All the commands and their explaination

All the essential Git operations, categorized by their function in the development workflow. Maximum commands are in the squence like what you will be doing after this --

I. Initial Setup and Configuration

These commands are run once on your local machine to identify you and configure Git's behavior.

Command	Description	Practical Code Example
git config --global user.name	Sets the name that will be attached to all your commits.	git config --global user.name "Your Name"
git config --global user.email	Sets the email that will be attached to all your commits.	git config --global user.email "your.email@example.com"
git config --list	Shows all current configuration settings.	git config --list
git init	Initializes a new local Git repository in the current directory.	git init
git clone [url]	Downloads an existing project from a remote repository (e.g., GitHub).	git clone https://github.com/user/project.git

II. Basic Workflow (Tracking Changes)

These commands are used constantly to manage files in the three states of the Git three-tree architecture: Working Directory, Staging Area, and Local Repository.

Command	Description	Practical Code Example
git status	Shows the current state of the repository (modified, staged, untracked files).	git status
git add [file]	Moves specific modified files from the Working Directory to the Staging Area.	git add index.html

git add .	Moves ALL modified files from the Working Directory to the Staging Area.	git add .
git commit -m "[message]"	Records the staged snapshot permanently to the Local Repository history.	git commit -m "Add initial structure for the header"
git commit --amend	Rewrites the most recent commit (useful for fixing typos or adding forgotten files).	git commit --amend -m "Updated header structure and fixed CSS path"
git log	Displays the history of commits in reverse chronological order.	git log
git log --oneline	Displays a concise, one-line summary of the commit history.	git log --oneline
git diff	Shows unstaged changes between the Working Directory and the Staging Area.	git diff
git diff --staged	Shows staged changes ready to be committed.	git diff --staged

III. Branching and Merging

Branching allows developers to work on features or fixes in isolation.

Command	**Description**	**Practical Code Example**
git branch	Lists all local branches in the repository.	git branch
git branch [name]	Creates a new branch (does not switch to it).	git branch feature/user-auth
git checkout [branch]	Switches your Working Directory to the specified branch.	git checkout feature/user-auth
git checkout -b [name]	Creates a new branch AND switches to it (shorthand for the two commands above).	git checkout -b fix/typo-on-homepage
git merge [branch]	Integrates changes from the specified branch into the current branch (creating a new merge commit).	git merge feature/user-auth

git branch -d [name]	Deletes the specified local branch (must be fully merged).	git branch -d feature/user-auth
git push origin --delete [branch]	Deletes a branch from the remote repository.	git push origin --delete old-feature

IV. Remote Operations (Syncing with the Server)

These commands manage interaction between your Local Repository and the shared Remote Repository (e.g., on GitHub, GitLab).

Command	**Description**	**Practical Code Example**
git remote -v	Lists the remote repositories you have configured.	git remote -v
git push [remote] [branch]	Uploads your local branch commits to the remote repository.	git push origin main
git pull [remote] [branch]	Fetches changes from the remote repository and *automatically merges* them into your current local branch.	git pull origin main
git fetch [remote]	Downloads new data from a remote repository but does not merge or modify your local Working Directory.	git fetch origin
git remote add [name] [url]	Connects your local repository to a new remote server.	git remote add upstream https://github.com/project/original.git

V. Advanced Workflow Commands

These are powerful commands used for cleaning up history, moving commits, or integrating changes in a specific way.

Rebase vs. Merge

Command	**Description**	**Use Case**

git rebase [branch]	Rewrites history: Moves a sequence of commits to a new base commit, creating a linear history.	To keep a cleaner, non-interrupted project history.
Example:	Rebases your current branch onto main. All your new commits will appear *after* the latest main commit.	git rebase main
git merge [branch]	Preserves history: Combines two commit histories by creating a new merge commit.	To show exactly when and where a feature branch was merged.

Cherry-Pick

Command	**Description**	**Practical Code Example**
git cherry-pick [commit-hash]	Applies the changes introduced by a single existing commit from one branch onto your currently checked-out branch.	git cherry-pick 1e45f8a

VI. Utility and Cleanup

Command	**Description**	**Practical Code Example**
git stash	Temporarily saves all modified, staged, and unstaged changes, allowing you to switch branches quickly without committing.	git stash save "WIP on user profile"
git stash pop	Restores the most recently stashed changes and removes the stash from the list.	git stash pop
git tag [name]	Creates a permanent pointer (tag) to a specific point in history, usually used for release versions.	git tag v1.0.0
git reset --hard [hash]	DANGEROUS: Moves the current branch head to the specified commit, discarding all changes (unstaged and staged) since that commit.	git reset --hard HEAD~1 (Discards the last commit)
git restore [file]	Newer alternative to checkout: Discards changes in the Working Directory or Staging Area.	git restore index.js (Unstages or reverts index.js to the last committed version)

Forking (Concept)

"Forking" is not a Git command; it's a platform operation (e.g., on GitHub). When you fork a repository, you create a complete copy of the repository on your account. You then git clone this copy, make changes, and use a git push to your fork. To contribute back to the original project, you submit a Pull Request (PR), which the maintainers of the original project can then review and git merge.

NOTE1: All the des. Has been given already, still I am giving this in an arranged manner as you guys requested and after this if you do not practice it you wil be never getting it.

NOTE2: The remote, main, or any other path i.e the calls are according to my system. Kindly change before running. Basic:: take a project – add – change it – add features if u want to – commit – push – pull – merge – do whatever you want .. in betweeen check for status log and your account github also.
NOTE3: before push or pull or anything – add your origin ie-> repo and github account for example -- git remote add origin https://github.com/chongder6/chat_PDF -> this one for my you add yours.