Git & GitHub

Known as version control system
staging area →review, adjust, remove the changes

Git vs GitHub

- **Git** is a version control system that runs locally and tracks changes in files.
- **GitHub** is an online platform that hosts Git repositories for collaboration.
- Simple analogy: **Git is the coffee; GitHub is the coffee shop.**

What Git Does

Git records:

- What changed
- When it changed
- Who changed it

It stores multiple versions of files, so you can restore previous versions anytime.

Core Git Areas

Git works in three main areas:

1. **Working Directory** – Where you create and edit files
2. **Staging Area** – Where you prepare changes before committing. The Staging Area is an intermediate step between working directory and commit. It lets you choose which changes you want to include in the next commit.
3. **Repository** – Where commits are permanently saved

Flow:

Working → Staging → Repository

.

Git & GitHub Commands – Detailed Explanation

Repository Setup Commands

git init

Purpose:
Initializes a new Git repository in the current folder.

What it does internally:

- Creates a hidden. git folder
- Starts tracking changes in that directory

When to use:
When starting a brand-new project locally.

git init

git clone <repository-url>

Purpose:
Creates a local copy of a remote repository (GitHub, GitLab, etc.)

What it does internally:

- Downloads all files
- Copies full commit history
- Automatically links local repo with remote

git clone https://github.com/user/repo.git

Checking Repository Status

git status

Purpose:
Shows the current state of the working directory and staging area.

Displays:

- Modified files
- New (untracked) files
- Staged files

- Deleted files

git status

This command is your **best friend** in Git.

Untracked = “Git hasn’t met this file yet.” –>new file

Adding Files to Staging Area

git add <file>

Purpose:
Stages a specific file.

git add 1.txt

git add <folder/file>

Stages a file inside a folder.

git add myfolder/3.txt

git add .

Best practice command

Stages:
✓ New files
✓ Modified files
Deleted files
Files inside subfolders

git add .

stage the change within the current directory you are in.

git add *

Stages:
New and modified files
Does NOT stage deleted files
Does NOT include subfolders

git add *
git add --all or git add -A
Stages **everything across the entire project**, including deletions.

git add –all
git add -A

git add *.txt

Stages all .txt files in the current directory. Except deleted one
git add *.txt

Removing Files

git rm <file>

Deletes the file **and stages the deletion**.
It can't delete uncommitted modification files
git rm 4.txt

git rm --force <file>
Force deletes a modified file without committing changes.
git rm -f 4.txt

git rm --cached <file>
Removes file from Git tracking **but keeps it locally**. Will be untracked.
git rm --cached 4.txt
Used when you want Git to ignore a file.

git rm -r <folder>
Deletes a folder recursively.
git rm -r myfolder

Committing Changes

git commit -m "message"

Purpose:
Saves staged changes permanently to repository history.
git commit -m "Added new feature"

Each commit includes:

- Author name
- Email
- Timestamp
- Unique commit ID

Git identity setup (First time only)

git config --global user.name "Your Name"

git config --global user.email "you@email.com"

for local repo we must replace global by “local”

Undoing Changes

git reset

Description:
git reset removes files from the staging area but keeps the changes in the working directory.

Simple meaning:
It unstages changes without deleting your work.
it does not bring back the deleted file.

git reset --hard

Dangerous command

- Removes staged changes
- Removes working directory changes
- Restores last commit exactly
  it brings back the deleted file.
  It does not delete any commits; it stays on the current commit and only deletes all uncommitted changes.
  git reset --hard

git reset HEAD~

- Removes staged changes
- delete last commit exactly, bring it to the working directory. But it keeps file changes.

git restore <file or folder name>

Restores file to last committed state.
git restore 1.txt

git restore --staged <file or . >

Removes file from staging area only.
git restore --staged 1.txt

git restore .

Restores entire repository to last commit.
git restore .

git revert
git revert <commit-id>
What happens?
Suppose commit history:
A → B → C → D
You revert commit C.
After running:
git revert C
History becomes:
A → B → C → D → E
Where:

- E is a new commit
- E cancels changes made in C

So:
C is not deleted
A new commit undoes C

Viewing Commit History

git log
Shows full commit history.
Press **Q** to exit.

git log –oneline
Compact commit history.

Comparing Changes

git diff
Shows unstaged changes.

git diff <commit1> <commit2>

Compares two commits.
git diff abc123 def456

Branching Commands

----> review test and manage the code on another place then can be merged to main

git branch
Lists all branches.

git branch <new branch name>
creating a new branch. Exact copy of main

git checkout <branch>

Switches branches.
git checkout development

git checkout <commit id>
going back to that commit

git checkout -b <branch>

Creates and switches to a new branch.
git checkout -b feature-login

Merging Branches

git merge <branch>

Merges another branch into current branch.
git merge development

-m using for comment

Merge Conflict

A merge conflict occurs when Git cannot automatically merge changes because the same part of a file was modified differently in two branches.
The conflict must be resolved manually by choosing or combining the changes, **then committing the resolved file**. This is the solve

git rebase <branch name>
git rebase moves the commits of one branch on top of another branch, creating a linear history.

It rewrites commit history instead of creating a merge commit.

Remote Repository Commands

git push origin <branch name>

Uploads local commits to remote.
git push origin main

git fetch
Downloads remote changes **without merging**.
git fetch
then
git merge

git pull

Fetches + merges remote changes.

Git Stash (Temporary Save uncommitted works)

git stash

Saves unfinished work temporarily.
git stash
git stash list
Shows all stashes.

git stash pop

Restores and removes stash.

git stash apply

Restores but keeps stash.

for specific stash
➜ git-journey git:(main) git stash pop stash@{0}

➜ git-journey git:(main) git stash apply stash@{0}

git stash drop

Deletes stash permanently.

git stash drop

Pull Requests (GitHub Concept)

Not a command, but a **GitHub workflow**:

- Create branch
- Push changes
- Open Pull Request
- Review
- Merge into main

Created by
Sowmik vadro