Git: the practical notebook

A complete practical course for beginners and everyday collaborators

Keep your work. Understand your history. Build with others.

A guided lesson in the confident notebook style: warm paper, blue ink, clear rules and highlights that help you remember.

What you will learn

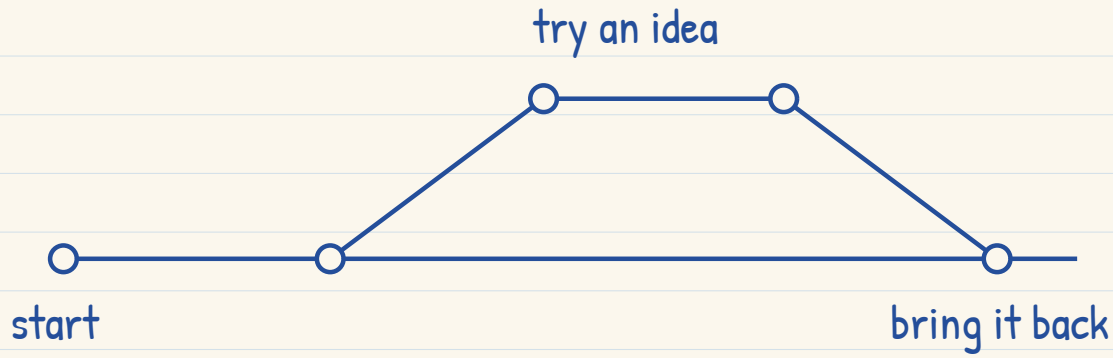
Create a repository, make meaningful commits, work on branches, share through GitHub, resolve conflicts and recover from common mistakes.

How to use this notebook

Read one lesson, type the commands in a disposable practice folder, then answer the checkpoint. Commands use Bash: Linux, macOS or Git Bash on Windows. Replace YOUR-NAME and repository URLs with your own values.

> The handwriting is for learning. Commands stay in a precise monospace font so you can copy them safely.

Edition 1.0 / 19 October 2026 / By depriver.tech



Your learning route

A course you can follow in short sessions

01 / Build the foundation - pages 3-6

Understand Git, configure your identity, create a repository and read changes.

02 / Work with others - pages 7-10

Branches, remotes, a pull-request workflow and conflict resolution.

03 / Stay in control - pages 11-17

Ignore rules, undo choices, stashes, history tools, releases and advanced workflows.

04 / Put it into practice - pages 18-21

A complete exercise, answer checks, troubleshooting and command reference.

05 / Keep learning - page 22

Official references and a glossary you can return to.

Use a practice repository first. A clean git status is your checkpoint before merging, rebasing or switching tasks.

01 / Think in snapshots

Understand what Git is protecting

Git records versions of a project on your computer. A commit identifies a snapshot plus information such as its parent, author and message. You can inspect earlier work without keeping folders named final-final-v3.

Four places to understand

Working tree: the files you edit. Staging area: the exact changes selected for the next commit. Local repository: the history stored in .git. Remote: another repository you exchange commits with.

```
edit files  ->  git add  ->  git commit  ->  git push
working tree   staging      local history   remote
```

A branch is a movable name pointing at a commit. HEAD normally refers to the branch you have checked out. A commit hash identifies a particular commit; branch names can move as you work.

Git and GitHub

Git is the version-control tool. GitHub is one hosting and collaboration service. Local commits work offline; sharing with a remote usually needs a connection and authentication.

> Checkpoint: saving a file is not committing it, and committing it is not uploading it.

git-scm.com/docs/gittutorial

02 / Set up once

Install Git, then choose your commit identity

Install Git using the instructions for your operating system at git-scm.com/install. On Windows, open Git Bash for this course. On macOS or Linux, open your terminal.

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

Replace the example identity. Your name and email become part of future commits. If you use GitHub and prefer email privacy, use the exact no-reply address shown in your account settings.

Project-specific identity

Run the next command inside a repository to override the global email for that project. This does not change old commits.

```
git config user.email "work@example.com"
git help commit
```

Commit identity is not a login. HTTPS credentials or SSH keys authenticate you to your remote host.

git-scm.com/docs/git-config

03 / Your first repository

Create a small notebook project

```
mkdir git-notebook
cd git-notebook
git init -b main
printf "# My Git notebook\n" > README.md
git status
git add README.md
git diff --staged
git commit -m "Add notebook introduction"
git status
```

Before git add, README.md is untracked. After git add, it is staged. After the commit, git status should report a clean working tree. The first commit is now your starting point.

Make the second snapshot

```
printf "\nI am learning Git.\n" >> README.md
git diff
git add README.md
git commit -m "Describe notebook purpose"
git log --oneline
```

Expected result: two commits, newest first. Their hashes will differ from anyone else's example.

git-scm.com/docs/git-init

04 / Read before you commit

See the exact change, then save one clear idea

```
git status --short
git diff
git diff --staged
git diff HEAD
git log --oneline --graph --decorate --all
```

git diff compares working files with the staging area. --staged compares the staging area with the last commit. git diff HEAD shows tracked changes relative to the last commit. New untracked file content is not included until staged.

Stage deliberately

```
git add README.md
git add -p
git restore --staged README.md
```

git add -p lets you choose individual change sections. Unstaging keeps the file edits. If you edit a file after staging it, add it again to include the latest content.

Write useful messages

Prefer "Fix broken guide download link" over "updates". Keep unrelated fixes in separate commits. Check for private keys, passwords and generated files before committing.

Checkpoint: staged and unstaged changes can exist in the same file at the same time.

git-scm.com/docs/git-diff

05 / Branch without fear

Work on one idea without changing main

```
git switch -c feature/checklist
printf "%s\n" "- Review before committing" > checklist.md
git add checklist.md
git commit -m "Add review checklist"
git switch main
git merge feature/checklist
git branch -d feature/checklist
```

The feature branch moves when you commit. Switching back to main restores its snapshot. Merging brings the feature history into the current branch. Use git branch to confirm the current branch before a merge.

Two ordinary merge outcomes

Fast-forward: main has no separate new work, so Git moves its pointer forward. A merge commit: both branches advanced, so Git joins the histories with a commit that has two parents.

```
git log --oneline --graph --all
git branch
```

A branch is cheap. Use clear names such as feature/docs-archive or fix/download-link.

git-scm.com/book/en/v2/Git-Branching-Basic-Branching-and-Merging

06 / Connect to a remote

Publish your local history or clone existing work

Create an empty repository on your hosting service before connecting this existing project. Do not initialise it with a README for this exercise. Replace YOUR-NAME below.

```
git remote add origin https://github.com/YOUR-NAME/git-notebook.git
git remote -v
git push -u origin main
```

origin is a conventional remote name. -u records an upstream relationship so later git push and git pull know the usual destination for this branch. Follow your host's HTTPS credential-manager or SSH setup instructions; do not put tokens in a remote URL.

Starting from an existing repository

```
git clone https://github.com/YOUR-NAME/git-notebook.git
cd git-notebook
git remote -v
```

Clone creates a new folder containing the project and its history. Use clone instead of init when joining a repository that already exists.

Never commit credentials. If authentication fails, check the remote URL, your account access and your host's credential setup.

git-scm.com/docs/git-push

07 / A team workflow

Fetch, review, integrate, then share

```
git switch main
git status
git fetch origin
git log --oneline main..origin/main
git pull --ff-only
git switch -c feature/lesson-notes
```

Fetch updates your local view of the remote without merging into your working branch. origin/main is that last fetched view. Pull fetches and then integrates. --ff-only refuses to combine diverged histories automatically.

Open a pull request

Edit and test your files, stage only the intended changes, commit, then push the feature branch. On your host, open a pull request into main. Explain the problem, the change and how you tested it.

```
git add README.md
git commit -m "Add lesson notes"
git push -u origin feature/lesson-notes
```

Respond to feedback with additional commits. After the reviewed change is merged on the host, switch to main and run git pull --ff-only. A squash merge may leave a different commit history from a normal merge.

A pull request is a hosting-service workflow. It is not a command built into Git.

git-scm.com/docs/git-fetch

08 / Resolve a conflict

A conflict asks you to choose the right final content

When branches change the same lines differently, Git may pause the merge. Start from clean work so it is easier to recover. Read git status and open every file listed as unmerged.

```
<<<<<<< HEAD
Title from your current branch
=======
Title from the incoming branch
>>>>>>> feature/lesson-notes
```

Edit this region into the final text you want. Keep one version, combine them, or write a better version. Remove all marker lines, then test the result.

```
git add README.md
git diff --staged
git merge --continue
git status
```

Need to stop?

```
git merge --abort
```

Use --abort during an unresolved merge. For a rebase conflict, use git rebase --continue or git rebase --abort instead. Resolving a conflict is a content decision, not simply clicking "accept both".

git-scm.com/docs/git-merge

09 / Keep your repository clean

Ignore generated files and keep secrets out

Create .gitignore in the project root. These example patterns suit many web projects; adjust them to your stack.

```
# .gitignore
node_modules/
dist/
.env
.env.*
!.env.example
*.log
.DS_Store
```

Ignore rules affect untracked files. If .env was already tracked, adding it to .gitignore does not remove it from Git history.

```
git rm --cached .env
git add .gitignore
git commit -m "Stop tracking local environment file"
```

Run that removal only if .env is tracked. It keeps the local file while removing it from the next snapshot. If a secret was exposed, revoke or rotate it promptly; deleting the latest copy does not erase previous commits.

For large applications and ZIP downloads, use release assets or object storage. Keep source and metadata in Git; use Git LFS only when it fits your hosting workflow.

git-scm.com/docs/gitignore

10 / Undo the right thing

Choose the scope before choosing the command

Unstage, but keep your edits

```
git restore --staged README.md
```

This changes the staging area, not the working copy. It is useful when you selected too much for a commit.

Discard unstaged edits to one file

```
git diff -- README.md
git restore -- README.md
```

The second command replaces the working copy from the index by default. Unstaged edits are lost. Review or copy them elsewhere first if you might need them.

Bring back a version from a commit

```
git restore --source=COMMIT_HASH -- README.md
git diff -- README.md
```

Replace COMMIT_HASH with an actual hash. This restores the working file from that snapshot; inspect it, then stage and commit if you want to keep it.

restore changes file content or staging. revert records an inverse commit. reset can move the current branch. They solve different problems.

git-scm.com/docs/git-restore

11 / Repair committed work

Prefer a new correction when history is shared

Reverse an ordinary shared commit

```
git status
git revert COMMIT_HASH
git push
```

Use a clean working tree and a real commit hash. Revert records a new inverse change and preserves the original history. It can conflict; resolve and use git revert --continue, or git revert --abort. Reverting a merge needs special care with its mainline parent.

Fix your latest unpushed commit

```
git add README.md
git commit --amend --no-edit
```

Amend replaces the latest commit with a new one, including staged changes. Omit --no-edit to revise the message. Do this only when rewriting that commit will not disrupt collaborators.

Reset: know the difference

--soft moves the branch while keeping staging and files. The default --mixed also resets staging but keeps working edits. --hard resets tracked files too and can destroy uncommitted work. Do not use it as a routine fix.

Avoid force-pushing shared main. Rewriting published history needs an agreed team process.

git-scm.com/docs/git-revert

12 / Pause and recover

Use a stash for interruptions and reflog for lost pointers

```
git stash push -u -m "Draft lesson edits"
git stash list
git stash show -p "stash@{0}"
git stash apply "stash@{0}"
```

-u includes untracked files, but not ignored files. apply restores the changes and keeps the stash entry. After checking the result, remove it with git stash drop "stash@{0}". Applying a stash can produce conflicts.

Find a commit you lost track of

```
git reflog
git show COMMIT_HASH
git branch rescue/lesson COMMIT_HASH
```

Reflog records recent local reference movements. If the commit still exists, creating a branch gives it a lasting name. Reflog is local and entries expire; it is not a backup of every unsaved edit.

> For important unfinished work, a commit on a temporary branch is easier to explain and back up than a forgotten stash.

git-scm.com/docs/git-stash

13 / Read the project history

Find what changed and why

```
git log --oneline --graph --decorate --all
git log -p -- README.md
git show COMMIT_HASH
git blame README.md
git log --grep="download" --oneline
```

Log shows history; show inspects a commit; blame associates current lines with the last changes that touched them. Use blame to find context, not to assign fault.

Find the first bad commit

```
git bisect start
git bisect bad
git bisect good GOOD_COMMIT_HASH
# Test the checked-out version, then mark it:
git bisect good
# Or: git bisect bad
# Repeat until Git identifies the change.
git bisect reset
```

Begin with clean work, one known bad revision and one known good revision. Bisect tests intermediate history to narrow the search. Use git bisect skip if a revision cannot be tested.

A reproducible test makes history useful. Record the symptom before starting your search.

git-scm.com/docs/git-bisect

14 / Release a known version

Tag a commit so you can find the shipped code

```
git switch main
git pull --ff-only
# Run your project checks first.
git tag -a v1.0.0 -m "First notebook release"
git push origin v1.0.0
git show v1.0.0
```

An annotated tag gives a release a stable name and message. Pushing a branch does not automatically push all tags. Publish the specific tag you intend to release.

Source versus download files

The Git tag identifies source history. A hosting-service release can attach built applications or ZIP files. Record the version, target platform, changes and installation steps alongside those files.

Make a source-only ZIP

```
git archive --format=zip --output=source-v1.zip v1.0.0
```

This exports tracked files from the tag. It does not include the .git history, ignored build files or a compiled application unless those files were tracked.

For your Docs Archive, keep a title, version, file type, size and short description with every download.

git-scm.com/docs/git-tag

15 / Beyond the basics

Understand these tools before adding them to daily work

Rebase a private feature branch

```
git fetch origin
git switch feature/lesson-notes
git rebase origin/main
```

Rebase replays your branch commits on a new base and creates new commit identities. Use it for your own unshared work unless your team has agreed otherwise. Conflicts pause the replay; stage resolutions and use git rebase --continue, or --abort.

Copy one change

```
git cherry-pick COMMIT_HASH
```

Cherry-pick applies the selected commit's change to your current branch as a new commit. Confirm the destination branch and test it; copying a change can introduce dependencies or conflicts.

Work on another branch in another folder

```
git worktree add ../notebook-review main
git worktree list
```

Worktrees share repository objects while keeping separate working directories. The branch must not already be checked out in another worktree. Remove a finished, clean worktree with git worktree remove ../notebook-review.

git-scm.com/docs/git-rebase

16 / Practice lab: build it

A fresh local exercise; no GitHub account required

Run these commands in a new parent directory. This lab creates its own folder and sets an example identity only inside it. Do not use an existing project folder.

```
mkdir git-notebook-lab
cd git-notebook-lab
git init -b main
git config user.name "Git Learner"
git config user.email "learner@example.invalid"
printf "# Git lab\n" > README.md
git add README.md
git commit -m "Start Git lab"
git switch -c feature/notes
printf "One idea per commit.\n" > notes.txt
git add notes.txt
git commit -m "Add commit reminder"
git switch main
git merge feature/notes
git log --oneline --graph --all
```

What should you see?

Two commits, a clean working tree and notes.txt on main. Because main did not diverge, this merge fast-forwards. Keep this folder for the next page.

Extra task: edit notes.txt, stage it, then unstage it without losing the edit. Inspect status at every step; restore the edit before continuing.

17 / Practice lab: resolve it

Create a deliberate conflict, then fix it

```
git switch -c feature/title
printf "# Team notebook\n" > README.md
git add README.md
git commit -m "Use team title"
git switch main
printf "# Personal notebook\n" > README.md
git add README.md
git commit -m "Use personal title"
git merge feature/title
```

Expected: a content conflict in README.md. Open it and inspect the markers. For this exercise, choose a combined title, then complete the merge:

```
printf "# Shared Git notebook\n" > README.md
git add README.md
git commit -m "Resolve notebook title"
git status
git log --oneline --graph --all
```

Answer checks

The final README contains the combined title, status is clean, and the graph shows a merge commit joining the two histories. Try git show HEAD to inspect it.

18 / When something goes wrong

Read the message, then choose the smallest fix

Not a Git repository

Check your current folder with pwd and ls -a. Use cd to enter the project. Do not initialise random parent folders to silence the message.

Nothing to commit

The files may be unchanged, ignored or outside this repository. Inspect git status --short and git check-ignore -v FILE.

Push rejected / non-fast-forward

Fetch and inspect remote changes. Integrate using your team's merge or rebase policy, test, then push. Do not blindly force-push.

Local changes would be overwritten

Commit, stash or deliberately discard those changes before switching or merging. Git is protecting your working files.

Detached HEAD

You checked out a commit rather than a branch. Keep new work with git switch -c rescue/my-work before leaving it.

An editor opened unexpectedly

Git may be asking for a commit message. Save and close the editor to continue, or cancel intentionally. Use -m for simple commit messages.

19 / Your desk reference

The everyday loop: inspect, change, review, commit, share

Start
```
git init -b main / git clone URL
```

Inspect
```
git status / git diff / git diff --staged
```

Stage
```
git add FILE / git add -p
```

Save
```
git commit -m "Describe the change"
```

History
```
git log --oneline --graph --all
```

Branch
```
git switch -c feature/name
```

Combine
```
git switch main / git merge feature/name
```

Remote
```
git fetch origin / git pull --ff-only
```

Share
```
git push -u origin BRANCH
```

Unstage
```
git restore --staged FILE
```

Undo shared
```
git revert COMMIT_HASH
```

Pause
```
git stash push -u -m "Draft"
```

Recover
```
git reflog / git branch rescue HASH
```

Release
```
git tag -a v1.0.0 -m "Release"
```

Before pushing: correct branch, reviewed diff, no secrets, checks passed, clear commit message.

20 / Keep this page nearby

Glossary, official references and next steps

Repository: project history. Commit: a recorded snapshot. Index: staging area. Branch: a movable commit reference. HEAD: your current position. Remote: another repository. Upstream: the branch used for usual tracking. Conflict: a change Git needs you to resolve.

Official references

Install and configuration
https://git-scm.com/install

Git tutorial
https://git-scm.com/docs/gittutorial

Branching and merging
https://git-scm.com/book/en/v2/Git-Branching-Basic-Branching-and-Merging

Restore, revert and reset
https://git-scm.com/docs/git-restore

Remote workflow
https://git-scm.com/docs/git-fetch

Stash and recovery
https://git-scm.com/docs/git-reflog

Advanced workflow
https://git-scm.com/docs/git-rebase

Command reference
https://git-scm.com/docs

Command documentation reviewed 19 October 2026. This practical course covers everyday Git plus selected advanced tools; it is not a replacement for the full command reference.

Your next step: use a feature branch and a reviewed pull request for one real improvement.