Mastering Git & GitHub: From Zero to Collaboration

A Comprehensive Hands-on Course

Abderrahman Bouanani && Houcine Gahboub

ENSAA

October 27, 2025

Course Overview

- Understand version control concepts
- Master Git commands and workflows
- Collaborate effectively using GitHub
- Handle real-world scenarios
- Learn best practices and troubleshooting

Course Structure

Learning Objectives

By the end of this session, you will be able to:

- Explain the importance of version control
- Set up and configure Git
- Create and manage repositories
- Track changes with commits

Why Version Control?

Scenario: The "One Small Change" Problem

- Your program is working perfectly.
- You change "just one little thing"...
- Your program breaks.
- You try to change it back...
- Your program is still broken!

What is Version Control?

A system that records changes to files over time

Think of it as a **time machine** for your code!

Key Features

- **History**: See who changed what and when.
- **Backup**: Recover any previous version.
- **Collaboration**: Work with others seamlessly.

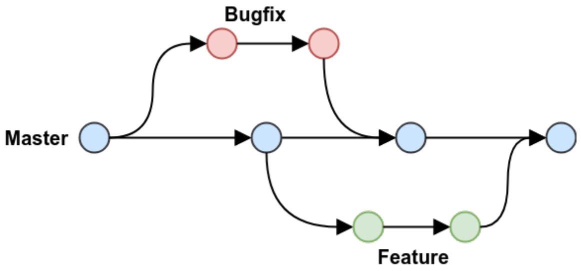
Analogy

Like "Track Changes" in a document, but supercharged for code.

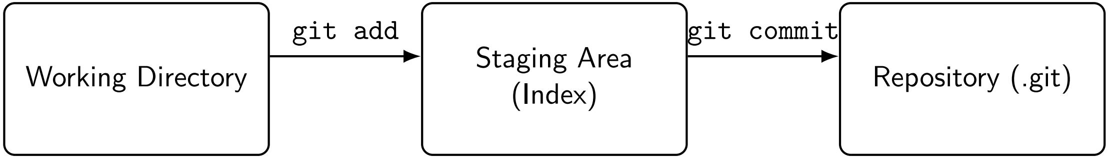
What is Git?

A distributed version control system

- Created by Linus Torvalds in 2005
- Initially developed for Linux kernel development
- Now the most widely used VCS in the world



Core Git Concepts: The Three Areas



- **Working Directory**: Files you actively modify.
- **Staging Area**: Draft of your next commit.
- **Repository**: Full history stored in .git.

Essential Git Commands

Basic Commands

`git init`	Initialize a new repository
`git add`	Stage changes for commit
`git commit`	Save changes to the repository
`git log`	View the commit history
`git status`	Check the status of your files

Useful Options

`git status -s`	Compact status view
`git log --oneline`	Compact log view
`git commit -am`	Add & commit in one step

Working with Branches

Scenario 2: The "Works on My Machine" Problem

- You want to add something new to your project,
- but you're afraid this small change might break everything,
- and you don't want to mess up the version that's already working well.
- Branches to the rescue!

Working with Branches

What are Branches?

Branches are independent lines of development. They let you work on features or fixes without affecting main.

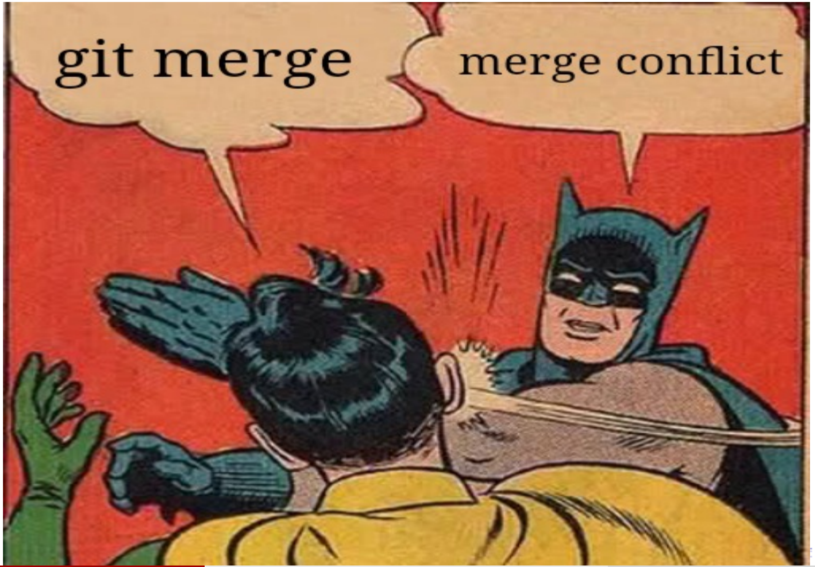
Branch Commands

`git branch`	List all branches
`git switch -c <name>`	Create a new branch
`git switch <name>`	Switch branches
`git merge <name>`	Merge a branch

Branching Strategy

- **main**: Production code
- **develop**: Integration branch
- **feature/***: New features
- **hotfix/***: Urgent fixes

Merge Conflicts



How to Resolve Merge Conflicts

What is a Merge Conflict?

Happens when Git cannot automatically merge because two branches edit the **same line** or one deletes a file the other modified.

Step-by-Step Resolution

1. Open the conflicted file(s)
2. Look for conflict markers:
   - `<<<<<<< HEAD` (your changes)
   - `=======`
   - `>>>>>>> branch-name` (their changes)
3. Edit code to keep desired changes & remove markers
4. Save the file
5. Stage the resolved file: `git add <file>`

Example in a File

```
/* style.css */
.title {
<<<<<<< HEAD
  color: blue;
=======
  color: red;
>>>>>>> feature-new-color
}
```

Key Takeaways

What We've Learned

- Version control is essential for tracking changes
- Git provides a powerful way to manage project history
- Basic workflow: modify $\rightarrow$ stage $\rightarrow$ commit

End of Session 1

Next: Session 2 — Collaboration on GitHub