Git Basics - Version Control Fundamentals

What is Git? 🤔

Git is a **distributed version control system** that tracks changes in your code over time.

Why Use Git?

✅ Benefits

- 📝 Track every change
- 🔄 Work with teams seamlessly
- 🌿 Experiment safely with branches
- ⏮️ Undo mistakes easily
- 🌍 Collaborate from anywhere

💡 Real-World Use

- Software development
- Documentation management
- Configuration tracking
- Design file versioning
- Any text-based project

Git was created by Linus Torvalds in 2005 for Linux kernel development

Core Concepts 📚

Understanding Git's building blocks

Repository (Repo)

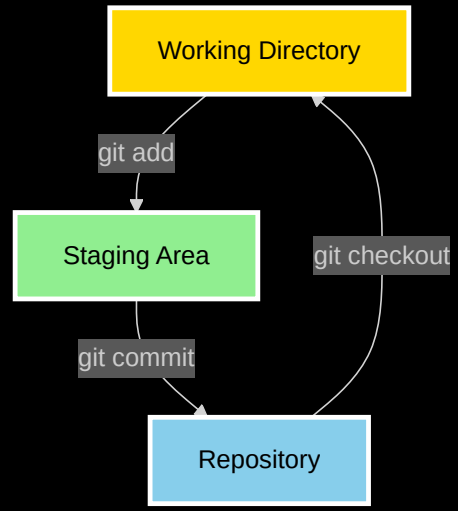
A folder tracked by Git containing your project and its history.

Working Directory

Your current project files where you make changes.

Staging Area

A preparation area for your next commit.



The three states of Git

Core Concepts - Commands 💻

Now let's see the commands for each concept

Initialize Repository

```
# Create a new Git repository
git init
```

Creates a `.git` folder to track changes

Stage Files

```
# Stage a specific file
git add filename.txt

# Stage all changes
git add .
```

Prepares files for the next commit

Check Status

```
# See current state
git status
```

Shows tracked/untracked files and changes

View Changes

```
# See unstaged changes
git diff
```

Compare working directory with staging

Commits - The Heart of Git ❤️

A commit is a **snapshot** of your project at a specific point in time.

Anatomy of a Commit

```
commit a1b2c3d4e5f6789...
Author: Jane Doe <jane@example.com>
Date:   Mon Sep 30 2025

    Add authentication

    - Login/logout
    - Password hashing
```

Creating Commits

```
git add auth.js login.html
git commit -m "Add login"
```

Best Practices

- ✅ Atomic & focused
- ✅ Clear messages
- ✅ Present tense

Basic Git Workflow 🔄

The everyday cycle of working with Git

```
# Step 1: Check current status
git status

# Step 2: Make changes (edit files in your editor)
echo "Hello Git" > readme.txt

# Step 3: See what changed
git status
git diff

# Step 4: Stage your changes
git add readme.txt  # Or: git add .

# Step 5: Commit your changes
git commit -m "Add readme file with introduction"

# Step 6: View commit history
git log --oneline --graph
```

💡 **Tip:** Commit often, push when stable!

Branching 🌿

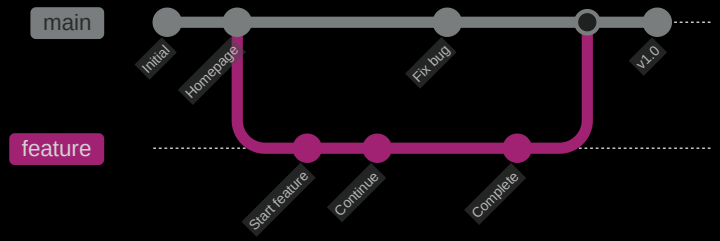
The Superpower of Git

Branches let you:

- 🔬 Experiment without breaking main code
- 👥 Work on multiple features simultaneously
- 🎯 Isolate bug fixes from new features
- 🤝 Collaborate without conflicts

Understanding Branches

Think of branches as parallel universes for your code



Branch Commands

```
# Create a new branch
git branch feature-login

# Switch to a branch
git checkout feature-login

# Create and switch (shortcut)
git checkout -b feature-login
```

```
# List all branches
git branch

# List with details
git branch -v

# Delete a branch
git branch -d feature-login
```

Merging Branches 🔀

Combining changes from different branches

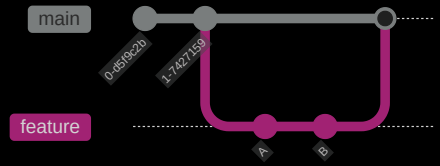
Merge Example

```
# Switch to the branch you want to merge INTO
git checkout main

# Merge the feature branch
git merge feature-login
```

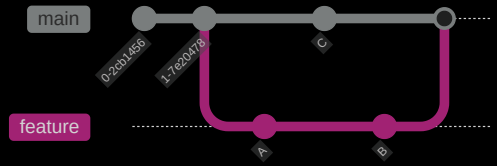
Fast-Forward Merge

No divergent changes



Three-Way Merge

Divergent changes



Merge Conflicts 💥

When Git can't automatically merge changes

What Causes Conflicts?

Two branches modify the same lines in the same file

Example Conflict

```
// <<<<<<< HEAD
function greet() {
  console.log("Hello!");
}
// =======
function greet() {
  console.log("Hi there!");
}
// >>>>>>> feature
```

Conflict markers (<<<, ===, >>>) shown with //

Resolving Conflicts

1. Open the conflicted file
2. Choose which changes to keep
3. Remove conflict markers
4. Stage and commit

```
git add resolved-file.js
git commit -m "Resolve conflict"
```

Interactive Example: Git States 🎨

Watch how files move through Git states:

📄 New File

Working Directory git add →

✅ File is now safely stored in Git history!

Remote Repositories 🌍

Collaborating with others

What is a Remote?

A repository hosted on a remote server like:

- **GitHub** 🌟
- **GitLab** 🦊
- **Bitbucket** 🪣

Enables team collaboration and backup

Essential Commands

Clone repository:

```
git clone <url>
```

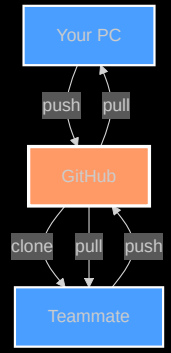
Add remote:

```
git remote add origin <url>
```

Push/Pull:

```
git push origin main
git pull origin main
```

How It Works



Share code via remote

Essential Git Commands 🛠️

Your daily toolkit

Status & Info

```
git status
git log --oneline --graph --all
git diff
git show <commit-hash>
```

Branching

```
git branch <name>
git checkout <name>
git checkout -b <name>
git branch -d <name>
```

Staging & Committing

```
git add <file>
git add .
git commit -m "message"
git commit --amend
```

Remote Operations

```
git push origin <branch>
git pull origin <branch>
git fetch origin
git clone <url>
```

Undoing Changes ⏮️

Mistakes happen - Git has your back!

Undo Working Directory

```
# Discard changes
git checkout -- filename.txt
git checkout -- .
```

Unstage Files

```
# Unstage a file
git reset HEAD filename.txt
# Newer syntax
git restore --staged filename.txt
```

Undo Commits

```
# Keep changes
git reset --soft HEAD~1
# Discard changes
git reset --hard HEAD~1
# Revert safely
git revert <commit-hash>
```

Pro Tip 💡

Use `git reflog` to see all actions and recover from mistakes!

Git Best Practices 🏆

Tips from professional developers

1. Commit Messages

```
# ✅ Good
git commit -m "Fix login bug on mobile"
# ❌ Bad
git commit -m "fixed stuff"
```

- Use present tense - Be specific and descriptive

2. Commit Frequency

- Commit early, commit often
- Each commit = one logical change
- Don’t commit broken code

3. Branching

- Keep main/master stable
- Create branches for features
- Use descriptive names

```
feature/user-authentication
bugfix/header-alignment
```

4. .gitignore

Ignore files that shouldn’t be tracked

```
node_modules/
.env
*.log
dist/
```

Practical Example: Feature Development 💻

1-3. Start Feature

```
git status
git checkout -b feature/user-profile
# Edit profile.js, profile.css, profile.html
```

4-6. Stage & Commit

```
git status
git add profile.js profile.css profile.html
git commit -m "Add user profile page"
```

7-8. Continue & Push

```
git commit -m "Add picture upload"
git commit -m "Add edit functionality"
git push -u origin feature/user-profile
```

9-11. Merge & Cleanup

```
git checkout main
git pull origin main
git merge feature/user-profile
git push origin main
git branch -d feature/user-profile
```

Git Cheat Sheet 📋

Setup

```
git config --global user.name "Your Name"
git config --global user.email "email@example.com"
git init
git clone <url>
```

Daily Workflow

```
git status
git add <file>
git commit -m "message"
git push
git pull
```

Branching

```
git branch
git checkout -b <branch>
git merge <branch>
git branch -d <branch>
```

Inspection

```
git log
git log --oneline --graph
git diff
git show <commit>
```

Undo

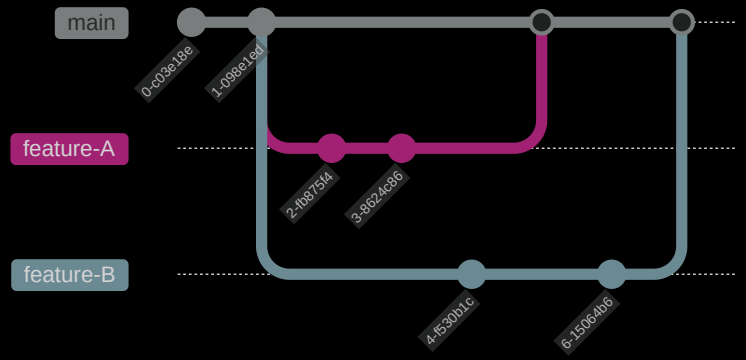
```
git checkout -- <file>
git reset HEAD <file>
git revert <commit>
git reset --hard HEAD~1
```

Remote

```
git remote -v
git remote add origin <url>
git fetch
git pull
git push origin <branch>
```

Common Git Workflows 👥

How teams use Git in the real world



Steps:

1. Create feature branch
2. Work on feature
3. Push to remote
4. Open Pull Request
5. Code review

Benefits:

- ✅ Isolated development
- ✅ Easy code review
- ✅ Safe experimentation
- ✅ Clear history

Pull Requests (PRs) 🔍

Code review and collaboration

What is a Pull Request?

A request to merge your changes into another branch, with built-in code review.

Creating a PR

1. Push your branch

```
git push origin feature-login
```

2. Go to GitHub/GitLab 3. Click "New Pull Request" 4. Select branches 5. Add description 6. Request reviewers

PR Best Practices

- Write clear descriptions
- Keep PRs small and focused
- Link to related issues
- Respond to feedback promptly
- Update branch with main regularly
- Ensure CI passes

💡 **Tip:** Use draft PRs for early feedback

Helpful Resources 📚

Continue your Git journey

📖 Documentation

- Official Git Docs
- GitHub Guides
- Atlassian Git Tutorial

📺 Videos

- Git and GitHub for Beginners (freeCodeCamp)
- Git Tutorial for Beginners (Programming with Mosh)

📄 Cheat Sheets

- GitHub Git Cheat Sheet
- GitLab Git Cheat Sheet

🎮 Interactive Learning

- Learn Git Branching
- Git Immersion

🎓 Remember: Everyone struggles with Git at first!

The key is consistent practice and not being afraid to make mistakes. Git has your back!