GIT & GITHUB FOR DEVELOPERS

A practical guide to tracking changes and collaborating on software

LEARNSAPPEAL | ORIGINAL COURSE

Save useful history. Share changes clearly. Build together.

Learn Git locally, connect a repository to GitHub, and follow a reviewable workflow from idea to merged change.

What you need

A computer, a text editor, and Git installed. GitHub is a hosting and collaboration service; Git is the version control tool on your computer. You can practice the local Git lessons without a GitHub account. Examples use the command line, but the same ideas apply in graphical tools.

Course map: Work safely with Git (Modules 1-6), then collaborate and maintain a project on GitHub (Modules 7-12). Includes command examples, exercises with answers, projects, and a glossary.

How to use this course

Use a practice folder first. Run a command, read its output, and check git status before and after each step. Make small commits that describe one coherent change. When uncertain, inspect status and diff before choosing a command that rewrites history or discards files.

By the end: explain repositories, commits, branches, and remotes; create and inspect commits; merge work; handle simple conflicts; push a branch; open a pull request; review changes; and maintain a project with clear documentation and safe credentials.

Course map

- 1. Version control and Git
- 2. Create a repository
- 3. Stage and commit changes
- 4. Inspect history and undo carefully
- 5. Branches and focused work
- 6. Merge changes and resolve conflicts
- 7. Remotes, clone, push, and pull
- 8. Pull requests and code review
- 9. Issues, discussions, and project flow
- 10. README, .gitignore, and releases
- 11. GitHub Actions and automation
- 12. Security and capstone workflow

WORK SAFELY WITH GIT

1. Version control and Git

Version control records changes over time so you can compare versions, restore earlier work, and see why a change was made. Git stores project history in a repository on your computer. GitHub hosts repositories online and adds collaboration features such as pull requests, issues, and automation. A commit is a saved snapshot with a message and parent history.

Example

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

Check your understanding

- What does Git track?
- How does GitHub relate to Git?

Suggested answers

1. Changes to files and project history.
2. GitHub hosts repositories and adds collaboration tools; Git works locally too.

2. Create a repository

A repository contains files and their history. Start a new repository with git init, or copy an existing one with git clone. The .git directory holds Git metadata. Add a .gitignore file early for generated files, local settings, and secrets that should not be committed. Inspect the folder before initializing Git.

Example

```
mkdir course-catalog
cd course-catalog
git init
git status
# Or get an existing project:
# git clone <repository-url>
```

Check your understanding

- What does git init create?
- When is git clone useful?

Suggested answers

1. A new local Git repository in the current folder.
2. When copying an existing remote repository and its history.

3. Stage and commit changes

Git work has three useful places to understand: the working tree, the staging area, and committed history. Edit files in the working tree; stage selected changes with git add; save them in a commit with git commit. Staging lets you choose exactly which changes belong together. Review the staged diff before committing.

Example

```
git status
git add README.md
git diff --staged
git commit -m "Add course catalog overview"
```

Check your understanding

- What does staging let you choose?
- Which command previews staged changes?

Suggested answers

1. Which file changes go into the next commit.
2. git diff --staged.

4. Inspect history and undo carefully

Use status to understand the current state, diff to inspect edits, and log to review commits. To discard unstaged changes in a file, restore it only after confirming you do not need them. To unstage a file, restore it from the index. Avoid reset --hard or force-push unless you understand the effect and have a safe recovery plan.

Example

```
git status
git diff
git diff --staged
git log --oneline --decorate
# Unstage a file but keep its edits:
git restore --staged README.md
```

Check your understanding

- How can you inspect edits before committing?
- What does restore --staged do?

Suggested answers

1. Use git diff and git status.
2. Removes it from staging while keeping working-tree edits.

5. Branches and focused work

A branch is a movable name for a line of development. Branches let you work on a change without immediately changing the main branch. Use short, descriptive names and keep each branch focused. Commit related steps, then push the branch when you want to share it.

Example

```
git switch -c add-course-search
# Edit files and commit a focused change
git add .
git commit -m "Add course search field"
git switch main
```

Check your understanding

- Why use a branch?
- What should a branch name communicate?

Suggested answers

1. To isolate work until it is ready to combine.
2. The feature or fix being worked on.

6. Merge changes and resolve conflicts

A merge combines histories. If changes do not overlap, Git may combine them automatically. If both branches changed the same part, a conflict needs a person to choose the correct final content. Open the marked file, remove conflict markers, review the result, run checks, stage the resolved file, and finish the merge.

Example

```
git switch main
git merge add-course-search
# If a conflict occurs:
# 1. open the marked file and edit the final content
# 2. remove conflict markers
# 3. run checks and inspect the result
git add path/to/resolved-file
git commit
```

Check your understanding

- What is a merge conflict?
- What should you do after editing a conflicted file?

Suggested answers

1. A change Git cannot combine automatically.
2. Review, test, stage the resolved file, then complete the merge.

COLLABORATE AND MAINTAIN

7. Remotes, clone, push, and pull

A remote is a named link to another repository, often hosted on GitHub. Clone downloads a repository and sets up its remote. Push sends local commits to a remote branch. Fetch downloads remote history without integrating it; pull fetches and integrates changes. Before pushing shared work, check the branch and commits you are sending.

Example

```
git remote -v
git clone https://github.com/OWNER/REPO.git
git switch -c update-course-list
git push -u origin update-course-list
# Fetch remote changes for inspection:
git fetch origin
```

Check your understanding

- What does git fetch do?
- What does -u set up on a first push?

Suggested answers

1. Downloads remote commits and references without merging them into your current branch.
2. An upstream tracking relationship for the branch.

8. Pull requests and code review

A pull request (PR) proposes merging changes from one branch into another. It provides a description, a diff, discussion, review, and automated check results. Keep the change focused, explain why it is useful, list how you checked it, and mention known risks. Respond to feedback with new commits and merge only when required checks and approvals are satisfied.

Example

```
PR description:
What changed: add a topic filter to the course list.
Why: learners can find relevant material faster.
Checks: manually reviewed empty and matching results.
Follow-up: add analytics after privacy review.
```

Check your understanding

- What is a pull request for?
- Why describe checks in the PR?

Suggested answers

1. Proposing, discussing, reviewing, and merging a branch's changes.
2. So reviewers can understand what validation was performed.

9. Issues, discussions, and project flow

Issues can track bugs, questions, and planned work. A good report includes steps to reproduce, expected behavior, actual behavior, and relevant context without exposing private data. Labels and milestones help organize work. Link a pull request to its issue so people can follow the change from problem to resolution.

Example

```
Issue report:
Title: Course filter misses the no-results state
Steps: choose a topic with no matching courses
Expected: an explanatory empty message
Actual: blank result area
Context: browser and page version
```

Check your understanding

- What makes a bug report actionable?
- Why link a PR to an issue?

Suggested answers

1. Clear reproduction steps and expected versus actual behavior.
2. It connects the fix to the reason for the work.

10. README, .gitignore, and releases

A README helps someone understand, install, and use a project. Include purpose, requirements, setup, usage, and contribution notes. .gitignore prevents selected untracked files from being added accidentally, but it does not remove a file already committed. Tags can mark a release point; release notes summarize changes and limitations.

Example

```
README sections:
# Project name
## What it does
## Requirements and setup
## How to run it
## How to contribute
## License and contact

# Example .gitignore entries:
.env
__pycache__/
node_modules/
```

Check your understanding

- Does adding a file to .gitignore untrack it if it was committed already?
- What should release notes explain?

Suggested answers

1. No. It only affects files not already tracked.
2. What changed, important fixes, known limitations, and upgrade notes.

11. GitHub Actions and automation

GitHub Actions can run automated workflows when repository events occur, such as a push or pull request. A workflow file describes triggers and jobs; jobs contain steps that run commands or reusable actions. Start with a simple check such as formatting or tests. Protect secrets, limit permissions, and review third-party actions before relying on them.

Example

```
Workflow idea:
on pull request:
  install dependencies
  run formatter check
  run tests
  report pass or failure

A failing required check can block merge until fixed.
```

Check your understanding

- What can trigger a workflow?
- Why review action permissions and secrets?

Suggested answers

1. Events such as pushes or pull requests.
2. A workflow can access code and credentials; limit its access.

12. Security and capstone workflow

Do not commit passwords, tokens, private keys, or personal data. Use a credential manager or protected secret settings. If a secret is committed, removing it from the latest file is not enough; rotate or revoke it immediately and follow the project response steps because it may remain in history. Use branch protections and review changes before merging.

Example

```
Capstone: publish a course update
1. Create a branch for one focused feature.
2. Commit a README and course link change.
3. Push the branch to GitHub.
4. Open a PR with context and checks.
5. Review the diff and automation results.
6. Merge, then verify the main branch.
```

Check your understanding

- What should you do if a token is committed?
- Why inspect the diff before merging?

Suggested answers

1. Revoke or rotate it immediately, then follow incident steps; assume it may remain in history.
2. To catch unintended changes and confirm the proposal matches its description.

Practice projects

1. Personal portfolio repository

Create a repository with a README and a small page. Add a .gitignore. Make separate commits for page structure and styling, then inspect the history and diffs.

2. Collaborative course update

Create a feature branch, update a course description, push it, and draft a pull request with a summary and verification steps. Review the diff as if you were a teammate.

3. Team contribution guide

Write a short guide for branch naming, commit messages, pull request descriptions, review expectations, and how to report a bug.

Project checklist
Is the change focused? Can someone understand the history? Is the diff reviewed? Are secrets excluded? Is the pull request clear? Are checks and recovery steps documented?

Command quick reference

Command	Purpose
git status	Show current changes and branch
git diff	Review unstaged edits
git add FILE	Stage selected file changes
git commit -m MESSAGE	Save staged changes
git switch -c NAME	Create and switch to a branch
git log --oneline	Review compact history
git fetch origin	Download remote history for inspection
git push -u origin BRANCH	Share branch and set upstream

Glossary

Term	Meaning
Branch	A movable reference to a line of project history.
Commit	A saved snapshot with a message and parent history.
Diff	A view of changes between versions or states.
Git	A distributed version control system.
GitHub	A hosting and collaboration platform for Git repositories.
Merge	Combining changes from different lines of history.
Pull request	A proposal to merge one branch into another, with discussion and review.
Remote	A named reference to another repository.
Repository	Project files and their recorded history.
Staging area	The place where selected changes are prepared for the next commit.
Workflow	An automated sequence of jobs triggered by repository events.
Working tree	The checked-out files currently visible in the project folder.

Further study

These official references cover Git concepts and GitHub collaboration. Interfaces and product details can change, so use current documentation for settings in your account.

- Pro Git. *The entire book.* https://git-scm.com/book/en/v2
- GitHub Docs. *About pull requests.* https://docs.github.com/en/pull-requests/get-started/about-pull-requests
- GitHub Docs. *Understanding GitHub Actions.* https://docs.github.com/en/actions/get-started/understanding-github-actions
- GitHub Docs. *About protected branches.* https://docs.github.com/en/repositories/configuring-branches-and-merges-in-your-repository/managing-protected-branches/about-protected-branches

Course completion checklist

I can create and inspect a repository; stage and commit focused changes; use branches; merge and resolve conflicts; connect a remote; contribute through a pull request; review changes; document a project; and protect credentials and important branches.