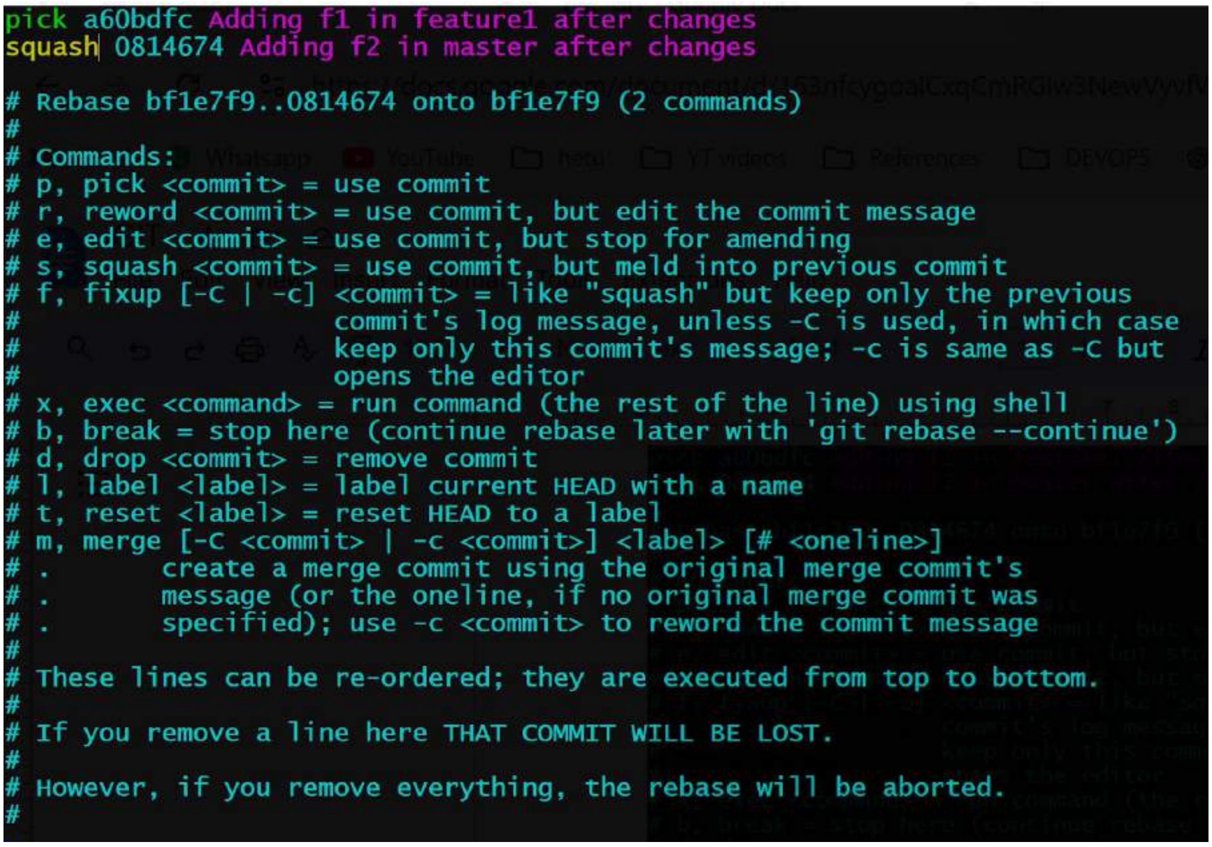
```
ashis@DESKTOP-K9O7KNT MINGW64 ~/Desktop/DevOps/DevOps - Jun 2024/3.GIT/Practicing git (master)
$ git log --oneline
0814674 (HEAD -> master) Adding f2 in master after changes
a60bdfc (feature1) Adding f1 in feature1 after changes
bf1e7f9 Add f2
34797c8 Add f1

ashis@DESKTOP-K9O7KNT MINGW64 ~/Desktop/DevOps/DevOps - Jun 2024/3.GIT/Practicing git (master)
$ git rebase -i HEAD~2
```

- On git rebase -i HEAD~2, a text editor will open.



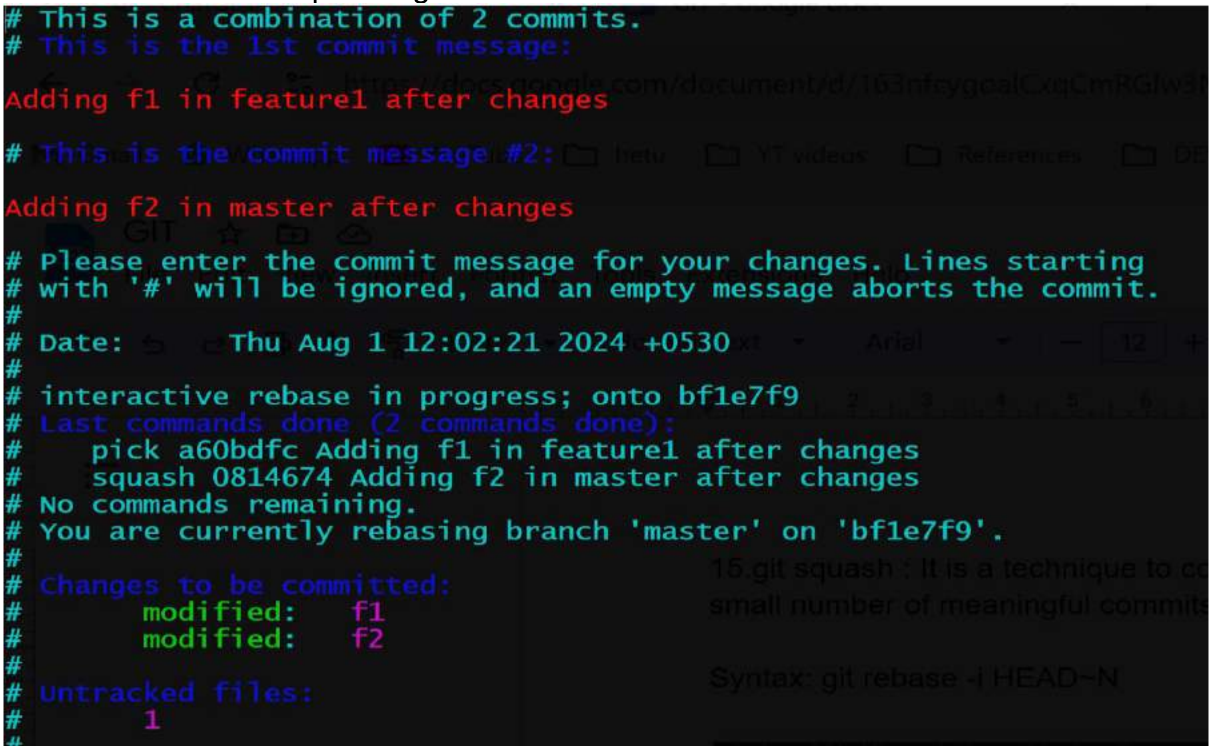
```
pick a60bdfc Adding f1 in feature1 after changes
pick 0814674 Adding f2 in master after changes

# Rebase bf1e7f9..0814674 onto bf1e7f9 (2 commands)
#
# Commands:
# p, pick <commit> = use commit
# r, reword <commit> = use commit, but edit the commit message
# e, edit <commit> = use commit, but stop for amending
# s, squash <commit> = use commit, but meld into previous commit
# f, fixup [-C | -c] <commit> = like "squash" but keep only the previous
#                    commit's log message, unless -C is used, in which case
#                    keep only this commit's message; -c is same as -C but
#                    opens the editor
# x, exec <command> = run command (the rest of the line) using shell
# b, break = stop here (continue rebase later with 'git rebase --continue')
# d, drop <commit> = remove commit
# l, label <label> = label current HEAD with a name
# t, reset <label> = reset HEAD to a label
# m, merge [-C <commit> | -c <commit>] <label> [# <oneline>]
# .       create a merge commit using the original merge commit's
# .       message (or the oneline, if no original merge commit was
# .       specified); use -c <commit> to reword the commit message
#
# These lines can be re-ordered; they are executed from top to bottom.
#
# If you remove a line here THAT COMMIT WILL BE LOST.
#
# However, if you remove everything, the rebase will be aborted.
#
```

- Replace the word pick with squash

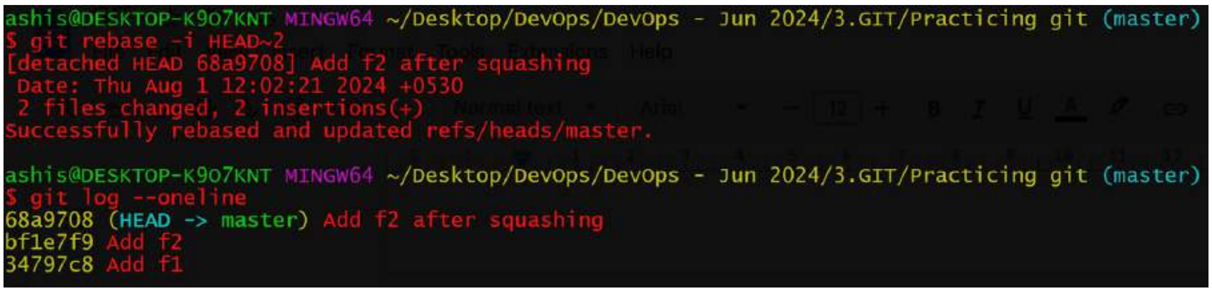


```
pick a60bdfc Adding f1 in feature1 after changes
squash 0814674 Adding f2 in master after changes

# Rebase bf1e7f9..0814674 onto bf1e7f9 (2 commands)
#
# Commands:
# p, pick <commit> = use commit
# r, reword <commit> = use commit, but edit the commit message
# e, edit <commit> = use commit, but stop for amending
# s, squash <commit> = use commit, but meld into previous commit
# f, fixup [-C | -c] <commit> = like "squash" but keep only the previous
#                    commit's log message, unless -C is used, in which case
#                    keep only this commit's message; -c is same as -C but
#                    opens the editor
# x, exec <command> = run command (the rest of the line) using shell
# b, break = stop here (continue rebase later with 'git rebase --continue')
# d, drop <commit> = remove commit
# l, label <label> = label current HEAD with a name
# t, reset <label> = reset HEAD to a label
# m, merge [-C <commit> | -c <commit>] <label> [# <oneline>]
# .       create a merge commit using the original merge commit's
# .       message (or the oneline, if no original merge commit was
# .       specified); use -c <commit> to reword the commit message
#
# These lines can be re-ordered; they are executed from top to bottom.
#
# If you remove a line here THAT COMMIT WILL BE LOST.
#
# However, if you remove everything, the rebase will be aborted.
#
```

- After replacing the word pick with squash, another editor will open.
- Edit the comment as you want and save it.
- In this case, I have removed both the comments and added the comment “Add f2 after squashing” in the editor and saved it.

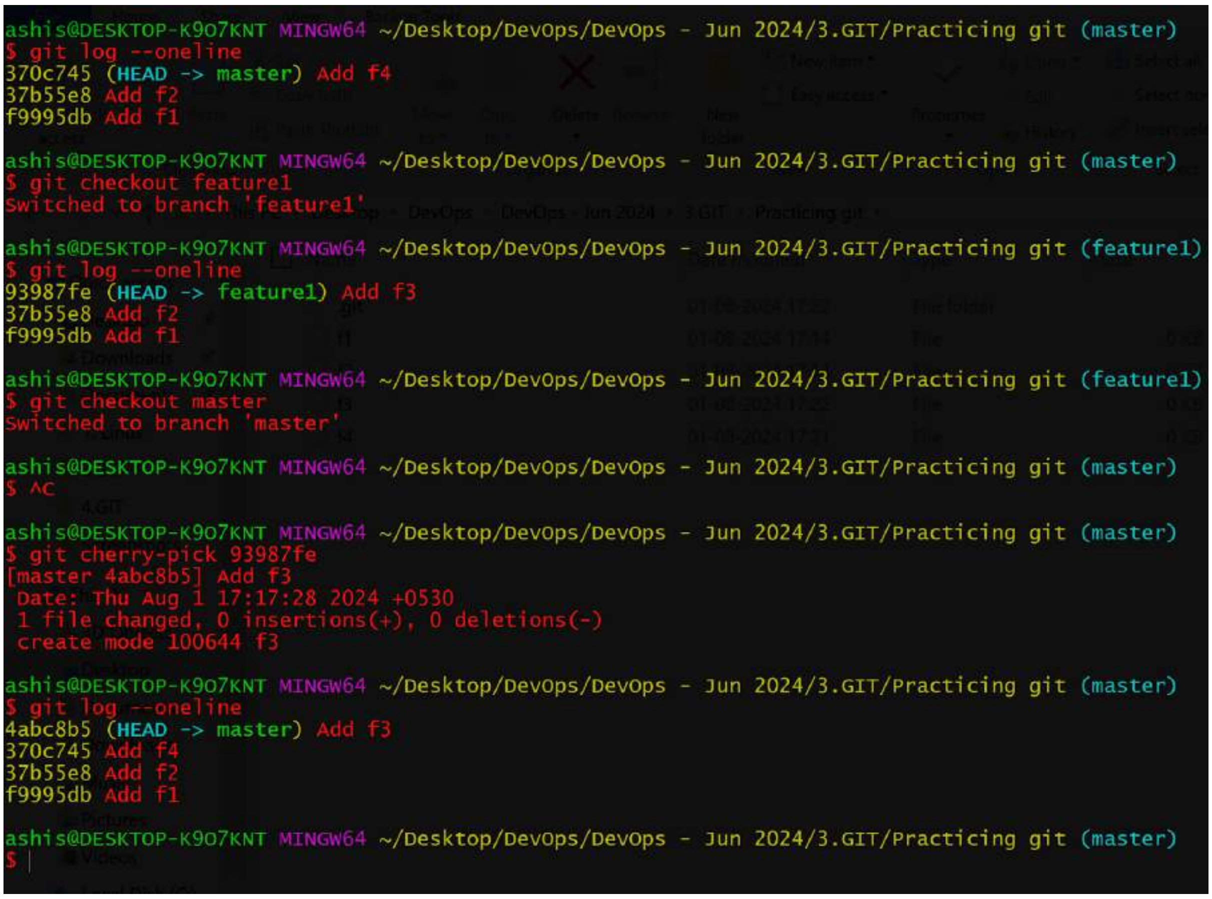


```
# This is a combination of 2 commits.
# This is the 1st commit message:

Adding f1 in feature1 after changes

# This is the commit message #2:

Adding f2 in master after changes

# Please enter the commit message for your changes. Lines starting
# with '#' will be ignored, and an empty message aborts the commit.
#
# Date:      Thu Aug 1 12:02:21 2024 +0530
#
# interactive rebase in progress; onto bf1e7f9
# Last commands done (2 commands done):
#    pick a60bdfc Adding f1 in feature1 after changes
#    squash 0814674 Adding f2 in master after changes
# No commands remaining.
# You are currently rebasing branch 'master' on 'bf1e7f9'.
#
# Changes to be committed:
#       modified:   f1
#       modified:   f2
#
# Untracked files:
#       1
#
```


```
ashis@DESKTOP-K907KNT MINGW64 ~/Desktop/DevOps/DevOps - Jun 2024/3.GIT/Practicing git (master)
$ git rebase -i HEAD~2
[detached HEAD 68a9708] Add f2 after squashing
 Date: Thu Aug 1 12:02:21 2024 +0530
 2 files changed, 2 insertions(+)
Successfully rebased and updated refs/heads/master.

ashis@DESKTOP-K907KNT MINGW64 ~/Desktop/DevOps/DevOps - Jun 2024/3.GIT/Practicing git (master)
$ git log --oneline
68a9708 (HEAD -> master) Add f2 after squashing
bf1e7f9 Add f2
34797c8 Add f1
```
****

17.git cherry-pick : sometimes we may commit the file wrongly in other branch instead of committing on the required branch, then we can use cherry-pick to merge that commit to the branch that we require.

Syntax: git cherry-pick <commit_id> → to merge specific commit on branch(currently checkout branch)


```
ashis@DESKTOP-K907KNT MINGW64 ~/Desktop/DevOps/DevOps - Jun 2024/3.GIT/Practicing git (master)
$ git log --oneline
370c745 (HEAD -> master) Add f4
37b55e8 Add f2
f9995db Add f1

ashis@DESKTOP-K907KNT MINGW64 ~/Desktop/DevOps/DevOps - Jun 2024/3.GIT/Practicing git (master)
$ git checkout feature1
Switched to branch 'feature1'

ashis@DESKTOP-K907KNT MINGW64 ~/Desktop/DevOps/DevOps - Jun 2024/3.GIT/Practicing git (feature1)
$ git log --oneline
93987fe (HEAD -> feature1) Add f3
37b55e8 Add f2
f9995db Add f1

ashis@DESKTOP-K907KNT MINGW64 ~/Desktop/DevOps/DevOps - Jun 2024/3.GIT/Practicing git (feature1)
$ git checkout master
Switched to branch 'master'

ashis@DESKTOP-K907KNT MINGW64 ~/Desktop/DevOps/DevOps - Jun 2024/3.GIT/Practicing git (master)
$ ^C

ashis@DESKTOP-K907KNT MINGW64 ~/Desktop/DevOps/DevOps - Jun 2024/3.GIT/Practicing git (master)
$ git cherry-pick 93987fe
[master 4abc8b5] Add f3
 Date: Thu Aug 1 17:17:28 2024 +0530
 1 file changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 f3

ashis@DESKTOP-K907KNT MINGW64 ~/Desktop/DevOps/DevOps - Jun 2024/3.GIT/Practicing git (master)
$ git log --oneline
4abc8b5 (HEAD -> master) Add f3
370c745 Add f4
37b55e8 Add f2
f9995db Add f1

ashis@DESKTOP-K907KNT MINGW64 ~/Desktop/DevOps/DevOps - Jun 2024/3.GIT/Practicing git (master)
$ |
```

- f1 f2 f4 in master, added and committed
- f3 in feature1, added and committed.
- I have wrongly committed f3 in the feature1, so did git cherry-pick <commit_ID> and commit on the master.

****

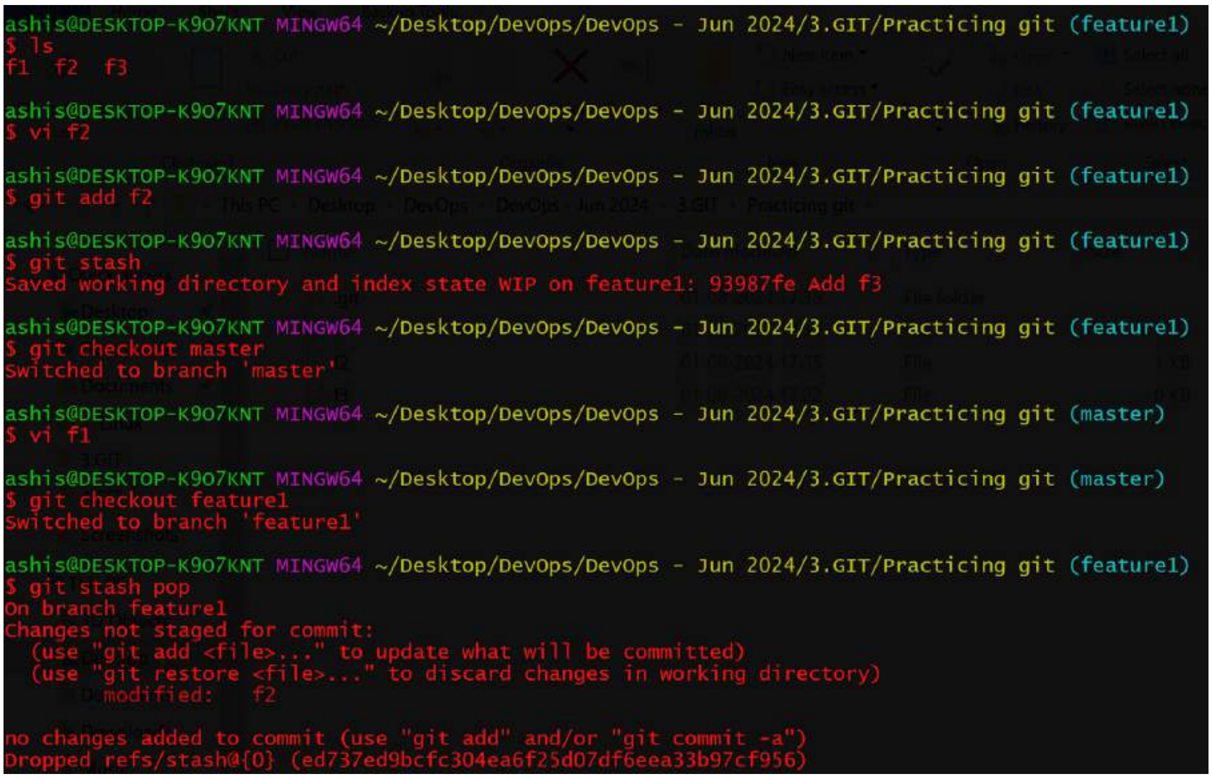
18.git stash:

- If I am working on 1 branch and I get critical work/bug to be fixed on other branch, I don't want to commit changes in current branch, so I will do git stash where the files will be stored in a temporary area, and switched to another branch , I will fix the issue and come back to the previous branch to continue my work.

- I need to do git stash pop, to get back files from a temporary area..

Syntax:

- git stash
- git stash pop



```
ashis@DESKTOP-K9O7KNT MINGW64 ~/Desktop/DevOps/DevOps - Jun 2024/3.GIT/Practicing git (feature1)
$ ls
f1  f2  f3

ashis@DESKTOP-K9O7KNT MINGW64 ~/Desktop/DevOps/DevOps - Jun 2024/3.GIT/Practicing git (feature1)
$ vi f2

ashis@DESKTOP-K9O7KNT MINGW64 ~/Desktop/DevOps/DevOps - Jun 2024/3.GIT/Practicing git (feature1)
$ git add f2

ashis@DESKTOP-K9O7KNT MINGW64 ~/Desktop/DevOps/DevOps - Jun 2024/3.GIT/Practicing git (feature1)
$ git stash
Saved working directory and index state WIP on feature1: 93987fe Add f3

ashis@DESKTOP-K9O7KNT MINGW64 ~/Desktop/DevOps/DevOps - Jun 2024/3.GIT/Practicing git (feature1)
$ git checkout master
Switched to branch 'master'

ashis@DESKTOP-K9O7KNT MINGW64 ~/Desktop/DevOps/DevOps - Jun 2024/3.GIT/Practicing git (master)
$ vi f1

ashis@DESKTOP-K9O7KNT MINGW64 ~/Desktop/DevOps/DevOps - Jun 2024/3.GIT/Practicing git (master)
$ git checkout feature1
Switched to branch 'feature1'

ashis@DESKTOP-K9O7KNT MINGW64 ~/Desktop/DevOps/DevOps - Jun 2024/3.GIT/Practicing git (feature1)
$ git stash pop
On branch feature1
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:   f2

no changes added to commit (use "git add" and/or "git commit -a")
Dropped refs/stash@{0} (ed737ed9bcfc304ea6f25d07df6eea33b97cf956)
```

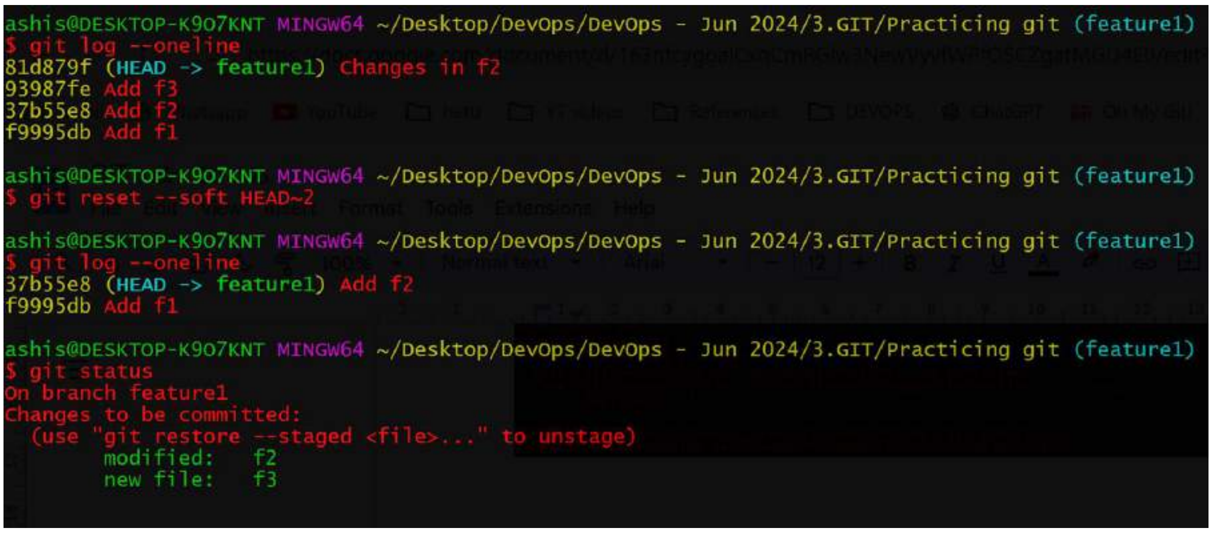
****

19. git reset : undo the committed changes, history will be removed .

Syntax:

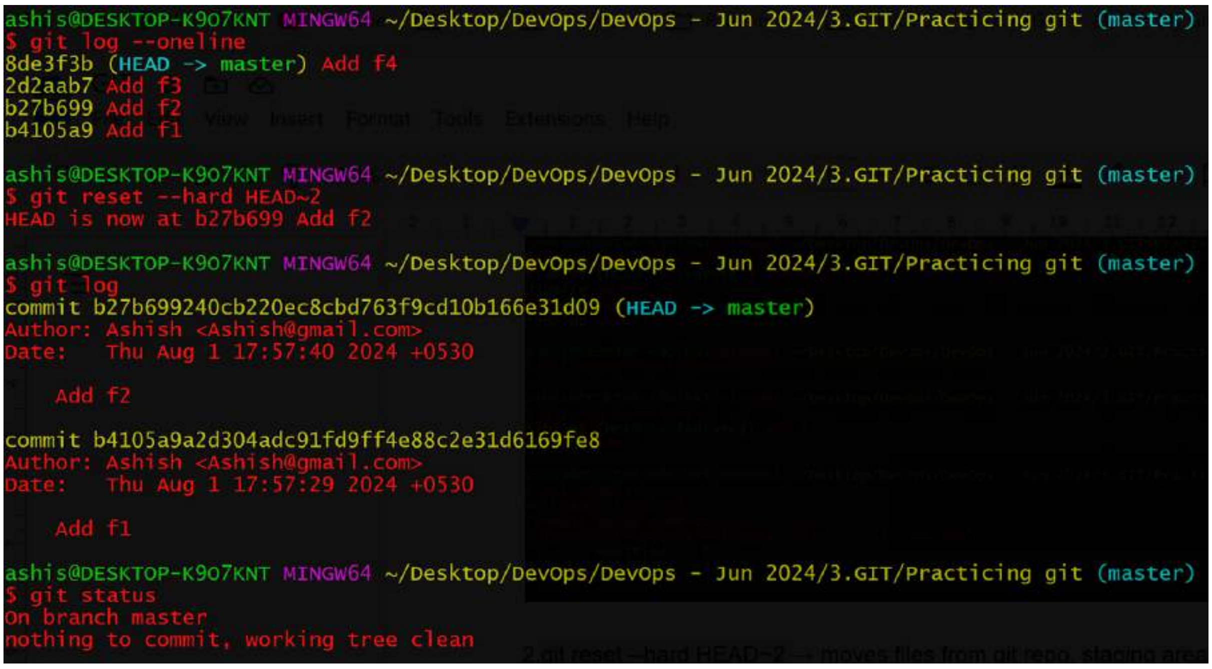
- git reset --mixed HEAD~N → moves files from staging area to workspace
- git reset --soft HEAD~N → moves files from git repo to staging area, history will removed
- git reset --hard HEAD~N → moves files from git repo, staging area, workspace and commit ID are also removed

a.git reset -- soft HEAD~2 → moves files from git repo to staging area, history will removed



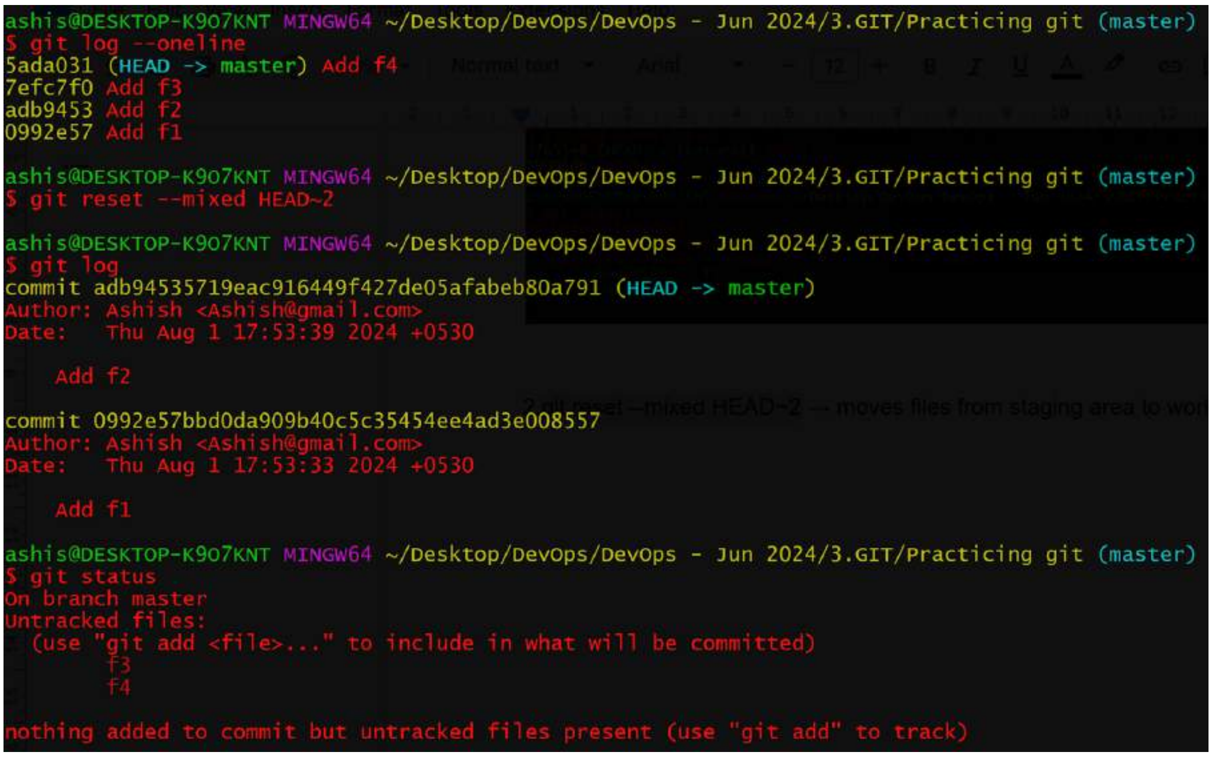
```
ashis@DESKTOP-K907KNT MINGW64 ~/Desktop/DevOps/DevOps - Jun 2024/3.GIT/Practicing git (feature1)
$ git log --oneline
81d879f (HEAD -> feature1) Changes in f2
93987fe Add f3
37b55e8 Add f2
f9995db Add f1

ashis@DESKTOP-K907KNT MINGW64 ~/Desktop/DevOps/DevOps - Jun 2024/3.GIT/Practicing git (feature1)
$ git reset --soft HEAD~2

ashis@DESKTOP-K907KNT MINGW64 ~/Desktop/DevOps/DevOps - Jun 2024/3.GIT/Practicing git (feature1)
$ git log --oneline
37b55e8 (HEAD -> feature1) Add f2
f9995db Add f1

ashis@DESKTOP-K907KNT MINGW64 ~/Desktop/DevOps/DevOps - Jun 2024/3.GIT/Practicing git (feature1)
$ git status
On branch feature1
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        modified:   f2
        new file:   f3
```

b. git reset --hard HEAD~2 → moves files from git repo, staging area, workspace and commit ID are also removed



```
ashis@DESKTOP-K907KNT MINGW64 ~/Desktop/DevOps/DevOps - Jun 2024/3.GIT/Practicing git (master)
$ git log --oneline
8de3f3b (HEAD -> master) Add f4
2d2aab7 Add f3
b27b699 Add f2
b4105a9 Add f1

ashis@DESKTOP-K907KNT MINGW64 ~/Desktop/DevOps/DevOps - Jun 2024/3.GIT/Practicing git (master)
$ git reset --hard HEAD~2
HEAD is now at b27b699 Add f2

ashis@DESKTOP-K907KNT MINGW64 ~/Desktop/DevOps/DevOps - Jun 2024/3.GIT/Practicing git (master)
$ git log
commit b27b699240cb220ec8cbd763f9cd10b166e31d09 (HEAD -> master)
Author: Ashish <Ashish@gmail.com>
Date:   Thu Aug 1 17:57:40 2024 +0530

    Add f2

commit b4105a9a2d304adc91fd9ff4e88c2e31d6169fe8
Author: Ashish <Ashish@gmail.com>
Date:   Thu Aug 1 17:57:29 2024 +0530

    Add f1

ashis@DESKTOP-K907KNT MINGW64 ~/Desktop/DevOps/DevOps - Jun 2024/3.GIT/Practicing git (master)
$ git status
On branch master
nothing to commit, working tree clean
```

c . git reset --mixed HEAD~2 → moves files from staging area to workspace



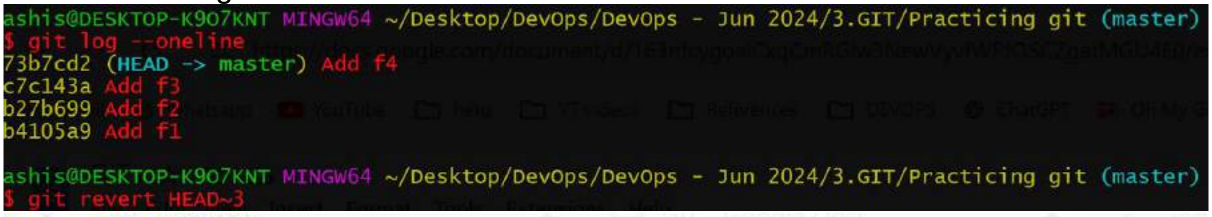
```
ashis@DESKTOP-K907KNT MINGW64 ~/Desktop/DevOps/DevOps - Jun 2024/3.GIT/Practicing git (master)
$ git log --oneline
5ada031 (HEAD -> master) Add f4
7efc7f0 Add f3
adb9453 Add f2
0992e57 Add f1

ashis@DESKTOP-K907KNT MINGW64 ~/Desktop/DevOps/DevOps - Jun 2024/3.GIT/Practicing git (master)
$ git reset --mixed HEAD~2

ashis@DESKTOP-K907KNT MINGW64 ~/Desktop/DevOps/DevOps - Jun 2024/3.GIT/Practicing git (master)
$ git log
commit adb94535719eac916449f427de05afabeb80a791 (HEAD -> master)
Author: Ashish <Ashish@gmail.com>
Date:   Thu Aug 1 17:53:39 2024 +0530

    Add f2

commit 0992e57bbd0da909b40c5c35454ee4ad3e008557
Author: Ashish <Ashish@gmail.com>
Date:   Thu Aug 1 17:53:33 2024 +0530

    Add f1

ashis@DESKTOP-K907KNT MINGW64 ~/Desktop/DevOps/DevOps - Jun 2024/3.GIT/Practicing git (master)
$ git status
On branch master
Untracked files:
  (use "git add <file>..." to include in what will be committed)
        f3
        f4

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

****

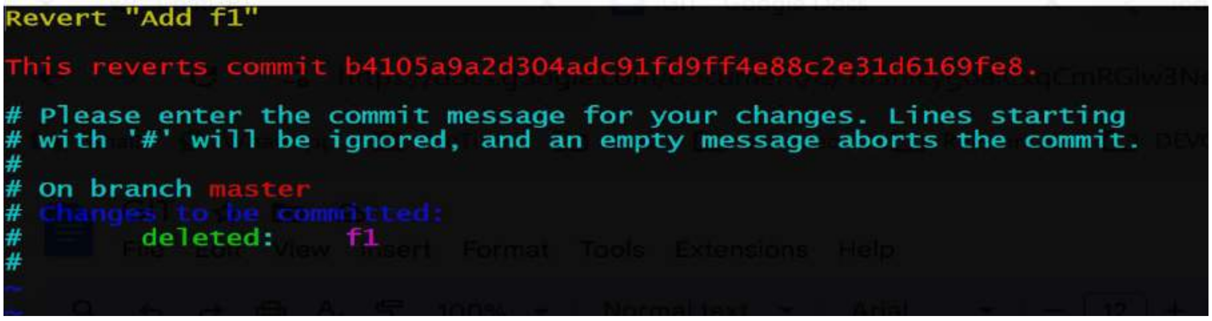
20. git revert - (forward) undo the committed changes , history will be retained, we can track the changes in git log.

Syntax:
git revert HEAD~N
git revert commitID

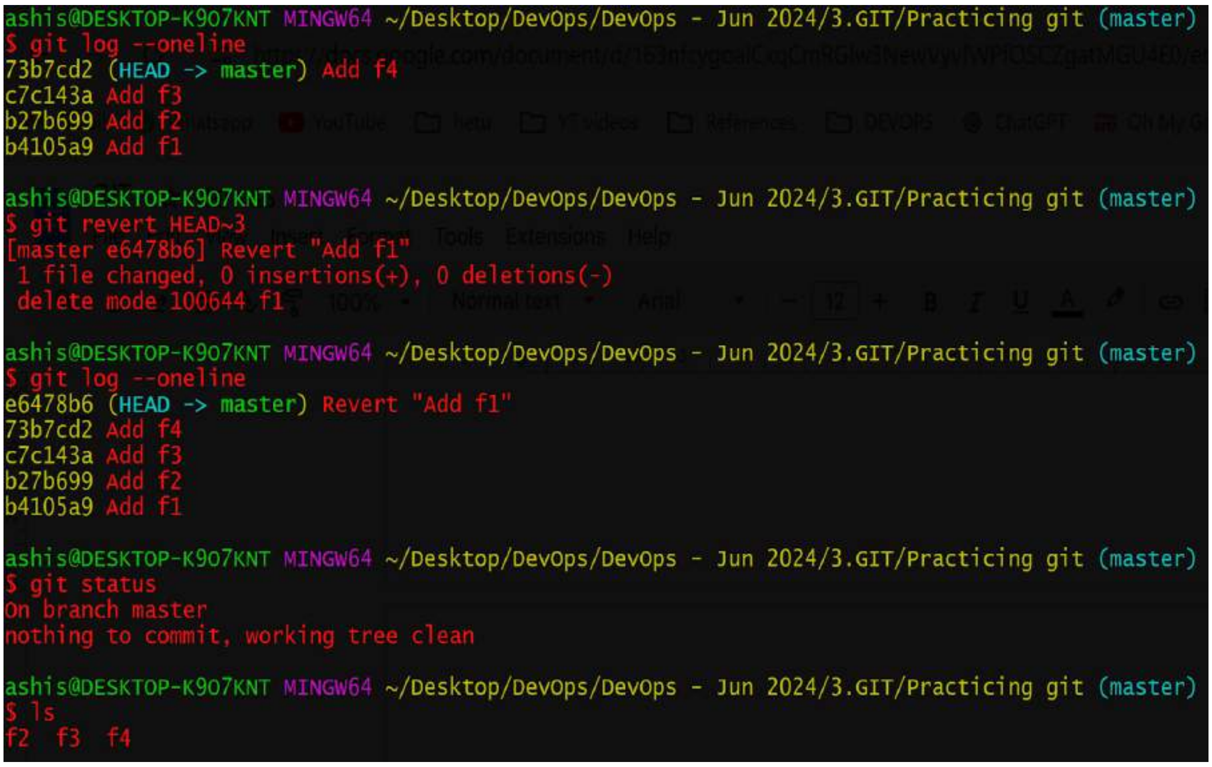
On, git revert HEAD~3, a new text editor will open.
Edit the message in the editor.



```
ashis@DESKTOP-K907KNT MINGW64 ~/Desktop/DevOps/DevOps - Jun 2024/3.GIT/Practicing git (master)
$ git log --oneline
73b7cd2 (HEAD -> master) Add f4
c7c143a Add f3
b27b699 Add f2
b4105a9 Add f1

ashis@DESKTOP-K907KNT MINGW64 ~/Desktop/DevOps/DevOps - Jun 2024/3.GIT/Practicing git (master)
$ git revert HEAD~3
```



```
Revert "Add f1"

This reverts commit b4105a9a2d304adc91fd9ff4e88c2e31d6169fe8.

# Please enter the commit message for your changes. Lines starting
# with '#' will be ignored, and an empty message aborts the commit.
#
# On branch master
# Changes to be committed:
#       deleted:    f1
#
~
```



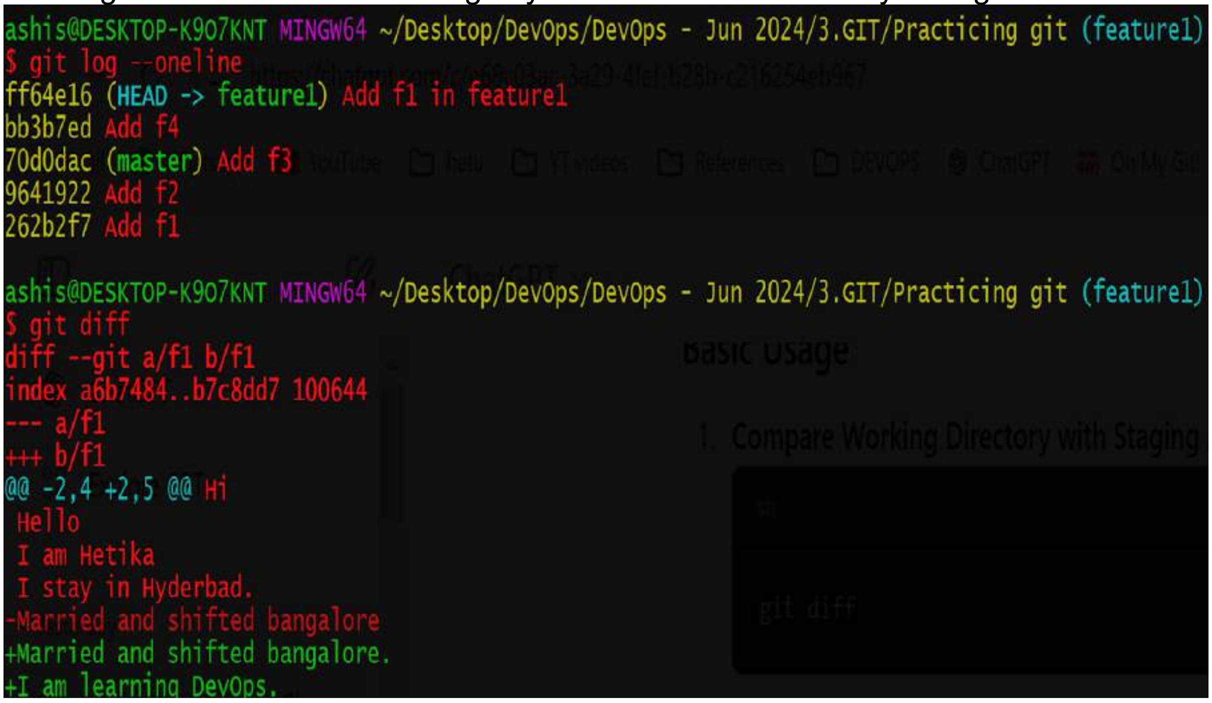
```
ashis@DESKTOP-K907KNT MINGW64 ~/Desktop/DevOps/DevOps - Jun 2024/3.GIT/Practicing git (master)
$ git log --oneline
73b7cd2 (HEAD -> master) Add f4
c7c143a Add f3
b27b699 Add f2
b4105a9 Add f1

ashis@DESKTOP-K907KNT MINGW64 ~/Desktop/DevOps/DevOps - Jun 2024/3.GIT/Practicing git (master)
$ git revert HEAD~3
[master e6478b6] Revert "Add f1"
 1 file changed, 0 insertions(+), 0 deletions(-)
 delete mode 100644 f1

ashis@DESKTOP-K907KNT MINGW64 ~/Desktop/DevOps/DevOps - Jun 2024/3.GIT/Practicing git (master)
$ git log --oneline
e6478b6 (HEAD -> master) Revert "Add f1"
73b7cd2 Add f4
c7c143a Add f3
b27b699 Add f2
b4105a9 Add f1

ashis@DESKTOP-K907KNT MINGW64 ~/Desktop/DevOps/DevOps - Jun 2024/3.GIT/Practicing git (master)
$ git status
On branch master
nothing to commit, working tree clean

ashis@DESKTOP-K907KNT MINGW64 ~/Desktop/DevOps/DevOps - Jun 2024/3.GIT/Practicing git (master)
$ ls
f2  f3  f4
```

****

21. git diff: It is used to display the differences between various Git objects such as commits, branches, files, and more.

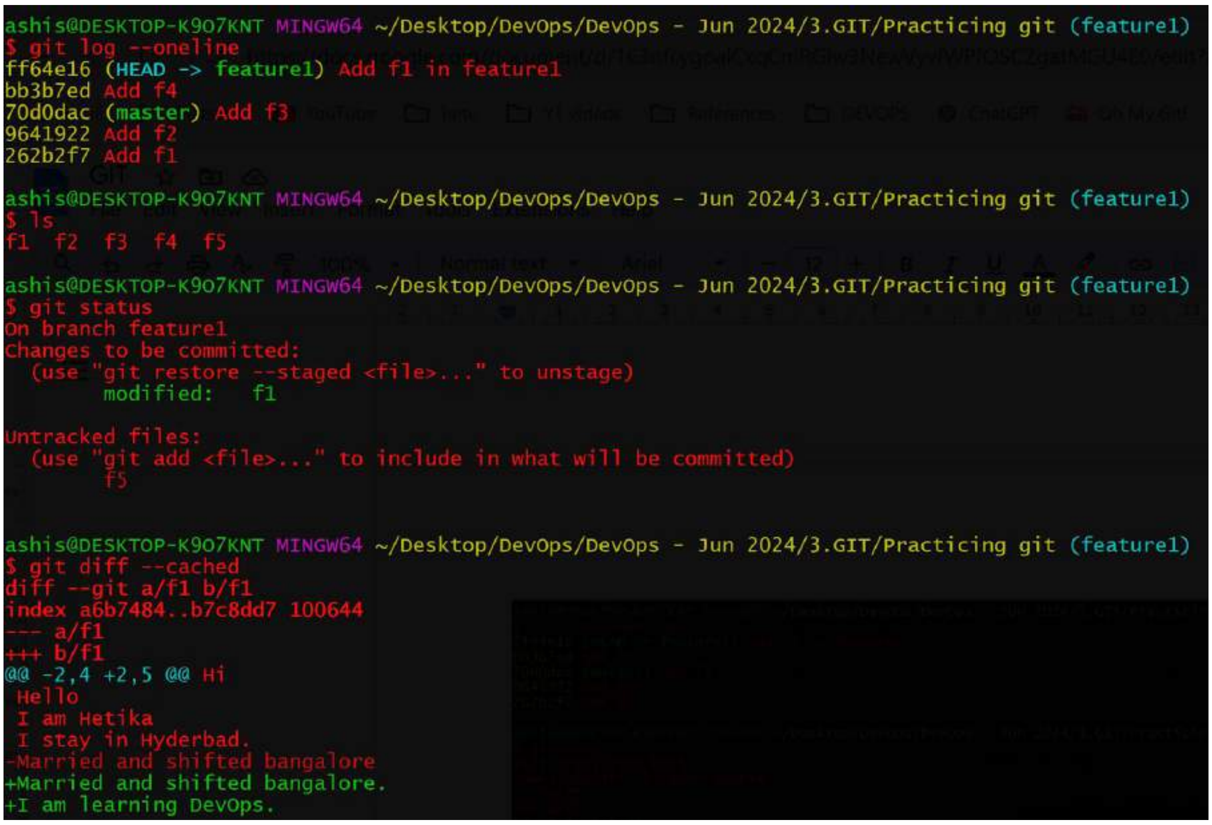
Syntax:

- git diff → To see the changes you've made but haven't yet staged



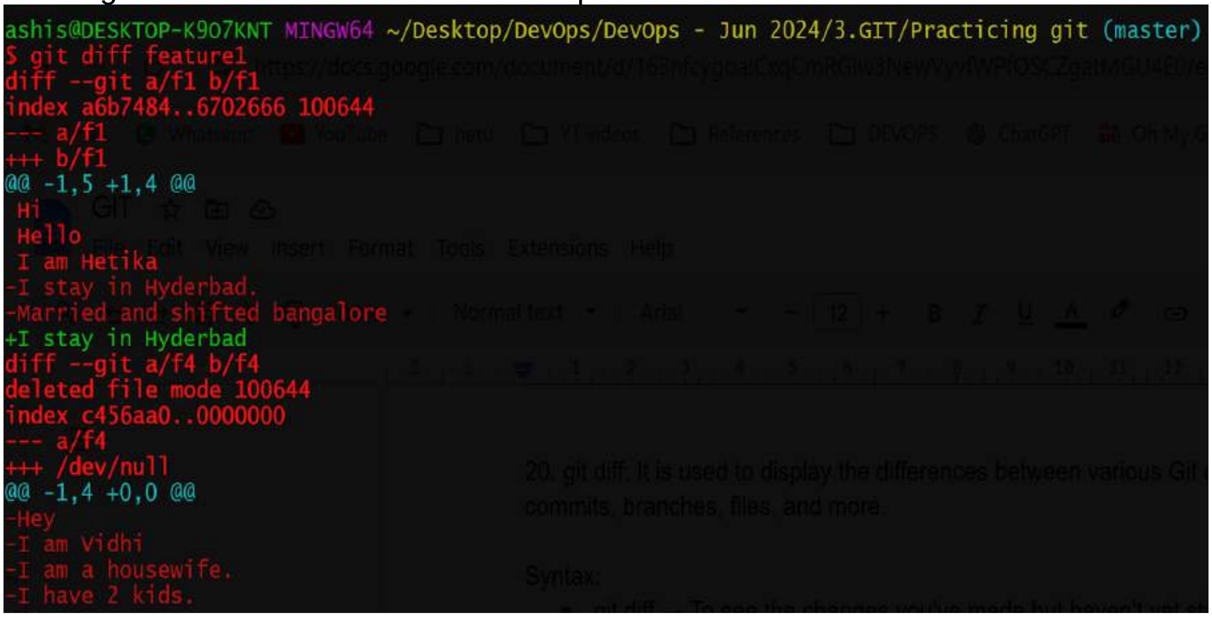
```
ashis@DESKTOP-K907KNT MINGW64 ~/Desktop/DevOps/DevOps - Jun 2024/3.GIT/Practicing git (feature1)
$ git log --oneline
ff64e16 (HEAD -> feature1) Add f1 in feature1
bb3b7ed Add f4
70d0dac (master) Add f3
9641922 Add f2
262b2f7 Add f1

ashis@DESKTOP-K907KNT MINGW64 ~/Desktop/DevOps/DevOps - Jun 2024/3.GIT/Practicing git (feature1)
$ git diff
diff --git a/f1 b/f1
index a6b7484..b7c8dd7 100644
--- a/f1
+++ b/f1
@@ -2,4 +2,5 @@ Hi
 Hello
 I am Hetika
 I stay in Hyderbad.
-Married and shifted bangalore
+Married and shifted bangalore.
+I am learning DevOps.
```

- git diff --cached → To see the changes you've staged that will be included in the next commit



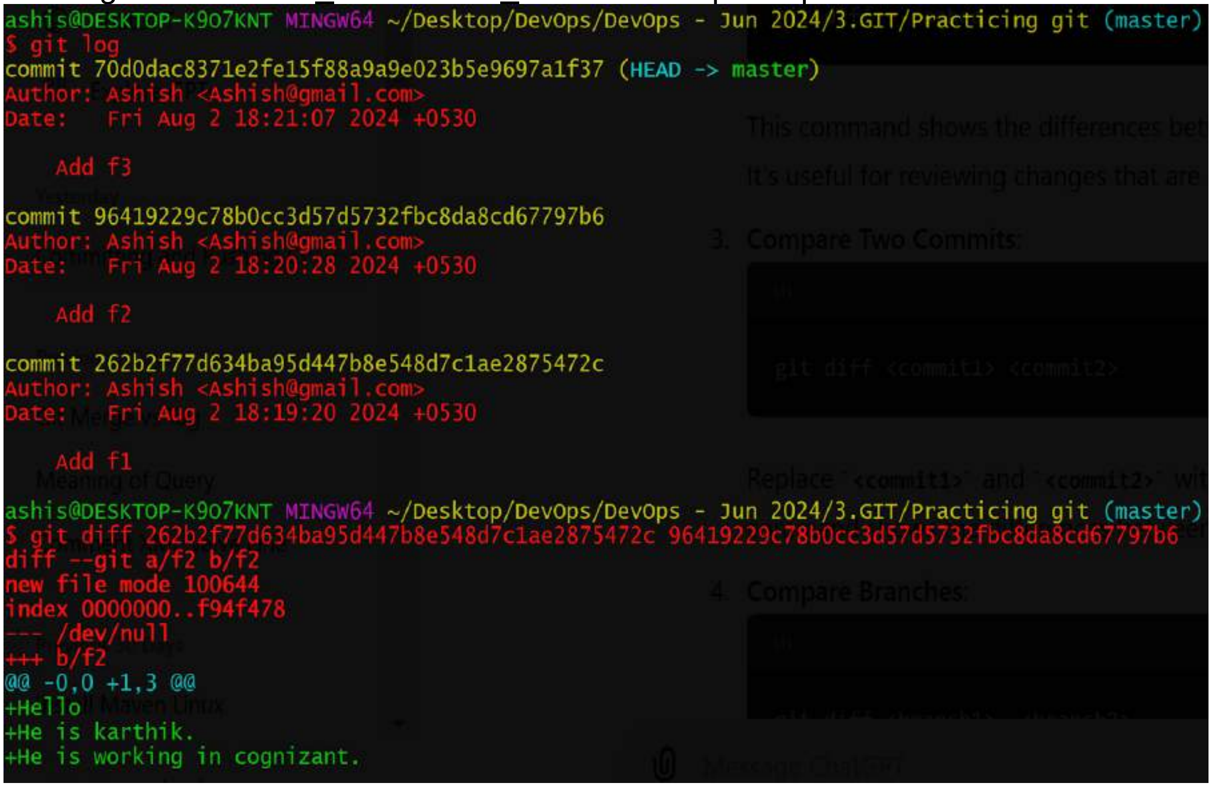
```
ashis@DESKTOP-K907KNT MINGW64 ~/Desktop/DevOps/DevOps - Jun 2024/3.GIT/Practicing git (feature1)
$ git log --oneline
ff64e16 (HEAD -> feature1) Add f1 in feature1
bb3b7ed Add f4
70d0dac (master) Add f3
9641922 Add f2
262b2f7 Add f1

ashis@DESKTOP-K907KNT MINGW64 ~/Desktop/DevOps/DevOps - Jun 2024/3.GIT/Practicing git (feature1)
$ ls
f1  f2  f3  f4  f5

ashis@DESKTOP-K907KNT MINGW64 ~/Desktop/DevOps/DevOps - Jun 2024/3.GIT/Practicing git (feature1)
$ git status
On branch feature1
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        modified:   f1

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        f5


ashis@DESKTOP-K907KNT MINGW64 ~/Desktop/DevOps/DevOps - Jun 2024/3.GIT/Practicing git (feature1)
$ git diff --cached
diff --git a/f1 b/f1
index a6b7484..b7c8dd7 100644
--- a/f1
+++ b/f1
@@ -2,4 +2,5 @@ Hi
 Hello
 I am Hetika
 I stay in Hyderbad.
-Married and shifted bangalore
+Married and shifted bangalore.
+I am learning DevOps.
```

- git diff <branch-name> → To compare the current branch with another branch



```
ashis@DESKTOP-K907KNT MINGW64 ~/Desktop/DevOps/DevOps - Jun 2024/3.GIT/Practicing git (master)
$ git diff feature1
diff --git a/f1 b/f1
index a6b7484..6702666 100644
--- a/f1
+++ b/f1
@@ -1,5 +1,4 @@
 Hi
 Hello
 I am Hetika
-I stay in Hyderbad.
-Married and shifted bangalore
+I stay in Hyderbad
diff --git a/f4 b/f4
deleted file mode 100644
index c456aa0..0000000
--- a/f4
+++ /dev/null
@@ -1,4 +0,0 @@
-Hey
-I am Vidhi
-I am a housewife.
-I have 2 kids.
```

- git diff <commit_1> <commit_2> → To compare specific commits



```
ashis@DESKTOP-K907KNT MINGW64 ~/Desktop/DevOps/DevOps - Jun 2024/3.GIT/Practicing git (master)
$ git log
commit 70d0dac8371e2fe15f88a9a9e023b5e9697a1f37 (HEAD -> master)
Author: Ashish <Ashish@gmail.com>
Date:   Fri Aug 2 18:21:07 2024 +0530

    Add f3

commit 96419229c78b0cc3d57d5732fbc8da8cd67797b6
Author: Ashish <Ashish@gmail.com>
Date:   Fri Aug 2 18:20:28 2024 +0530

    Add f2

commit 262b2f77d634ba95d447b8e548d7c1ae2875472c
Author: Ashish <Ashish@gmail.com>
Date:   Fri Aug 2 18:19:20 2024 +0530

    Add f1

ashis@DESKTOP-K907KNT MINGW64 ~/Desktop/DevOps/DevOps - Jun 2024/3.GIT/Practicing git (master)
$ git diff 262b2f77d634ba95d447b8e548d7c1ae2875472c 96419229c78b0cc3d57d5732fbc8da8cd67797b6
diff --git a/f2 b/f2
new file mode 100644
index 0000000..f94f478
--- /dev/null
+++ b/f2
@@ -0,0 +1,3 @@
+Hello
+He is karthik.
+He is working in cognizant.
```

****

22. git bisect:

- It is a powerful command in Git used for binary searching through a commit history to find which commit introduced a bug or regression.
- The git bisect command helps you perform a binary search through your commit history.
- This is particularly useful when you have a bug or regression in your project and you need to identify the exact commit where the issue was introduced.
- Instead of manually checking each commit, git bisect automates the process by narrowing down the range of commits that could be responsible.

<u>How to Use git bisect:</u>
Start the Bisect Session:
Begin by running git bisect start. This initialises the bisecting process.

Mark Good and Bad Commits:

- Tell Git which commits are known to be good and bad:
  - Use git bisect good <commit> → to mark a commit as good (before the bug was introduced).
  - Use git bisect bad <commit> → to mark a commit as bad (where the bug is present).

Git Performs Binary Search:

- Git will automatically checkout a commit between the known good and bad commits.
- Test your project to see if the bug is present in the checked out commit.

Repeat Until Found:

- Based on your testing, use git bisect good or git bisect bad to continue the process.
- Git will halve the remaining commits to check, narrowing down to the specific commit introducing the bug.

Identify the Culprit Commit:

- Once Git identifies the first bad commit, it will output the commit hash.
- You can then examine this commit to understand what changes may have caused the bug.

End the Bisect Session:

- After finding the culprit commit, end the bisect session with git bisect reset.

Here’s a simplified example of how git bisect might be used:

```
# Start bisecting
git bisect start

# Mark a known good commit
git bisect good v1.0

# Mark a known bad commit
git bisect bad HEAD   # HEAD is typically the current commit

# Git will automatically checkout a commit between v1.0 and HEAD
# Test your project to see if the bug is present

# Based on testing, mark commits as good or bad
git bisect good     # or git bisect bad

# Continue until Git identifies the first bad commit
# Once found, note down the commit hash for further investigation

# End the bisect session
git bisect reset
```

By using git bisect, you can efficiently pinpoint when a bug or regression was introduced, facilitating quicker resolution and minimising the time spent on manual investigation.

****

23. .git ignore:

- The .gitignore file is a text file used by Git to specify files and directories that should be ignored and not tracked by Git.

Purpose of .gitignore:

- Git tracks changes to files in your project's directory, but there are often files and directories that you don't want Git to track. These might include:
  1. Compiled binaries (e.g., .class, .pyc).
  2. Log files generated by your system or application.
  3. Configuration files that differ between developers' environments.
  4. Files containing sensitive information (like passwords or API keys).

By specifying these files and patterns in a .gitignore file, you prevent them from being added to the Git repository accidentally.
****
24. git instaweb

- The git instaweb command is a Git subcommand that launches a local web server to browse your Git repository using a web browser.
- It's a convenient way to visualise your Git repository's history and contents in a graphical interface without leaving your development environment.

****
25. git drop : if we want to delete commits we use this git drop.
****
26. git remote: The git remote command lets you create, view, and delete connections to other repositories.
****
27. git hook : git hooks are scripts that run automatically every time a particular event occurs in a git repository.
****
28. git blame : it will show who has modified the code(each line of the code)
****
29. git fork:

- GitHub fork is a copy of a repository (repo) that sits in your account rather than the account from which you forked the data from.
- Once you have forked a repo, you own your forked copy.
- This means that you can edit the contents of your forked repository without impacting the parent repo.

****
30.git show:

- git-show is a command line utility that is used to view expanded details on Git objects such as blobs, trees, tags, and commits.
- git-show has specific behaviour per object type.
- Blobs show the direct content of the blob.
- Commits show a commit log message and a diff output of the changes in the commit.
- Git show is similar to git log but it also shows which line and what has been modified

****
31. git stash drop:

- When you are done with the stashed item or want to remove it from the list,run the git ‘stash drop’ command.
- It will remove the last added stash item by default, and it can also remove a specific item if you include it as an argument.

****
32. git rm: To remove the file from the staging area and also off your disk ‘git rm’ is used.
****
33. Git Hotfix Branches:

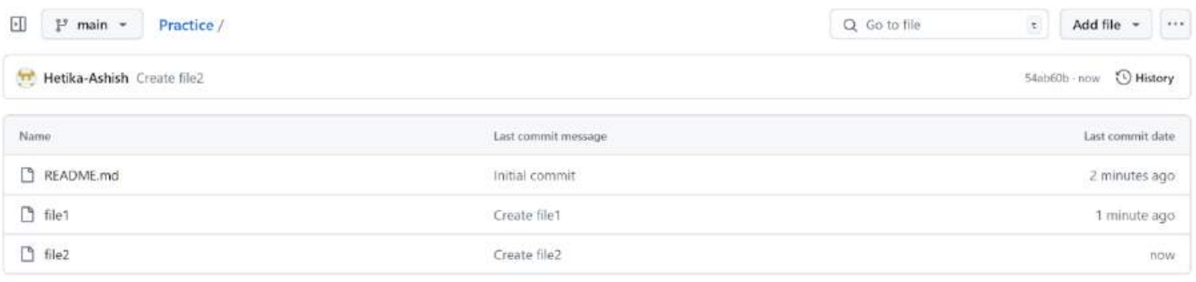
- Hotfix branches are very much like release branches in that they are also meant to prepare for a new production release, albeit unplanned.
- They arise from the necessity to act immediately upon an undesired state of a live production version.

- When a critical bug in a production version must be resolved immediately, a hotfix branch may be branched off from the corresponding tag on the master branch that marks the production version.

The essence is that work of team members (on the develop branch) can continue, while another person is preparing a quick production fix.
****

GITHUB:

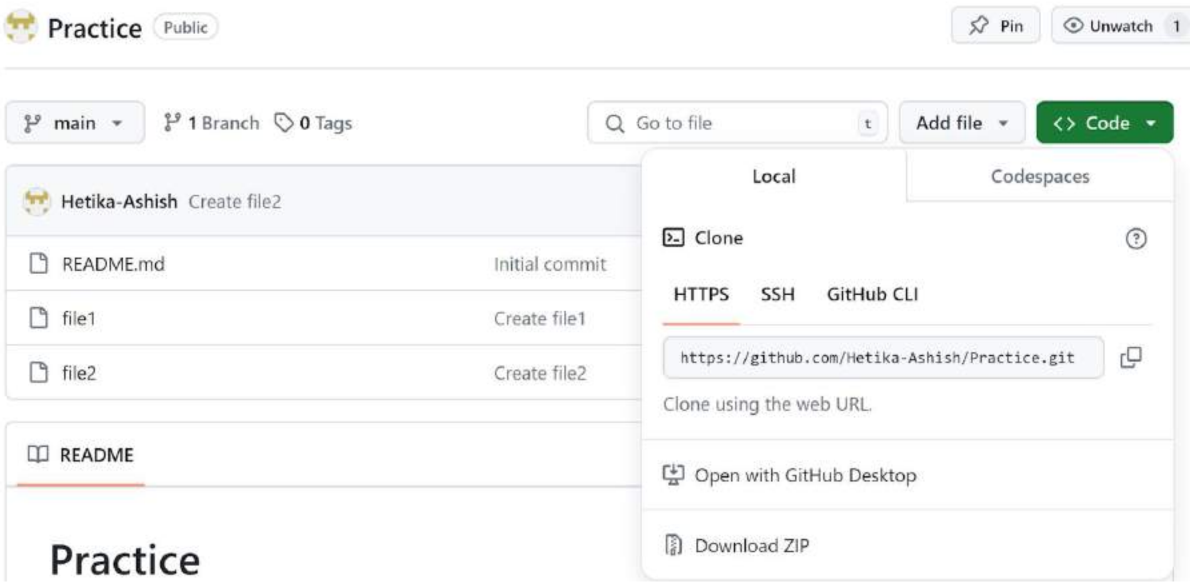
In github, we have created a new repository named practice and created two files namely, file1 and f2 with some content in it and committed it.



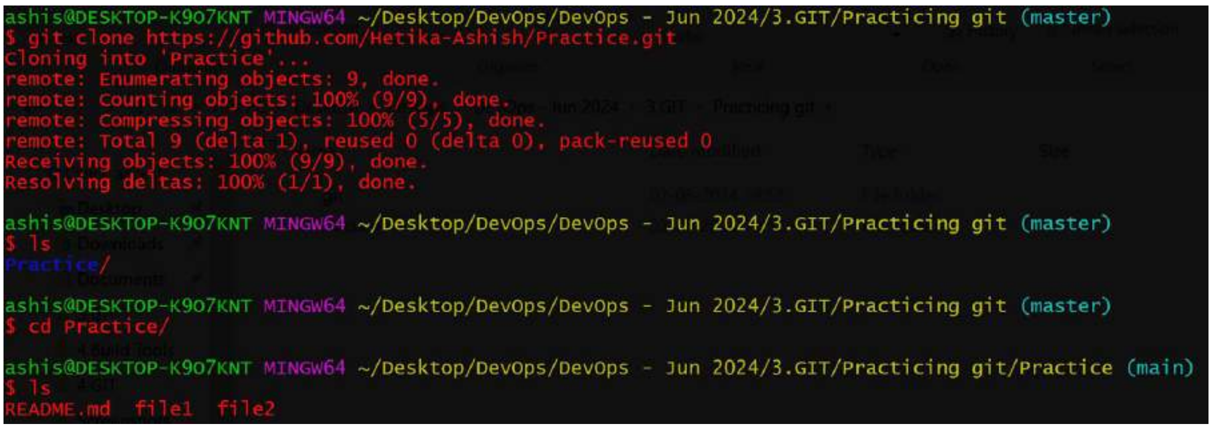
Hetika-Ashish Create file2 54ab60b · now

Name	Last commit message	Last commit date
README.md	Initial commit	2 minutes ago
file1	Create file1	1 minute ago
file2	Create file2	now

Now, we want to clone this repo in our local repo.
We can use git clone <URL> to bring the remote repo to the local repo.

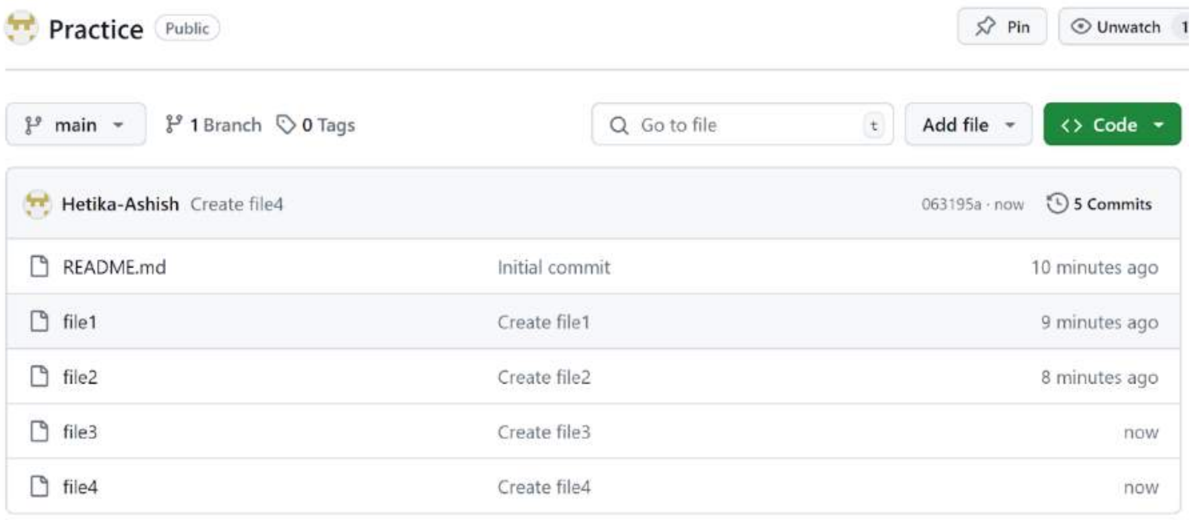


In this case it will be git clone https://github.com/Hetika-Ashish/Practice.git

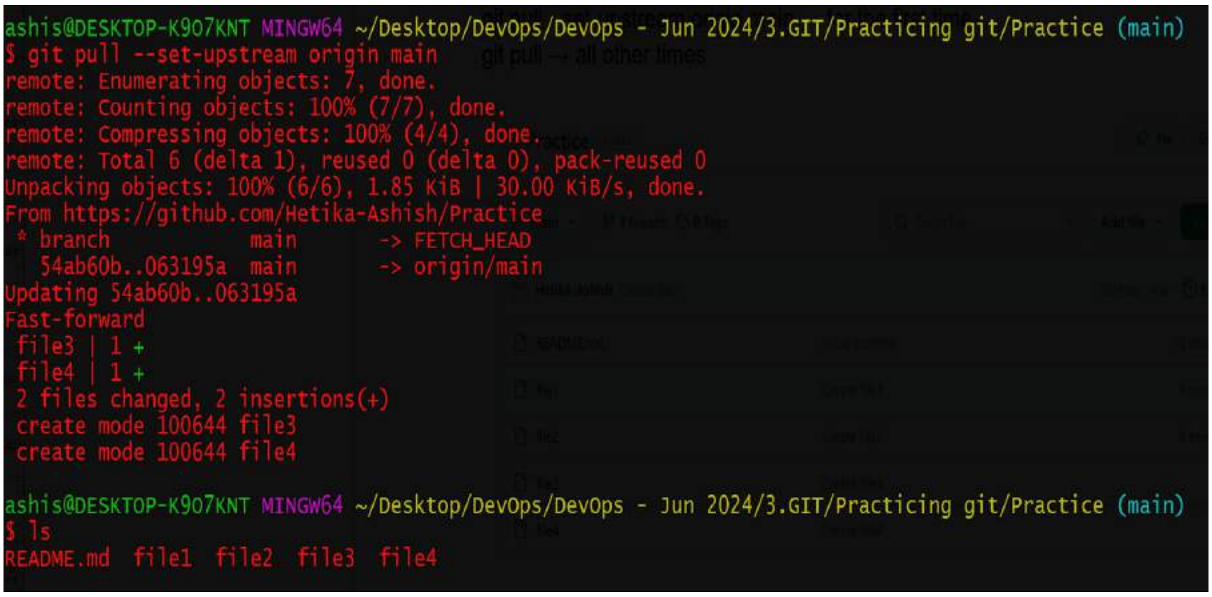


```
ashis@DESKTOP-K9O7KNT MINGW64 ~/Desktop/DevOps/DevOps - Jun 2024/3.GIT/Practicing git (master)
$ git clone https://github.com/Hetika-Ashish/Practice.git
Cloning into 'Practice'...
remote: Enumerating objects: 9, done.
remote: Counting objects: 100% (9/9), done.
remote: Compressing objects: 100% (5/5), 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.

ashis@DESKTOP-K9O7KNT MINGW64 ~/Desktop/DevOps/DevOps - Jun 2024/3.GIT/Practicing git (master)
$ ls
Practice/

ashis@DESKTOP-K9O7KNT MINGW64 ~/Desktop/DevOps/DevOps - Jun 2024/3.GIT/Practicing git (master)
$ cd Practice/

ashis@DESKTOP-K9O7KNT MINGW64 ~/Desktop/DevOps/DevOps - Jun 2024/3.GIT/Practicing git/Practice (main)
$ ls
README.md  file1  file2
```

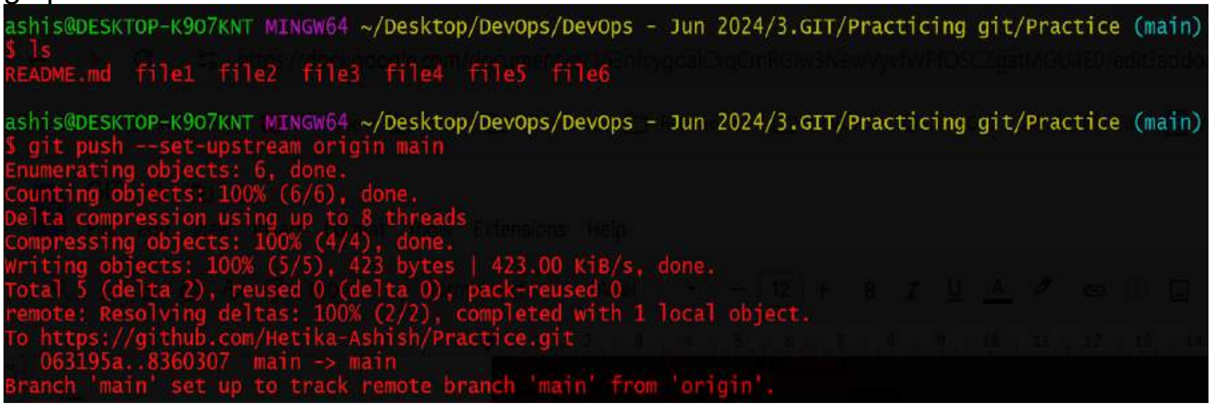
After cloning, I created two new files, file3 and file4 in github.
To bring those files to local we have to use
git pull --set-upstream origin main → for the first time
git pull → all other times



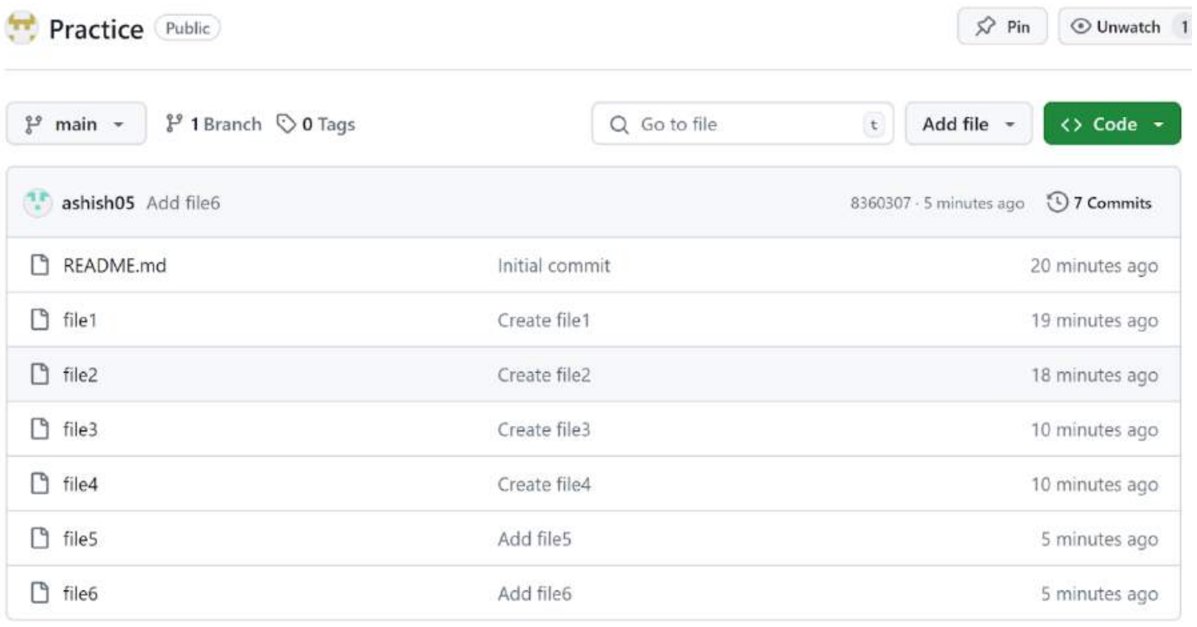
README.md	Initial commit	10 minutes ago
file1	Create file1	9 minutes ago
file2	Create file2	8 minutes ago
file3	Create file3	now
file4	Create file4	now


```
ashis@DESKTOP-K907KNT MINGW64 ~/Desktop/DevOps/DevOps - Jun 2024/3.GIT/Practicing git/Practice (main)
$ git pull --set-upstream origin main
remote: Enumerating objects: 7, done.
remote: Counting objects: 100% (7/7), done.
remote: Compressing objects: 100% (4/4), done.
remote: Total 6 (delta 1), reused 0 (delta 0), pack-reused 0
Unpacking objects: 100% (6/6), 1.85 KiB | 30.00 KiB/s, done.
From https://github.com/Hetika-Ashish/Practice
 * branch            main       -> FETCH_HEAD
   54ab60b..063195a  main       -> origin/main
Updating 54ab60b..063195a
Fast-forward
 file3 | 1 +
 file4 | 1 +
 2 files changed, 2 insertions(+)
 create mode 100644 file3
 create mode 100644 file4

ashis@DESKTOP-K907KNT MINGW64 ~/Desktop/DevOps/DevOps - Jun 2024/3.GIT/Practicing git/Practice (main)
$ ls
README.md  file1  file2  file3  file4
```

Now, I have created two files, file5 and file6 in my local and want to push them to my remote repo.
To do so use command:
git push --set-upstream origin main →for the first time
git push → all other times


```
ashis@DESKTOP-K907KNT MINGW64 ~/Desktop/DevOps/DevOps - Jun 2024/3.GIT/Practicing git/Practice (main)
$ ls
README.md  file1  file2  file3  file4  file5  file6

ashis@DESKTOP-K907KNT MINGW64 ~/Desktop/DevOps/DevOps - Jun 2024/3.GIT/Practicing git/Practice (main)
$ git push --set-upstream origin main
Enumerating objects: 6, done.
Counting objects: 100% (6/6), done.
Delta compression using up to 8 threads
Compressing objects: 100% (4/4), done.
Writing objects: 100% (5/5), 423 bytes | 423.00 KiB/s, done.
Total 5 (delta 2), reused 0 (delta 0), pack-reused 0
remote: Resolving deltas: 100% (2/2), completed with 1 local object.
To https://github.com/Hetika-Ashish/Practice.git
   063195a..8360307  main -> main
Branch 'main' set up to track remote branch 'main' from 'origin'.
```

On refreshing the github, we can find the files that we have pushed.



Practice Public

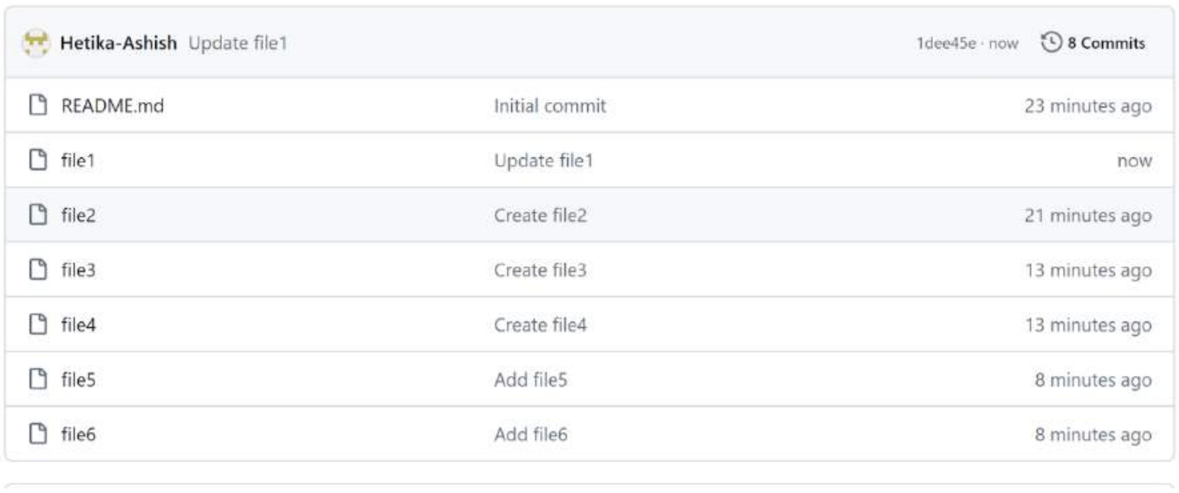
Pin | Unwatch 1

main | 1 Branch | 0 Tags | Go to file | Add file | Code

ashish05 Add file6 — 8360307 · 5 minutes ago — 7 Commits

README.md	Initial commit	20 minutes ago
file1	Create file1	19 minutes ago
file2	Create file2	18 minutes ago
file3	Create file3	10 minutes ago
file4	Create file4	10 minutes ago
file5	Add file5	5 minutes ago
file6	Add file6	5 minutes ago

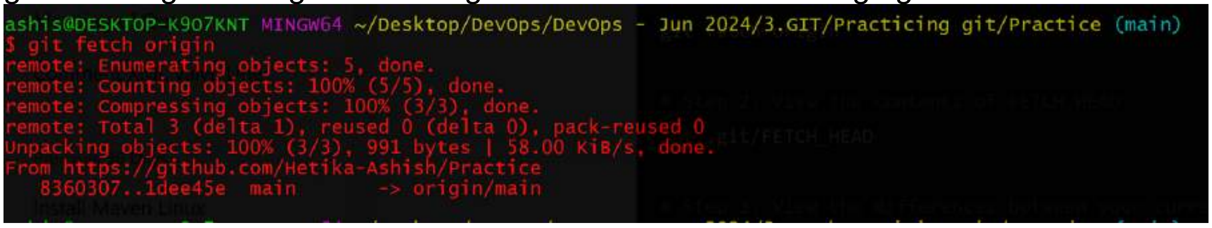
Now, I have made some changes in the file1 in the remote repo and committed it.



Hetika-Ashish Update file1 — 1dee45e · now — 8 Commits

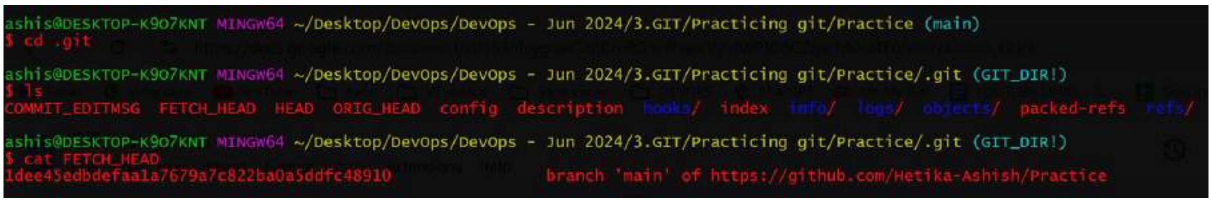
README.md	Initial commit	23 minutes ago
file1	Update file1	now
file2	Create file2	21 minutes ago
file3	Create file3	13 minutes ago
file4	Create file4	13 minutes ago
file5	Add file5	8 minutes ago
file6	Add file6	8 minutes ago

Use command,
git fetch origin → to get the changes in the local without merging



```
ashis@DESKTOP-K9O7KNT MINGW64 ~/Desktop/DevOps/DevOps - Jun 2024/3.GIT/Practicing git/Practice (main)
$ git fetch origin
remote: Enumerating objects: 5, done.
remote: Counting objects: 100% (5/5), done.
remote: Compressing objects: 100% (3/3), done.
remote: Total 3 (delta 1), reused 0 (delta 0), pack-reused 0
Unpacking objects: 100% (3/3), 991 bytes | 58.00 KiB/s, done.
From https://github.com/Hetika-Ashish/Practice
   8360307..1dee45e  main       -> origin/main
```

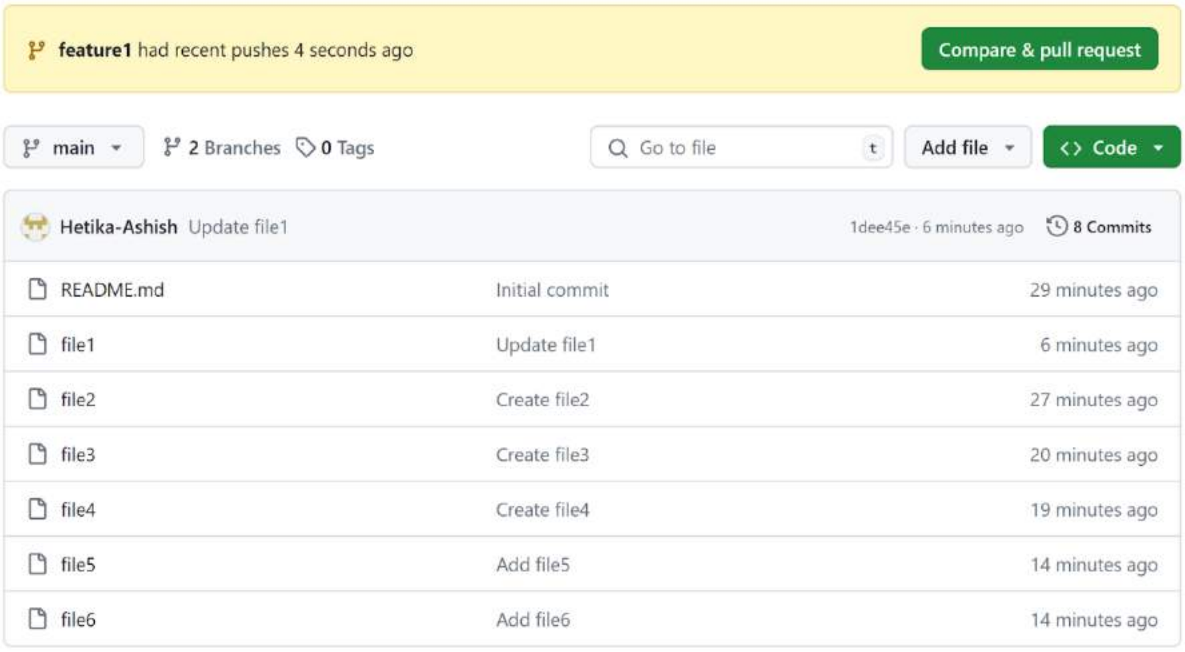
The changes will be reflecting the separate branch called FETCH_HEAD. (present in the .git folder)



```
ashis@DESKTOP-K9O7KNT MINGW64 ~/Desktop/DevOps/DevOps - Jun 2024/3.GIT/Practicing git/Practice (main)
$ cd .git

ashis@DESKTOP-K9O7KNT MINGW64 ~/Desktop/DevOps/DevOps - Jun 2024/3.GIT/Practicing git/Practice/.git (GIT_DIR!)
$ ls
COMMIT_EDITMSG  FETCH_HEAD  HEAD  ORIG_HEAD  config  description  hooks/  index  info/  logs/  objects/  packed-refs  refs/

ashis@DESKTOP-K9O7KNT MINGW64 ~/Desktop/DevOps/DevOps - Jun 2024/3.GIT/Practicing git/Practice/.git (GIT_DIR!)
$ cat FETCH_HEAD
1dee45edbdefaa1a7679a7c822ba0a5ddfc48910            branch 'main' of https://github.com/Hetika-Ashish/Practice
```

- NOTE: git pull = git fetch + merge

Now, I created a new branch feature1 in the github and edited the file1. On the top we can see a message asking us to compare and pull requests.

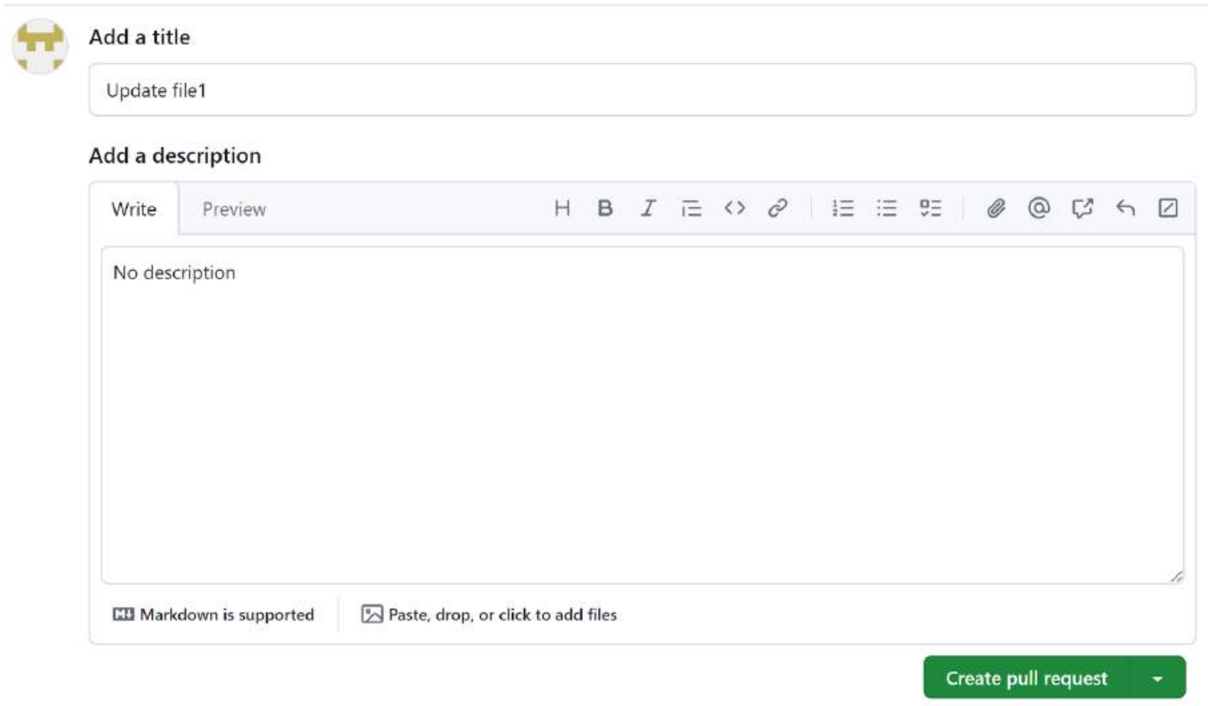


feature1 had recent pushes 4 seconds ago

Hetika-Ashish Update file1 1dee45e · 6 minutes ago 8 Commits

README.md	Initial commit	29 minutes ago
file1	Update file1	6 minutes ago
file2	Create file2	27 minutes ago
file3	Create file3	20 minutes ago
file4	Create file4	19 minutes ago
file5	Add file5	14 minutes ago
file6	Add file6	14 minutes ago

On clicking it


Add a title

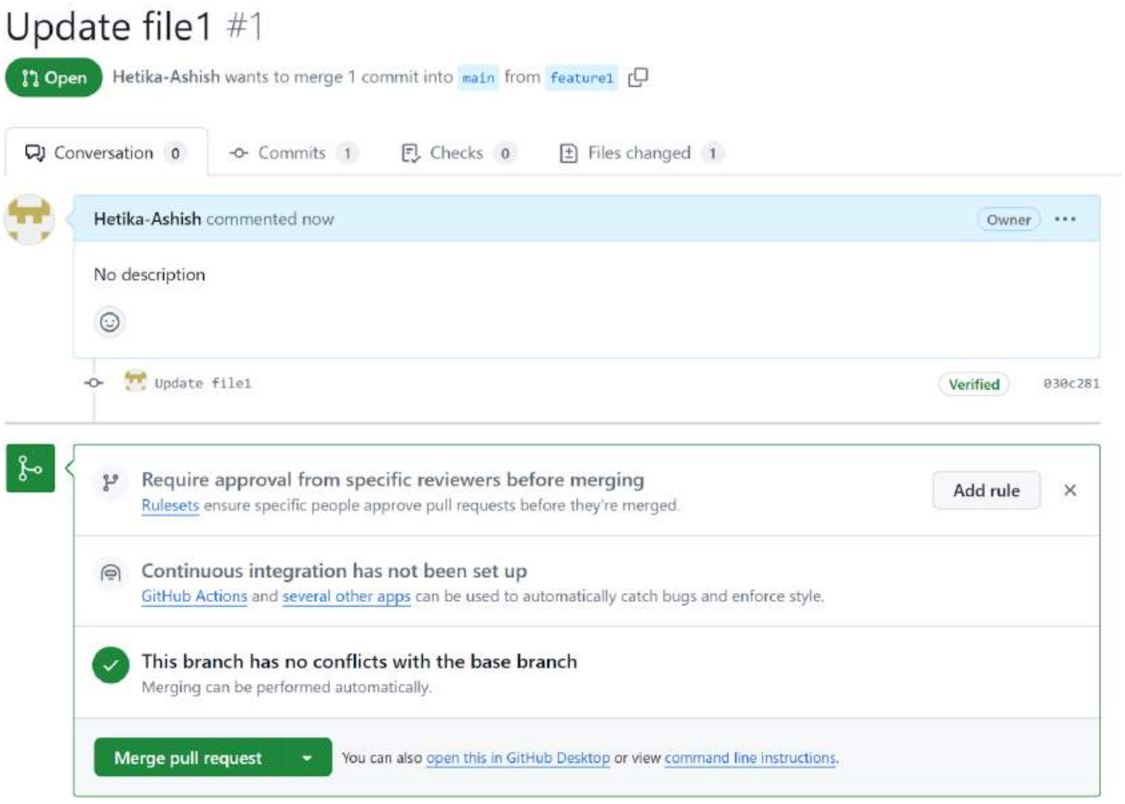
Update file1

Add a description

No description

Now, give the description that you want and click on create pull request.

On clicking it,


Update file1 #1

Hetika-Ashish wants to merge 1 commit into main from feature1

Hetika-Ashish commented now

No description

Update file1 Verified 030c281

Require approval from specific reviewers before merging

Rulesets ensure specific people approve pull requests before they're merged.

Continuous integration has not been set up

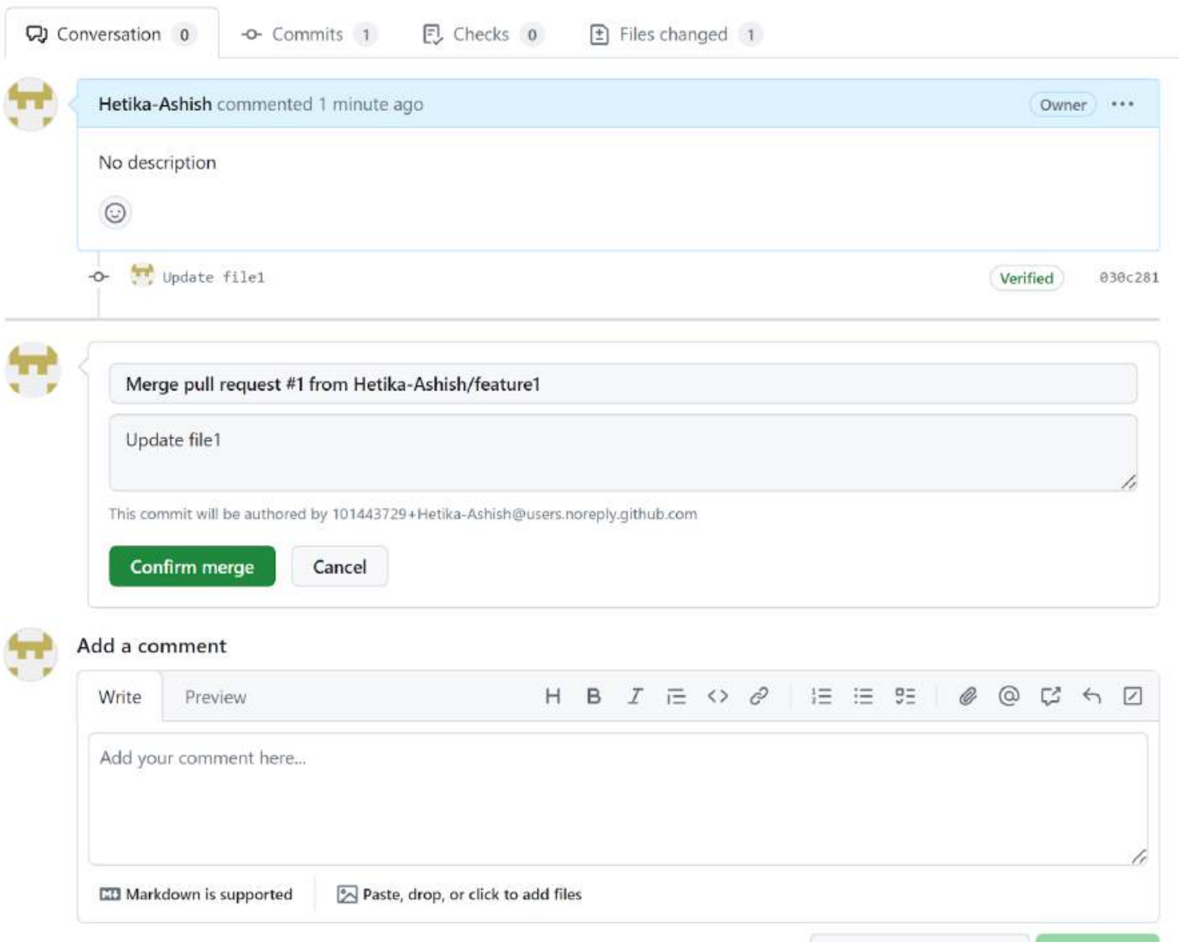
GitHub Actions and several other apps can be used to automatically catch bugs and enforce style.

This branch has no conflicts with the base branch

Merging can be performed automatically.

You can also open this in GitHub Desktop or view command line instructions.

Now, click on merge pull request


Conversation 0 · Commits 1 · Checks 0 · Files changed 1

Hetika-Ashish commented 1 minute ago · Owner

No description

Update file1 · Verified · 030c281

Merge pull request #1 from Hetika-Ashish/feature1

Update file1

This commit will be authored by 101443729+Hetika-Ashish@users.noreply.github.com

Confirm merge · Cancel

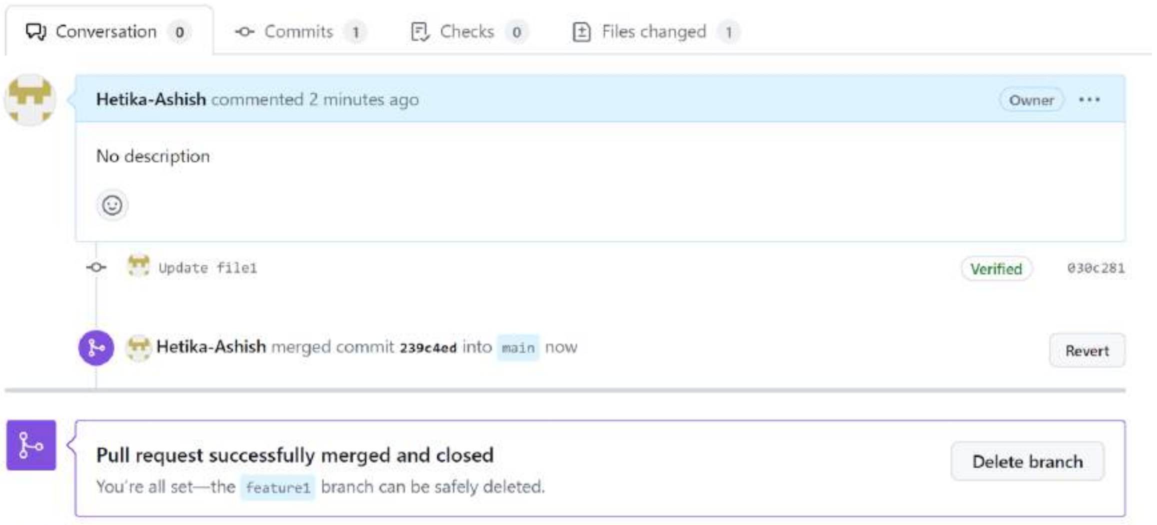
Add a comment

Write · Preview

Add your comment here...

Markdown is supported · Paste, drop, or click to add files

Next, click in confirm merge.


Conversation 0 · Commits 1 · Checks 0 · Files changed 1

Hetika-Ashish commented 2 minutes ago · Owner

No description

Update file1 · Verified · 030c281

Hetika-Ashish merged commit 239c4ed into main now · Revert

Pull request successfully merged and closed

You're all set—the feature1 branch can be safely deleted.

Delete branch

Pull request is successfully merged and closed.