Git Guide for future Full-Stack Developers

by Alek Mugnozzo - mugnozzo.xyz - mugnozzo.net

---

1. What Git Is (and What It Isn’t)

1.1 Version Control in One Sentence

Git is a **version control system**: it keeps track of changes to files over time so you can **go back**, **compare**, and **collaborate safely**.

1.2 Git vs GitHub / GitLab

- **Git**: the tool, runs on your computer, works offline
- **GitHub / GitLab**: hosting platforms for Git repositories

Think of Git as *the engine*, GitHub as *the parking lot + collaboration space*.

1.3 Core Vocabulary

- **Repository (repo)**: a project tracked by Git
- **Commit**: a snapshot of your project at a given time
- **Branch**: an independent line of development

---

2. Mental Model (Very Important)

2.1 Snapshots, Not Diffs

Each commit is a **snapshot** of the entire project, not just a list of changes.

Git figures out what changed for you.

2.2 The Three Zones

```
Working Directory  ->  Staging Area  ->  Repository
(files)                (git add)        (git commit)
```

- **Working directory**: your files as you edit them
- **Staging area**: what *will* be in the next commit
- **Repository**: the history of commits

2.3 HEAD and Current Branch

- **HEAD**: where you are right now
- Usually points to the **latest commit of the current branch**

---

3. Installing and First-Time Setup

3.1 Installing Git

- Windows: Git for Windows
- macOS: Xcode Command Line Tools or Homebrew
- Linux: system package manager

3.2 Basic Configuration

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

Optional but recommended:

```
git config --global init.defaultBranch main
git config --global core.editor nano
```

3.3 HTTPS vs SSH

- **HTTPS**: easier to start with
- **SSH**: better long-term (no passwords)

For beginners: **start with HTTPS**.

---

4. Creating and Cloning Repositories

4.1 Creating a Repository

```
git init
```

Creates a hidden `.git/` folder that contains the whole history.

4.2 The `.git` Folder (Conceptually)

Contains:

- commits
- branches
- configuration

👉 If you delete it, the project is no longer a Git repo.

4.3 Cloning a Repository

```
git clone <url>
```

Creates:

- a project folder
- the full history
- a remote called `origin`

---

5. The Daily Workflow

5.1 `git status` (Your Best Friend)

```
git status
```

Use it **all the time**.

5.2 Adding Files to the Staging Area

```
git add file.txt
git add .
```

Partial add (advanced but powerful):

```
git add -p
```

5.3 Committing Changes

```
git commit -m "Add login form"
```

5.4 Viewing History

```
git log
git log --oneline
```

5.5 Inspecting Changes

```
git diff          # unstaged changes
git diff --staged # staged changes
```

---

6. Good Commit Habits

6.1 Writing Good Commit Messages

Good:

```
Add user registration validation
Fix navbar layout on mobile
```

Bad:

```
update
stuff
fix
```

6.2 Commit Size

- One idea per commit
- If you need "and", split the commit

6.3 When to Commit

- Code works
- Tests pass (if any)
- You could explain *why* this commit exists

---

7. Undoing Mistakes (Without Panic)

7.1 Unstaging a File

```
git restore --staged file.txt
```

7.2 Discarding Local Changes

```
git restore file.txt
```

⚠️ This deletes local changes.

7.3 Amend Last Commit

```
git commit --amend
```

Useful for:

- fixing commit message
- adding forgotten files

7.4 Reverting a Commit (Safe)

```
git revert <commit>
```

Creates a *new* commit that undoes another one.

7.5 Reset (Danger Zone)

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

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

Only use `--hard` if you **know exactly** what you are doing.

7.6 Reflog (Emergency Tool)

```
git reflog
```

Shows **everything Git knows you did**, even deleted commits.

---

8. Branches

8.1 Why Branches Exist

Branches let you:

- try ideas safely
- work in parallel
- avoid breaking `main`

8.2 Creating and Switching

```
git switch -c feature/login
git switch main
```

8.3 Naming Conventions

- `feature/login`
- `fix/navbar-bug`
- `chore/update-deps`

8.4 Comparing Branches

```
git log main..feature/login
git diff main..feature/login
```

---

9. Merging

9.1 Fast-Forward vs Merge Commit

- **Fast-forward**: linear history
- **Merge commit**: explicit merge

9.2 Merging a Branch

```
git switch main
git merge feature/login
```

9.3 Merge Conflicts

Conflicts happen when Git cannot decide automatically.

9.4 Resolving Conflicts

1. Open conflicted files
2. Decide what to keep
3. Remove conflict markers
4. `git add`
5. `git commit`

---

10. Rebase (Optional)

10.1 Rebase vs Merge

- **Merge**: preserves history
- **Rebase**: rewrites history

Rule:

> Never rebase commits that are already shared.

10.2 Interactive Rebase (Cleanup)

```
git rebase -i HEAD~3
```

Use to:

- squash commits
- rewrite messages

---

11. Working With Remotes

11.0 Bare Repositories (Important Concept)

What Is a Bare Repository?

A **bare repository** is a Git repository **without a working directory**.

- No project files
- No `index.html`, no `src/`, no code to edit
- Only Git's internal data (commits, branches, tags)

In practice, a bare repo looks like what is usually inside a `.git/` folder.

Why Bare Repositories Exist

Bare repositories are meant to be used as **shared central repositories**:

- a place to **push to**
- a place to **pull from**
- **not** a place to write code

This is exactly how Git hosting platforms work internally.

Example: Creating a Bare Repository

```
git init --bare project.git
```

Notes:

- The `.git` extension is a **convention**, not a requirement
- You should never work directly inside this repo

Bare Repo vs Normal Repo

Normal repository	Bare repository
Has working files	No working files

Normal repository	Bare repository
Used for coding	Used for sharing
`git status` makes sense	`git status` is meaningless

Typical Real-World Setup

```
/central-repos/project.git   (bare repo)

/dev/alek/project            (normal repo)
/dev/sam/project             (normal repo)
```

All developers:

- clone from the bare repo
- push to the bare repo

Important Rule

> **Never push to a non-bare repository that someone is actively working in.**

This can corrupt the working directory of others.

---

11. Working With Remotes

11.1 Remotes and `origin`

```
git remote -v
```

11.2 Fetch, Pull, Push

```
git fetch
git pull
git push
```

- **fetch**: download history
- **pull**: fetch + merge/rebase
- **push**: send your commits

11.3 Upstream Branch

```
git push -u origin feature/login
```

---

12. Pull Requests (Team Workflow)

Typical flow:

1. Create branch
2. Commit changes
3. Push branch
4. Open PR
5. Review
6. Merge

Good PRs are:

- small
- focused
- easy to read

---

13. Tags and Releases

13.1 What Tags Are For

- mark versions
- releases

```
git tag v1.0.0
git push --tags
```

---

14. Stashing (Parking Work)

```
git stash
git stash list
```

```
git stash pop
```

Use when:

- you must switch branches quickly

---

15. Ignoring Files

15.1 `.gitignore`

Common examples:

```
node_modules/
.env
.idea/
```

15.2 Already Committed Files

```
git rm --cached file.txt
```

---

16. Debugging With Git

16.1 Blame

```
git blame file.txt
```

16.2 Bisect (Advanced)

Find which commit introduced a bug.

---

17. Common Beginner Problems

- Detached HEAD
- Conflicts panic
- Committed on wrong branch
- Push rejected

👉 All of these are **normal**.

---

18. Recommended Workflow for School Projects

18.1 Solo Projects

- work on `main`
- commit often
- push regularly

18.2 Group Projects

- protect `main`
- feature branches
- pull requests only

---

19. Mini Cheat Sheet

Daily:

```
git status
git add
git commit
git pull
git push
```

Fixing:

```
git restore
git revert
git reflog
git stash
```

---

20. Exercises

1. Create a repo and make 5 clean commits
2. Create a branch and merge it

3. Trigger and resolve a conflict
4. Recover a deleted commit using reflog

> Git is not about commands. It is about **confidence**, **safety**, and **collaboration**.