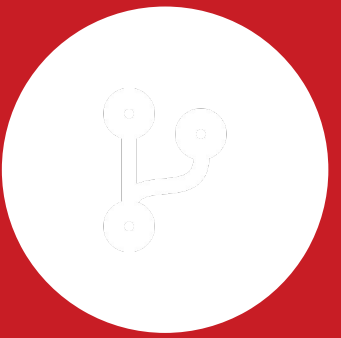
HANDS-ON WORKSHOP • BEGINNER TO ADVANCED

GIT & GITHUB

Mastering Version Control & Collaborative Development

From your first commit to production deployment — a complete, practical training course.



`git`

20 SLIDES · 30+ COMMANDS · 1 CAPSTONE PROJECT

Why Version Control? Understanding VCS

Problems Version Control Solves

- No history of who changed what, or when
- Multiple people overwrite each other's work
- Impossible to safely undo a bad change
- No reliable way to back up or share code

Types of Version Control Systems

Type	How it works	Examples
Local VCS	Versions stored only on one machine, in a local database	RCS
Centralized (CVCS)	Single central server holds history; clients check out files	SVN, CVS, Perforce
Distributed (DVCS)	Every clone is a full repository with complete history	Git, Mercurial

Git is a Distributed VCS: every developer has the full project history locally, enabling offline work, fast branching, and no single point of failure.

Installing & Configuring Git

Windows

Download Git for Windows from git-scm.com, run the installer, keep default options (Git Bash included).

Linux (Debian/Ubuntu)

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

macOS

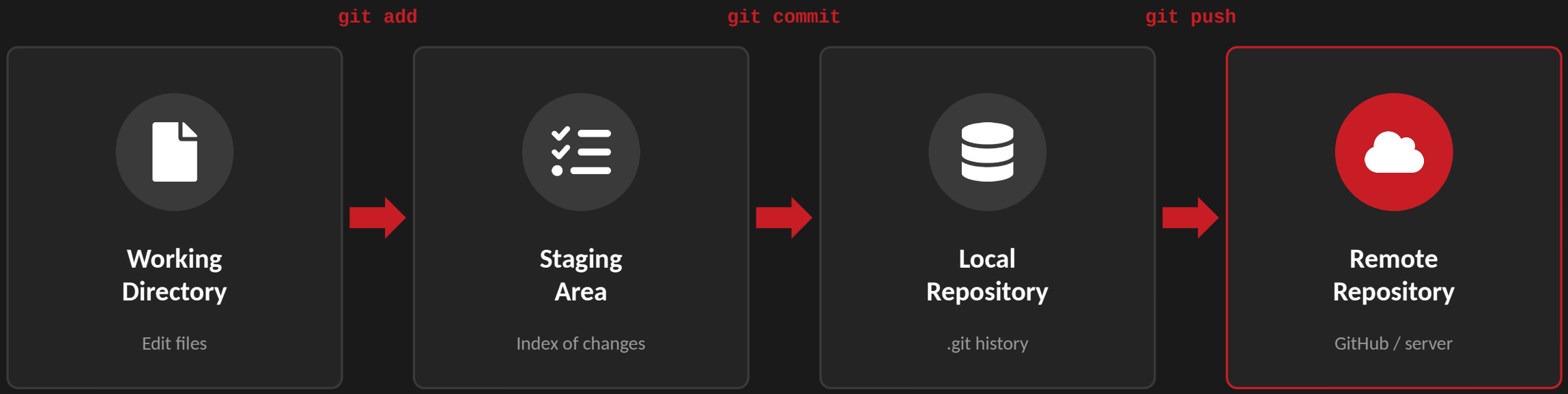
```
brew install git
# or: xcode-select --install
```

Essential Configuration & Verification Commands

Command	Purpose
git --version	Verify Git installed correctly
git config --global user.name "Name"	Set your commit author name
git config --global user.email "you@x.com"	Set your commit author email
git config --global core.editor "code --wait"	Set default editor
git config --list	View all active configuration

Git's Internal Workflow

How a change travels from your editor to a shared remote



git restore / git reset move changes backward through these stages — covered next.

Getting Started: Your First Commands

Command	Description	Example
git init	Creates a new local Git repository in the current folder	git init
git clone <url>	Copies an existing remote repository to your machine	git clone https://github.com/user/repo.git
git status	Shows staged, unstaged, and untracked file changes	git status
git add <file>	Moves changes into the staging area	git add index.html | git add .
git commit -m "msg"	Saves staged changes to local history with a message	git commit -m "Add homepage layout"
git log	Displays commit history	git log --oneline --graph
git diff	Shows line-by-line differences between states	git diff HEAD~1 HEAD

Undoing Changes Safely

Command	Description	Example
git restore <file>	Discards unstaged changes in the working directory	git restore app.js
git restore --staged <f>	Unstages a file, keeping its edits	git restore --staged app.js
git reset <mode>	Moves HEAD/branch pointer; --soft keeps changes staged, --mixed unstages, --hard discards	git reset --hard HEAD~1
git revert <commit>	Creates a new commit that undoes a prior commit — safe for shared history	git revert a1b2c3d
git rm <file>	Removes a file from working dir and staging area	git rm old.txt
git mv <old> <new>	Renames/moves a tracked file	git mv app.js server.js

Best practice: use git revert on commits already pushed/shared; reserve git reset --hard for local, unpublished work only.

Branching & Merging

Command	Description
git branch <name>	Creates a new branch
git checkout <branch>	Switches to an existing branch (legacy)
git switch <branch>	Switches branches (modern, safer replacement)
git merge <branch>	Combines another branch's history into the current one
git rebase <branch>	Replays commits on top of another branch for a linear history

Feature Branch Flow

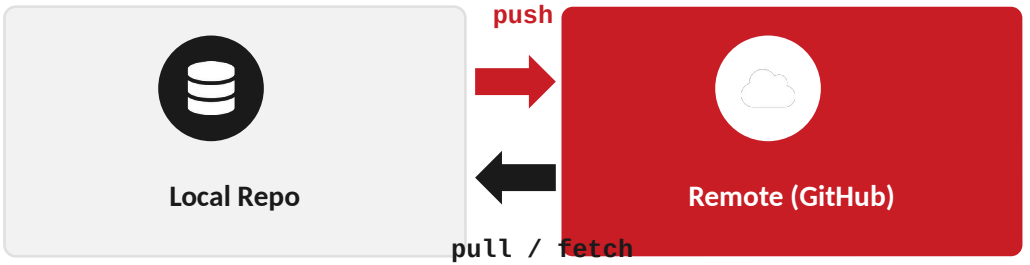

```
git checkout -b feature/login
git switch main
git merge feature/login
```

Advanced & Everyday-Useful Commands

Command	Description	Example
git stash	Temporarily shelves uncommitted changes	git stash / git stash pop
git cherry-pick <c>	Applies a specific commit from another branch	git cherry-pick a1b2c3d
git tag <name>	Marks a specific commit, often for releases	git tag v1.0.0
git blame <file>	Shows who last modified each line	git blame index.html
git show <commit>	Displays details and diff of one commit	git show a1b2c3d
git reflog	Logs every HEAD movement — recovers 'lost' commits	git reflog
git clean -fd	Removes untracked files and directories	git clean -fd

Working with Remotes

Command	Description
`git remote -v`	Lists configured remotes and their URLs
`git remote add origin <url>`	Links a local repo to a remote
`git fetch`	Downloads remote commits without merging them
`git pull`	Fetches AND merges remote changes into current branch
`git push`	Uploads local commits to the remote repository



fetch vs pull:

git fetch downloads changes into a hidden remote-tracking branch (origin/main) so you can review before merging. git pull does fetch + merge in one step — faster, but riskier on shared branches.

Introducing GitHub

GitHub hosts Git repositories in the cloud and adds collaboration tools on top.

Repositories

Hosted project + full history

Commits

Saved snapshots in the cloud

Branches

Parallel lines of development

Pull Requests

Propose & review code changes

Forks

Your own copy of someone's repo

Issues

Track bugs & feature requests

Releases

Packaged, versioned snapshots

Stars

Bookmark / endorse a repo

Watchers

Get notified of repo activity

Discussions

Q&A and community conversation

GitHub Desktop & GitHub CLI

GitHub Desktop

- Visual GUI client for Git & GitHub
- Drag-and-drop staging, visual diff viewer
- One-click branch creation & PR creation
- Great for beginners and non-CLI workflows
- Built-in conflict resolution helper

GitHub CLI (gh)

```
gh auth login
gh repo create my-app --public
gh repo clone user/repo
gh pr create --fill
gh pr view --web
gh issue list
```

Manage repos, issues, and pull requests entirely from the terminal — ideal for scripting and CI automation.

Connecting a Local Repo to GitHub

1. Create a free account at github.com and verify your email.
2. Click New Repository → name it → choose Public/Private → Create.
3. Clone it locally: git clone https://github.com/user/repo.git
4. Or link an existing folder: git remote add origin <url>
5. Push your first commit: git push -u origin main
6. Collaborate: teammates git pull to receive your changes.

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

Pull Requests & Resolving Merge Conflicts

Collaborating with Pull Requests

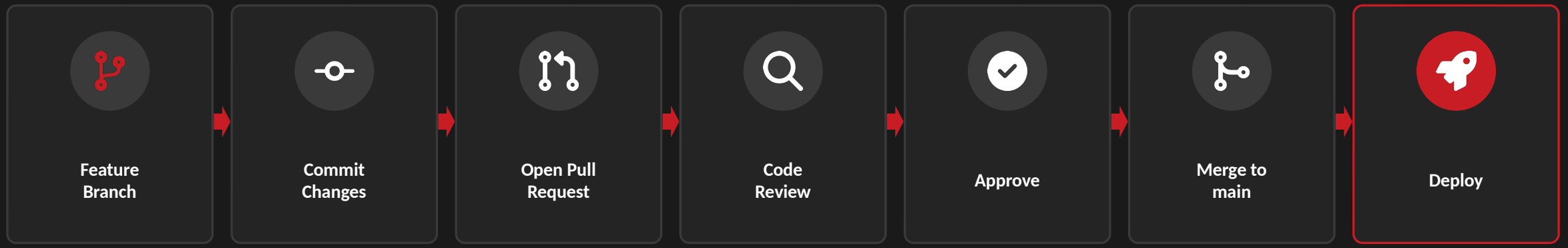
1. Create a feature branch
2. Push branch & open a Pull Request
3. Team reviews & comments on code
4. Address feedback with new commits
5. PR is approved & merged into main

Resolving a Merge Conflict

```
<<<<<<< HEAD
const title = "Home";
=======
const title = "Dashboard";
>>>>>>> feature/rename
```

1. Open the conflicting file(s)
2. Edit to keep the correct code, remove markers
3. git add <file> → git commit
4. Push — the PR updates automatically

The Complete Developer Workflow



Feature branching → commits → pull request → review & approval → merge → automated deployment.

Authentication: Personal Access Tokens

Setup: Settings → Developer settings → enable Developer Mode → Personal access tokens

Classic Token

- One token, broad scopes (repo, workflow, admin:org)
- Simple to create; harder to audit/limit blast radius
- Use for quick scripts & legacy tool compatibility

Fine-Grained Token

- Scoped to specific repositories only
- Granular permissions per resource (Contents, Issues, PRs)
- Recommended for production & least-privilege access

Task	Command
Use token with HTTPS	git push https://<TOKEN>@github.com/user/repo.git
Store credentials	git config --global credential.helper store

Security best practices: never commit tokens to code, set the shortest useful expiration, grant the minimum scopes needed, and rotate/revoke tokens regularly.

Webhooks: Event-Driven Automation

A webhook is an HTTP POST that GitHub sends to a URL you configure whenever a chosen event occurs in your repository — powering CI/CD, notifications, and bots.

Common Events

- push — code pushed to a branch
- pull_request — opened, updated, merged
- release — a new release is published
- issues — opened, closed, labeled

Security

- Every payload is signed with a webhook secret
- Verify the X-Hub-Signature-256 HMAC on receipt
- Always receive over HTTPS
- Reject payloads that fail verification

Build & Deploy a Website with GitHub Pages

Build a simple site: index.html, style.css, script.js

git init → git add . → git commit -m "Initial site"

Create a GitHub repo and push: git remote add origin … && git push -u origin main

Settings → Pages → Source: main branch / root → Save

Visit the live URL: https://username.github.io/repo-name

Edit, commit, push → Pages auto-redeploys within a minute

Advanced (optional): *auto-deploy with GitHub Actions — add .github/workflows/deploy.yml so every push to main triggers build & publish, no manual step needed.*

Project Hygiene & the GitHub Ecosystem

.gitignore

Exclude build artifacts, secrets, node_modules from version control

README & Markdown

Document setup, usage, and badges using Markdown syntax

Semantic Commits

feat:, fix:, docs:, chore: prefixes for readable history

Branching Strategy

Git Flow (release branches) vs GitHub Flow (main + short-lived branches)

Releases & Versioning

Tag with SemVer (v1.2.0); publish Release notes

GitHub Actions

YAML workflows that build, test, and deploy on every push

Security Tools

Dependabot alerts + auto-PRs, Code Scanning (CodeQL)

Issues & Project Boards

Track work with labels, milestones, and Kanban boards

Common Errors & the Recovery Cheat Sheet

Problem	Fix
Detached HEAD state	git switch -c new-branch (save work on a new branch)
Authentication failed (password)	Use a Personal Access Token, not your account password
Merge conflict markers left in file	Edit file, remove <<<< ==== >>>> markers, git add, git commit
Accidentally deleted a commit/branch	git reflog → git checkout -b recovered <sha>
Committed to the wrong branch	git cherry-pick <sha> onto correct branch, then git reset --hard on the wrong one
Need to undo last commit, keep edits	git reset --soft HEAD~1

QUICK CHEAT SHEET

```
status · add · commit -m · push · pull · branch · switch · merge · log --oneline · stash · reflog · reset --hard · revert
```

Interview Prep & Capstone Recap

Frequently Asked Interview Questions

- What's the difference between git merge and git rebase?
- How would you recover a deleted commit?
- Explain git fetch vs git pull.
- What happens in a 3-way merge conflict?
- Classic vs fine-grained Personal Access Tokens?

Capstone Lab

- Initialize a repo and push it to GitHub
- Create a feature branch, open a Pull Request
- Resolve a deliberate merge conflict with a partner
- Tag a release (v1.0.0) with notes
- Deploy the project live via GitHub Pages
- Recover a deleted commit using git reflog

You've gone from your first git init to a deployed, collaborative, production-style workflow. Keep committing.