

Mastering Git & GitHub: From Zero to Collaboration

A Comprehensive Hands-on Course

Abderrahman Bouanani

ENSAA

October 11, 2025

Course Overview

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

Course Structure

- 1 Session 1: Version Control & Git Fundamentals
- 2 Session 2: Advanced Git & GitHub
- 3 Session 3: Collaboration on GitHub
- 4 Conclusion

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
- Navigate Git history



Agenda

- Introduction to Version Control
- Git Installation & Setup
- Basic Git Commands
- Hands-on Exercises

Git Workflow

Working Directory → Staging Area → Repository

Why Version Control?

Scenario 1: The "One Small Change" Problem

- Your program is working
- You change "just one thing"
- Your program breaks
- You change it back
- Your program is still broken!

Why Version Control? (cont.)

Scenario 2: The "Works on My Machine" Problem

- Your program worked well enough yesterday
- You made improvements last night...
- ...but haven't gotten them to work yet
- You need to turn in your program now

Scenario 3: The Integration Problem

- You change one part of a program – it works
- Your co-worker changes another part – it works
- You put them together – it doesn't work
- What changed? Which change broke what?

Scenario 4: The Parallel Development Problem

- You make improvements to a class
- Your co-worker makes different improvements to the same class
- How do you merge these changes without losing work?

What is Version Control?

A system that records changes to files over time

Think of it like a time machine for your code!

Key Features

- **History:** See who changed what and when
- **Backup:** Recover previous versions
- **Collaboration:** Work together without conflicts

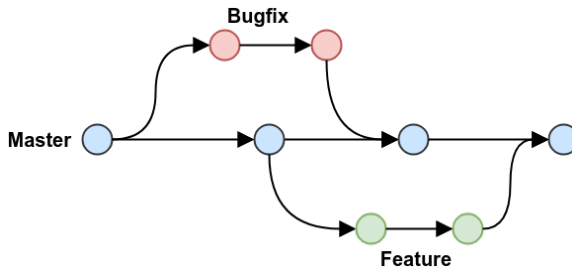
Real-world Example

Like "Track Changes" in Word, but much more powerful!

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



Key Features of Git

Core Capabilities

- Complete project history
- Offline functionality
- Branching and merging
- Data integrity



Why Choose Git?

Key Advantages

- Distributed workflow
- Fast performance
- Strong community support
- Cross-platform compatibility

Fun Fact





The name "Git" is British slang meaning "unpleasant person" - a playful joke by Linus Torvalds!

Industry Standard

Git has become the de facto standard for version control in software development.

Core Git Concepts: The Basics





Key Components

-  **Repository** Your project's folder
-  **Commit** A saved "checkpoint"
-  **Branch** Parallel version
-  **Remote** Cloud version (GitHub)

Video Game Analogy

- Commits = Save points
- Branches = Story paths
- Remote = Cloud save

Workflow

- 1  Make changes
- 2  Stage changes
- 3  Commit changes
- 4  Push to remote

Git Workflow

Understanding Git Workflow

Git's workflow is built around three main areas that help track and manage changes in your project. This structure ensures version control and collaboration efficiency.

The Basic Cycle

- 1 Modify files in working directory
- 2 Stage changes `git add`
- 3 Commit changes `git commit`

Working Directory

Where you make changes to files

Staging Area

Preview of next commit

Essential Git Commands: Getting Started

Repository Setup

`git init` Start new repo
`git clone` Copy existing repo
`git status` Check status

[Space for init/clone diagram]

Example

```
$ git init
$ echo "# My Project" > README.md
$ git add README.md
$ git commit -m "Initial commit"
```


Essential Git Commands: Daily Work

Basic Commands

```
git add Stage changes
git commit Save changes
git log View history
git diff See changes
```

[Space for command examples]

Useful Options

```
git status -s Compact status
git log --oneline Compact log
git commit -am Add & commit
```

[Space for more examples]

Hands-on Exercise

Task: Create Your First Repository

- 1 Create a new directory: `mkdir my-first-repo`
- 2 Navigate into it: `cd my-first-repo`
- 3 Initialize Git: `git init`
- 4 Create a README.md file with your project description
- 5 Stage the file: `git add README.md`
- 6 Commit your changes: `git commit -m "Initial commit"`

Check Your Work

Run `git status` to verify your changes are committed

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 → stage → commit

Next Steps

- Practice basic Git commands
- Explore more Git features
- In the next session: Branching and Merging

Session 2: What You'll Learn

Advanced Git

- Branching and merging
- Undoing changes
- Using .gitignore

[Space for Git workflow diagram]

GitHub Basics

- Creating repositories
- Pushing code
- Basic collaboration

[Space for GitHub screenshot]

What is Branching?

Parallel Universes for Your Code

Branching allows you to create separate lines of development that can evolve independently.

- **Isolation:** Work on features/fixes without affecting the main codebase
- **Experimentation:** Try new ideas safely
- **Collaboration:** Multiple people can work simultaneously
- **Organization:** Keep different work streams separate

Working with Branches

What are Git Branches?

Git branches are independent lines of development that allow you to work on different features, fixes, or experiments without affecting the main codebase. They enable parallel development and help organize work efficiently.

Branch Commands

```
git branch List branches  
git branch <name> Create branch  
git checkout <name> Switch branch  
git merge <name> Merge branch
```

Branching Strategy

- Main/Master - Production code
- Develop - Integration branch
- Feature/* - New features
- Hotfix/* - Quick fixes

Advanced Git: Undoing Changes

What is Undoing Changes?

Git provides powerful tools to undo changes at different stages of your workflow. Understanding these tools helps you recover from mistakes and maintain a clean project history.

Undo Commands

`git restore <file>` Discard changes
`git revert <commit>` Create undo commit
`git reset` Move branch pointer
`git clean` Remove untracked files

Be Careful!

Some commands modify history.

- Use revert for shared branches
- Use reset carefully on local branches

What is GitHub?

- Cloud-based Git hosting
- Collaboration platform
- Open source community hub
- Professional portfolio

[GitHub features]

Key Features

- Code hosting
- Issue tracking
- Pull requests
- GitHub Actions (CI/CD)
- GitHub Pages

Git + GitHub Workflow

Basic Commands

`git remote add origin <url>` Connect to GitHub

`git push -u origin main` First push

`git clone <url>` Get existing repo

`git pull` Get latest changes

[Workflow diagram: Local ↔ GitHub]

Best Practice

Always pull before you push to avoid conflicts!

Hands-on: First GitHub Repository

Steps

- 1 Create GitHub account
- 2 New repository
- 3 Connect local repo
- 4 Push your code

[Screenshot: GitHub new repo]

[Terminal: git commands]

Remember

Don't forget to add a meaningful README.md!

Session 3: Team Collaboration

What We'll Cover

- Forking repositories
- Pull requests
- Code reviews
- Team workflows

[Collaboration diagram]

Key Benefits

- Work in parallel
- Review changes
- Maintain code quality
- Document decisions

[Team workflow]

Forking Workflow

Step 1: Fork

- 1 Find a repository on GitHub
- 2 Click "Fork" button
- 3 Creates your copy

Step 2: Clone

```
git clone <your-fork-url>
```

[Fork button screenshot]

[Repository structure]

Pull Requests

What is a Pull Request?

A way to propose and review changes before merging them into the main project.

Creating a PR

- 1 Push changes to your fork
- 2 Click "New Pull Request"
- 3 Add description
- 4 Request reviews

[PR creation flow]

[Good PR example]

Code Reviews

Reviewing Code

- Check for bugs and edge cases
- Verify code style and best practices
- Suggest improvements
- Ensure tests are present

Example Review

- "Consider adding error handling here"
- "This could be more efficient if..."
- "Don't forget to update the docs"

Best Practices

- Be constructive and kind
- Explain the 'why' behind suggestions
- Keep reviews focused and actionable
- Review in a timely manner

[Code review interface]

GitHub Features for Teams

Project Management

Issues Track bugs and features

Projects Kanban-style boards

Discussions Team conversations

Wiki Project documentation

Automation

- GitHub Actions (CI/CD)
- Branch protection rules
- Auto-merge PRs
- Code scanning

[GitHub features overview]

Team Workflow

- 1 Create issue
- 2 Branch off main
- 3 Make changes
- 4 Open PR
- 5 Review and discuss
- 6 Merge and deploy

[Workflow visualization]

GitHub Features Overview

Feature	Description
Issues	Track bugs and features
Pull Requests	Suggest and review code changes
Actions	Automate tasks (CI/CD)
Wiki	Project documentation
Projects	Kanban-style task management
Discussions	Team conversations
Security	Vulnerability alerts and Dependabot

Benefits

- Build a public portfolio
- Learn from other developers
- Contribute to tools you use
- Improve coding skills

Best Practices

- Use branches and PRs to avoid conflicts
- Communicate through Issues and Discussions
- Follow the project's contribution guidelines
- Keep changes focused and well-documented

Practice Idea

- Work in pairs
- One creates a repository, the other forks it
- Collaborate using pull requests and issues

Conclusion

- You learned what version control is
- You mastered Git basics and advanced commands
- You discovered GitHub collaboration tools

Next Steps:

- Practice Git commands every day
- Build your portfolio on GitHub
- Contribute to open-source projects

Thank You!

Questions or feedback?