BY LEARNSTACK

2026 Edition

Git & GitHub Complete Handbook

From Zero to Professional - Version Control, Collaboration & DevOps

Git Basics | Branching | GitHub Actions | Pull Requests | CI/CD | Open Source | Team Workflows

Learn. Build. Grow.

linktr.ee/LearnStack

Developer Mastery Series

About This Handbook

This handbook is a complete, beginner-friendly guide to Git and GitHub. It starts with the reason version control exists, then builds toward professional workflows such as pull requests, code review, GitHub Actions, CI/CD, open-source contribution, security, and team-level DevOps practices.

It is written for beginners with zero Git knowledge, developers who want to work professionally, team leads who want a shared workflow vocabulary, and computer science students preparing for internships, placements, and developer interviews.

INFO: What You Will Learn

You will learn how Git thinks, how to use commands safely, how GitHub collaboration works, how production teams review and ship code, and how to answer Git and GitHub questions in interviews.

TIP: How to Use This PDF

Read the first three chapters slowly. They build the mental model. Then practice commands inside a disposable test repository before using them on a real project.

Learning Path

ARCHITECTURE: Beginner -> Intermediate -> Advanced -> Professional

Beginner: init, add, commit, status
Intermediate: branch, merge, rebase, stash
Advanced: reflog, bisect, worktree, hooks
Professional: PRs, reviews, Actions, CI/CD, security

Use this handbook linearly the first time. Later, keep the command reference and interview section nearby as a daily developer reference.

Table of Contents

INFO: Premium Learning Roadmap

The page numbers below point to chapter openers. Subsections are listed so you can quickly jump to the exact skill you want to practice.

01 Introduction to Version Control & Git 6

What is Version Control? 6
Types of VCS 6
What is Git? 6
Snapshot Model 7
Why Git won 7

02 Installing & Configuring Git 10

Install Git 10
First-Time Configuration 10
Config Levels 10
Tools 11
First Repository 11

03 Git Core Concepts (The Mental Model) 14

Three Trees 14
Commits 14
.git Directory 14
.gitignore 15
HEAD 15
Object Model 15

04 Essential Git Commands 19

init 19
clone 19
status 19
add 20
commit 20
log 20
diff 21
show 21
rm/mv 21
restore/reset 22
stash 22
tag 22
clean 22

05 Branching & Merging 27

Branches 27
Branch Commands 27
Merging 27
Conflicts 28
Rebasing 28
Strategies 28

Table of Contents (continued)

INFO: Premium Learning Roadmap

The page numbers below point to chapter openers. Subsections are listed so you can quickly jump to the exact skill you want to practice.

06 Introduction to GitHub 33

What is GitHub? 33
Account Setup 33
Repositories 33
Remote Operations 34
Repo Features 34
Profile README 34

08 Advanced Git Techniques 40

cherry-pick 40
bisect 40
reflog 40
worktree 41
submodules 41
blame 41
hooks 42
log tricks 42
signing 42

10 Security, SSH & Authentication 50

HTTPS vs SSH 50
SSH Keys 50
PATs 50
Secret Scanning 51
Credential Manager 51
2FA 51
.gitignore Security 52

07 Collaborating on GitHub 36

Pull Requests 36
Code Review 36
Forking 36
Issues 37
Discussions 37
Teams 37
Branch Protection 38

09 GitHub Actions & CI/CD 45

CI/CD 45
Actions Concepts 45
First Workflow 45
Python Workflow 46
Triggers 46
Secrets 46
Deployments 47
Caching 47
Matrix Builds 47
Marketplace 48

Table of Contents (continued)

> **INFO: Premium Learning Roadmap**
>
> The page numbers below point to chapter openers. Subsections are listed so you can quickly jump to the exact skill you want to practice.

11 Open Source Contribution 53

Why Contribute 53
Find Projects 53
Contribution Workflow 53
Licenses 54
CONTRIBUTING.md 54
Hacktoberfest 54
Your Project 55

12 Git in Professional Teams & DevOps 56

Team Workflows 56
Conventional Commits 56
SemVer 56
DevOps Pipelines 57
Monorepos 57
Review Culture 57
Aliases 58
Performance 58

13 Interview Preparation (Git & GitHub) 59

Top 40 Q&A 59
Scenario Questions 59
Interview Tips 59

14 Quick Reference Cheat Sheet 67

Setup 67
Staging 67
Branching 67
Remote 68
Undoing 68
Inspection 68
Stashing 69
Tags 69
GitHub CLI 69
Aliases 70

15 Resources & Further Learning 71

Official Docs 71
Video Channels 71
Interactive Learning 71
Books 72
Tools 72
LearnStack CTA 72

01

LearnStack Developer Mastery Series | Learn. Build. Grow.

1.1 What is Version Control?

> **INFO: Definition**
>
> Version Control System (VCS) is software that records changes to files over time so you can view history, restore previous versions, compare changes, and collaborate without overwriting each other.

The simplest way to understand version control is to imagine a project folder that remembers every important checkpoint. When you break something, you do not panic because you can go back. When two people change the same file, the VCS helps combine the work safely.

- Problem it solves: What if I break everything?
- Problem it solves: Who changed this line and why?
- Problem it solves: How do two developers work on the same project at once?
- Problem it solves: How do we safely release and roll back production code?

A short history: SCCS appeared in the 1970s, CVS made network collaboration common, SVN improved centralized workflows, and Git arrived in 2005 as a fast distributed system built for the Linux kernel.

> **REAL WORLD: Why companies care**
>
> Every professional software team needs a trustworthy history of code changes. Without version control, releases become risky, debugging becomes guesswork, and collaboration becomes chaos.

1.2 Types of Version Control Systems

Feature	Local VCS	Centralized VCS (SVN)	Distributed VCS (Git)
Where history lives	Only on your machine	On one central server	On every developer's machine
Network needed	No	Often yes	Only for sharing
If server dies	No shared server	Team can be blocked	Every clone is a backup
Branching	Limited	Usually heavy	Lightweight and fast
Best for	Solo experiments	Older enterprise workflows	Modern software teams

1.3 What is Git?

INFO: Git in one sentence

Git is a free, open-source distributed version control system designed for speed, data integrity, and non-linear workflows such as branching and merging.

Git was created by Linus Torvalds in 2005 after the Linux kernel project needed a powerful replacement for its previous version control arrangement. The design goal was simple: make branching cheap, make history reliable, and make local operations extremely fast.

Git	GitHub
A version control tool installed on your computer.	A cloud platform that hosts Git repositories and adds collaboration features.
Works offline.	Requires an account and internet for collaboration.
Commands include git add, git commit, git branch.	Features include pull requests, issues, Actions, and organizations.

REAL WORLD: Professional reality

Google, Microsoft, Netflix, Spotify, startups, agencies, open-source projects, and almost every modern engineering organization use Git-based workflows. Your future workplace almost certainly expects Git fluency.

1.4 How Git Stores Data - The Snapshot Model

INFO: Snapshots, not simple diffs

Git stores project snapshots. A commit represents the state of your project at a point in time. Internally Git is efficient and reuses unchanged objects, but the mental model is a series of snapshots.

ARCHITECTURE: Commit timeline with snapshots

Commit A -> Commit B -> Commit C -> Commit D
| | | |
Snapshot Snapshot Snapshot Snapshot
of files of files of files of files

Each commit points to a full project tree. If a file did not change, Git can reuse the same object, which keeps storage efficient.

The three states

ARCHITECTURE: Working Directory -> Staging Area -> Repository

Working Directory -- git add --> Staging Area -- git commit --> Repository
(your files) (index) (.git history)

Modified means changed in your working folder. Staged means selected for the next commit. Committed means safely recorded in Git history.

State	Meaning	Command that moves it forward
Modified	A tracked file changed in the working directory.	git add
Staged	The change is selected for the next commit.	git commit
Committed	The snapshot is stored in the repository.	git push to share it remotely

1.5 Why Git Became the Industry Standard

- Speed: most operations are local and do not need the network.
- Branching: branches are lightweight pointers, so teams can experiment safely.
- Distributed design: every clone contains complete project history.
- Data integrity: Git identifies content using hashes and detects corruption.
- Ecosystem: GitHub, GitLab, Bitbucket, CI/CD tools, and IDEs all integrate deeply with Git.

TIP: Developer habit

Even if you never collaborate with anyone, Git is still one of the most important developer habits you can build. A clean commit history becomes your personal undo system, learning journal, and proof of work.

Practice Lab - Create a disposable Git learning folder

BASH

```
mkdir git-learning
cd git-learning
git init
echo "hello git" > notes.txt
git status
```

$ Output:

```
Initialized empty Git repository in .../git-learning/.git/
On branch main
Untracked files:
notes.txt
```

TIP: Lab Habit

Type the commands yourself instead of copying them. The muscle memory matters. After each command, run git status so Git tells you exactly what changed.

TIP: Professional Checklist

Use this checklist before you move on. In real teams, these small habits prevent large mistakes.

- You can explain Git vs GitHub without mixing them up.
- You understand Working Directory, Staging Area, and Repository.
- You know that a commit is a project snapshot.
- You know why teams use branches for safe experimentation.

02

LearnStack Developer Mastery Series | Learn. Build. Grow.

REAL WORLD: Setup is part of professionalism

A developer who can install Git, configure identity, connect an editor, and verify the setup can start contributing to a project immediately. Poor setup causes confusing commits, missing author names, and failed pushes.

2.1 Installing Git

Windows

BASH

```
# Download Git for Windows
# https://git-scm.com/download/win
# Run installer - use recommended settings
git --version
```

$ Output:

```
git version 2.x.x.windows.x
```

CAUTION: Windows terminal choice

On Windows, use Git Bash, PowerShell, or Windows Terminal. Plain CMD can work, but Git Bash gives a more Unix-like experience that matches most tutorials and professional workflows.

macOS

BASH

```
# Option 1: Xcode Command Line Tools
xcode-select --install
# Option 2: Homebrew (recommended if you use Homebrew)
brew install git
git --version
```

$ Output:

```
git version 2.x.x
```

Linux - Ubuntu / Debian

BASH

```
sudo apt update
sudo apt install git -y
git --version
```

$ Output:

```
git version 2.x.x
```

Linux - Fedora / RHEL

```bash
sudo dnf install git -y
git --version
```

$ Output:

```
git version 2.x.x
```

2.2 First-Time Git Configuration

> **INFO: Identity matters**
>
> Git stores your name and email inside every commit. Set them before your first real commit. They become part of the permanent project history and are visible to teammates.

```bash
# Set your identity (required)
git config --global user.name "Your Full Name"
git config --global user.email "you@example.com"
# Set your default editor
git config --global core.editor "code --wait"
# Set default branch name to main
git config --global init.defaultBranch main
# Verify your configuration
git config --list
```

$ Output:

```
user.name=Your Full Name
user.email=you@example.com
core.editor=code --wait
init.defaultbranch=main
```

> **TIP: GitHub contribution graph**
>
> Use the same email as your GitHub account. This links your local commits to your GitHub profile contribution graph and verified authorship.

2.3 Configuration Levels

Level	Flag	Typical file	Scope
System	--system	/etc/gitconfig	All users on the machine
Global	--global	~/.gitconfig	Current operating-system user
Local	--local	.git/config	Only the current repository

Most developers use global configuration for name, email, editor, aliases, and default branch. Use local configuration when a specific repository needs a different email, signing key, or remote behavior.

2.4 Recommended IDEs & Git GUI Tools

Tool	Type	Platform	Free?	Best for
VS Code + GitLens	IDE + Git UI	All	Yes	Beginners and professionals
GitHub Desktop	GUI client	Windows/macOS	Yes	Beginners who want visual commits
GitKraken	GUI client	All	Freemium	Visual branch workflows
Sourcetree	GUI client	Windows/macOS	Yes	Atlassian users
Terminal / Bash	CLI	All	Yes	Professional speed and automation

2.5 Your First Git Repository

BASH

```
# Create a new project and initialize Git
mkdir my-first-project
cd my-first-project
git init
# Create a file
echo "# My First Project" > README.md
# Stage the file
git add README.md
# Make your first commit
git commit -m "Initial commit: add README"
# Verify
git log --oneline
```

$ Output:

```
Initialized empty Git repository in .../my-first-project/.git/
[main (root-commit) a1b2c3d] Initial commit: add README
a1b2c3d Initial commit: add README
```

TIP: Repository origin

Every repository starts with either git init or git clone. Everything else - staging, committing, branching, pushing, and pulling - builds on that foundation.

2.6 Common Setup Problems

Problem	Likely cause	Fix
git is not recognized	Git executable not in PATH	Reinstall Git and enable command-line tools
Wrong author on commits	user.name or user.email misconfigured	Set global or local identity correctly

LearnStack Free Preview

This was a free preview. Get the full book on LearnStack.

Visit: https://www.learnstack.co.in

Digital PDF delivery is handled through Gumroad email after purchase.