1. Configuration Commands

`git config`

Description: Sets Git configuration values (user identity, editor, aliases, etc.).

- `git config --global user.name "Ribhav Jain"`
- `git config --global user.email "ribhav@example.com"`

Use Case: Set your identity and preferences globally or per repository.
Tip: Use `git config --list` to verify current settings.

Local configuration:

- `git config --local <setting> <value>`
- `git config --local user.email "jane.doe@work.com"`
  Description: Overrides global settings for the current repository (stored in `.git/config`).
  Use Case: Use a different email for work vs personal projects.

🌿 2. Repository Basics

`git init`

Description: Initializes a new Git repository by creating a hidden `.git/` folder.
Use Case: Begin tracking changes in a local project.

`git clone`

Description: Clones a remote repository (with full history and remote connection).

- `git clone https://github.com/user/repo.git`
  Use Case: Download and work on a project from a remote repository.

📂 3. Working Directory & Index

`git status`

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

- `git status`
  Use Case: See which files are untracked, modified, or staged.

`git add`

Description: Stages file changes for the next commit.

- `git add file.txt` – Add specific file
- `git add .` – Stage all changes
  Tip: Use `git add -p` to **interactively stage hunks** of changes.

`git commit`

Description: Commits staged changes to the local repository with a message.

- `git commit -m "Add feature X"`
  Tip: Combine add and commit for tracked files: `git commit -am "Fix bug"`

🌲 4. Branching

`git branch`

Description: List, create, or delete branches. Creates a new branch based on the current commit (HEAD). It doesn't switch to the new branch.

- `git branch` – List local branches
- `git branch -a` – List local and remote-tracking branches
- `git branch feature-x` – Create new branch
- `git branch -d old-feature` – Delete branch

Tip: Use descriptive names like `feature/login`, `bugfix/payment-issue`.

`git checkout` / `git switch`

Description: Switch between branches or restore files.

- `git checkout feature-x` – Switch to branch
- `git checkout -b new-feature` – Create and switch
- `git switch -c feature-x` – Newer alternative (Git 2.23+)

Tip: Ensure a clean working directory before switching branches(no uncommitted changes or stash your changes before switching to avoid issues).

`git merge`

Description: Merges changes from one branch into the current one.

- `git merge feature-x`
  Use Case: Integrate completed features into main branches.
  Tip: Always switch to the target branch before merging.

`git rebase`
Description: Reapplies commits on top of another base tip (rewrites history).

- `git rebase main`
  Use Case: Create a clean, linear history before merging.
  Tip: Avoid rebasing shared branches. Use `git fetch` + `git rebase origin/main` to keep your branch up-to-date.
- **Tip: Powerful but dangerous if misused.** *Never* rebase commits that have already been pushed and shared with others, unless you coordinate carefully with your team, as it rewrites history. Merge conflicts during rebase can be more complex to resolve than merge conflicts during a standard merge.

🌍 5. Remote Repositories

`git remote`
Description: Manages remote repository connections.

- `git remote -v` – List current remotes
- `git remote add origin <url>` – Add new remote
  Tip: `origin` is the default alias for your main remote.

`git fetch`
Description: Downloads objects and refs from the remote but does not merge.

- `git fetch origin`
  Use Case: Review changes made on the remote before merging or rebasing.
  Tip: After fetch:
- `git merge origin/main`
- `git rebase origin/main`

`git pull`
Description: Fetch + merge from the upstream branch.

- `git pull origin main`
  Tip: Can lead to merge conflicts; some prefer manual fetch and rebase for control.

`git push`
Description: Uploads your commits to a remote repository.

- `git push origin main`
- `git push -u origin feature-x` – Set upstream tracking

Tip:

- First-time push needs `-u` or `--set-upstream`
- **Use `git push --force` or `git push --force-with-lease` with extreme caution.** It overwrites the remote history and can cause problems for collaborators. `--force-with-lease` is slightly safer as it checks if the remote branch has changed since your last pull. Generally avoid forcing pushes on shared branches.

🩹 6. Undoing Changes

`git restore`

Description: Restores file content from last commit or unstages files.

- `git restore file.txt` – Revert file
- `git restore --staged file.txt` – Unstage file
- **Warning: This permanently loses the changes you made since the last commit.**

`git reset`

Description: Unstages files or resets branch history.

- `git reset file.txt` – Unstage
- `git reset --hard HEAD~1` – Roll back last commit

Modes:

- `--soft` – Keep changes staged
- `--mixed` (default) – Unstage but keep changes
- `--hard` – Discard everything (destructive)

Tip: Avoid `--hard` on shared history.

`git revert <commit>`

Description: Safely undo a commit by creating a new inverse commit.

`git clean -fd`

Description: Deletes untracked files/directories.

- `-f` – Force
- `-d` – Include directories

- `-n` – Preview deletions
  Use Case: Clean up build artifacts or logs or other generated files that are not (and should not be) tracked by Git.

🧳 7. Stashing

`git stash`
Description: Temporarily saves uncommitted changes.

- `git stash`
  `git stash push -m "WIP on bugfix"`

Usage:

- `git stash list` – View saved stashes
- `git stash apply stash@{1}` – Reapply a specific stash
- `git stash pop` – Apply and remove latest stash
- `git stash drop stash@{2}` – Delete specific stash
- `git stash clear` – Delete all stashes

🏷️ 8. Tags

`git tag`
Description: Labels a specific commit.

- `git tag v1.0` – Lightweight tag
- `git tag -a v1.1 -m "Release v1.1"` – Annotated tag

Pushing tags:

- `git push origin v1.0`
- `git push origin --tags` – Push all tags

Deleting tags:

- `git tag -d v1.0` – Delete local tag

📜 9. Logs and History

`git log`
Description: Shows commit history.

- `git log`
- `git log --oneline --graph --all --decorate`

Modifiers:

- `--oneline` – Short format
- `--graph` – ASCII graph
- `--stat` – File change summary
- `-p` – Show diffs
- `--author="Name"` – Filter by author

`git show <commit>`
Description: Shows details of a specific commit.

`git diff`
Description: Compare file changes across states.

- `git diff` – Working dir vs staged
- `git diff --staged` – Staged vs HEAD
- `git diff HEAD~1 HEAD` – Between commits

`git blame <file>`
Description: Shows who last modified each line.
Use Case: Audit responsibility or understand changes.

🧪 10. Useful Extras

`git cherry-pick <commit>`
Description: Apply a specific commit from another branch into the current one.
Use Case: Bring over an isolated change without merging the full branch.

🧠 Pro Tips

- **Use `.gitignore`** to exclude temporary or sensitive files.
- **Use `git reflog`** to recover lost commits or see HEAD history.