Setup

- `git config --global user.name "Your Name"`
- `git config --global user.email "you@example.com`

Starting a Project

- `git init - Initialize a local Git repo.`
- `git clone - Copy a remote repo to local.`

Basic Workflow git status - Show status of changes.

- `git add filename- Stage a file.`
- `git add . - Stage all files.`
- `git commit -m "message" - Save your staged changes.`
- `git push origin main - Push commits to remote repo.`
- `git pull origin main - Fetch + merge from remote`

Branching

- `git branch - List branches.`
- `git branch <branchname>- Create a new branch.`
- `git checkout <branchname> - Switch to a branch.`
- `git checkout -b <branchname> - Create + switch branch.`
- `git merge <branchname> - Merge another branch into current.`
- `git branch <branchname> -d - Delete a branch`

Syncing

- `git fetch - Download changes (don't merge).`
- `git pull --rebase - Fetch and reapply local changes.`

Undoing

- `git diff - See unstaged changes.`
- `git restore - Undo unstaged changes.`
- `git reset HEAD - Unstage a staged file.`
- `git revert - Undo a commit safely.`
- `git reset --hard - Dangerous! Full rollback`

History

- `git log - Show commit history.`
- `git log --oneline --graph --all - Visual branch history`

Quick Git Flow

git clone -> git branch -> git checkout -> git add -> git commit -> git push

Got it — you want a **Git Commands Cheat Sheet** focusing on:

- reset
- revert
- stash
- cherry-pick
- rebase
- merge

Here's a clean, quick-reference cheat sheet for you:

Git Commands Cheat Sheet

git reset

- **Move HEAD and optionally modify working directory and index.**
- Soft Reset (keep changes staged):

```
git reset --soft <commit>
```

- Mixed Reset (unstage changes):

```
git reset --mixed <commit>   # Default
```

- Hard Reset (discard changes):

```
git reset --hard <commit>
```

- Reset a single file:

```
git reset HEAD <file>
```

git revert

- **Create a new commit that undoes changes from a previous commit.**
- Revert a single commit:

```
git revert <commit>
```

- Revert a range of commits:

```
git revert <oldest_commit>^..<newest_commit>
```

(Note: Unlike reset, revert is safe for shared branches.)

git stash

- **Save and clear uncommitted changes temporarily.**
- Stash changes:

```
git stash
```

- Stash with message:

```
git stash save "message"
```

- List stashed changes:

```
git stash list
```

- Apply last stash:

```
git stash apply
```

- Apply specific stash:

```
git stash apply stash@{2}
```

- Drop a stash:

```
git stash drop stash@{0}
```

- Pop last stash (apply and remove):

```
git stash pop
```

git cherry-pick

- **Apply a specific commit from one branch onto another.**
- Cherry-pick a commit:

```
git cherry-pick <commit>
```

- Cherry-pick multiple commits:

```
git cherry-pick <commit1> <commit2>
```

- Continue after conflicts:

```
git cherry-pick --continue
```

git rebase

- **Move or combine commits to rewrite history.**
- Rebase current branch onto another:

```
git rebase <branch>
```

- Interactive rebase (edit, squash commits):

```
git rebase -i <commit>^
```

- Continue after fixing conflicts:

```
git rebase --continue
```

- Abort rebase:

```
git rebase --abort
```

git merge

- **Merge another branch into your current branch.**
- Merge a branch:

```
git merge <branch>
```

- Merge with a commit message:

```
git merge --no-ff <branch> -m "Merge message"
```

- Resolve conflicts if any, then:

```
git commit
```

Quick Tip:
Use `git log --oneline --graph --all` to visualize merges, cherry-picks, and rebases easily!

Git Commands Cheat Sheet

git reset

Move HEAD and optionally modify working directory and index.

Soft Reset (keep changes staged):

```
git reset --soft <commit
```

Mixed Reset (unstage changes):

```
git reset --mixed <commit
```

Hard Reset (discard changes)

```
git reset --hard <commit
```

git revert

Create a new commit that undees changes from a previous commit

```
git revert <commit
```

Revert a range of commits:

```
git revert <oldest_commit>
    ^ .<newest_commit>
```

(Note: Unlike reset, revert is safe for shared branches.)

git stash

Save and clear uncommitted changes temporarily.

Stash changes

```
git stash
```

Stash with message

```
git stash save "message"
```

List stashed changes

```
git stash list
```

git cherry-pick

Apply a specific commit from one branch onto anoher

Cherry-pick a commit

```
git cherry-pick <commit>
```

Cherry-pick multiple commits

```
git cherry-pick <commit><comm2>
```

Continue after conflicts

```
git cherry-pick --coninue
```

git rebase

Move or combine commits to rewrite history

Rebase current branch onto another

```
git rebase <branch
```

Interactive rebase (edit squash co...)

```
git rebase –i <commit>^
```

Continue after fixing conflicts

```
git rebase --continue
```

git merge

Merge another branch into your current branch

Merge a branch

```
git merge <branch
```

Merge with a commit message

```
git merge —no-ff <branch
    –m "Merge message"
```

Resolve conflicts if any, then

Git: Working Area vs Staging Area

1. Working Directory (Working Area)

- **What it is:**
  The **Working Directory** (or Working Area) is your **local project folder** — the actual files and code you're editing right now.
- **What happens here:**
  - You create, delete, and modify files.
  - Changes are **NOT tracked** by Git until you tell it to (using `git add`).
- **Analogy:**
  Think of it like **your desk** where you're actively writing and editing documents.

---

2. Staging Area (Index)

- **What it is:**
  The **Staging Area** (also called the Index) is a **temporary area** where you collect all changes that you want to commit together.
- **What happens here:**
  - You use `git add <file>` to move changes from the Working Directory to the Staging Area.
  - Only files in the Staging Area will be included in your next `git commit`.
- **Analogy:**
  It's like a **basket** where you gather the papers you want to submit. Once you're ready, you send them all together (commit).

Quick flow:

```
scss

[You edit files] → (Working Directory)

      ↓ git add

[Files prepared for commit] → (Staging Area)

      ↓ git commit

[Files saved in repository] → (Git history)
```

Quick Commands:

Task	Command
Check current status (working + staging area)	`git status`
Add file to staging area	`git add <file>`
Add all changes to staging	`git add .`
Commit staged changes	`git commit -m "Message"`
Unstage a file	`git reset HEAD <file>`