

# Git Fundamentals – Summer Practice

## Course Support

Welcome to your Git journey! This guide was written to support your learning during the summer practice and beyond. It's written in a casual but informative tone – just like I'd explain it to you during our sessions.

### 1. Introduction to Version Control

Git is a free and open-source distributed version control system. It helps developers manage changes to code, collaborate smoothly, and keep a full history of their project. Whether you're working alone or in a team, Git gives you control over the evolution of your codebase.

### 2. Why Do We Need Version Control?

Software development is messy – and that's normal! Code evolves constantly, features get added, bugs get fixed, and mistakes happen.

A version control system helps us handle all of this safely and with confidence.

With Git, you can:

- **Track Changes Over Time**

Keep a full history of your project. You can see what changed, who changed it, and when. You can always roll back to a previous version if needed.

- **Collaborate Efficiently**

Multiple people can work on the same codebase without stepping on each other's toes. Git lets you safely merge everyone's work together.

- **Experiment Safely (Branching)**

Try out new features or fixes in isolated branches. Once you're happy with your work, you can merge it back into the main branch.

- **Maintain Code Quality**

Using Pull Requests and code reviews, teams can spot mistakes early, suggest improvements, and maintain high standards.

- **Ensure Backup and Recovery**

Your code is not just on your laptop. By pushing it to GitHub or GitLab, your project stays safe even if your machine isn't.

- **Keep an Audit Trail**

Every commit is a snapshot with a message. This helps you (and your team) understand the history and intent behind every change. It's debugging gold.

"Six months ago, only me and God understood this code. Now only God understands." — That's what happens when you don't use version control properly 😞

### **3. Git vs GitHub (and others)**

- Git is the tool.
- GitHub, GitLab, and Bitbucket are platforms that host your Git repositories online and provide collaboration features (issues, pull requests, CI/CD, etc).

You can use Git without GitHub, but GitHub makes teamwork much easier.

### **4. Installing & Configuring Git**

Before using Git, make sure you:

1. Install Git: <https://git-scm.com/downloads>
2. Create an account on GitHub, GitLab, or any other provider you prefer.

## Global Configuration

Run these commands in your terminal to set your name and email:

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

These are used in every commit to track who made the change.

## Local Configuration (optional)

If you want to use different credentials for personal vs work projects:

```
cd my-awesome-project
# then run:
git config user.name "Work Name"
git config user.email "you@work.com"
```

# 5. Creating a Repository

## Option 1: Clone from GitHub (most common)

1. Create a new repo on GitHub
2. Run this locally:

```
git clone https://github.com/your-username/your-repo.git
cd your-repo
```

You're ready to work!

## Option 2: Start Locally and Connect to Remote Later

```
mkdir my-project  
cd my-project  
git init
```

Now add a remote:

```
git remote add origin https://github.com/your-username/your-repo.git  
git push -u origin main
```

This sets up your local repo to push to GitHub.

## 6. Git Basics: add, commit, status, log

These are the fundamental Git commands you'll use daily:

```
# Check Your Current Status. See what's changed since your last commit. Red  
# files = unstaged, green = staged.  
git status  
  
# Add Changes  
git add <file>  
  
# Stage changes for commit. You can also use . to stage everything  
git add .  
  
# Commit Changes. Make sure the message clearly explains the change!  
git commit -m "describe what you did"  
  
# View Commit History. Press q to exit the log view.  
git log --oneline --graph --all  
  
# See Differences  
git diff          # unstaged vs working directory  
git diff --staged # staged vs last commit
```

## 7. Branching & Merging

```
# Create a New Branch
git checkout -b feature/new-cool-thing

# Switch Between Branches
git checkout main

# Merge Your Branch into Main
git checkout main
git merge feature/new-cool-thing

# Delete a Merged Branch
git branch -d feature/new-cool-thing
```

Pro tip: Always pull the latest changes from main before starting a new branch.

## 8. Working with Remotes

```
# Push Your Changes
git push

# If it's the first push of a new branch:
git push -u origin your-branch-name

# Pull Latest Changes. This fetches + merges changes from the remote.
git pull

# Just Fetch (no merge)
git fetch
```


## 9. GitHub Workflow: Pull Requests & Reviews

- Fork or clone the repo
- Create a new branch
- Make changes and push to your branch
- Go to GitHub and open a Pull Request
- Request a code review
- After approval, merge it into main

Pull Requests let your teammates comment, suggest changes, and keep the code clean.

## 10. Resolving Merge Conflicts

Conflicts happen when Git can't auto-merge changes. You'll see conflict markers like:



```
<<<<<<< HEAD
code from your branch
=====
code from main
>>>>>>> main
```

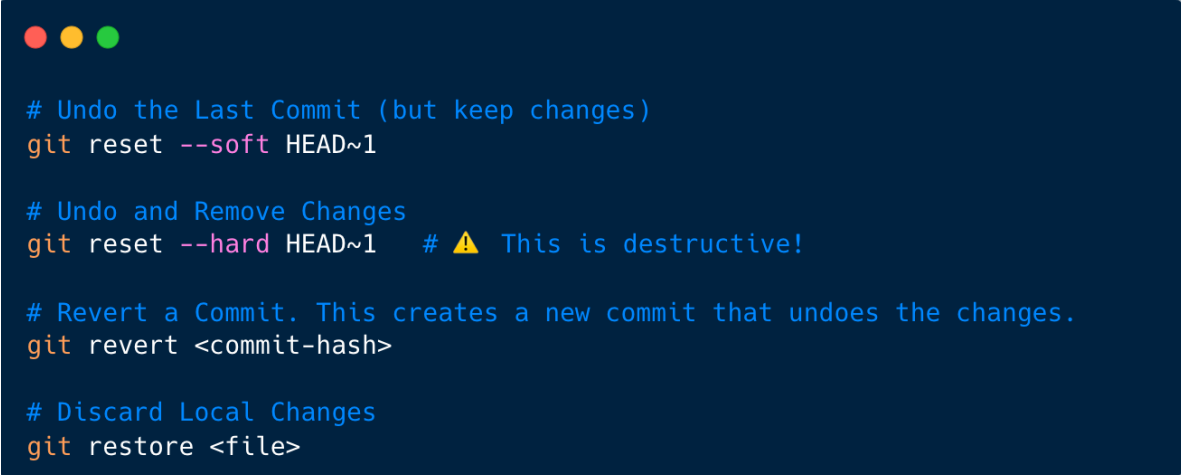
Manually fix the file, and remove the conflict markers, leaving only the code desired, then:



```
git add <file>
git commit
```

Done!

## 11. Undoing Mistakes



```
# Undo the Last Commit (but keep changes)
git reset --soft HEAD~1

# Undo and Remove Changes
git reset --hard HEAD~1 # ⚠ This is destructive!

# Revert a Commit. This creates a new commit that undoes the changes.
git revert <commit-hash>

# Discard Local Changes
git restore <file>
```

## 12. Git Bisect: Finding the Bug Like a Detective

Let's say your app is broken now, but it worked 10 commits ago. Instead of guessing, use:

```
git bisect start
git bisect bad      # current bad commit
git bisect good abc1234 # known good commit
```

Git will now check out a commit halfway between. Run your tests:

- If it works: `git bisect good`
- If it's broken: `git bisect bad`

Repeat until Git finds the exact commit that introduced the bug. Then run:

```
git bisect reset
```

## 13. Tips, Tricks & Best Practices

- Commit often, with clear messages
- Use branches for every new feature or fix
- Always pull before you push
- Use `.gitignore` to avoid committing junk files
- Clean history = clean mind
- Rebase before merging, if you know what you're doing 🤔
- Review others' code with kindness, but don't be afraid to suggest improvements