GIT

1. What is Git?

Git is a version control system that tracks every change made to your files, allowing you to roll back to a previous version if you make a mistake. Git is currently the most popular version control system in the world.

There are two types of source code management:

- Centralised version control system
- Distributed version control system

1. Centralized Version Control (CVCS)

In a centralized system, there is a single central server that contains the complete project history.

- How it works: Developers "check out" a single version of the files from the server, make changes, and then "commit" them back to the server.
- Network Dependency: You must be connected to the internet/network to do almost anything (view history, commit changes, etc.).
- Risk: The central server is a single point of failure. If the server crashes and you don't have a backup, the entire project history could be lost.

2. Distributed Version Control (DVCS) — Git

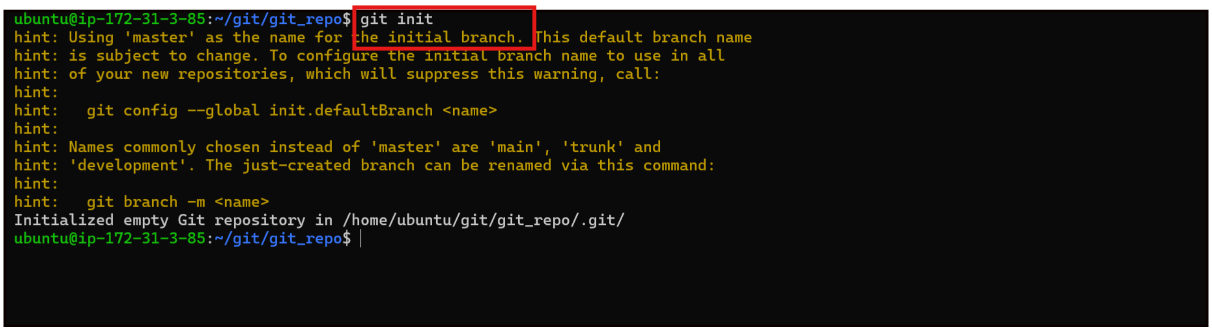
In a distributed system like Git, every developer has a complete copy of the entire project history on their own computer.

- How it works: When you "clone" a repository, you aren't just getting the latest version of the code; you are getting the entire history. You commit changes to your local repository first.
- Network Independence: You can work, commit, and view history offline. You only need the internet when you want to "push" your changes to others or "pull" their updates.
- Security: Every developer’s computer acts as a full backup. If the main server (like GitHub) goes down, you can restore it using any developer's local copy.

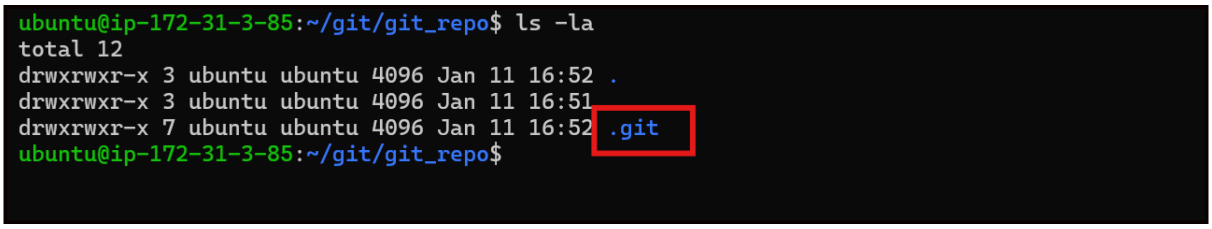
2. Commands:

1. Configuration & Initialization:

- git init: Transforms the current directory into a Git repository. It creates a hidden .git folder where all metadata and version history are stored.

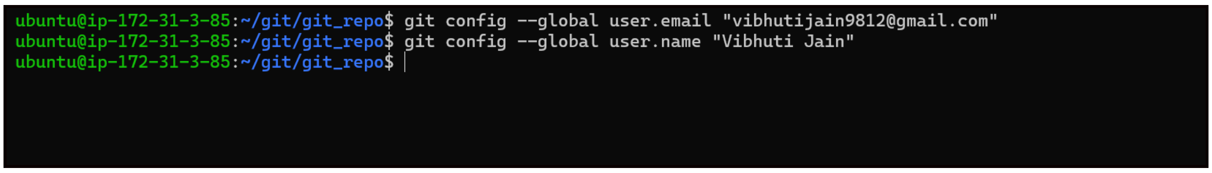


```
ubuntu@ip-172-31-3-85:~/git/git_repo$ git init
hint: Using 'master' as the name for the initial branch. This default branch name
hint: is subject to change. To configure the initial branch name to use in all
hint: of your new repositories, which will suppress this warning, call:
hint:
hint:   git config --global init.defaultBranch <name>
hint:
hint: Names commonly chosen instead of 'master' are 'main', 'trunk' and
hint: 'development'. The just-created branch can be renamed via this command:
hint:
hint:   git branch -m <name>
Initialized empty Git repository in /home/ubuntu/git/git_repo/.git/
ubuntu@ip-172-31-3-85:~/git/git_repo$
```



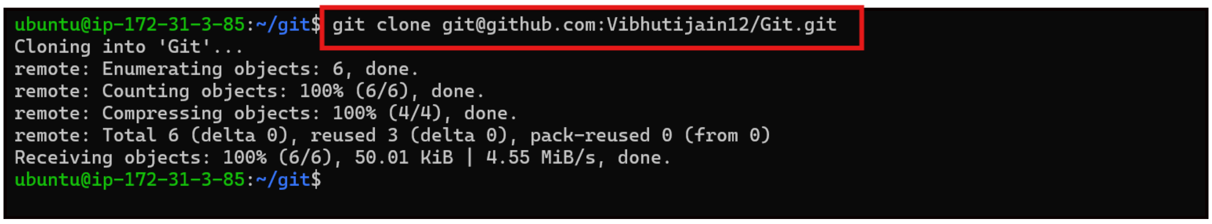
```
ubuntu@ip-172-31-3-85:~/git/git_repo$ ls -la
total 12
drwxrwxr-x 3 ubuntu ubuntu 4096 Jan 11 16:52 .
drwxrwxr-x 3 ubuntu ubuntu 4096 Jan 11 16:51 ..
drwxrwxr-x 7 ubuntu ubuntu 4096 Jan 11 16:52 .git
ubuntu@ip-172-31-3-85:~/git/git_repo$
```

- git config --global user.name "Name" & git config --global user.email "email" - Sets your identity. Every Git commit uses this information to identify you as the author. Using the --global flag ensures these settings apply to all repositories on your system.



```
ubuntu@ip-172-31-3-85:~/git/git_repo$ git config --global user.email "vibhutijain9812@gmail.com"
ubuntu@ip-172-31-3-85:~/git/git_repo$ git config --global user.name "Vibhuti Jain"
ubuntu@ip-172-31-3-85:~/git/git_repo$
```

- git clone <url> - Downloads an existing repository from a remote source (like GitHub) to your local machine. It automatically sets up the remote connection.

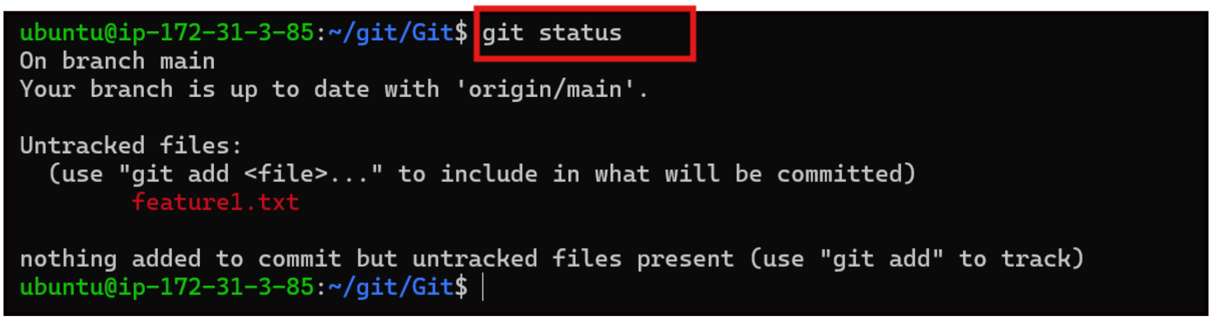


```
ubuntu@ip-172-31-3-85:~/git$ git clone git@github.com:Vibhutijain12/Git.git
Cloning into 'Git'...
remote: Enumerating objects: 6, done.
remote: Counting objects: 100% (6/6), done.
remote: Compressing objects: 100% (4/4), done.
remote: Total 6 (delta 0), reused 3 (delta 0), pack-reused 0 (from 0)
Receiving objects: 100% (6/6), 50.01 KiB | 4.55 MiB/s, done.
ubuntu@ip-172-31-3-85:~/git$
```

2. Tracking Changes (The Core Cycle)

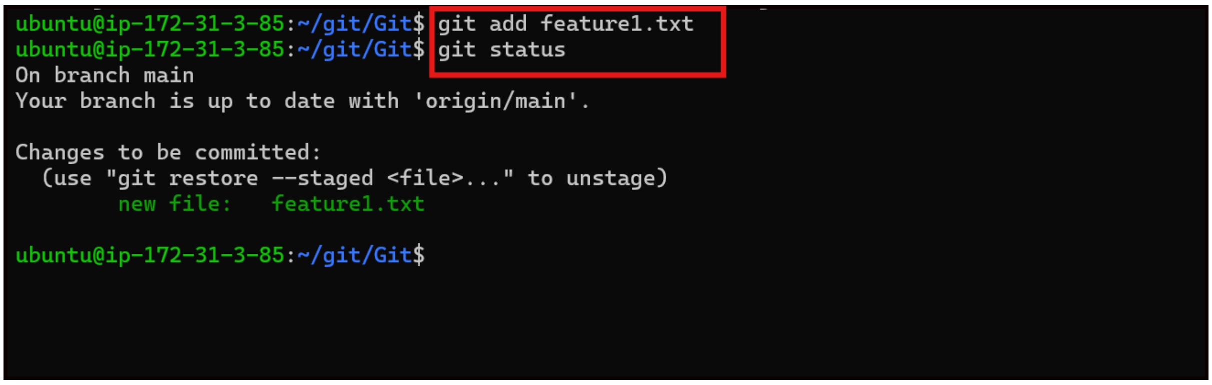
This is the "daily bread" of working with Git.

- git status - Your "health check" command. It tells you which files are untracked, modified, or staged for the next commit.



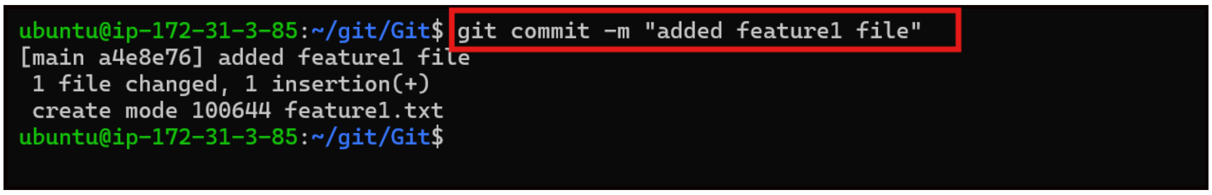
```
ubuntu@ip-172-31-3-85:~/git/Git$ git status
On branch main
Your branch is up to date with 'origin/main'.

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        feature1.txt

nothing added to commit but untracked files present (use "git add" to track)
ubuntu@ip-172-31-3-85:~/git/Git$
```

- git add <file> - Moves changes from the Working Directory to the Staging Area (Index). This tells Git, "I want to include these specific changes in my next snapshot."



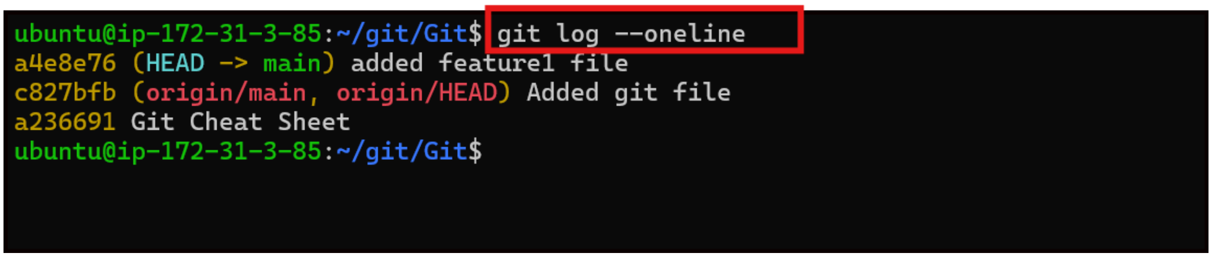
```
ubuntu@ip-172-31-3-85:~/git/Git$ git add feature1.txt
ubuntu@ip-172-31-3-85:~/git/Git$ git status
On branch main
Your branch is up to date with 'origin/main'.

Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        new file:   feature1.txt

ubuntu@ip-172-31-3-85:~/git/Git$
```

- git commit -m "message" - Takes everything in the Staging Area and wraps it into a permanent snapshot (version) in the Local Repository.



```
ubuntu@ip-172-31-3-85:~/git/Git$ git commit -m "added feature1 file"
[main a4e8e76] added feature1 file
 1 file changed, 1 insertion(+)
 create mode 100644 feature1.txt
ubuntu@ip-172-31-3-85:~/git/Git$
```

- git log --oneline - Displays a condensed history of your commits. Each line shows the commit SHA (ID) and the commit message.

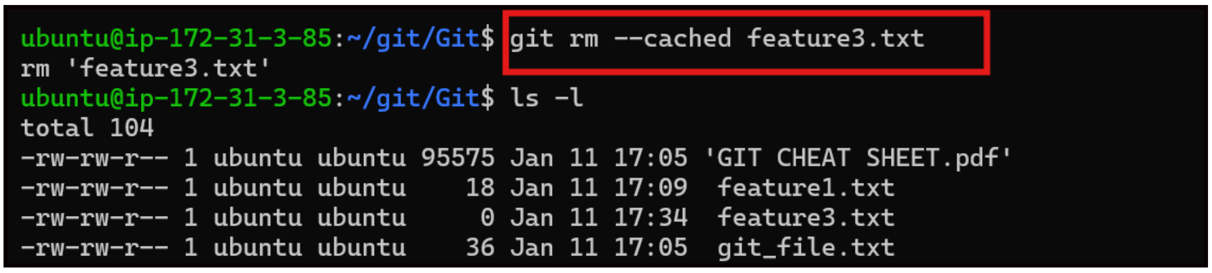


```
ubuntu@ip-172-31-3-85:~/git/Git$ git log --oneline
a4e8e76 (HEAD -> main) added feature1 file
c827bfb (origin/main, origin/HEAD) Added git file
a236691 Git Cheat Sheet
ubuntu@ip-172-31-3-85:~/git/Git$
```

3. Undoing & Moving Files

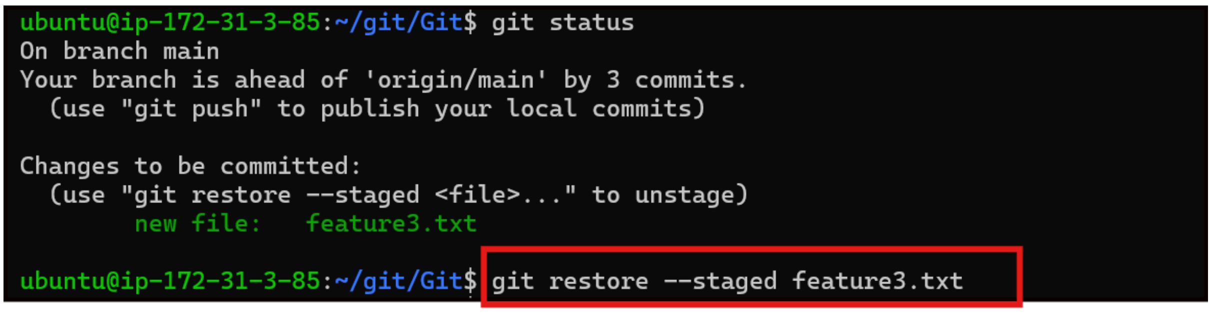
Sometimes you need to correct a mistake or reorganize your project.

- git rm --cached <file> - Removes a file from the Staging Area but keeps it in your local folder. It effectively "untracks" the file.



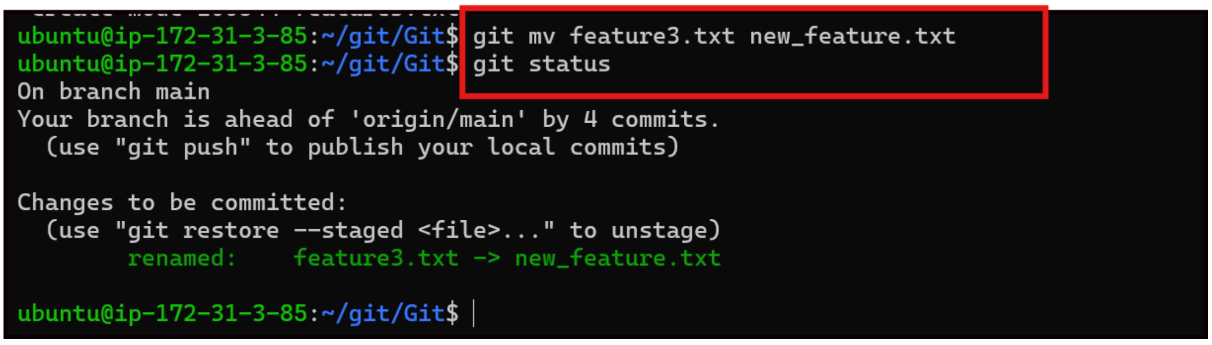
```
ubuntu@ip-172-31-3-85:~/git/Git$ git rm --cached feature3.txt
rm 'feature3.txt'
ubuntu@ip-172-31-3-85:~/git/Git$ ls -l
total 104
-rw-rw-r-- 1 ubuntu ubuntu 95575 Jan 11 17:05 'GIT CHEAT SHEET.pdf'
-rw-rw-r-- 1 ubuntu ubuntu    18 Jan 11 17:09  feature1.txt
-rw-rw-r-- 1 ubuntu ubuntu     0 Jan 11 17:34  feature3.txt
-rw-rw-r-- 1 ubuntu ubuntu    36 Jan 11 17:05  git_file.txt
```

- git restore <file> - Discards local changes in a file and restores it to the state of the last commit. Use this if you made a mistake and want to start over on that file.

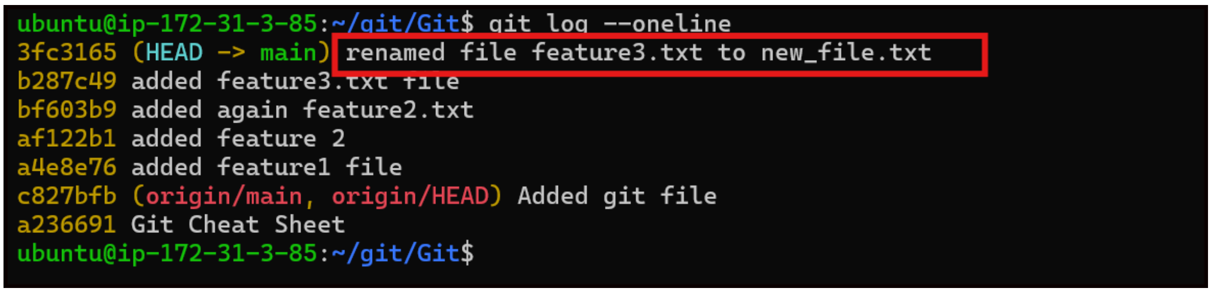


```
ubuntu@ip-172-31-3-85:~/git/Git$ git status
On branch main
Your branch is ahead of 'origin/main' by 3 commits.
  (use "git push" to publish your local commits)

Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        new file:   feature3.txt

ubuntu@ip-172-31-3-85:~/git/Git$ git restore --staged feature3.txt
```

git mv <old_name> <new_name> - Renames or moves a file. Git tracks this as a deletion of the old name and an addition of the new name.



```
ubuntu@ip-172-31-3-85:~/git/Git$ git mv feature3.txt new_feature.txt
ubuntu@ip-172-31-3-85:~/git/Git$ git status
On branch main
Your branch is ahead of 'origin/main' by 4 commits.
  (use "git push" to publish your local commits)

Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        renamed:    feature3.txt -> new_feature.txt

ubuntu@ip-172-31-3-85:~/git/Git$ |
```

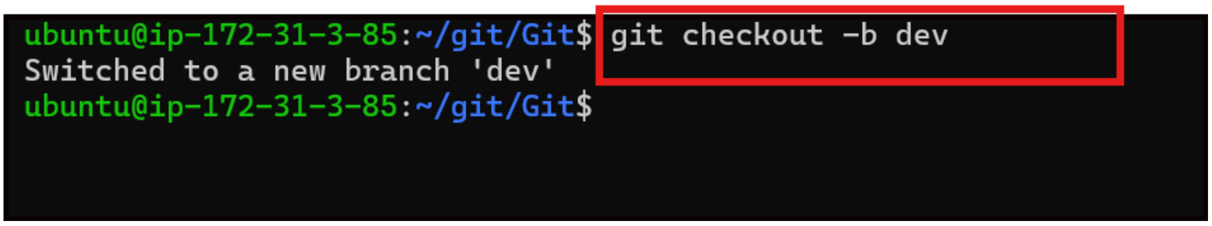


```
ubuntu@ip-172-31-3-85:~/git/Git$ git log --oneline
3fc3165 (HEAD -> main) renamed file feature3.txt to new_file.txt
b287c49 added feature3.txt file
bf603b9 added again feature2.txt
af122b1 added feature 2
a4e8e76 added feature1 file
c827bfb (origin/main, origin/HEAD) Added git file
a236691 Git Cheat Sheet
ubuntu@ip-172-31-3-85:~/git/Git$
```

4. Branching & Switching:

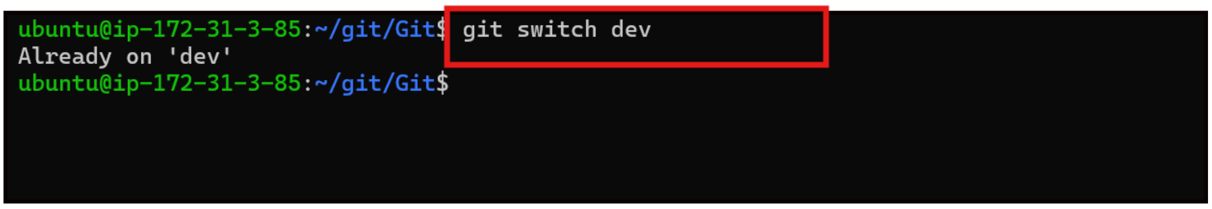
Branches allow you to work on new features without breaking the "main" code.

- git checkout -b <branch_name> - A shortcut command that creates a new branch and immediately switches you to it.



```
ubuntu@ip-172-31-3-85:~/git/Git$ git checkout -b dev
Switched to a new branch 'dev'
ubuntu@ip-172-31-3-85:~/git/Git$
```

- git switch <branch_name> (or git checkout) - Moves you from your current branch to an existing one. git switch is the modern, more intuitive command for this specific task.

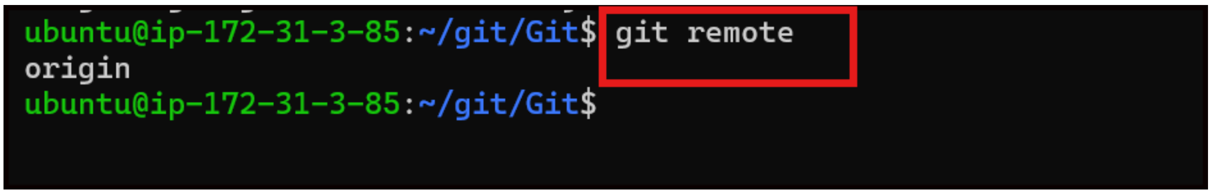


```
ubuntu@ip-172-31-3-85:~/git/Git$ git switch dev
Already on 'dev'
ubuntu@ip-172-31-3-85:~/git/Git$
```

5. Remote Collaboration:

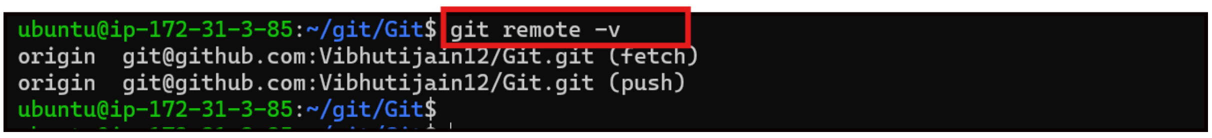
Sharing your work with the world (or your team).

- git remote - The git remote command is used to manage the connections between your local repository and "remotes"—the versions of your project hosted on servers like GitHub, GitLab, or Bitbucket. Quick check of which remotes are connected.



```
ubuntu@ip-172-31-3-85:~/git/Git$ git remote
origin
ubuntu@ip-172-31-3-85:~/git/Git$
```

- git remote -v - Lists remotes with their full URLs.

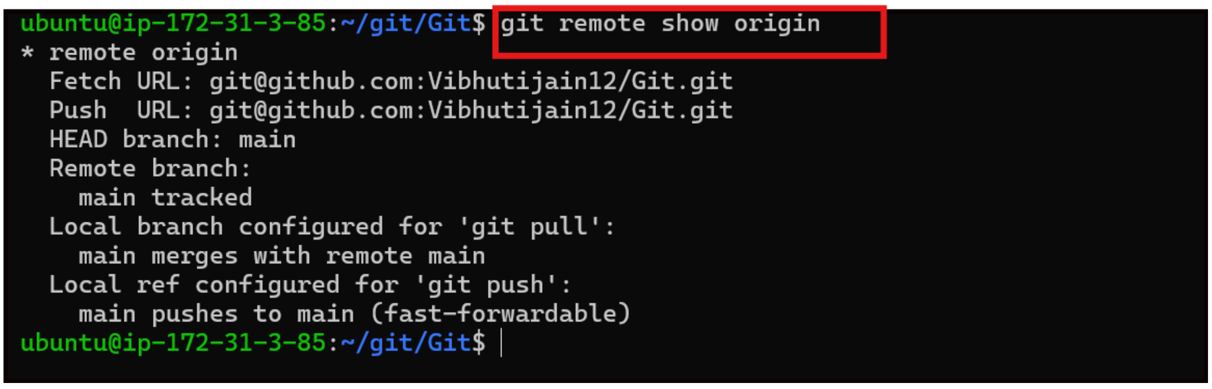


```
ubuntu@ip-172-31-3-85:~/git/Git$ git remote -v
origin  git@github.com:Vibhutijain12/Git.git (fetch)
origin  git@github.com:Vibhutijain12/Git.git (push)
ubuntu@ip-172-31-3-85:~/git/Git$
```

- git remote add <name> <url> - Links a local repo to a remote URL.

Practical: git remote add origin https://github.com/user/repo.git

- git remote show <name> - Shows which branches are tracked and if your local repo is up-to-date.



```
ubuntu@ip-172-31-3-85:~/git/Git$ git remote show origin
* remote origin
  Fetch URL: git@github.com:Vibhutijain12/Git.git
  Push  URL: git@github.com:Vibhutijain12/Git.git
  HEAD branch: main
  Remote branch:
    main tracked
  Local branch configured for 'git pull':
    main merges with remote main
  Local ref configured for 'git push':
    main pushes to main (fast-forwardable)
ubuntu@ip-172-31-3-85:~/git/Git$ 
```

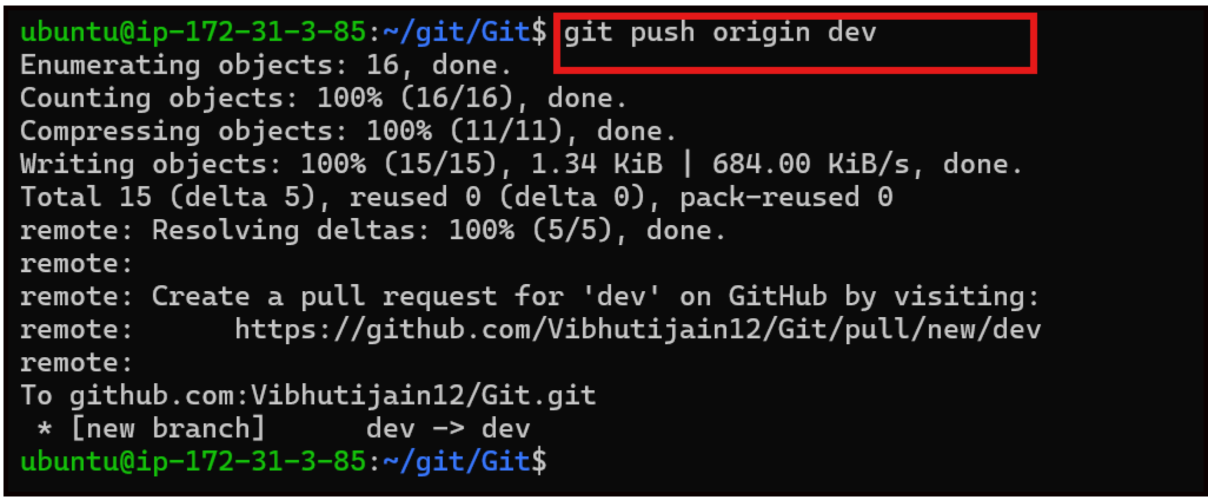
- git remote remove <name> - Unlinks the remote from your local repo.

What is "Origin"?

You will see the word origin everywhere in Git. It is not a special command; it is simply the default name (alias) Git gives to the primary remote server you are communicating with.

When you run' git clone', Git automatically adds a remote named' origin' for you. If you start a repository locally with' git init', you need to add it yourself.

- git push origin <branch> - Uploads your local repository commits to a remote repository (like GitHub). Origin is the default name for your remote server, and main (or master) is the branch name.



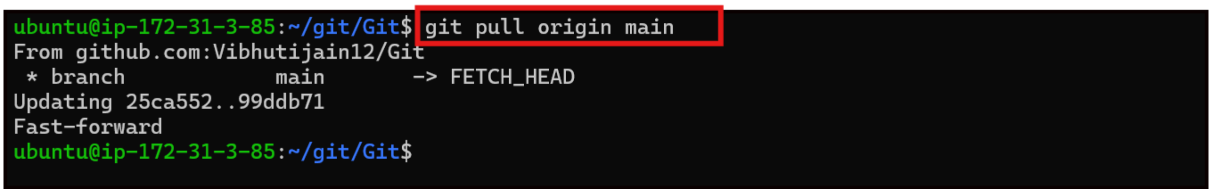
```
ubuntu@ip-172-31-3-85:~/git/Git$ git push origin dev
Enumerating objects: 16, done.
Counting objects: 100% (16/16), done.
Compressing objects: 100% (11/11), done.
Writing objects: 100% (15/15), 1.34 KiB | 684.00 KiB/s, done.
Total 15 (delta 5), reused 0 (delta 0), pack-reused 0
remote: Resolving deltas: 100% (5/5), done.
remote:
remote: Create a pull request for 'dev' on GitHub by visiting:
remote:      https://github.com/Vibhutijain12/Git/pull/new/dev
remote:
To github.com:Vibhutijain12/Git.git
 * [new branch]      dev -> dev
ubuntu@ip-172-31-3-85:~/git/Git$
```

- git fetch - Downloads the latest history/metadata from the remote repository but does not change your local files.

Practical: Use this to see what your teammates have done without automatically merging their code into yours.

- git pull origin <branch> - A combination of git fetch followed by git merge. It downloads changes and immediately tries to integrate them into your local branch.

  Practical: Use this to update your local code with the latest changes from the server.



```
ubuntu@ip-172-31-3-85:~/git/Git$ git pull origin main
From github.com:Vibhutijain12/Git
 * branch            main       -> FETCH_HEAD
Updating 25ca552..99ddb71
Fast-forward
ubuntu@ip-172-31-3-85:~/git/Git$
```

6. What are Untracked, Staged, and Committed in Git?

Untracked (The "Waiting Room")

- The Concept: The file exists on your hard drive, but it's not part of Git's version control history. If you delete it now, Git can't help you get it back.
- How to see it: Run git status. Untracked files usually appear in red.

Staged (The "Photo Lineup")

When you run git add <file>, you move the file into the Staging Area (also called the Index).

- The Concept: You are telling Git, "I want the current version of this file to be included in my next snapshot."
- Key Detail: If you stage a file, then change it again before committing, you have to run git add again to stage the new changes. Otherwise, Git will only commit the version that was staged.
- How to see it: Run git status. Staged files usually appear in green under "Changes to be committed."

Committed (The "Final Photo")

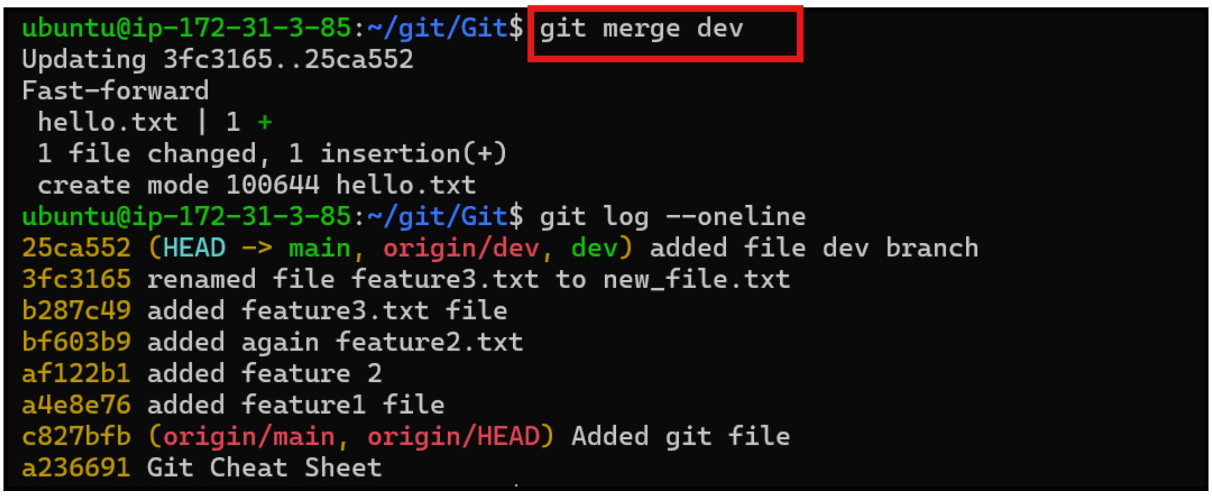
When you run git commit -m "message", Git takes a snapshot of everything in the Staging Area and saves it permanently in the .git directory.

- The Concept: The changes are now a part of the project's history. You can travel back to this exact moment at any time in the future.

Here are the details for these advanced Git commands. These are the tools that help you manage complex workflows, fix errors, and collaborate with others.

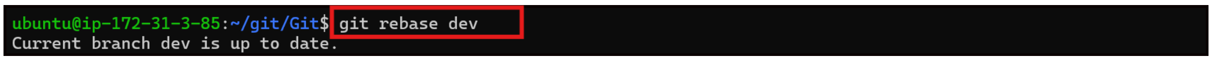
7. Integrating Changes:

- git merge <branch-name> - Combines the work from another branch into your current branch. If you are on main and run git merge feature-login, Git will take all the commits from the login branch and integrate them into main.



```
ubuntu@ip-172-31-3-85:~/git/Git$ git merge dev
Updating 3fc3165..25ca552
Fast-forward
 hello.txt | 1 +
 1 file changed, 1 insertion(+)
 create mode 100644 hello.txt
ubuntu@ip-172-31-3-85:~/git/Git$ git log --oneline
25ca552 (HEAD -> main, origin/dev, dev) added file dev branch
3fc3165 renamed file feature3.txt to new_file.txt
b287c49 added feature3.txt file
bf603b9 added again feature2.txt
af122b1 added feature 2
a4e8e76 added feature1 file
c827bfb (origin/main, origin/HEAD) Added git file
a236691 Git Cheat Sheet
```

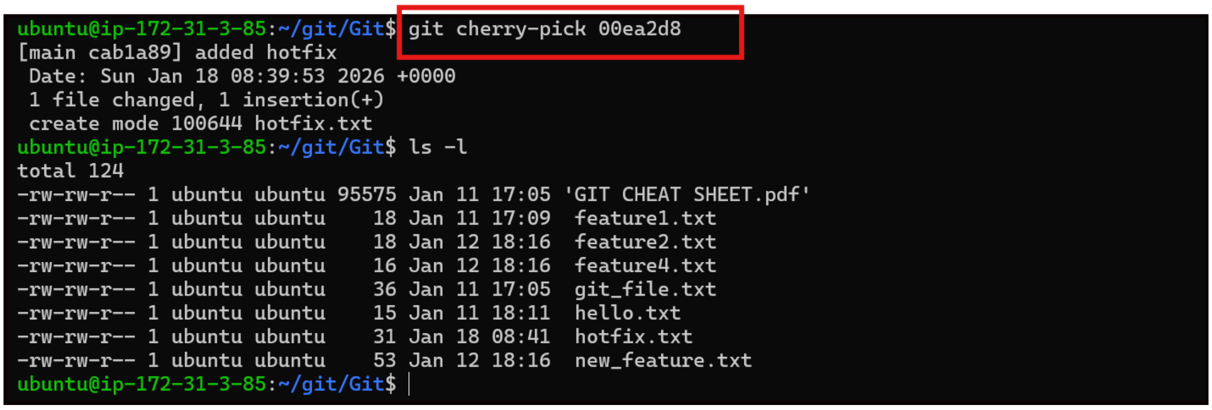
- git rebase <branch name> - Rebase is the process of moving or combining a sequence of commits to a new base commit.



```
ubuntu@ip-172-31-3-85:~/git/Git$ git rebase dev
Current branch dev is up to date.
```

- git cherry-pick - Allows you to pick **one specific commit** from another branch and apply it to your current branch.

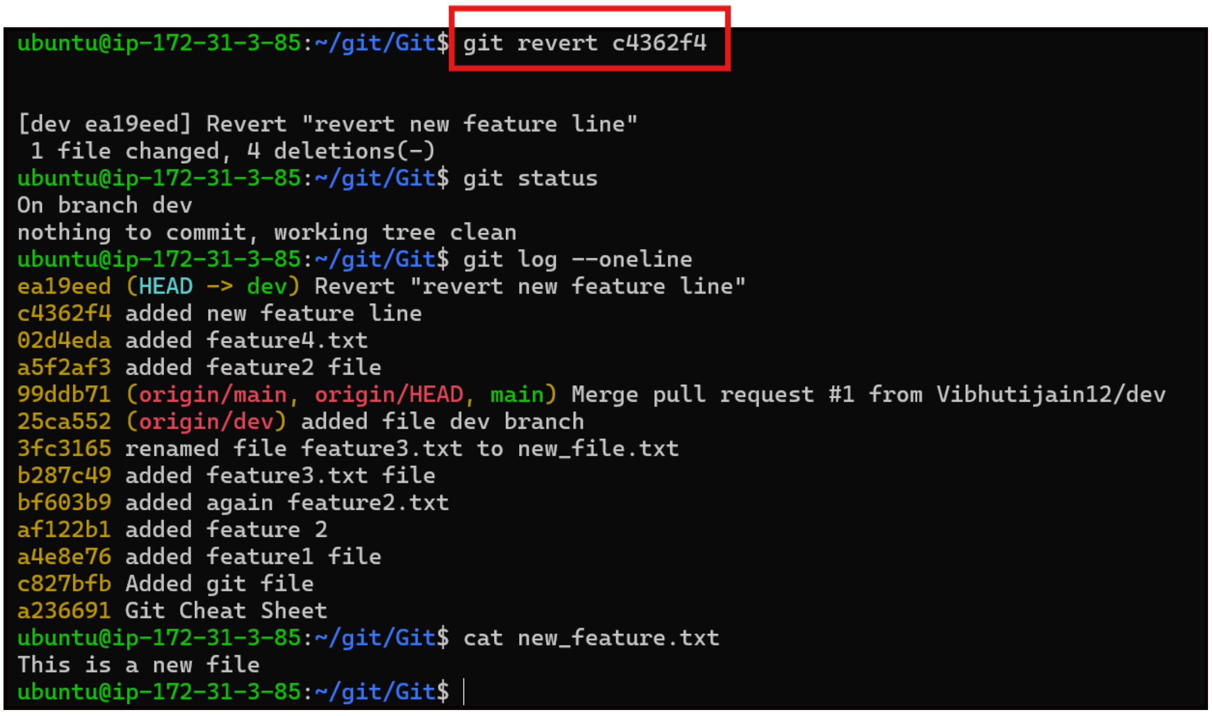
  Practical: Useful when you have a bug fix on a "work-in-progress" branch that you want to move to "production" without taking all the unfinished features.



```
ubuntu@ip-172-31-3-85:~/git/Git$ git cherry-pick 00ea2d8
[main cab1a89] added hotfix
 Date: Sun Jan 18 08:39:53 2026 +0000
 1 file changed, 1 insertion(+)
 create mode 100644 hotfix.txt
ubuntu@ip-172-31-3-85:~/git/Git$ ls -l
total 124
-rw-rw-r-- 1 ubuntu ubuntu 95575 Jan 11 17:05 'GIT CHEAT SHEET.pdf'
-rw-rw-r-- 1 ubuntu ubuntu    18 Jan 11 17:09  feature1.txt
-rw-rw-r-- 1 ubuntu ubuntu    18 Jan 12 18:16  feature2.txt
-rw-rw-r-- 1 ubuntu ubuntu    16 Jan 12 18:16  feature4.txt
-rw-rw-r-- 1 ubuntu ubuntu    36 Jan 11 17:05  git_file.txt
-rw-rw-r-- 1 ubuntu ubuntu    15 Jan 11 18:11  hello.txt
-rw-rw-r-- 1 ubuntu ubuntu    31 Jan 18 08:41  hotfix.txt
-rw-rw-r-- 1 ubuntu ubuntu    53 Jan 12 18:16  new_feature.txt
ubuntu@ip-172-31-3-85:~/git/Git$
```

8. Undoing & Fixing Mistakes:

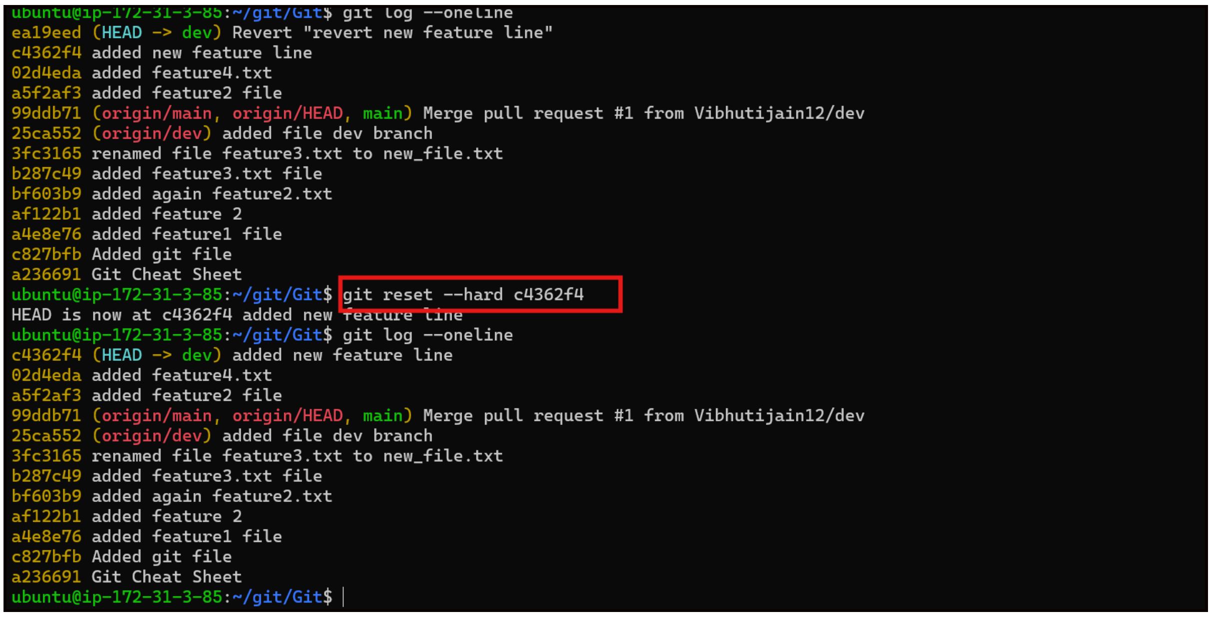
- git revert <commit-hash> - Creates a new commit that does the exact opposite of a previous commit.



```
ubuntu@ip-172-31-3-85:~/git/Git$ git revert c4362f4


[dev ea19eed] Revert "revert new feature line"
 1 file changed, 4 deletions(-)
ubuntu@ip-172-31-3-85:~/git/Git$ git status
On branch dev
nothing to commit, working tree clean
ubuntu@ip-172-31-3-85:~/git/Git$ git log --oneline
ea19eed (HEAD -> dev) Revert "revert new feature line"
c4362f4 added new feature line
02d4eda added feature4.txt
a5f2af3 added feature2 file
99ddb71 (origin/main, origin/HEAD, main) Merge pull request #1 from Vibhutijain12/dev
25ca552 (origin/dev) added file dev branch
3fc3165 renamed file feature3.txt to new_file.txt
b287c49 added feature3.txt file
bf603b9 added again feature2.txt
af122b1 added feature 2
a4e8e76 added feature1 file
c827bfb Added git file
a236691 Git Cheat Sheet
ubuntu@ip-172-31-3-85:~/git/Git$ cat new_feature.txt
This is a new file
ubuntu@ip-172-31-3-85:~/git/Git$ 
```

- git reset <commit-hash> - oves the current branch tip back to a previous commit. It "rewrites" history.

Practical: Use --hard to completely wipe out changes, or --soft to keep your work in the staging area. **Warning:** Avoid using this on shared remote branches as it can confuse your teammates.

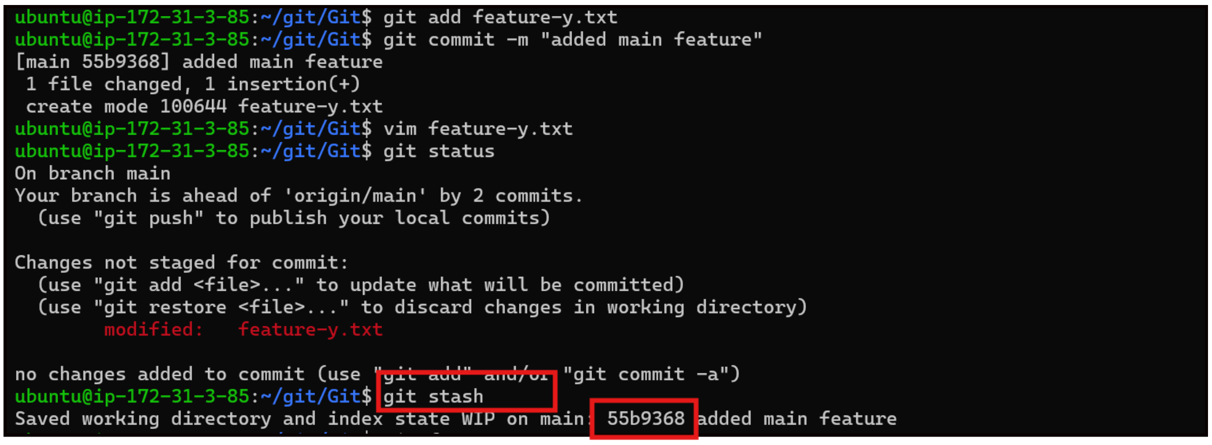


```
ubuntu@ip-172-31-3-85:~/git/Git$ git log --oneline
ea19eed (HEAD -> dev) Revert "revert new feature line"
c4362f4 added new feature line
02d4eda added feature4.txt
a5f2af3 added feature2 file
99ddb71 (origin/main, origin/HEAD, main) Merge pull request #1 from Vibhutijain12/dev
25ca552 (origin/dev) added file dev branch
3fc3165 renamed file feature3.txt to new_file.txt
b287c49 added feature3.txt file
bf603b9 added again feature2.txt
af122b1 added feature 2
a4e8e76 added feature1 file
c827bfb Added git file
a236691 Git Cheat Sheet
ubuntu@ip-172-31-3-85:~/git/Git$ git reset --hard c4362f4
HEAD is now at c4362f4 added new feature line
ubuntu@ip-172-31-3-85:~/git/Git$ git log --oneline
c4362f4 (HEAD -> dev) added new feature line
02d4eda added feature4.txt
a5f2af3 added feature2 file
99ddb71 (origin/main, origin/HEAD, main) Merge pull request #1 from Vibhutijain12/dev
25ca552 (origin/dev) added file dev branch
3fc3165 renamed file feature3.txt to new_file.txt
b287c49 added feature3.txt file
bf603b9 added again feature2.txt
af122b1 added feature 2
a4e8e76 added feature1 file
c827bfb Added git file
a236691 Git Cheat Sheet
ubuntu@ip-172-31-3-85:~/git/Git$ 
```

Git Stash Commands

- git stash - Takes all your modified tracked files and staged changes and saves them on a stack of unfinished changes. It then reverts your working directory to match the HEAD commit (a clean state).

  Practical: Use this when you are 50% done with a feature but need to switch branches immediately to fix a production bug.

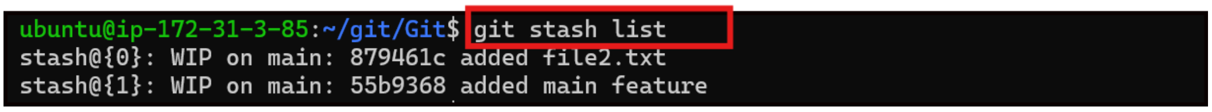


```
ubuntu@ip-172-31-3-85:~/git/Git$ git add feature-y.txt
ubuntu@ip-172-31-3-85:~/git/Git$ git commit -m "added main feature"
[main 55b9368] added main feature
 1 file changed, 1 insertion(+)
 create mode 100644 feature-y.txt
ubuntu@ip-172-31-3-85:~/git/Git$ vim feature-y.txt
ubuntu@ip-172-31-3-85:~/git/Git$ git status
On branch main
Your branch is ahead of 'origin/main' by 2 commits.
  (use "git push" to publish your local commits)

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:   feature-y.txt

no changes added to commit (use "git add" and/or "git commit -a")
ubuntu@ip-172-31-3-85:~/git/Git$ git stash
Saved working directory and index state WIP on main: 55b9368 added main feature
```

- git stash list - Displays all the stashes you have saved, organized as a stack.

  Practical: It shows them in the format stash@{0}: WIP on master....

  Note: stash@{0} is always the most recent one you created. As you add more, the older ones move to higher numbers (1, 2, 3...).

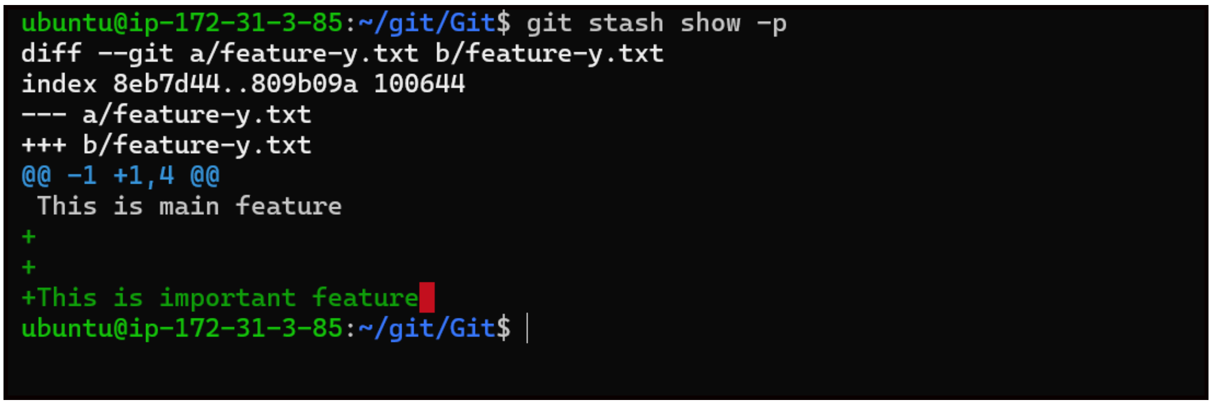


```
ubuntu@ip-172-31-3-85:~/git/Git$ git stash list
stash@{0}: WIP on main: 879461c added file2.txt
stash@{1}: WIP on main: 55b9368 added main feature
```

- git stash show - Gives you a summary of the changes in your most recent stash (or a specific one).

Practical Usage:

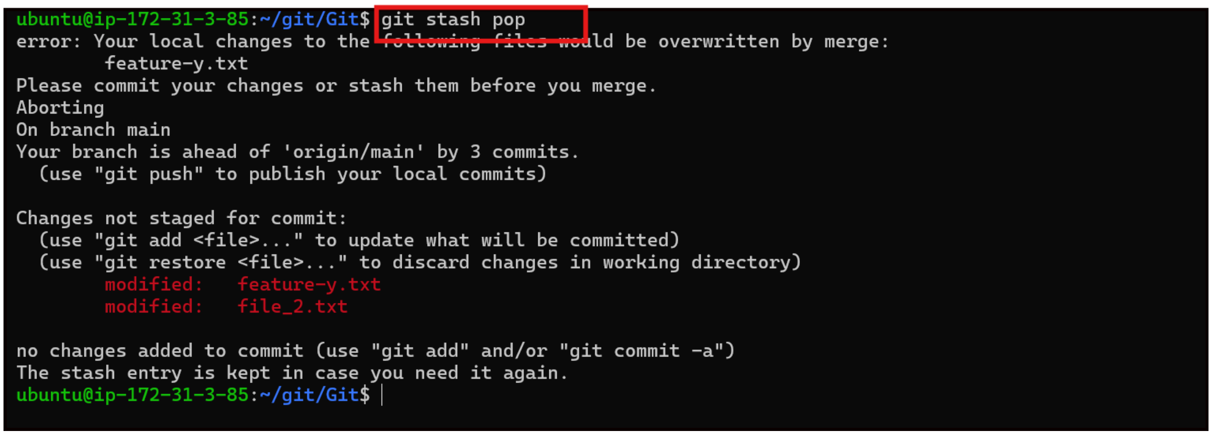
- git stash show: Shows a summary (files changed).
- git stash show -p: Shows the full "patch" (the actual lines of code changed).
- git stash show stash@{1}: Shows changes for a specific item in the list.



```
ubuntu@ip-172-31-3-85:~/git/Git$ git stash show -p
diff --git a/feature-y.txt b/feature-y.txt
index 8eb7d44..809b09a 100644
--- a/feature-y.txt
+++ b/feature-y.txt
@@ -1 +1,4 @@
 This is main feature
+
+
+This is important feature
ubuntu@ip-172-31-3-85:~/git/Git$
```

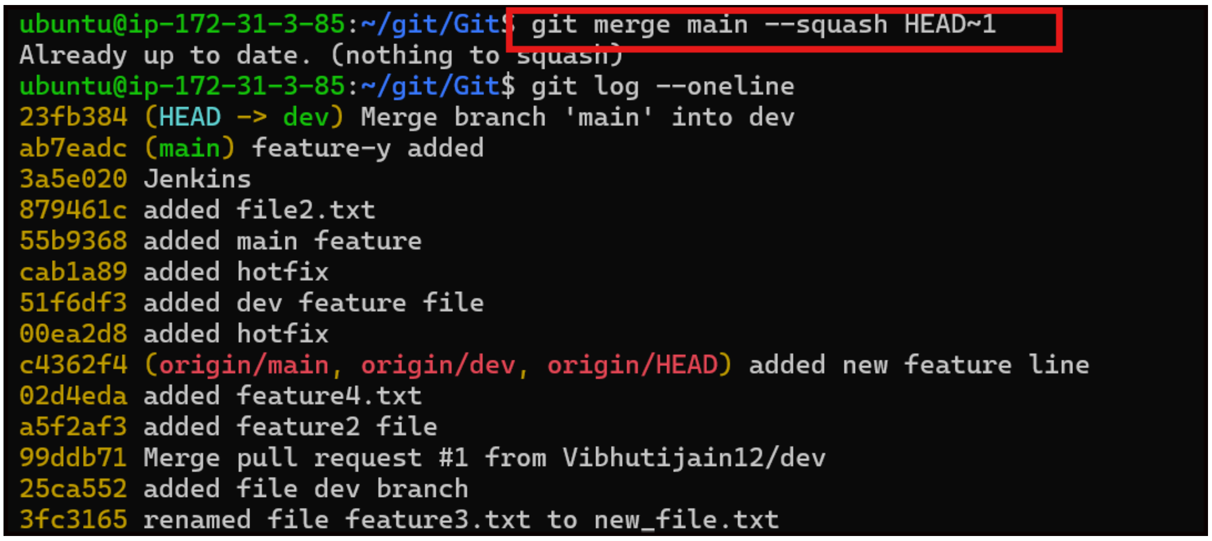
- git stash pop - Removes the most recent stash from the stack and applies it to your current working directory.

  Practical: This is the "Paste and Delete" command. It brings your work back and cleans up the stash list at the same time.

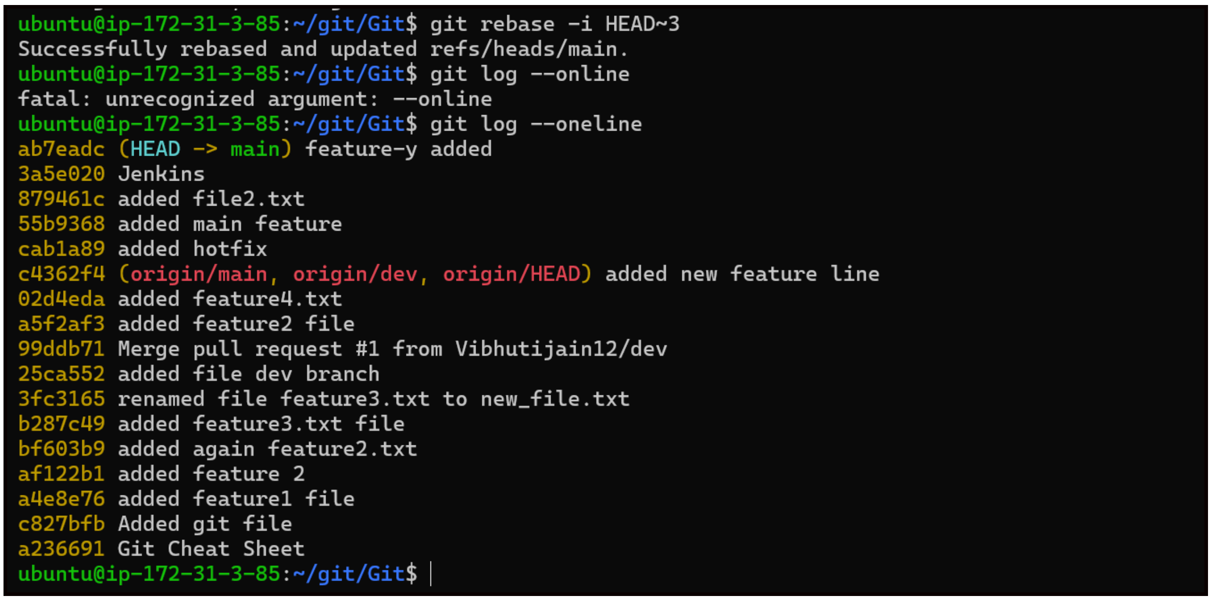


```
ubuntu@ip-172-31-3-85:~/git/Git$ git stash pop
error: Your local changes to the following files would be overwritten by merge:
        feature-y.txt
Please commit your changes or stash them before you merge.
Aborting
On branch main
Your branch is ahead of 'origin/main' by 3 commits.
  (use "git push" to publish your local commits)

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:   feature-y.txt
        modified:   file_2.txt

no changes added to commit (use "git add" and/or "git commit -a")
The stash entry is kept in case you need it again.
ubuntu@ip-172-31-3-85:~/git/Git$
```

- git squash - **"Squashing"** is the process of combining multiple commit snapshots into a single, clean commit. It is a powerful way to clean up your project history before merging a feature into the main codebase.



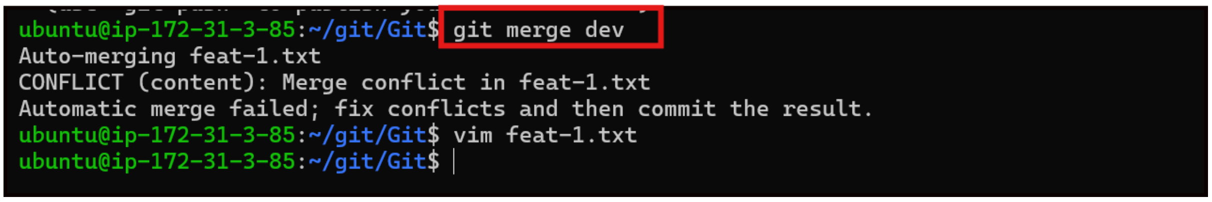
```
ubuntu@ip-172-31-3-85:~/git/Git$ git merge main --squash HEAD~1
Already up to date. (nothing to squash)
ubuntu@ip-172-31-3-85:~/git/Git$ git log --oneline
23fb384 (HEAD -> dev) Merge branch 'main' into dev
ab7eadc (main) feature-y added
3a5e020 Jenkins
879461c added file2.txt
55b9368 added main feature
cab1a89 added hotfix
51f6df3 added dev feature file
00ea2d8 added hotfix
c4362f4 (origin/main, origin/dev, origin/HEAD) added new feature line
02d4eda added feature4.txt
a5f2af3 added feature2 file
99ddb71 Merge pull request #1 from Vibhutijain12/dev
25ca552 added file dev branch
3fc3165 renamed file feature3.txt to new_file.txt
```



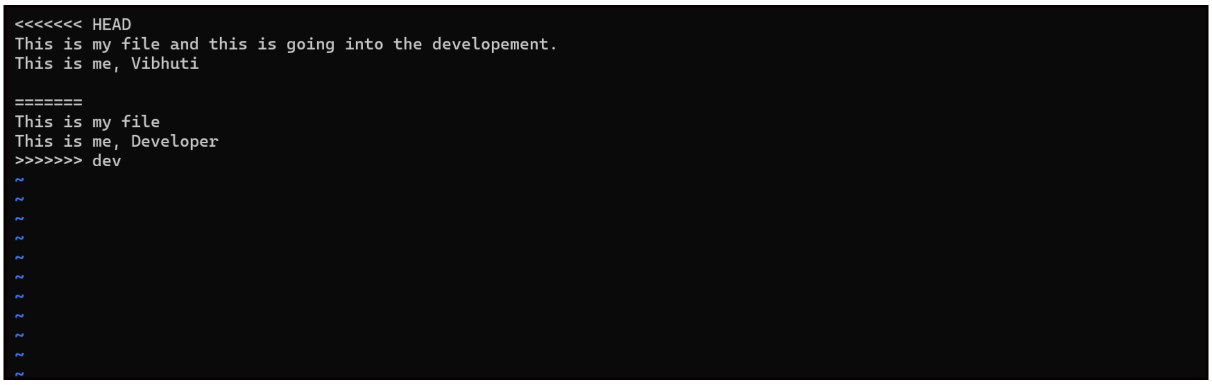
```
ubuntu@ip-172-31-3-85:~/git/Git$ git rebase -i HEAD~3
Successfully rebased and updated refs/heads/main.
ubuntu@ip-172-31-3-85:~/git/Git$ git log --online
fatal: unrecognized argument: --online
ubuntu@ip-172-31-3-85:~/git/Git$ git log --oneline
ab7eadc (HEAD -> main) feature-y added
3a5e020 Jenkins
879461c added file2.txt
55b9368 added main feature
cab1a89 added hotfix
c4362f4 (origin/main, origin/dev, origin/HEAD) added new feature line
02d4eda added feature4.txt
a5f2af3 added feature2 file
99ddb71 Merge pull request #1 from Vibhutijain12/dev
25ca552 added file dev branch
3fc3165 renamed file feature3.txt to new_file.txt
b287c49 added feature3.txt file
bf603b9 added again feature2.txt
af122b1 added feature 2
a4e8e76 added feature1 file
c827bfb Added git file
a236691 Git Cheat Sheet
ubuntu@ip-172-31-3-85:~/git/Git$
```

- git merge conflicts - A **Merge Conflict** is an event that occurs when Git is unable to automatically resolve differences in code between two commits.

  While Git is very smart at merging files, it only understands **lines of code**, not the **logic** of the code. If two different changes are made to the same line of the same file, or if one person deletes a file that another person is modifying, Git gets "confused" and asks you to decide which version to keep.



```
ubuntu@ip-172-31-3-85:~/git/Git$ git merge dev
Auto-merging feat-1.txt
CONFLICT (content): Merge conflict in feat-1.txt
Automatic merge failed; fix conflicts and then commit the result.
ubuntu@ip-172-31-3-85:~/git/Git$ vim feat-1.txt
ubuntu@ip-172-31-3-85:~/git/Git$
```



```
<<<<<<< HEAD
This is my file and this is going into the developement.
This is me, Vibhuti

=======
This is my file
This is me, Developer
>>>>>>> dev
~
~
~
~
~
~
~
~
~
~
~
```

✅ How to Resolve a Conflict

You must manually edit the file to fix the "clash."

1. **Open the file:** Use your code editor (VS Code, IntelliJ, etc.).
2. **Edit the code:** Remove the markers (`<<<<`, `====`, `>>>>`) and rewrite the lines so they look exactly how you want them to. You can keep Version A, Version B, or a mix of both.
3. **Stage the fix:** Tell Git you've resolved the issue by running:
   - `git add <filename>`
4. **Complete the merge:**
   - `git commit -m "Resolved merge conflict in index.html."`