

A BEGINNER'S MANUAL

GROK EDITION

GitHub

Save, share, and collaborate on your work, with Git explained from zero

GitHub

Save, share, and collaborate on your work, with Git explained from zero

GROK EDITION

Concise edition, revised and fact-checked by Grok Bot in GrokAllManualsV1.

GitHub is where most of the world's software lives, and it's where AI coding tools like Claude Code, Cursor, and Copilot expect your projects to be. It keeps a complete history of your work, lets you undo almost any mistake, and makes it easy to work with other people (and AI agents) without stepping on each other's toes.

This manual starts with the ideas, moves on to clicking around the GitHub website with nothing installed, and ends with the everyday commands, publishing a website, automation, and keeping your account secure.

HOW TO READ THIS MANUAL

Tip — a shortcut or good habit.

Warning — easy to get wrong.

`$ command` — text you type, exactly as shown.

Note — useful background.

Danger — can lose data, money, or access.

`Settings > Privacy` — a menu path to click.

New words appear in **bold** the first time and are collected in the Glossary at the back.

Contents

01 Welcome to GitHub	7
1.1 Git and GitHub are two different things	7
1.2 Why bother?	7
1.3 Words you'll hear constantly	8
02 Create Your Account	9
2.1 Sign up	9
2.2 Turn on two-factor authentication	9
2.3 Make your profile friendly	10
03 Your First Repository, Entirely in the Browser	11
3.1 Create a repository	11
3.2 Edit a file and make your first commit	11
3.3 Upload files	12
3.4 Writing in Markdown	12
04 Install Git on Your Computer	13
4.1 Install Git	13
4.2 Tell Git who you are	13
4.3 Option A: GitHub Desktop (easiest)	14
4.4 Option B: the GitHub CLI (best for AI tools)	14
05 The Everyday Workflow	15
5.1 The three places your work lives	15
5.2 Get a repository onto your computer	15
5.3 Save and share changes	16
5.4 Writing good commit messages	17
5.5 Looking back in time	17
5.6 Ignoring files with .gitignore	17
06 Branches and Pull Requests	18
6.1 The branch-and-PR workflow	18
6.2 Three ways to merge	19
6.3 Merge conflicts, without panic	19

6.4	Protect your main branch	20
6.5	Contributing to someone else's project	20
07	Issues, Projects, and Discussion	21
7.1	Issues	21
7.2	Projects	21
7.3	Notifications and mentions	21
08	Publish a Website with GitHub Pages	22
09	Automate with GitHub Actions	23
10	GitHub and AI	25
10.1	GitHub Copilot	25
10.2	Claude Code and other agents	25
10.3	Codespaces: a computer in the cloud	26
11	Security and Good Habits	27
11.1	Never commit secrets	27
11.2	Personal access tokens	27
11.3	Keep dependencies healthy	28
11.4	Checklist for a healthy repository	28
12	Plans and Pricing	29
13	Troubleshooting	30
14	Step-by-Step Projects	31
14.1	Project 1: Contribute to another project	31
14.2	Project 2: Resolve a merge conflict	32
14.3	Project 3: Undo common mistakes	33
14.4	Project 4: Tag and publish a release	34
14.5	Project 5: Keep a fork up to date	34
15	Git Command Reference	35
15.1	Setup	35
15.2	Everyday	36
15.3	Branches	36
15.4	Remotes	37
15.5	History and investigation	37

15.6	Stash and clean	37
16	GitHub CLI (gh) Reference	38
16.1	Most-used commands	38
16.2	Full command list	39
17	Markdown on GitHub	45
17.1	Formatting	45
17.2	GitHub extras	46
17.3	README template	46
18	GitHub Actions Cookbook	47
18.1	Anatomy of a workflow	47
18.2	Triggers	48
18.3	Ready-to-use workflows	49
18.4	Secrets and variables	51
18.5	Debugging a failed run	51
19	Security and Repository Settings Walkthrough	52
19.1	Account security	52
19.2	Protect <code>main</code>	52
19.3	Dependabot and code security	53
19.4	Personal access tokens	53
19.5	If you leak a secret	53
20	Frequently Asked Questions	54
21	Working as a Team on GitHub	55
21.1	Organizations vs personal accounts	55
21.2	Step by step: set up a team repo	55
21.3	Reviewing a pull request well	57
21.4	Merge methods	58
21.5	Team habits that prevent pain	58
22	GitHub Desktop, Step by Step	59
22.1	First-time setup	59
22.2	The everyday loop	59
22.3	Resolving a conflict in Desktop	60
23	.gitignore Templates	61

23.1	Every official template (root folder, October 2026)	61
23.2	Add one to an existing repo	62
23.3	Already committed a file you meant to ignore?	62
23.4	Lines worth adding to almost any project	62
24	Releases, Tags, and Versioning	63
24.1	Semantic versioning in one table	63
24.2	Make a release (browser)	63
24.3	Make a release (terminal)	64
24.4	Automate releases with Actions	64
25	Contributing to Open Source	65
25.1	Your first contribution, step by step	65
25.2	Keep your fork up to date	66
25.3	Etiquette	66
	Quick Reference	67
	Glossary	68

Welcome to GitHub

You’ve probably heard “push it to GitHub” or “it’s in the repo.” This chapter explains what those words mean and why GitHub is worth learning, even if you never plan to call yourself a programmer.

IN THIS CHAPTER

- The difference between Git and GitHub
- Why GitHub matters, especially when you work with AI
- The handful of words you’ll hear constantly

1.1 Git and GitHub are two different things

Git is a free program that runs on your computer. It watches a folder and records snapshots of the files in it over time, so you can see what changed, when, and why, and go back to any earlier version. Git was created in 2005 and is used almost everywhere.

GitHub is a website (github.com, owned by Microsoft) that stores copies of Git projects online. It adds everything you need to work with others: a web interface, discussions, task tracking, reviews, automation, and website hosting.

NOTE

A good analogy: Git is like the “track changes” and version history in a word processor; GitHub is like a shared cloud drive built around that history.

1.2 Why bother?

- **A safety net.** Every saved snapshot is permanent. Broke something? Go back.
- **A backup.** Your work lives on GitHub’s servers as well as your computer.

- **A shared workspace.** Several people, or several AI agents, can work on the same project at once.
- **A portfolio.** Your public projects show what you can build.
- **AI tools expect it.** Claude Code, Copilot, Cursor, Lovable, Bolt, and others connect to GitHub to read, change, and publish your code.

1.3 Words you'll hear constantly

WORD	PLAIN-ENGLISH MEANING
Repository (repo)	A project folder that Git tracks, plus its entire history
Commit	A saved snapshot of your files, with a message describing the change
Branch	A separate line of work, so you can try things without touching the main version
Main	The default branch: the "official" version of the project
Merge	Combine the changes from one branch into another
Pull request (PR)	A proposal to merge a branch, with a place to discuss and review it
Issue	A to-do item, bug report, or idea, tracked on GitHub
Clone	Download a full copy of a repository to your computer
Push / Pull	Send your commits up to GitHub / bring others' commits down
Fork	Your own copy of someone else's repository on GitHub

Don't worry about memorizing these; each one gets its own explanation as we go, and they're all in the Glossary.

Create Your Account

2.1 Sign up

CREATE A FREE GITHUB ACCOUNT

- 1 Go to github.com.**
Click **Sign up**.
- 2 Enter your email, a password, and a username.**
Your username becomes part of your web address (`github.com/username`), so pick something you're happy to share professionally.
- 3 Verify.**
Solve the quick puzzle and enter the code GitHub emails you.
- 4 Choose the Free plan.**
It includes unlimited public and private repositories. You can upgrade later.

TIP

Choose a username that's short and easy to say aloud. Changing it later breaks old links to your work.

2.2 Turn on two-factor authentication

Two-factor authentication (2FA) means signing in needs your password *plus* a second proof, such as a code from your phone. GitHub requires 2FA for anyone who contributes code, so set it up now.

ENABLE 2FA

- 1 Open settings.**
Click your profile picture (top right), then **Settings**.

- 2 **Find the security page.**
Go to [Password and authentication](#).
- 3 **Enable two-factor authentication.**
Click **Enable two-factor authentication**.
- 4 **Scan the QR code** with an authenticator app such as Google Authenticator, Microsoft Authenticator, 1Password, or Authy, and type the 6-digit code it shows.
- 5 **Save your recovery codes.**
Download them and keep them somewhere safe, offline. They're your only way back in if you lose your phone.
- 6 **Add a passkey (optional, recommended).**
A **passkey** lets you sign in with your fingerprint, face, or device PIN instead of typing a password.

 DANGER

Lose your phone *and* your recovery codes, and GitHub support usually cannot restore your account. Print the recovery codes and store them with your important papers.

2.3 Make your profile friendly

Click your profile picture, then **Your profile**, then **Edit profile** to add a photo, a short bio, and links. Optional but nice: create a repository named exactly the same as your username, and its README appears at the top of your profile page.

Your First Repository, Entirely in the Browser

You can do real work on GitHub without installing anything. Let's create a repository, edit a file, and save a commit, all from the website.

3.1 Create a repository

NEW REPOSITORY

- 1 Click the **+** menu (top right) and choose **New repository**.
- 2 **Name it.**
For example `my-first-repo`. Use lowercase letters and dashes.
- 3 **Add a description**, like "Learning GitHub."
- 4 **Choose Public or Private.**
Public means anyone can see it; private means only you and people you invite.
- 5 **Tick "Add a README file."** A README is the welcome page of your repository.
- 6 **Click Create repository.**

You're now looking at your repository's home page. The file list is in the middle, the README is shown below it, and tabs across the top lead to **Issues**, **Pull requests**, **Actions**, and **Settings**.

3.2 Edit a file and make your first commit

EDIT AND COMMIT

- 1 **Open README.md** by clicking its name.

- 2 Click the pencil icon (Edit this file).
- 3 Change the text.
Add a line such as `This is where I'm learning GitHub!`
- 4 Click **Commit changes...** (top right).
- 5 Write a **commit message**, a short note saying what you did: `Add welcome line to README`.
- 6 Choose “**Commit directly to the main branch**” and click **Commit changes**.

Congratulations: that's a **commit**. Click the clock icon labeled **commits** on the repository page to see the history, including GitHub's original “Initial commit.”

3.3 Upload files

On the repository page, click **Add file**, then **Upload files**, and drag in documents, images, or code. Write a commit message and click **Commit changes**.

TIP

Press the `.` (period) key on any repository page to open **github.dev**, a full code editor that runs in your browser. It's great for editing several files at once.

3.4 Writing in Markdown

README files, issues, and comments on GitHub use **Markdown**, a simple way to format text with symbols:

MARKDOWN

```
# Big heading
## Smaller heading

**bold**, *italic*, and `code`

- a bullet
- another bullet

1. a numbered step
2. another step

\[a link\]\(https://github.com\)
- [ ] a checkbox to tick off
```

Install Git on Your Computer

Working in the browser is great for small edits. For real projects, and for AI coding tools that work on your computer, you'll want Git installed locally. You can drive it three ways: a friendly app (**GitHub Desktop**), the **command line**, or let an **AI agent** run the commands for you.

4.1 Install Git

SYSTEM	HOW
Windows	Download the installer from git-scm.com and accept the defaults. It includes Git Bash and Git Credential Manager , which handles signing in to GitHub.
macOS	Open Terminal and type <code>git --version</code> . If Git isn't installed, macOS offers to install the developer tools. Or use Homebrew: <code>brew install git</code> .
Linux	Ubuntu/Debian: <code>sudo apt install git</code> · Fedora: <code>sudo dnf install git</code>

Check it worked:

TERMINAL

```
git --version
```

4.2 Tell Git who you are

Every commit is labeled with a name and email. Set them once:

TERMINAL

```
git config --global user.name "Your Name"
git config --global user.email "you@example.com"
git config --global init.defaultBranch main
```

 TIP

To keep your real email private, use the **noreply** address GitHub gives you. Find it at [Settings](#) > [Emails](#) (it looks like `12345+username@users.noreply.github.com`) and tick **Keep my email addresses private**.

4.3 Option A: GitHub Desktop (easiest)

Download **GitHub Desktop** from desktop.github.com (Windows and Mac). Sign in with your GitHub account, and it handles authentication for you. Every action in the next chapter has a button in Desktop: **Clone**, **Commit**, **Push origin**, **Fetch**, **Create branch**, **Create pull request**.

4.4 Option B: the GitHub CLI (best for AI tools)

The **GitHub CLI**, a program called `gh`, lets you, and AI agents like Claude Code, do GitHub things from the terminal: create repositories, open pull requests, manage issues.

SYSTEM	INSTALL COMMAND
macOS	<code>brew install gh</code>
Windows	<code>winget install --id GitHub.cli</code> (or the installer from cli.github.com)
Linux	Follow the apt/dnf instructions linked from cli.github.com

Then sign in. It also sets up Git to use your GitHub login:

TERMINAL

```
gh auth login
```

Answer the questions: choose **GitHub.com**, **HTTPS**, **Yes** to authenticate Git, and **Login with a web browser**. Copy the one-time code, paste it into the browser page that opens, and approve.

 NOTE

GitHub stopped accepting your account password for Git operations years ago. You sign in through the browser (GitHub Desktop, `gh auth login`, or Git Credential Manager), or with an SSH key or personal access token. More on tokens in Chapter 10.

The Everyday Workflow

Almost all Git work is the same loop: **change files** → **stage** → **commit** → **push**. Here it is with the command line; GitHub Desktop has a button for each step.

5.1 The three places your work lives

PLACE	WHAT IT IS
Working folder	The files you see and edit on your computer
Staging area	Changes you've marked as "ready to go in the next commit"
Repository	The saved history of commits, both on your computer (<i>local</i>) and on GitHub (the <i>remote</i> , usually named <code>origin</code>)

5.2 Get a repository onto your computer

CLONE A REPOSITORY

- 1 Copy the address.**
On the repository page, click the green **Code** button and copy the HTTPS URL.
- 2 Open a terminal** in the folder where you keep projects.
- 3 Clone it.**

TERMINAL

```
git clone https://github.com/your-username/my-first-repo.git
cd my-first-repo
```

Or, with the GitHub CLI: `gh repo clone your-username/my-first-repo`.

Starting from a folder you already have instead? Turn it into a repository and publish it in one go:

TERMINAL

```
cd my-project
git init
gh repo create my-project --private --source=. --push
```

5.3 Save and share changes

THE DAILY LOOP

1 See what changed.

TERMINAL

```
git status
```

2 Stage the changes you want to save.

A dot means “everything in this folder.”

TERMINAL

```
git add .
```

3 Commit with a clear message.

TERMINAL

```
git commit -m "Add contact page"
```

4 Push to GitHub.

TERMINAL

```
git push
```

5 Get other people's changes before you start work next time.

TERMINAL

```
git pull
```

5.4 Writing good commit messages

A commit message is a note to your future self. Make it short and specific, and start with a verb:

- Good: `Fix crash when cart is empty` · `Add opening hours to homepage`
- Not so good: `stuff` · `changes` · `asdf`

5.5 Looking back in time

COMMAND	WHAT IT SHOWS
<code>git log --oneline</code>	A compact list of past commits
<code>git diff</code>	Exactly what you changed since the last commit
<code>git show <commit-id></code>	The details of one commit
<code>git restore <file></code>	Throw away uncommitted changes to a file
<code>git revert <commit-id></code>	Create a new commit that undoes an old one, safely

⚠ WARNING

Be careful with commands that contain `--force`, `reset --hard`, or `clean`. They can permanently discard work. If an AI agent wants to run one, read what it's about to do first.

5.6 Ignoring files with .gitignore

Some files should never go into a repository: passwords, huge downloads, and temporary files. List them in a file named `.gitignore` at the top of your project:

```
.GITIGNORE

# secrets
.env
# dependencies (re-downloadable)
node_modules/
# system junk
.DS_Store
```

GitHub offers ready-made `.gitignore` templates when you create a repository.

Branches and Pull Requests

Branches let you work on something new without risking the version that works. **Pull requests** are how you propose, review, and merge that work. This is how professional teams, and AI coding agents, work every day.

6.1 The branch-and-PR workflow

MAKE A CHANGE ON A BRANCH

1 Create and switch to a new branch.

TERMINAL

```
git switch -c add-menu-page
```

2 Make your changes, then commit them as usual (`git add .` and `git commit -m "..."`).

3 Push the branch to GitHub.

TERMINAL

```
git push -u origin add-menu-page
```

4 Open a pull request.

GitHub shows a yellow banner with **Compare & pull request**. Click it. Or run `gh pr create`.

5 Describe the change.

Write what you did and why. Click **Create pull request**.

6 Review.

Teammates (or AI reviewers) comment on specific lines, and automated checks run.

7

Merge.

When everyone's happy, click **Merge pull request**, then **Delete branch**.

8

Update your computer.

TERMINAL

```
git switch main
git pull
```

 TIP

Even if you work alone, use pull requests for anything an AI agent writes. The PR page gives you a clean, line-by-line view of every change before it reaches `main`.

6.2 Three ways to merge

When you merge a pull request, GitHub offers:

OPTION	WHAT HAPPENS	USE WHEN
Create a merge commit	Keeps every commit plus a “merge” commit	You want the full history
Squash and merge	Combines all the PR's commits into one	You want a tidy history (popular)
Rebase and merge	Replays commits on top of main, no merge commit	Advanced teams with linear history

6.3 Merge conflicts, without panic

A **merge conflict** happens when two branches changed the same lines differently, and Git can't decide which to keep. GitHub (or your editor) shows the conflict like this:

TEXT

```
<<<<<< main
Open 7am to 3pm
=====
Open 8am to 4pm
>>>>>> update-hours
```

Delete the markers, keep the right text (or a mix), save, then commit. In a pull request you can often click **Resolve conflicts** and fix it in the browser. Or ask your AI agent: “resolve the merge conflicts on this branch and explain your choices.”

6.4 Protect your main branch

In a repository's [Settings](#) > [Rules](#) > [Rulesets](#) (or **Branches** on older layouts) you can require that changes to `main` go through a pull request, that checks pass, or that someone approves first. This is a great safety net when AI agents push code.

6.5 Contributing to someone else's project

You can't push to other people's repositories directly. Instead:

1. Click **Fork** to make your own copy.
2. Clone your fork, make a branch, commit, and push.
3. Open a pull request from your fork back to the original project.

Issues, Projects, and Discussion

GitHub isn't just for code. Its planning tools work for any project.

7.1 Issues

An **issue** is a tracked item: a bug, a task, a question, or an idea. Click the **Issues** tab, then **New issue**. Give it a title and description, and optionally:

- **Assignees:** who's responsible (this can be an AI agent such as Copilot).
- **Labels:** colored tags like `bug`, `design`, or `good first issue`.
- **Milestone:** group issues toward a goal or deadline.

Write `Fixes #12` in a pull request's description and issue #12 closes automatically when the PR merges.

7.2 Projects

Projects are flexible boards and spreadsheets built from your issues and pull requests, like a lightweight Trello or Notion database. From your profile or organization, open **Projects**, then **New project**, and choose a **Board** (columns like *To do*, *In progress*, *Done*), a **Table**, or a **Roadmap**.

7.3 Notifications and mentions

Type `@username` in any comment to notify that person. Click the bell icon to see your notifications. **Watch** a repository (top of its page) to follow everything that happens in it.

Publish a Website with GitHub Pages

GitHub Pages turns a repository into a free public website, which is perfect for portfolios, documentation, and small business sites.

PUBLISH A SITE

- 1 Put your site in a repository.**
At minimum, an `index.html` file at the top level.
- 2 Open the repository's settings.**
Click **Settings**, then **Pages** in the left sidebar.
- 3 Choose the source.**
Under **Build and deployment**, set **Source** to **Deploy from a branch**.
- 4 Pick the branch.**
Choose `main` and the `/ (root)` folder, then click **Save**.
- 5 Wait a minute or two.**
Refresh the Pages settings; your address appears at the top: `https://your-username.github.io/repository-name/`.

NOTE

On the Free plan, Pages works for **public** repositories. A repository named `your-username.github.io` becomes your main site at `https://your-username.github.io`. You can also connect your own domain name under **Custom domain**.

Automate with GitHub Actions

GitHub Actions runs tasks automatically when something happens in your repository: test the code on every push, publish a site, or send a message when an issue opens. Each automation is a **workflow**, written in a **YAML** file (a simple text format of names and values, indented with spaces) inside the folder

`.github/workflows/`.

`.GITHUB/WORKFLOWS/CHECK.YML`

```
name: Check my site
on: [push, pull_request]

jobs:
  test:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v7
      - name: Say hello
        run: echo "Running checks on ${github.ref_name}"
```

Push that file and click the **Actions** tab to watch it run. A green check mark means it passed; a red X means it failed (click it to see why).

Starting from scratch? The **Actions** tab offers ready-made workflow templates for most languages and hosting services.

PLAN	INCLUDED ACTIONS MINUTES PER MONTH (PRIVATE REPOS)
Free	2,000
Pro / Team	3,000
Enterprise	50,000

Actions on **public** repositories using GitHub's standard runners are free.

 TIP

You rarely need to write workflows by hand. Ask your AI tool: “add a GitHub Actions workflow that runs my tests on every pull request.”

GitHub and AI

10.1 GitHub Copilot

GitHub Copilot is GitHub’s own AI assistant. It suggests code as you type in VS Code and other editors, answers questions in a chat panel, reviews pull requests, and, as the **Copilot coding agent**, can take an issue you assign to it, work in the background, and open a pull request.

COPILOT PLAN	PRICE	HIGHLIGHTS
Free	\$0	2,000 code completions/month, limited chat and agent use
Pro	\$10/month	Unlimited completions; coding agent; monthly AI credits
Pro+	\$39/month	More AI credits; third-party agents such as Claude and Codex
Max	\$100/month	The largest monthly AI credit allowance
Business / Enterprise	Contact sales	Organization controls and policies

Prices from github.com/features/copilot/plans, October 2026. Copilot measures heavier use in **AI credits** (1 credit = \$0.01).

10.2 Claude Code and other agents

AI coding agents use GitHub as their shared workspace:

- **Claude Code** runs `git` and `gh` commands for you. Ask “commit this and open a pull request.” Its cloud sessions at claude.ai/code clone your repositories and push branches back.
- Run `/install-github-app` inside Claude Code to install the Claude GitHub App, then mention `@claude` in an issue or pull request to have it do the work.
- **Cursor**, **Lovable**, **Bolt**, **Replit**, and similar tools can connect your GitHub account to save projects to a repository.

10.3 Codespaces: a computer in the cloud

Codespaces gives you a ready-to-code environment in your browser, with a full VS Code editor, terminal, and tools, running on GitHub's servers. On a repository, click **Code**, choose the **Codespaces** tab, then **Create codespace on main**. Personal accounts get 120 core-hours and 15 GB of storage free each month (a 2-core machine uses 2 core-hours per hour).

TIP

Stop codespaces you aren't using. Click **Code**, then **Codespaces**, then the **...** menu, then **Stop codespace**. Idle ones stop automatically after 30 minutes by default, but they still use storage until deleted.

Security and Good Habits

11.1 Never commit secrets

A **secret** is anything that grants access: passwords, API keys, tokens. Once pushed, assume a secret is compromised, even if you delete it afterwards, because it remains in the history.

- Keep secrets in a `.env` file and list `.env` in `.gitignore`.
- For Actions, store secrets in `Settings` > `Secrets and variables` > `Actions`.
- GitHub's **push protection** blocks many common secret formats from being pushed to public repositories, and you can turn it on for others under `Settings` > `Advanced Security` (called **Code security** on some accounts).

⚠️ LEAKED A KEY?

Revoke or rotate it immediately at the service that issued it (for example, generate a new API key and delete the old one). Removing it from GitHub is not enough.

11.2 Personal access tokens

Some tools ask for a **personal access token** (PAT), a password-like string that lets a program act on your behalf. Prefer **fine-grained** tokens, which you can limit to specific repositories and permissions:

CREATE A FINE-GRAINED TOKEN

- 1 **Open developer settings.**
Profile picture, then `Settings` > `Developer settings` > `Personal access tokens` > `Fine-grained tokens`.
- 2 **Click Generate new token.**
- 3 **Name it and set an expiration**, for example 90 days.

4

Choose repository access*Only select repositories is safest.*

5

Pick permissions.

Grant only what the tool needs (for example, **Contents: Read and write** and **Pull requests: Read and write**).

6

Generate and copy the token.

You'll only see it once. Store it in a password manager.

11.3 Keep dependencies healthy

Turn on **Dependabot** ([Settings](#) > [Advanced Security](#)) to get alerts, and automatic pull requests, when a library your project uses has a known security problem.

11.4 Checklist for a healthy repository

- ☐ A README explaining what the project is and how to run it
- ☐ A `.gitignore` that excludes secrets and build files
- ☐ `main` protected by a ruleset requiring pull requests
- ☐ Dependabot alerts on
- ☐ 2FA on your account, recovery codes stored safely
- ☐ Tokens scoped to the minimum and set to expire

Plans and Pricing

PLAN	PRICE	INCLUDES
Free	\$0	Unlimited public and private repos and collaborators; 2,000 Actions minutes and 500 MB Packages storage per month
Pro (individual)	\$4/month	3,000 Actions minutes, 2 GB Packages, protected branches and Pages on private repos
Team	\$4 per user/month	Pro features for organizations, plus team access controls
Enterprise	\$21 per user/month	50,000 Actions minutes, 50 GB Packages, SSO, advanced compliance

From github.com/pricing and GitHub Docs, October 2026. Copilot and Codespaces are billed separately (Chapter 10).

TIP

The Free plan is plenty for learning, personal projects, and most small businesses.

Troubleshooting

PROBLEM	FIX
<code>fatal: not a git repository</code>	You're not inside a repository folder. <code>cd</code> into it, or run <code>git init</code> .
<code>Authentication failed</code> / asked for a password	Sign in with <code>gh auth login</code> , GitHub Desktop, or Git Credential Manager. Your account password won't work.
<code>rejected ... (fetch first)</code> when pushing	Someone else pushed first. Run <code>git pull</code> , fix any conflicts, then <code>git push</code> .
<code>Please tell me who you are</code>	Run the two <code>git config --global user...</code> commands from Chapter 4.
Committed to the wrong branch	Ask your AI agent, or run <code>git switch -c new-branch</code> to keep the work on a new branch before switching back.
Accidentally committed a secret	Revoke the secret at its source first, then remove it from the repo.
Pages site shows 404	Check that <code>index.html</code> is in the folder you chose, wait a few minutes, and check the Actions tab for errors.

Step-by-Step Projects

Reading about Git only goes so far. These projects walk you through the situations you'll actually run into, one keystroke at a time.

IN THIS CHAPTER

- Project 1: Contribute to someone else's project (fork → PR)
- Project 2: Resolve a merge conflict
- Project 3: Undo common mistakes
- Project 4: Tag and publish a release
- Project 5: Keep a fork up to date

14.1 Project 1: Contribute to another project

1

Fork it.

On the project's GitHub page, click **Fork**. You now have your own copy under your account.

2

Clone your fork.

TERMINAL

```
git clone https://github.com/YOU/project.git
cd project
```

3

Add the original as `upstream`.

TERMINAL

```
git remote add upstream https://github.com/OWNER/project.git
```

4 Make a branch.

```
git switch -c fix-typo-readme
```

5 Edit, then commit.

```
git add README.md && git commit -m "Fix typo in install steps"
```

6 Push to your fork.

```
git push -u origin fix-typo-readme
```

7 Open the pull request.

GitHub shows a **Compare & pull request** button. Explain *what* you changed and *why*.

8 Respond to review.

Push more commits to the same branch and the PR updates automatically.

 TIP

Read the project's `CONTRIBUTING.md` first. Many projects want an issue opened before a PR.

14.2 Project 2: Resolve a merge conflict

A **conflict** happens when two branches change the same lines. Git can't guess which version you want, so it asks you.

1 Update and merge.**TERMINAL**

```
git switch my-branch
git fetch origin
git merge origin/main
```

Git reports `CONFLICT (content): Merge conflict in index.html`.

2 Open the file.

You'll see markers:

TEXT

```
<<<<<<< HEAD
<h1>Welcome!</h1>
=====
<h1>Hello there</h1>
>>>>>>> origin/main
```

- 3 **Choose.**
Keep one side, both, or write a new version. Delete all three marker lines.
- 4 **Mark it resolved.**
`git add index.html`
- 5 **Finish.**
`git commit` (Git suggests a message).
- 6 **Push.**
`git push`

NOTE

VS Code and Cursor show **Accept Current / Accept Incoming / Accept Both** buttons above each conflict. GitHub can also resolve simple conflicts in the browser, from the PR page.

14.3 Project 3: Undo common mistakes

I WANT TO...	COMMAND	SAFE?
Discard unsaved edits to one file	<code>git restore file.txt</code>	Loses those edits
Unstage a file (keep the edits)	<code>git restore --staged file.txt</code>	Yes
Fix the last commit's message (not pushed yet)	<code>git commit --amend -m "New message"</code>	Yes, if not pushed
Add a forgotten file to the last commit (not pushed)	<code>git add f && git commit --amend --no-edit</code>	Yes, if not pushed
Undo a pushed commit safely	<code>git revert <commit></code>	Yes. Creates a new "undo" commit
Move the branch back but keep the changes	<code>git reset --soft HEAD~1</code>	Yes, locally
Throw away local commits and changes	<code>git reset --hard <commit></code>	Destructive
Recover "lost" commits	<code>git reflog</code> , then <code>git switch -c rescue <hash></code>	Yes
Put work aside temporarily	<code>git stash</code> , later <code>git stash pop</code>	Yes
Committed to <code>main</code> by mistake (not pushed)	<code>git switch -c new-branch</code> , then <code>git switch main && git reset --hard origin/main</code>	Careful

! DANGER

Never `reset --hard`, rebase, or force-push commits that others may have pulled. Use `git revert` for anything already shared. If you must force-push your own branch, use `git push --force-with-lease`, which refuses if someone else pushed in the meantime.

14.4 Project 4: Tag and publish a release

1 Tag the commit.

```
git tag -a v1.0.0 -m "First release"
```

2 Push the tag.

```
git push origin v1.0.0
```

3 Create the release.

On GitHub go to [Releases](#) > [Draft a new release](#), choose the tag, and click **Generate release notes**. Or, from the terminal:

TERMINAL

```
gh release create v1.0.0 --generate-notes
```

4 Attach files (optional), such as a PDF or a zip, by dragging them into the release form or adding paths after the tag in `gh release create`.

Version numbers usually follow **semantic versioning**: `MAJOR.MINOR.PATCH`. Bump MAJOR for breaking changes, MINOR for new features, and PATCH for fixes.

14.5 Project 5: Keep a fork up to date

TERMINAL

```
git switch main
git fetch upstream
git merge upstream/main
git push origin main
```

Or click **Sync fork** on your fork's GitHub page.

Git Command Reference

Every Git command a beginner-to-intermediate user needs, grouped by task. Type `git help <command>` for the official manual page.

IN THIS CHAPTER

- Setup, everyday, branching, remote, history, and repair commands
- Useful configuration

15.1 Setup

COMMAND	WHAT IT DOES
<code>git config --global user.name "Your Name"</code>	Name stamped on your commits
<code>git config --global user.email "you@example.com"</code>	Email on your commits (GitHub's no-reply address works)
<code>git config --global init.defaultBranch main</code>	New repos start on <code>main</code>
<code>git config --global pull.rebase false</code>	<code>git pull</code> merges (the beginner-friendly default)
<code>git config --global core.editor "code --wait"</code>	Use VS Code for commit messages
<code>git config --list</code>	Show all settings
<code>git init</code>	Turn the current folder into a repository
<code>git clone <url></code>	Copy a repository to your computer

15.2 Everyday

COMMAND	WHAT IT DOES
<code>git status</code>	What changed, what's staged, which branch you're on
<code>git add <file></code> / <code>git add .</code>	Stage one file / everything in this folder
<code>git add -p</code>	Stage piece by piece, reviewing each change
<code>git commit -m "msg"</code>	Save a snapshot with a message
<code>git commit -am "msg"</code>	Stage tracked files and commit in one step
<code>git diff</code>	Changes not yet staged
<code>git diff --staged</code>	Changes staged for the next commit
<code>git log --oneline --graph --all</code>	Compact history of all branches
<code>git show <commit></code>	What one commit changed
<code>git rm <file></code> / <code>git mv a b</code>	Delete or rename, and stage it

15.3 Branches

COMMAND	WHAT IT DOES
<code>git branch</code>	List local branches
<code>git branch -a</code>	Include remote branches
<code>git switch <name></code>	Move to a branch
<code>git switch -c <name></code>	Create a branch and move to it
<code>git merge <branch></code>	Merge that branch into the current one
<code>git rebase main</code>	Replay your branch's commits on top of <code>main</code> (local branches only)
<code>git branch -d <name></code>	Delete a merged branch
<code>git branch -D <name></code>	Force-delete an unmerged branch
<code>git cherry-pick <commit></code>	Copy one commit onto the current branch

15.4 Remotes

COMMAND	WHAT IT DOES
<code>git remote -v</code>	List remotes and their URLs
<code>git remote add <name> <url></code>	Add a remote
<code>git fetch</code>	Download new commits without changing your files
<code>git pull</code>	Fetch and merge into the current branch
<code>git push</code>	Upload your commits
<code>git push -u origin <branch></code>	First push of a new branch, and set it to track
<code>git push --force-with-lease</code>	Overwrite your remote branch only if nobody else pushed
<code>git push origin --delete <branch></code>	Delete a branch on GitHub

15.5 History and investigation

COMMAND	WHAT IT DOES
<code>git log -p <file></code>	Every change ever made to a file
<code>git log --author="Name"</code>	Commits by one person
<code>git log -S "text"</code>	Commits that added or removed that text
<code>git blame <file></code>	Who last changed each line, and when
<code>git bisect start</code> / <code>good</code> / <code>bad</code>	Binary-search history for the commit that broke something
<code>git reflog</code>	Everywhere <code>HEAD</code> has been, which is your safety net
<code>git tag</code>	List tags

15.6 Stash and clean

COMMAND	WHAT IT DOES
<code>git stash</code>	Shelve uncommitted changes
<code>git stash list</code>	See your stashes
<code>git stash pop</code>	Re-apply and remove the latest stash
<code>git clean -n</code>	Preview untracked files that would be deleted
<code>git clean -fd</code>	Delete untracked files and folders

GitHub CLI (gh) Reference

`gh` brings GitHub to your terminal: pull requests, issues, releases, Actions, and more. Install it from cli.github.com, then run `gh auth login`.

IN THIS CHAPTER

- The ten `gh` commands you'll use most
- The full command list

16.1 Most-used commands

COMMAND	WHAT IT DOES
<code>gh auth login</code>	Sign in (browser or token)
<code>gh repo create my-app --private --source=. --push</code>	Create a GitHub repo from the current folder and push
<code>gh repo clone owner/repo</code>	Clone without copying the URL
<code>gh pr create --fill</code>	Open a PR using your commit messages
<code>gh pr list</code> / <code>gh pr view 12 --web</code>	List PRs / open one in the browser
<code>gh pr checkout 12</code>	Check out someone's PR locally to test it
<code>gh pr merge 12 --squash</code>	Merge a PR
<code>gh issue create</code> / <code>gh issue list</code>	File or list issues
<code>gh run watch</code>	Watch an Actions run live
<code>gh release create v1.0.0 --generate-notes</code>	Publish a release

16.2 Full command list

This list was generated from GitHub CLI version 2.102.0 (released September 30, 2026). Newer versions may add commands, so run `gh help` for yours.

COMMAND	WHAT IT DOES
<code>gh auth login</code>	Log in to a GitHub account
<code>gh auth logout</code>	Log out of a GitHub account
<code>gh auth refresh</code>	Refresh stored authentication credentials
<code>gh auth setup-git</code>	Setup git with GitHub CLI
<code>gh auth status</code>	Display active account and authentication state on each known GitHub host
<code>gh auth switch</code>	Switch active GitHub account
<code>gh auth token</code>	Print the authentication token gh uses for a hostname and account
<code>gh browse</code>	Transition from the terminal to the web browser to view and interact with:
<code>gh codespace code</code>	Open a codespace in Visual Studio Code
<code>gh codespace cp</code>	Copy files between local and remote file systems
<code>gh codespace create</code>	Create a codespace
<code>gh codespace delete</code>	Delete codespaces
<code>gh codespace edit</code>	Edit a codespace
<code>gh codespace jupyter</code>	Open a codespace in JupyterLab
<code>gh codespace list</code>	List codespaces
<code>gh codespace logs</code>	Access codespace logs
<code>gh codespace ports</code>	List ports in a codespace
<code>gh codespace rebuild</code>	Rebuild a codespace
<code>gh codespace ssh</code>	SSH into a codespace
<code>gh codespace stop</code>	Stop a running codespace
<code>gh codespace view</code>	View details about a codespace
<code>gh discussion create</code>	Create a new discussion (preview)
<code>gh discussion list</code>	List discussions in a repository (preview)
<code>gh discussion comment</code>	Add, edit, or delete a comment or a reply on a discussion (preview)
<code>gh discussion edit</code>	Edit a discussion (preview)
<code>gh discussion view</code>	View a discussion (preview)
<code>gh gist clone</code>	Clone a gist locally
<code>gh gist create</code>	Create a new gist
<code>gh gist delete</code>	Delete a gist
<code>gh gist edit</code>	Edit one of your gists

<code>gh gist list</code>	List your gists
<code>gh gist rename</code>	Rename a file in a gist
<code>gh gist view</code>	View a gist
<code>gh issue create</code>	Create a new issue
<code>gh issue list</code>	List issues in a repository
<code>gh issue status</code>	Show status of relevant issues
<code>gh issue close</code>	Close issue
<code>gh issue comment</code>	Add a comment to an issue
<code>gh issue delete</code>	Delete issue
<code>gh issue develop</code>	Manage linked branches for an issue
<code>gh issue edit</code>	Edit issues
<code>gh issue lock</code>	Lock issue conversation
<code>gh issue pin</code>	Pin an issue
<code>gh issue reopen</code>	Reopen issue
<code>gh issue transfer</code>	Transfer issue to another repository
<code>gh issue unlock</code>	Unlock issue conversation
<code>gh issue unpin</code>	Unpin an issue
<code>gh issue view</code>	View an issue
<code>gh org list</code>	List organizations for the authenticated user.
<code>gh pr create</code>	Create a pull request
<code>gh pr list</code>	List pull requests in a repository
<code>gh pr status</code>	Show status of relevant pull requests
<code>gh pr checkout</code>	Check out a pull request in git
<code>gh pr checks</code>	Show CI status for a single pull request
<code>gh pr close</code>	Close a pull request
<code>gh pr comment</code>	Add a comment to a pull request
<code>gh pr diff</code>	View changes in a pull request
<code>gh pr edit</code>	Edit a pull request
<code>gh pr lock</code>	Lock pull request conversation
<code>gh pr merge</code>	Merge a pull request
<code>gh pr ready</code>	Mark a pull request as ready for review
<code>gh pr reopen</code>	Reopen a pull request
<code>gh pr revert</code>	Revert a pull request
<code>gh pr review</code>	Add a review to a pull request
<code>gh pr unlock</code>	Unlock pull request conversation

<code>gh pr update-branch</code>	Update a pull request branch
<code>gh pr view</code>	View a pull request
<code>gh project close</code>	Close a project
<code>gh project copy</code>	Copy a project
<code>gh project create</code>	Create a project
<code>gh project delete</code>	Delete a project
<code>gh project edit</code>	Edit a project
<code>gh project field-create</code>	Create a field in a project
<code>gh project field-delete</code>	Delete a field in a project
<code>gh project field-list</code>	List the fields in a project
<code>gh project item-add</code>	Add a pull request or an issue to a project
<code>gh project item-archive</code>	Archive an item in a project
<code>gh project item-create</code>	Create a draft issue item in a project
<code>gh project item-delete</code>	Delete an item from a project by ID
<code>gh project item-edit</code>	Edit an item in a project
<code>gh project item-list</code>	List the items in a project
<code>gh project link</code>	Link a project to a repository or a team
<code>gh project list</code>	List the projects for an owner
<code>gh project mark-template</code>	Mark a project as a template
<code>gh project unlink</code>	Unlink a project from a repository or a team
<code>gh project view</code>	View a project
<code>gh release create</code>	Create a new release
<code>gh release list</code>	List releases in a repository
<code>gh release delete</code>	Delete a release
<code>gh release delete-asset</code>	Delete an asset from a release
<code>gh release download</code>	Download release assets
<code>gh release edit</code>	Edit a release
<code>gh release upload</code>	Upload assets to a release
<code>gh release verify</code>	Verify the attestation for a release
<code>gh release verify-asset</code>	Verify that a given asset originated from a release
<code>gh release view</code>	View information about a release
<code>gh repo create</code>	Create a new repository
<code>gh repo list</code>	List repositories owned by user or organization
<code>gh repo archive</code>	Archive a repository
<code>gh repo autolink</code>	Manage autolink references

<code>gh repo clone</code>	Clone a repository locally
<code>gh repo delete</code>	Delete a repository
<code>gh repo deploy-key</code>	Manage deploy keys in a repository
<code>gh repo edit</code>	Edit repository settings
<code>gh repo fork</code>	Create a fork of a repository
<code>gh repo gitignore</code>	List and view available repository gitignore templates
<code>gh repo license</code>	Explore repository licenses
<code>gh repo read-dir</code>	List a directory in a repository (preview)
<code>gh repo read-file</code>	Read a file from a repository (preview)
<code>gh repo rename</code>	Rename a repository
<code>gh repo set-default</code>	Configure default repository for this directory
<code>gh repo sync</code>	Sync a repository
<code>gh repo unarchive</code>	Unarchive a repository
<code>gh repo view</code>	View a repository
<code>gh skill install</code>	Install agent skills from a GitHub repository (preview)
<code>gh skill list</code>	List installed skills (preview)
<code>gh skill preview</code>	Preview a skill from a GitHub repository (preview)
<code>gh skill publish</code>	Validate and publish skills to a GitHub repository (preview)
<code>gh skill search</code>	Search for skills across GitHub (preview)
<code>gh skill update</code>	Update installed skills to their latest versions (preview)
<code>gh cache delete</code>	Delete GitHub Actions caches
<code>gh cache list</code>	List GitHub Actions caches
<code>gh run cancel</code>	Cancel a workflow run
<code>gh run delete</code>	Delete a workflow run
<code>gh run download</code>	Download artifacts generated by a workflow run
<code>gh run list</code>	List recent workflow runs
<code>gh run rerun</code>	Rerun a run
<code>gh run view</code>	View a summary of a workflow run
<code>gh run watch</code>	Watch a run until it completes, showing its progress
<code>gh workflow disable</code>	Disable a workflow
<code>gh workflow enable</code>	Enable a workflow
<code>gh workflow list</code>	List workflows
<code>gh workflow run</code>	Run a workflow by creating a workflow_dispatch event
<code>gh workflow view</code>	View the summary of a workflow
<code>gh co</code>	Check out a pull request in git

<code>gh agent-task create</code>	Create an agent task (preview)
<code>gh agent-task list</code>	List agent tasks (preview)
<code>gh agent-task view</code>	View an agent task session (preview)
<code>gh alias delete</code>	Delete set aliases
<code>gh alias import</code>	Import aliases from a YAML file
<code>gh alias list</code>	List your aliases
<code>gh alias set</code>	Create a shortcut for a gh command
<code>gh api</code>	Makes an authenticated HTTP request to the GitHub API and prints the response.
<code>gh attestation download</code>	Download an artifact's attestations for offline use
<code>gh attestation trusted-root</code>	Output trusted_root.jsonl contents, likely for offline verification
<code>gh attestation verify</code>	Verify an artifact's integrity using attestations
<code>gh completion</code>	Generate shell completion scripts for GitHub CLI commands.
<code>gh config clear-cache</code>	Clear the cli cache
<code>gh config get</code>	Print the value of a given configuration key
<code>gh config list</code>	Print a list of configuration keys and values
<code>gh config set</code>	Update configuration with a value for the given key
<code>gh copilot</code>	Runs the GitHub Copilot CLI.
<code>gh extension browse</code>	Enter a UI for browsing, adding, and removing extensions
<code>gh extension create</code>	Create a new extension
<code>gh extension exec</code>	Execute an installed extension
<code>gh extension install</code>	Install a gh extension from a repository
<code>gh extension list</code>	List installed extension commands
<code>gh extension remove</code>	Remove an installed extension
<code>gh extension search</code>	Search extensions to the GitHub CLI
<code>gh extension upgrade</code>	Upgrade installed extensions
<code>gh gpg-key add</code>	Add a GPG key to your GitHub account
<code>gh gpg-key delete</code>	Delete a GPG key from your GitHub account
<code>gh gpg-key list</code>	Lists GPG keys in your GitHub account
<code>gh label clone</code>	Clones labels from one repository to another
<code>gh label create</code>	Create a new label
<code>gh label delete</code>	Delete a label from a repository
<code>gh label edit</code>	Edit a label
<code>gh label list</code>	List labels in a repository
<code>gh licenses</code>	View license information for third-party libraries used in this build of the GitHub CLI.
<code>gh preview prompter</code>	Execute a test program to preview the prompter

<code>gh ruleset check</code>	View rules that would apply to a given branch
<code>gh ruleset list</code>	List rulesets for a repository or organization
<code>gh ruleset view</code>	View information about a ruleset
<code>gh search code</code>	Search within code
<code>gh search commits</code>	Search for commits
<code>gh search issues</code>	Search for issues
<code>gh search prs</code>	Search for pull requests
<code>gh search repos</code>	Search for repositories
<code>gh secret delete</code>	Delete secrets
<code>gh secret list</code>	List secrets
<code>gh secret set</code>	Create or update secrets
<code>gh ssh-key add</code>	Add an SSH key to your GitHub account
<code>gh ssh-key delete</code>	Delete an SSH key from your GitHub account
<code>gh ssh-key list</code>	Lists SSH keys in your GitHub account
<code>gh status</code>	The status command prints information about your work on GitHub across all the repositories you're subscribed to, including:
<code>gh variable delete</code>	Delete variables
<code>gh variable get</code>	Get variables
<code>gh variable list</code>	List variables
<code>gh variable set</code>	Create or update variables
<code>gh accessibility</code>	Learn about GitHub CLI's accessibility experiences
<code>gh actions</code>	Welcome to GitHub Actions on the command line.
<code>gh environment</code>	<code>GH_TOKEN</code> , <code>GITHUB_TOKEN</code> (in order of precedence): an authentication token that will be used when
<code>gh exit-codes</code>	gh follows normal conventions regarding exit codes.
<code>gh formatting</code>	By default, the result of <code>gh</code> commands are output in line-based plain text format.
<code>gh mintty</code>	MinTTY is the terminal emulator that comes by default with Git
<code>gh reference</code>	# gh reference
<code>gh telemetry</code>	gh collects telemetry to help us understand how the CLI is being used and to improve it.

Markdown on GitHub

README files, issues, PRs, and comments all use **GitHub Flavored Markdown**. This page is a complete cheat sheet.

IN THIS CHAPTER

- Formatting syntax
- GitHub-only extras: task lists, mentions, alerts, Mermaid diagrams
- A README template

17.1 Formatting

YOU TYPE	YOU GET
# Title ... ##### Small	Headings, levels 1–6
bold , <i>*italic*</i> , <del>~~strike~~</del>	bold , <i>italic</i> , ~~strike~~
<code>`code`</code>	Inline code
<code>```js ... ```</code>	A code block with syntax highlighting
- item or 1. item	Bulleted or numbered list
[text](https://url)	Link
![alt text](image.png)	Image
> quote	Blockquote
---	Horizontal rule
`	a

17.2 GitHub extras

FEATURE	SYNTAX	NOTES
Task list	<code>- [] todo / - [x] done</code>	Clickable checkboxes in issues and PRs
Mention	<code>@username</code>	Notifies that person
Issue / PR link	<code>#123</code>	Auto-links in the same repo
Closing keyword	<code>Fixes #123</code> in a PR description	Closes the issue when the PR merges
Alerts	<code>> [!NOTE]</code> , <code>> [!TIP]</code> , <code>> [!IMPORTANT]</code> , <code>> [!WARNING]</code> , <code>> [!CAUTION]</code>	Coloured callout boxes
Diagrams	A <code>```mermaid</code> code block	Renders flowcharts and sequence diagrams
Collapsible	<code><details><summary>Title</summary> ... </details></code>	Hides long logs
Footnote	<code>Text^[^1]</code> and <code>[^1]: Note</code>	Footnotes

17.3 README template

README.MD

Project Name

One sentence: what it does and who it's for.

Getting started

1. Install: ``npm install``
2. Run: ``npm start``
3. Open `http://localhost:3000`

Features

- Feature one
- Feature two

Contributing

Open an issue first, then a pull request.

License

MIT

GitHub Actions Cookbook

Actions run code automatically when something happens in your repository. This chapter gives you copy-paste workflows and explains every line.

IN THIS CHAPTER

- Anatomy of a workflow file
- Triggers reference
- Eight ready-to-use workflows
- Secrets and variables
- Debugging failed runs

18.1 Anatomy of a workflow

Workflows live in `.github/workflows/` as `.yaml` files.

```
.GITHUB/WORKFLOWS/CI.YML

name: CI                                # shown in the Actions tab
on:                                     # WHEN it runs
  push:
    branches: [main]
  pull_request:
jobs:                                   # WHAT runs
  test:
    runs-on: ubuntu-latest             # the virtual machine
    steps:
      - uses: actions/checkout@v7      # get your code
      - uses: actions/setup-node@v7
        with:
          node-version: 24
      - run: npm ci                    # a shell command
      - run: npm test
```

KEY	MEANING
<code>on</code>	Events that trigger the workflow
<code>jobs</code>	One or more jobs. They run in parallel unless linked with <code>needs</code>
<code>runs-on</code>	Runner type: <code>ubuntu-latest</code> , <code>windows-latest</code> , or <code>macos-latest</code>
<code>steps</code>	Ordered list of actions (<code>uses</code>) or commands (<code>run</code>)
<code>with</code>	Inputs to an action
<code>env</code>	Environment variables
<code>if</code>	A condition for running a job or step
<code>needs</code>	Make a job wait for another
<code>permissions</code>	What the automatic <code>GITHUB_TOKEN</code> is allowed to do

18.2 Triggers

TRIGGER	FIRES WHEN...
<code>push</code>	Commits are pushed (filter with <code>branches</code> , <code>paths</code> , <code>tags</code>)
<code>pull_request</code>	A PR is opened, updated, or reopened
<code>workflow_dispatch</code>	You click Run workflow (manual button)
<code>schedule</code>	A cron time, in UTC, e.g. <code>- cron: "0 6 * * 1"</code> = Mondays 06:00 UTC
<code>release</code>	A release is published
<code>issues</code> / <code>issue_comment</code>	Issues or comments are created or edited
<code>workflow_run</code>	Another workflow finishes

18.3 Ready-to-use workflows

1. Build a static site and deploy to Pages

YAML

```
on: { push: { branches: [main] }, workflow_dispatch: {} }
permissions: { contents: read, pages: write, id-token: write }
jobs:
  deploy:
    runs-on: ubuntu-latest
    environment: github-pages
    steps:
      - uses: actions/checkout@v7
      - uses: actions/configure-pages@v6
      - uses: actions/upload-pages-artifact@v5
        with: { path: "." }
      - uses: actions/deploy-pages@v5
```

2. Python tests

YAML

```
on: [push, pull_request]
jobs:
  test:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v7
      - uses: actions/setup-python@v7
        with: { python-version: "3.12" }
      - run: pip install -r requirements.txt
      - run: pytest
```

3. Test on several versions (matrix)

YAML

```
jobs:
  test:
    runs-on: ubuntu-latest
    strategy:
      matrix: { node: [22, 24] }
    steps:
      - uses: actions/checkout@v7
      - uses: actions/setup-node@v7
        with: { node-version: "${{ matrix.node }}" }
      - run: npm ci && npm test
```

4. A scheduled job

YAML

```
on:
  schedule: [{ cron: "30 12 * * *" }] # 12:30 UTC daily
  workflow_dispatch: {}
jobs:
  run:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v7
      - run: python scripts/daily_report.py
```

5. Upload a build file as an artifact

YAML

```
- run: npm run build
- uses: actions/upload-artifact@v7
  with: { name: site, path: dist/ }
```

6. Cache dependencies (setup actions have this built in)

YAML

```
- uses: actions/setup-node@v7
  with: { node-version: 24, cache: npm }
```

7. Only run when certain files change

YAML

```
on:
  push:
    paths: ["content/**", "build.mjs"]
```

8. Use a secret

YAML

```
- run: ./deploy.sh
  env:
    API_TOKEN: ${ secrets.API_TOKEN }
```

18.4 Secrets and variables

Add them in **Settings** > **Secrets and variables** > **Actions**. **Secrets** are encrypted and masked in logs. **Variables** are plain configuration values (`${{ vars.NAME }}`). Secrets aren't passed to workflows triggered by pull requests from forks.

⚠ DANGER

Pin third-party actions to a version tag you trust, or to a full commit hash for maximum safety. A compromised action can read your secrets. Give `permissions:` the minimum the workflow needs.

18.5 Debugging a failed run

1 Open the run.

Go to the **Actions** tab and click the red **X**.

2 Expand the failing step.

The error is usually at the bottom.

3 Re-run with debug logging.

Click **Re-run jobs**, then tick **Enable debug logging**.

4 Reproduce locally.

Run the same commands on your computer.

5 Ask AI.

Paste the log into an assistant and ask it to “explain this CI failure and propose the smallest fix.”

Security and Repository Settings Walkthrough

A guided tour of the settings that protect your work. Spend twenty minutes here once per important repository.

IN THIS CHAPTER

- Account security: 2FA and passkeys
- Protecting `main` with rulesets
- Dependabot and secret scanning
- Personal access tokens done right
- A security checklist

19.1 Account security

1 Turn on two-factor authentication.

Go to `Settings` > `Password and authentication`. Use an authenticator app or a passkey rather than SMS where you can.

2 Save your recovery codes somewhere offline, such as a printed copy or a password manager.

3 Review sessions and authorized apps.

Go to `Settings` > `Applications` and revoke anything you don't recognise.

19.2 Protect `main`

Go to `Settings` > `Rules` > `Rulesets` (or **Branches** for classic branch protection) and add a rule for `main`:

- ☐ Require a pull request before merging
- ☐ Require status checks to pass (pick your CI job)
- ☐ Block force pushes
- ☐ Restrict deletions
- ☐ Optionally, require approvals (useful on team repos)

19.3 Dependabot and code security

Under **Settings** > **Advanced Security** (called **Code security** on some accounts) you can turn on **Dependabot alerts** (warnings about vulnerable dependencies), **Dependabot security updates** (automatic fix PRs), and **secret scanning / push protection** (blocks pushes that contain known token formats). Availability depends on whether the repository is public or private and on your plan. GitHub's settings page shows which options you can use.

19.4 Personal access tokens

When a tool asks for a token, create a **fine-grained personal access token** at **Settings** > **Developer settings** > **Personal access tokens** :

- ☐ Limit it to only the repositories it needs
- ☐ Grant only the permissions it needs, read-only if possible
- ☐ Set an expiration date
- ☐ Store it in a password manager or secret store, never in code
- ☐ Revoke it when you stop using the tool

19.5 If you leak a secret

- 1 **Revoke or rotate it immediately** at the service that issued it. Deleting the commit isn't enough, because the secret is already exposed.
- 2 **Remove it from the code** and commit.
- 3 **Check logs** at that service for misuse.
- 4 **Optionally, rewrite history** with a tool like `git filter-repo` if the repository must not contain it. Coordinate with collaborators first.

Frequently Asked Questions

Is Git the same as GitHub? No. Git is the version-control tool on your computer. GitHub is a website that hosts Git repositories and adds collaboration features.

Public or private? Public repositories are visible to everyone. Private ones are visible only to people you invite. Choose private for anything personal. You can change it later in Settings.

Will deleting a file remove it from history? No. It's still in earlier commits. Treat anything you ever committed as permanent, which is why secrets must be rotated.

Merge, squash, or rebase when merging a PR? **Merge** keeps every commit. **Squash** combines the PR into one tidy commit, which is great for beginners. **Rebase** replays the commits in a straight line.

What's `origin`? The default nickname for the GitHub copy of your repository.

What's `HEAD`? A pointer to the commit you currently have checked out.

Can I use GitHub without the terminal? Yes. The website, GitHub Desktop, and the editors in VS Code and Cursor cover most tasks. The terminal is faster once you know it.

How big can files be? GitHub blocks files over 100 MB and warns above 50 MB. Use Git LFS or a release asset for large files.

Why does `git push` ask for a password that doesn't work? GitHub doesn't accept account passwords for Git over HTTPS. Sign in with `gh auth login`, Git Credential Manager, or SSH keys.

Working as a Team on GitHub

Working alone, you push to `main` and move on. Once there's a team, you need some rules: who can merge, who reviews what, and how work gets planned. This chapter sets those rules up step by step.

21.1 Organizations vs personal accounts

	PERSONAL ACCOUNT	ORGANIZATION
Owner	One person	A shared account, with owners and members
Best for	Your own projects	Clubs, companies and teams
Permissions	Collaborators per repo	Teams with roles (read, triage, write, maintain, admin) across repos
Billing	Your plan	The organization's plan

To make an organization: click your profile picture → **Your organizations** → **New organization**, or go to github.com/organizations/new. Choose the free plan to start.

21.2 Step by step: set up a team repo

1

Create teams.

In the organization, open **Teams** → **New team** (e.g. *developers*, *reviewers*).

2

Give access.

In the repo, go to **Settings** > **Collaborators and teams** → **Add teams**, and give *developers* the **Write** role.

3

Add CODEOWNERS.

Create `.github/CODEOWNERS` (example below). GitHub automatically requests reviews from the owners of the files a pull request changes.

4

Protect main.

Under **Settings** > **Rules** > **Rulesets**, create a branch ruleset targeting the default branch. Turn on *Require a pull request before merging* (1 approval), *Require review from Code Owners*, *Require status checks to pass*, and *Block force pushes*.

5

Add templates.

Create `.github/pull_request_template.md` and issue forms in `.github/ISSUE_TEMPLATE/` (examples below).

6

Plan the work.

Create a Project (board) for the organization and link the repo (see *Issues, Projects, and Discussion*).

`.GITHUB/CODEOWNERS`

```
# Later rules win. The last matching pattern takes precedence.
*                @my-org/developers
/docs/           @my-org/writers
/.github/        @my-org/admins
*.css            @alice
```

`.GITHUB/PULL_REQUEST_TEMPLATE.MD`

```
## What changed
<!-- One or two sentences -->

## Why
Closes #

## How to test
1.

## Checklist
- [ ] Tests pass locally
- [ ] Docs updated if needed
- [ ] No secrets or personal data added
```

```
.GITHUB/ISSUE_TEMPLATE/BUG.YML
```

```
name: Bug report
description: Something isn't working
labels: ["bug"]
body:
  - type: textarea
    id: what
    attributes:
      label: What happened?
      description: What did you expect instead?
    validations:
      required: true
  - type: textarea
    id: steps
    attributes:
      label: Steps to reproduce
      value: |
        1.
        2.
  - type: input
    id: version
    attributes:
      label: Version or browser
```

21.3 Reviewing a pull request well

- 1 Open the PR and read the description first. Is it clear *why* this change exists?
- 2 Open **Files changed**. Click + next to a line to leave a comment. Use **Add a suggestion** (the ± icon) to propose an exact replacement the author can accept with one click.
- 3 Check out the branch locally if the change is non-trivial: `gh pr checkout <number>`.
- 4 Finish with **Review changes** → *Comment, Approve* or *Request changes*.
- 5 Be kind and specific. “This could crash if `items` is empty. Maybe guard with `if (!items.length) return ?`” beats “wrong”.

21.4 Merge methods

METHOD	HISTORY RESULT	GOOD WHEN
Create a merge commit	Keeps every commit, plus a merge commit	You want the full branch history
Squash and merge	One commit per PR on main	Branches have messy work-in-progress commits (the most common team choice)
Rebase and merge	Commits replayed onto main, with no merge commit	Each commit is clean and meaningful

Choose which methods are allowed under [Settings](#) > [General](#) > [Pull Requests](#). Turn on **Automatically delete head branches** there too.

21.5 Team habits that prevent pain

- ☐ ☐ Small PRs, ideally under about 400 changed lines, so reviews are fast
- ☐ ☐ One issue per change, linked with “Closes #123” in the PR body
- ☐ ☐ Draft PRs for early feedback (**Create draft pull request**)
- ☐ ☐ Update your branch from main before asking for a final review
- ☐ ☐ Never commit secrets. Turn on push protection under the repo’s security settings
- ☐ ☐ Label issues (*bug, good first issue, needs triage*) so anyone can find work

GitHub Desktop, Step by Step

GitHub Desktop is a free app for Windows and macOS that does the everyday Git workflow with buttons instead of commands. Download it from desktop.github.com.

22.1 First-time setup

- 1 Install it and open it. Sign in with your GitHub account (on Windows **File > Options > Accounts**, on Mac **GitHub Desktop > Settings > Accounts**).
- 2 Set your name and email for commits under **Git** in the same Options/Settings window.
- 3 To get an existing repo: **File > Clone repository**, pick it from the list, choose a local folder, then **Clone**.
- 4 To start a new one: **File > New repository**, name it, tick **Initialize this repository with a README**, then **Publish repository** to put it on GitHub.

22.2 The everyday loop

YOU WANT TO	IN GITHUB DESKTOP
See what changed	The Changes tab lists files, and the right pane shows line-by-line differences
Save a snapshot	Tick the files, write a summary at the bottom left, click Commit to
Upload commits	Push origin (top bar)
Get others' changes	Fetch origin , then Pull origin
Make a branch	Current branch dropdown → New branch
Switch branch	Current branch dropdown → pick one. Desktop asks what to do with uncommitted changes

Open a pull request	After pushing a branch, click Preview Pull Request or Create Pull Request
Bring main into your branch	Branch > Update from main (named after your default branch)
Undo the last commit	History tab → right-click the latest commit → Undo commit (only before pushing)
Discard changes to a file	Right-click the file in Changes → Discard changes
Open in your editor	Repository > Open in (choose the editor in Options/Settings → Integrations)

22.3 Resolving a conflict in Desktop

- 1 When a merge or pull hits a conflict, Desktop lists the conflicted files.
- 2 Click **Open in** next to a file. Look for the markers `<<<<<<`, `=====` and `>>>>>>`.
- 3 Edit the file so it holds the final text you want, and delete all three markers.
- 4 Save. Desktop marks the file as resolved. When every file is resolved, click **Continue merge** or **Commit merge**.

.gitignore Templates

A `.gitignore` file tells Git which files never to track, such as dependencies, build output, secrets and system junk. GitHub keeps official templates at github.com/github/gitignore. When you create a repo on github.com, choose one from the **Add .gitignore** dropdown.

23.1 Every official template (root folder, October 2026)

Actionscript, Ada, AdventureGameStudio, Agda, Android, Angular, AppEngine, AppceleratorTitanium, ArchLinuxPackages, Autotools, Ballerina, C++, C, CFWheels, CMake, CUDA, CakePHP, ChefCookbook, Clojure, CodeIgniter, CommonLisp, Composer, Concrete5, Coq, CraftCMS, D, DM, Dart, Delphi, Deno, Dotnet, Drupal, EPiServer, Eagle, Emacs, Elixir, Elm, Erlang, ExpressionEngine, ExtJs, Fancy, Finale, Firebase, FlaxEngine, Flutter, ForceDotCom, Fortran, FuelPHP, GWT, Gcov, GitBook, GitHubPages, Gleam, Go, Godot, Gradle, Grails, HIP, Haskell, Haxe, IAR, IGORPro, Idris, JBoss, JENKINS_HOME, Java, Jekyll, Joomla, Julia, Katalon, KiCad, Kohana, Kotlin, LabVIEW, LangChain, Laravel, Lasal, Lean, Leiningen, LemonStand, Lilypond, Lithium, Lua, Luau, Magento, Maven, Mercury, MetaProgrammingSystem, ModelSim, Modelica, MoonBit, Nanoc, Nestjs, Nextjs, Nim, Nix, Node, OCaml, Objective-C, Opa, OpenCart, OracleForms, Packer, Perl, Phalcon, PlayFramework, Plone, Prestashop, Processing, PureScript, Python, Qooxdoo, Qt, R, ROS, Racket, Rails, Raku, ReScript, RhodesRhomobile, Ruby, Rust, SCons, SSDT-sqlproj, Salesforce, Sass, Scala, Scheme, Scrivener, Sdcc, SeamGen, SketchUp, Smalltalk, SolidWorks, Solidity-Remix, Stella, SugarCRM, Swift, Symphony, SymphonyCMS, TeX, Terraform, TestComplete, Textpattern, TurboGears2, TwinCAT3, Typo3, Unity, UnrealEngine, VBA, VVVV, VisualStudio, Waf, WordPress, Xojo, Yeoman, Yii, ZendFramework, Zephir, Zig, bun, ecu.test.

More templates for editors and operating systems (e.g. macOS, Windows, VisualStudioCode, JetBrains) live in the repo's `Global/` folder. Templates for less common tools are in `community/`.

23.2 Add one to an existing repo

TERMINAL

```
# Example: Node project
curl -L https://raw.githubusercontent.com/github/gitignore/main/Node.gitignore -o .gitignore
git add .gitignore && git commit -m "Add Node .gitignore"
```

23.3 Already committed a file you meant to ignore?

TERMINAL

```
echo ".env" >> .gitignore
git rm --cached .env          # stop tracking it, but keep the local copy
git commit -m "Stop tracking .env"
```

⚠ DANGER

If that file held a secret (API key, password), it is **still in the Git history**, and anyone who cloned the repo has it. **Rotate the secret now.** Removing it from history afterwards is optional cleanup, not a fix.

23.4 Lines worth adding to almost any project

.GITIGNORE ADDITIONS

```
# Secrets
.env
.env.*
!.env.example
# OS junk
.DS_Store
Thumbs.db
# Editors
.vscode/*
!.vscode/extensions.json
.idea/
# Logs
*.log
```

Releases, Tags, and Versioning

A **tag** marks a specific commit, such as `v1.2.0`. A **release** is a GitHub page attached to a tag, with notes and downloadable files.

24.1 Semantic versioning in one table

PART	BUMP WHEN	EXAMPLE
MAJOR	You break compatibility	1.4.2 → 2.0.0
MINOR	You add features without breaking anything	1.4.2 → 1.5.0
PATCH	You fix bugs only	1.4.2 → 1.4.3

24.2 Make a release (browser)

- 1 On the repo page, click **Releases** (right sidebar) → **Draft a new release**.
- 2 **Choose a tag**
type a new one, like `v1.0.0`, and choose **Create new tag on publish**.
- 3 Click **Generate release notes** to list the PRs merged since the last release, then edit the notes.
- 4 Attach any files, such as built PDFs or zip files.
- 5 Tick **Set as a pre-release** for betas, then **Publish release**.

24.3 Make a release (terminal)

TERMINAL

```
git tag -a v1.0.0 -m "First stable release"
git push origin v1.0.0
gh release create v1.0.0 --generate-notes ./dist/app.zip
gh release list
```

24.4 Automate releases with Actions

.GITHUB/WORKFLOWS/RELEASE.YML

```
name: Release
on:
  push:
    tags: ["v*"]
permissions:
  contents: write
jobs:
  release:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v7
      - run: gh release create "$GITHUB_REF_NAME" --generate-notes
    env:
      GH_TOKEN: ${ github.token }
```

Contributing to Open Source

25.1 Your first contribution, step by step

1 Find a project.

Search issues labeled `good first issue` (`github.com/issues?q=label:"good first issue"+is:open`), or browse a project you already use.

2 Read the rules.

Look for `CONTRIBUTING.md` and `CODE_OF_CONDUCT.md`. Follow them exactly.

3 Claim politely.

Comment on the issue: “I’d like to work on this. Is it still available?” Wait for a maintainer’s OK.

4 Fork.

Click **Fork** on the repo page, which makes your own copy. Or use `gh repo fork owner/repo --clone`.

5 Branch.

```
git switch -c fix-typo-in-readme
```

6 Change, commit, push to *your fork*.

7 Open the PR against the original repo. GitHub offers **Compare & pull request** after you push. Fill in the template and link the issue.

8 Respond to review.

Push more commits to the same branch, and the PR updates automatically.

25.2 Keep your fork up to date

TERMINAL

```
gh repo sync                # sync your fork's default branch with the original repo
# or, by hand:
git remote add upstream https://github.com/OWNER/REPO.git
git fetch upstream
git switch main && git merge upstream/main && git push
```

You can also click **Sync fork** on your fork's page on github.com.

25.3 Etiquette

DO	DON'T
Keep PRs focused on one thing	Mix a bug fix with reformatting the whole file
Explain what and why	Open a PR with an empty description
Be patient, since maintainers are often volunteers	Ping every day
Disclose AI-generated code if the project asks	Submit AI output you haven't read and tested
Report security bugs privately, using the repo's Security tab → Report a vulnerability if it's enabled	Open a public issue with exploit details

Quick Reference

Set up once: `git config --global user.name "Name"` · `git config --global user.email "you@example.com"` · `gh auth login`

Start: `git clone <url>` · or `git init` then `gh repo create <name> --private --source=. --push`

Daily loop: `git status` → `git add .` → `git commit -m "message"` → `git push` · Start of day: `git pull`

Branch: `git switch -c <name>` · `git push -u origin <name>` · `gh pr create` · back to main: `git switch main`

Look back: `git log --oneline` · `git diff` · undo a commit safely: `git revert <id>`

Website: [Settings](#) > [Pages](#) → Deploy from a branch → `main` / root

Glossary

.gitignore	A file listing which files Git should never track.
Branch	A separate line of development, letting you change things without affecting the main version.
Clone	Download a complete copy of a repository, including its history, to your computer.
Commit	A saved snapshot of your files with a message explaining the change.
Fork	Your own copy of someone else's repository on GitHub, used to propose changes.
Git	Free software that tracks the history of files in a folder.
GitHub	A website that hosts Git repositories and adds collaboration tools.
GitHub Actions	GitHub's automation system for running tasks when events happen in a repository.
GitHub CLI	The <code>gh</code> command-line program for working with GitHub from a terminal.
GitHub Pages	Free website hosting straight from a repository.
Issue	A tracked to-do, bug report, or idea on GitHub.
Main	The default branch, which holds the official version of a project.
Markdown	A simple text formatting language used across GitHub.
Merge	Combine the changes from one branch into another.
Merge conflict	When two branches changed the same lines differently, and a person must choose the result.
Passkey	A passwordless sign-in method using your device's fingerprint, face, or PIN.

Personal access token	A password-like string that lets a program act on your GitHub account with limited permissions.
Pull	Download new commits from GitHub to your computer.
Pull request	A proposal to merge one branch into another, with review and discussion.
Push	Upload your commits from your computer to GitHub.
README	The front-page document of a repository, usually <code>README.md</code> .
Remote	A copy of the repository hosted elsewhere, usually GitHub, named <code>origin</code> by default.
Repository	A project folder tracked by Git, together with its full history.
Ruleset	Settings that protect branches, for example requiring pull requests before changes reach main.
Secret	Any password, key, or token that grants access and must be kept private.
Staging area	The list of changes chosen to go into the next commit.
Two-factor authentication	Signing in with a password plus a second proof, such as a phone code.
Workflow	An automation defined in a YAML file for GitHub Actions.
YAML	A plain-text format for settings files, using indentation and <code>name: value</code> pairs.