

// EXAM TO WORK

Git / GitHub

GH-900 GitHub Foundations exam guide

Learn Git. Prepare for GH-900. Practise team workflows.

No experience needed. Plain-language explanations and practical examples to try.

145	9	151
quiz questions	hands-on labs	pages

✓ Local Git drills and live-tested GitHub labs · November 2026 edition

By Jey Singh

Independent study guide

Contents

Only what you need on the job. Nothing else.

00	Start Here	Terminal basics, install Git, your first commit	3

PART 1 Git foundations

01	The Big Picture	Git vs GitHub, the 3 areas, commits, branches, HEAD, inside .git	9
02	Setup Once	Identity, defaults, SSH or HTTPS, Windows, GitHub CLI	14
03	The Daily Loop	status, add, commit, diff, log, .gitignore	19
04	Branches	branch, switch, merge, delete	24
05	Remotes	clone, fetch, pull, push, forks, upstream	27
06	Merge Conflicts	Read them, fix them, stay calm	30
07	Rebase & Clean History	rebase, interactive rebase, force-with-lease	33
08	Undo Anything	restore, reset, revert, reflog, stash, cherry-pick	36
L1	Part 1 Drills	Four practice tickets on your laptop, no account needed	40

PART 2 GitHub the platform (exam core)

09	Accounts, Organizations, Plans	Account types, plans, profile, Desktop and Mobile	41
10	Repositories and Key Files	Key files, licenses, templates, Insights	45
11	Markdown on GitHub	Formatting, task lists, mentions, alerts, Mermaid	51
12	Issues and Discussions	Issues, linking PRs, forms, filters, Discussions	55
13	Pull Requests	GitHub Flow, reviews, PR states, merge buttons	61
14	Projects, Labels, Milestones	Layouts, fields, labels, milestones, workflows	67
15	Notifications, Saved Replies	Reasons, watching, inbox, saved replies	71
16	Pages, Gists and Wikis	Pages sites, secret gists, wikis vs docs	75
17	Codespaces and github.dev	Lifecycle, dev containers, cost, github.dev	79
18	Copilot and Actions on the Exam	Copilot modes and plans, Actions basics	85
19	Security, Roles, Administration	2FA, roles, teams, rulesets, EMU	92
20	Community and Open Source	First PR, Sponsors, Marketplace, InnerSource	100
L2	Part 2 Labs	Four work tickets on your own account	105
C2	Part 2 Exam Checkpoint	Exam facts, self-check, ten mixed questions	108

PART 3 At work

21	Team Workflows	Trunk-based, GitHub Flow, Git Flow, feature flags	110
22	Commit Messages, Tags & Releases	Conventional Commits, SemVer, tags	113
23	GitHub Actions (CI/CD)	Workflow YAML, tests on every PR, secrets	116
24	Security Habits	Secrets, push protection, signed commits, safe workflows	121
25	Power Tools	bisect, blame, log search, worktree, LFS	125
26	AI Agents & What's Coming	Copilot PRs, reviewing AI code, Git 3.0	128
27	Interview Prep	22 common questions with short answers	132
28	Practice Lab	Your first week on a team, an evidence check, a solo retry	134
29	Cheat Sheet	Two pages to keep open at your desk	140
30	When Something Breaks	Common error messages and safe first checks	142
OM	GH-900 Objective Map	Every exam skill, and the chapter that teaches it	144
GL	Glossary	About 90 terms in plain words	146
IX	Index	Topics, commands and features by name	148
	What Next	Practice, the other GitHub exams, corrections	150

How to use this guide

Type every command. Reading Git does not stick. Open a terminal and run the green *Try it* boxes as you go. Each one takes under 2 minutes.

Lines starting with `$` are what you type (do not type the `$`). Grey lines are comments. Dim lines are what Git prints back.

New to the terminal? Start with Chapter 00. It sets up a practice folder, `~/git-lab`, so nothing real can break, and walks you to your first commit. Any Git 2.39 or newer works (`git --version`).

BOX LEGEND

- **Try it** hands-on practice
- **At work** how teams really do it
- **Watch out** mistakes that hurt
- **Interview** asked in interviews
- **New** recent or coming changes
- **On the exam** what GH-900 expects
- **Check yourself** a short quiz ending each chapter

About this edition. *Git / GitHub Study Guide*, November 2026 edition, by Jey Singh. © 2026 Jey Singh. All rights reserved. This file is for your own use: please do not share or resell it. Corrections are listed in ERRATA.md at github.com/azureawscert-a11y/git-github-labs.

Independent publication. Not affiliated with, authorized, sponsored or endorsed by GitHub, Inc. or Microsoft Corporation. GitHub, GitHub Copilot and GitHub Actions are trademarks of GitHub, Inc. Microsoft and Azure are trademarks of the Microsoft group of companies. Git is a trademark of Software Freedom Conservancy. Other names belong to their owners.

All quiz and practice questions are original; this book contains no real exam questions. Researched and drafted with the help of AI tools, then reviewed, tested and edited by the author. GitHub changes often: when a screen differs from the book, trust the screen and check the docs. No guarantee that you will pass an exam or get a job.

00 Start Here

Never used a terminal? Good. This chapter takes you to your first commit in about 30 minutes.

AFTER THIS CHAPTER YOU CAN

- Open a terminal and move between folders
- Install Git and check that it works
- Tell Git your name and email
- Make your first commit
- Get out of any screen you get stuck in

Exam to work

This book assumes you know nothing about Git, GitHub or the command line. It takes you from your first command, to the GitHub Foundations exam (GH-900), to working on a real team. Every chapter builds on the one before, so start here even if you plan to skip ahead later.

Your goal	Your route	Watch for
New to Git	Read every chapter in order. Type every *Try it*. Do not skip the quizzes.	Green **Try it** boxes
Pass GH-900	This chapter, then chapters 1 to 8, then all of Part 2 and its Exam Checkpoint.	Teal **On the exam** boxes
Join a team	Chapters 1 to 8, then 12 to 14 and 19, then Part 3. Finish with the Practice Lab in chapter 28.	Blue **At work** boxes

How each lesson works. You read a short explanation, guess what will happen, run the command, check the result, and fix a mistake on purpose. The guessing and fixing are where you learn the most. Do not skip them.

Step 1: Open a terminal

A **terminal** is a window where you type commands instead of clicking. Git is used through a terminal at work, and the exam expects you to know the basic commands.

Computer	How to open it
Mac	Press **Cmd + Space**, type `Terminal`, press **Enter**.
Windows	Install Git first (Step 3). Then open the Start menu and choose **Git Bash**. Use Git Bash for every command in this book.
Linux	Press **Ctrl + Alt + T**, or find Terminal in your apps.

The line where you type is the **prompt**. On a Mac it ends in `%`. In Git Bash and Linux it ends in `$`. In this book, lines starting with `$` are what you type. Do not type the `$` itself. Press **Enter** after each line.

Step 2: Move around folders

The terminal is always "standing in" one folder, called the **current folder**. Commands work on the folder you are in. Most beginner mistakes come from being in the wrong folder, so learn these six commands first.

Type this	What it does
`pwd`	Prints the folder you are in now ("print working directory").
`ls`	Lists what is in this folder. `ls -a` also shows hidden items, whose names start with a dot.
`mkdir git-lab`	Makes a new folder called `git-lab` here.
`cd git-lab`	Moves you into the folder `git-lab`.
`cd ..`	Moves you up one folder.
`cd ~`	Takes you home. `~` means your home folder, for example `/Users/jey` on a Mac or `/c/Users/jey` in Git Bash.

Three keys save time. Tab finishes a folder or file name for you. The **Up arrow** brings back your last command. **Ctrl + C** stops whatever is running and gives you a fresh prompt.

TRY IT: MAKE YOUR PRACTICE FOLDER

Guess first: after these commands, what will the last `pwd` print?

```
$ cd ~
$ pwd
/Users/jey                      # your home folder, your name
$ mkdir git-lab
$ cd git-lab
$ pwd
/Users/jey/git-lab
```

Every exercise in this book happens inside `~/git-lab`, so nothing real can break. If `mkdir` says the folder already exists, that is fine. Just `cd` into it.

Step 3: Install Git and check it

Computer	How to install
Mac	Type `git --version`. If Git is missing, a window offers to install the "command line developer tools". Click **Install**, wait, then run `git --version` again.
Windows	Download Git for Windows from `git-scm.com` and run it. Keep the default choices. This installs Git, Git Bash and a login helper for GitHub.
Linux	On Ubuntu or Debian: `sudo apt install git`. It asks for your computer password.

```
$ git --version
git version 2.39.5 (Apple Git-154)      # any 2.39 or newer is fine
```

If you see `command not found: git`, Git is not installed yet, or the terminal was open while you installed it. Close the terminal, open a new one and try again.

Step 4: Tell Git who you are

Every commit records a name and an email. Git refuses to commit until you set them. Use your real name and, for now, the example email below. Chapter 2 swaps it for a private GitHub address before you put anything online.

```
$ git config --global user.name "Your Name"
$ git config --global user.email "you@example.com"
$ git config --global init.defaultBranch main     # new projects start on "main"
$ git config --global user.name                   # check it: prints your name
Your Name
```

`--global` means "for every project on this computer". You only do this once per computer.

Step 5: Your first commit

A **repository** (or **repo**) is a folder that Git watches. A **commit** is a saved snapshot of that folder, with a message saying what changed. You are about to make both.

TRY IT: SHOP-APP, YOUR PRACTICE PROJECT

```
$ cd ~/git-lab
$ git init shop-app               # make a new folder and turn it into a repo
Initialized empty Git repository in /Users/jey/git-lab/shop-app/.git/
$ cd shop-app
$ git status                      # "what does Git see?" Run it often.
On branch main
No commits yet
nothing to commit (create/copy files and use "git add" to track)
```

Now make a file. `echo` prints text, and `>` saves it into a file instead of the screen.

```
$ echo "# Shop App" > README.md
$ git status
Untracked files:
        README.md               # Git sees the file but is not saving it yet
```

Guess before you run the next two lines: after `git add`, what will `git status` call the file?

```
$ git add README.md               # choose this file for the next snapshot
$ git status
Changes to be committed:
        new file:   README.md
$ git commit -m "docs: add README" # save the snapshot with a message
[main (root-commit) f43904a] docs: add README
 1 file changed, 1 insertion(+)
$ git log --oneline               # list your commits
f43904a docs: add README
```

On Windows, `git add` may also print `LF will be replaced by CRLF`. That is normal: Git adjusts line endings for Windows (Chapter 2).

The short code (`f43904a`) is the commit's ID, called a **hash**. Yours will be different: every commit gets its own. The `docs:` at the start of the message is a label for the kind of change, such as `docs`, `fix` or `feat` (a new feature). It is optional, but most teams use it. Chapter 22 explains the style. You have just done the loop you will use every working day: **change, add, commit**. Chapter 1 explains what happened inside Git. Chapter 3 goes deeper on each command.

Step 6: Fix a mistake on purpose

Git commands only work inside a repo. Leave the repo and see what happens.

```
$ cd ..                           # up one folder, to ~/git-lab
$ git status
fatal: not a git repository (or any of the parent directories): .git
```

This is the error you will see most often as a beginner. It means "you are in the wrong folder". The fix is always the same: run `pwd` to see where you are, then `cd` into the project.

```
$ pwd
/Users/jey/git-lab
$ cd shop-app
$ git status
On branch main
nothing to commit, working tree clean             # back in the repo, all saved
```

Run `ls -a` here. You will see a hidden folder called `.git`. That folder *is* the repo: it holds every snapshot. Never delete it or edit it by hand.

Stuck? How to get out

You see	Do this
A screen of text ending in `:` or `(END)`	Press **q**. Git shows long output one page at a time. Space shows the next page.
A full-screen editor with `~` down the left side	That is the Vim editor. Press **Esc**, type `:q!` and press **Enter**. Chapter 2 sets an easier editor.
A `>` prompt that will not go away	You left a quote mark open. Press **Ctrl + C** and type the line again.
A command that keeps running	Press **Ctrl + C**.
`command not found`	Check the spelling (`gti` is a common typo), or install the tool.
`not a git repository`	Wrong folder. Run `pwd`, then `cd ~/git-lab/shop-app`.

START OVER SAFELY

If this chapter went wrong and you want a clean start, delete the practice repo and redo Step 5:

```
$ cd ~/git-lab
$ rm -rf shop-app                # deletes the folder for good, no undo
```

`rm -rf` skips the trash. Only use it inside `~/git-lab`, and check the name twice before you press Enter.

MOVE ON WHEN

- `pwd` prints a path ending in `git-lab/shop-app`.
- `git log --oneline` shows one commit: `docs: add README`.
- `git status` says `nothing to commit, working tree clean`.

Chapter 1 looks inside this repo. Chapter 3 adds files to it. Chapter 5 puts it on GitHub.

CHECK YOURSELF

Five questions. Answer them before you read the answers below.

1. You type `git status` and see `fatal: not a git repository`. What is the most likely cause?

A Git is not installed.
B Your commit failed.
C You are not inside the project folder.
D You have no internet connection.

2. Which command shows the folder you are in right now?

A `pwd`
B `ls`
C `cd ~`
D `git status`

3. You made a new file and ran `git status`. It lists the file under `Untracked files`. What do you run so it goes into your next commit?

A `git commit`
B `git init`
C `git log`
D `git add` and the file name.

4. `git log` fills the screen and the last line shows `:`. How do you get your prompt back?

A Close the terminal window.
B Press **q** .
C Type `exit` and press Enter.
D Press Enter until it ends.

5. Why do you set `user.name` and `user.email` before your first commit?

A Every commit records who made it, and Git will not commit without them.
B They are your GitHub login.
C They create your GitHub account.
D They are only needed for the exam.

ANSWERS

1. C. Git commands only work inside a repo. Run `pwd`, then `cd` into the project. If Git were missing you would see `command not found` instead.

2. A. `pwd` prints the current folder. `ls` lists what is inside it, and `cd ~` moves you home.

3. D. `git add` picks the file for the next snapshot. Then `git commit` saves it. You only run `git init` once, to create the repo.

4. B. Git shows long output one page at a time, and **q** quits that view. Closing the window works but loses your place.

5. A. The name and email go into every commit as its author. They are not a login. Logging in to GitHub is set up in Chapter 2.

L1 Part 1 Drills

Four work tickets on your own laptop. Git only: no GitHub account needed.

How the drills work

Each drill is a small mess like the ones you will meet at work. A **setup** script builds a practice repo in `~/git-lab/part1`. A **ticket** tells you what to fix, with "guess first" questions along the way. A **check** script prints PASS or FAIL for each item, with a hint when something is wrong. A **reset** script deletes the practice repo so you can start again.

`every drill` get the labs once, then the same steps

```
$ cd ~/git-lab
$ git clone https://github.com/azureawscert-a11y/git-github-labs.git
$ cd git-github-labs/part1/G1-split-the-commit
$ bash setup.sh       # builds a practice repo, then prints a cd line
# run the cd line it prints, do TICKET.md, then come back here:
$ cd ~/git-lab/git-github-labs/part1/G1-split-the-commit
$ bash check.sh       # PASS or FAIL for each item, with a hint
$ bash reset.sh       # asks you to type yes, then deletes the practice repo
```

- **You need** Git and a terminal (Start Here; Git Bash on Windows), and a text editor such as `nano`.
- **Scripts.** A `.sh` file is a **script**: terminal commands saved in a file, which `bash setup.sh` runs in order. They are short; read one first. Read `TICKET.md` on GitHub or with `cat TICKET.md`.
- **Mind the folder.** `setup.sh` leaves you in the labs folder. Run the `cd` line it prints and check with `pwd` before any `git` command.
- **No GitHub needed.** G2 to G4 make a folder, `team.git`, that plays GitHub. `push`, `fetch` and `pull` work with it as online. Got the labs earlier? `git pull` in the labs folder fetches new drills.

Drill	Ticket (folder part1/G1-...)	After	You practise
G1	Split the commit	Chapter 3	`git add -p`, staged versus unstaged, `git restore`
G2	Your teammate pushed first	Chapter 6	A rejected push, `git fetch`, a conflict with a teammate
G3	Clean up before review	Chapter 7	`git rebase -i`, `fixup`, a safe force push
G4	Wrong branch, urgent fix	Chapter 8	`git stash`, moving commits, a hotfix, `git reflog`

PRACTISE TWICE

Do each drill after its chapter, then again cold a few days later with `bash setup.sh b` (and `b` for check and reset). Finish b without hints and you own the skill.

13 Pull Requests

The single most important workflow you will use. Every line of code goes through a PR.

AFTER THIS CHAPTER YOU CAN

- Name the six steps of GitHub Flow in order
- Take a ticket from branch to merged PR
- Read a PR page and say what state it is in
- Review with comments, suggestions, approvals and change requests
- Keep a PR up to date and pick the right merge button

GitHub Flow in six steps

GitHub Flow is GitHub's name for the simple, branch-based way most teams work. The exam asks for it by name, so learn the six steps in order:

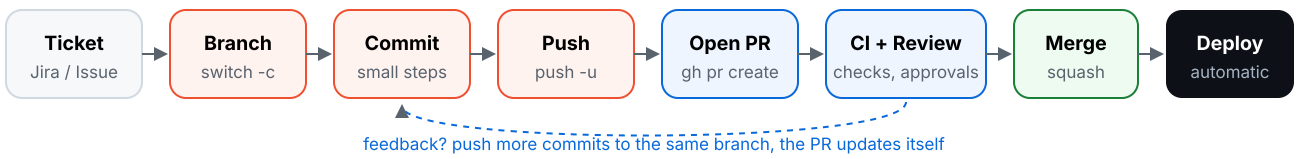
1. **Create a branch** with a short, descriptive name. It is a safe space that does not touch the default branch.
2. **Make changes** and commit them to the branch.
3. **Create a pull request** to ask for feedback. Mark it as a draft if you only want early advice.
4. **Address review comments.** Push more commits. The PR updates by itself.
5. **Merge your pull request** once it is approved.
6. **Delete your branch.** The PR and its commits stay in history.

Other models, such as trunk-based development and Git Flow, are compared in Chapter 21.

ON THE EXAM

Know the six steps and their order, and that GitHub Flow keeps the default branch always **deployable**, meaning ready to go live for users at any moment: work happens on short-lived branches and reaches it only through a reviewed pull request.

The flow on a real team



The full flow in commands

One new word first. **CI** (continuous integration) means automatic checks, such as tests and builds, that GitHub runs every time you push. They show on the PR as **checks**, green for pass and red for fail. Chapter 23 shows how to set them up.

```
$ git switch main && git pull                    # start from fresh main
$ git switch -c feature/JIRA-142-cart-total
  ...code, git add -p, git commit (a few times)...
$ git fetch && git rebase origin/main            # catch up before opening PR
$ git push -u origin feature/JIRA-142-cart-total
$ gh pr create --fill --draft                    # draft = "not ready, but CI please run"
$ gh pr ready                                    # mark ready for review
$ gh pr checks                                   # watch CI
  ...reviewer asks for changes: commit, push. PR updates automatically...
$ gh pr merge --squash --delete-branch           # once approved and green (merge buttons below)
$ git switch main && git pull && git branch -d feature/JIRA-142-cart-total
```

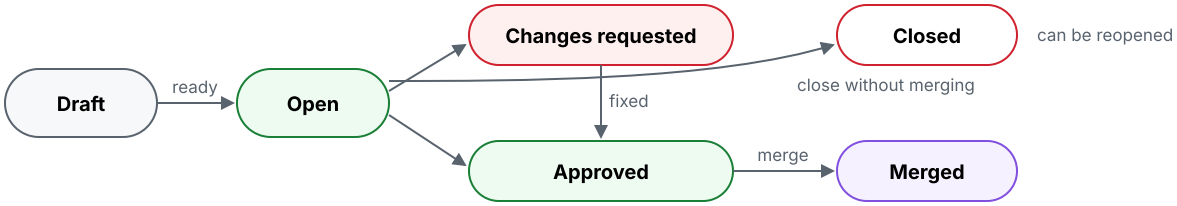
`gh pr` reviewing teammates' work from the terminal

```
$ gh pr list                    # open PRs in this repo
$ gh pr status                  # PRs you made / need your review
$ gh pr checkout 231            # pull PR #231 locally to run and test it
$ gh pr diff 231
$ gh pr review 231 --approve    # or --request-changes -b "reason"
$ gh pr view 231 --web          # open in browser
```

Anatomy of a PR page

Tab	What it is for
Conversation	The description, comments, reviews and a timeline of everything that happened. The **merge box** at the bottom says whether you can merge, and if not, what is missing: reviews, checks, or conflicts.
Commits	Every commit on the branch, in order. Useful when a reviewer asks "what changed since my last look?"
Checks	Results of automated tests, builds and scans, with logs for any that failed.
Files changed	The diff. This is where reviewers comment on lines, suggest changes and submit their review. Tick **Viewed** on each file to track your progress.

PR states



TRY IT: GREEN IS NOT ENOUGH

Start from: `~/git-lab/shop-app` after Chapter 23, with `main` up to date. The ticket says: *"Mug sale: lower the mug from 12.90 to 11.90. Change nothing else."* First, play the author, a hurried teammate or an AI agent:

```
$ git switch -c sale/mug
$ sed -i.bak 's/12\.90/119.00/' prices.txt && rm prices.txt.bak
$ echo "permissions: write-all" >> .github/workflows/ci.yml
$ git commit -am "fix: update mug price" && git push -u origin sale/mug
$ gh pr create --title "Mug sale price" --body "Price change only."
```

Guess first: will the **prices** check pass? It does: 119.00 has two decimals. Now be the reviewer. Open **Files changed** and score the PR:

Ask	This PR
Does it do what the ticket says?	No: 119.00, not 11.90.
Does everything else still work?	Yes: the other prices are untouched.
Does it stay in scope?	No: it also edits the CI workflow.
Do the tests cover the change?	No: the check reads the format, not the amount.
Are permissions safe?	No: `write-all` widens the token (Chapter 24).
Is the description true?	No: it says "price change only".

Comment on each problem line. GitHub does not let you approve or request changes on your own PR, so alone this is a self-review; with a friend, let them open the PR and use **Request changes**. Then fix it, check the diff is one line, and merge:

```
$ git restore --source=main .github/workflows/ci.yml
$ sed -i.bak 's/119\.00/11.90/' prices.txt && rm prices.txt.bak
$ git diff main          # one line: 12.90 becomes 11.90
$ git commit -am "fix: mug sale price 11.90" && git push
```

Green means the checks that exist passed. It does not mean the change is right: that is the reviewer's job.

30 When Something Breaks

Start from the message you see. Look first, change second.

AFTER THIS CHAPTER YOU CAN

- Find the cause of a common Git or GitHub error from its message
- Check safely before you change anything
- Pick the right fix and confirm it worked

Rule one: look before you touch

Most damage happens in the panic after an error, not from the error itself. Before you reset, force or delete anything, run the commands that only read:

```
$ git status            # where am I, what changed, anything in progress?
$ git log --oneline -5  # the last few commits
$ git remote -v         # which repo am I talking to?
$ gh auth status        # which GitHub account am I signed in as?
```

Then find your message in the table; the chapter number shows where the full fix lives. Asking a teammate? Paste the output of these four commands with the exact error.

Message to fix

You see	Check first (safe)	Usually means	Fix, then verify
`fatal: not a git repository`	`pwd`, `ls -a`	You are in the wrong folder.	`cd` into the repo; `git status` works again. (Ch 0)
`Permission denied (publickey)`	`ssh -T git@github.com`	No SSH key, or GitHub doesn't have it.	Add the key, or use the HTTPS URL; `ssh -T` greets you by name. (Ch 2)
`remote: Permission to OWNER/REPO.git denied to USER`, or 403	`gh auth status`	Wrong account, or no write access.	See the worked example below. (Ch 2)
`remote: Repository not found.`	`git remote -v`	Typo in the URL, or a private repo you can't see.	`git remote set-url origin URL`; `git fetch` works. (Ch 5)
`! [rejected] main -> main (fetch first)`	`git fetch`, `git status`	Someone pushed before you.	`git pull`, fix any conflict, push again. Never force a shared branch. (Ch 5)
`GH013: Repository rule violations`	Read the rule named under it	A ruleset blocks direct pushes, or a secret was found.	Push a branch and open a PR, or remove the secret. (Ch 19, 24)

`Your local changes ... would be overwritten by checkout`	`git status`	Unsaved work would be lost by switching.	Commit it, or `git stash`; then switch. (Ch 8)
`Automatic merge failed; fix conflicts`, or `Pulling is not possible because you have unmerged files`	`git status` lists the files	A merge is half done.	Fix the markers, `git add`, `git commit`. Or `git merge --abort`. (Ch 6)
`interactive rebase in progress`	`git status`	A rebase stopped halfway.	Fix, `git add`, `git rebase --continue`. Or `git rebase --abort`. (Ch 7)
`HEAD detached at abc1234`	`git status`, `git log --oneline -3`	You checked out a commit, not a branch.	`git switch main`. Made commits? `git switch -c rescue` first. (Ch 1)
Red X on a pull request	Click **Details** next to the check	A step failed; the log says which.	Fix on your branch and push; the check runs again. (Ch 23)
`Expected — Waiting for status to be reported`	Actions tab: did it run at all?	The required check never started.	Match the job name to the rule, fix `on:`, or merge the workflow to `main`. (Ch 23)
`No such file or directory`, `command not found`	`pwd`, `which git`	Wrong folder, or a tool is not installed.	`cd` to the right place, or install it. (Ch 0)

Worked example: denied on push

You push to your team's repo and get:

```
remote: Permission to acme/shop-app.git denied to jey-personal.
fatal: unable to access 'https://github.com/acme/shop-app.git/':
  The requested URL returned error: 403
```

Read it slowly: GitHub knows who you are (**jey-personal**), it just won't let that account write. The error is about who you are, so the fix is too: no reset, no force push, no new clone. Two likely causes:

Check	If you see	Then
`gh auth status`	Your personal account is active, not your work one.	`gh auth switch`, then `gh auth setup-git`, then push again.
The repo page on GitHub	The right account, but no write access.	Ask the owner for access, or fork and open a PR from your fork.

OM GH-900 Objective Map

Every skill the exam lists, and where this book teaches it.

The list below follows Microsoft's GH-900 skills outline, as of January 2026. Use it the week before your exam: for each line, flip to the chapter and check that you can explain it without looking. Recheck the outline on the Microsoft Learn GH-900 study guide before you book, because Microsoft can change it.

1. Understand Git and GitHub basics (25 to 30%)

Skill on the outline	Where in this book	Page
Purpose and benefits of version control	Ch. 01, Why version control	9
Difference between Git and GitHub	Ch. 01, Git vs GitHub	9
Key Git concepts: repositories, commits, branches	Ch. 01, Four words you must understand	9
GitHub accounts, organizations and enterprise options	Ch. 09, Three account types	41
The GitHub Flow	Ch. 13, GitHub Flow in six steps	61
Markdown in issues and pull requests	Ch. 11, The basics on one page	51
When to use GitHub Desktop and GitHub Mobile	Ch. 09, GitHub Desktop and GitHub Mobile	41

2. Work with GitHub repositories (10 to 15%)

Skill on the outline	Where in this book	Page
Structure and key files (README, LICENSE, CONTRIBUTING, CODEOWNERS, SECURITY)	Ch. 10, The key files	45
Create, organize and maintain repos with templates and branches	Ch. 10, Creating repos	45
Add and manage files	Ch. 10, Adding and managing files	45
Insights, stars, feature previews, metrics, dependency insights	Ch. 10, Seeing how a repo is doing	45
Best practices for maintenance and collaboration	Ch. 10, Maintenance habits	45

3. Collaborate using GitHub (10 to 15%)

Skill on the outline	Where in this book	Page
Issues, pull requests and discussions	Ch. 12, Anatomy of an issue, Discussions	55
Link PRs to issues; templates, filters, assignments	Ch. 12, Linking PRs to issues, Issue templates and forms, Finding things	55
Notifications and how to configure them	Ch. 15, Watching a repo, Settings worth changing	71
Gists, Wikis and GitHub Pages	Ch. 16, Which one?	75

4. Apply modern development practices (10 to 15%)

Skill on the outline	Where in this book	Page
Purpose and capabilities of GitHub Actions	Ch. 18, GitHub Actions (more in Ch. 23)	85
Copilot suggestions, agents, Agent Mode, multi-model	Ch. 18, The modes, Choosing a model	85
Copilot for Individuals, Business and Enterprise	Ch. 18, Plans	85
Codespaces and dev containers	Ch. 17, What a codespace is, Dev containers	79
When to use github.dev instead of a codespace	Ch. 17, github.dev: the instant editor	79

5. Manage projects with GitHub (5 to 10%)

Skill on the outline	Where in this book	Page
Projects and layout options	Ch. 14, Three layouts	67
Configure projects, labels, milestones, workflows	Ch. 14, Fields, Labels, milestones or fields?, Built-in workflows	67
Saved replies and assignees	Ch. 15, Saved replies (assignees: Ch. 14)	71
Project insights for progress and productivity	Ch. 14, Insights	67

6. Understand privacy, security and administration (10 to 15%)

Skill on the outline	Where in this book	Page
2FA and passkeys	Ch. 19, Securing your account	92
Permissions and roles for repositories and organizations	Ch. 19, Repository roles, Organization roles	92
Enterprise Managed Users and organization-wide Copilot policy	Ch. 19, Organization settings, Enterprise accounts and managed users	92
Repository privacy, visibility and branch protection	Ch. 19, Visibility, Protecting branches	92
Organization settings, teams and roles	Ch. 19, Teams, Organization settings	92

7. Explore the GitHub community (5 to 10%)

Skill on the outline	Where in this book	Page
Benefits of open source and GitHub Sponsors	Ch. 20, Why open source, GitHub Sponsors	100
How GitHub supports open-source projects	Ch. 20, How GitHub supports it	100
Following users and organizations	Ch. 20, Following, starring and Explore	100
GitHub Marketplace and its purpose	Ch. 20, GitHub Marketplace	100
InnerSource	Ch. 20, InnerSource	100
Forks, templates and discoverable repositories	Ch. 20, Fork, template or discoverable repo	100

Page numbers are where each chapter starts. Chapter 18 also covers Actions at exam level, and Chapter 23 builds real workflows.

That was the free sample

You have seen Start Here, the drills, a team chapter, the troubleshooting guide and the exam map.

What the full book adds

- **Start Here plus 30 chapters, 151 pages.** Git from your first commit to rebase and undo, all of GitHub for the GH-900 exam, and how teams really work: CI, releases, security, AI agents.
- **145 quiz questions** with explained answers, one quiz per chapter plus a mixed exam checkpoint.
- **4 local Git drills** with a checker that tells you what passed, each with a second variant to retry cold.
- **4 GitHub labs** written as work tickets, live-tested on GitHub.
- **A capstone**: your first week on a team, a handoff, an evidence check someone else can score, and a solo retry with no steps given.
- **Cheat sheet, glossary and index**, and an objective map showing where every exam skill is taught.

The Practice Pack

A separate PDF with two full practice exams, 60 questions each, in the exam's domain weights. Every answer is explained, names the tempting wrong choice and points to the chapter to review.

GET THE FULL BOOK

jeysingh.com/git

The labs are free and open to everyone: **github.com/azureawscert-a11y/git-github-labs**

Independent study guide. Not affiliated with, authorized or endorsed by GitHub or Microsoft. GitHub is a trademark of GitHub, Inc.