Case Study 1 Exercises: Setting Up a Project

- Create a new **public** repository named `my-blog-practice`, initialize with a README, then clone it locally using both GitHub Desktop and the CLI.
- In your local clone, build a blog folder structure (`content/`, `assets/`, `drafts/`) and add a `.gitignore` that excludes system files and build artifacts. Commit and push these changes with a descriptive message.
- Add a new Markdown post under `content/`, commit only that file, then view your commit history in GitHub Desktop.
- Introduce an unwanted file (e.g. `temp.log`), verify it's untracked, then update `.gitignore`, commit the change, and confirm the file no longer appears in "Changes."

Case Study 2: Branching, Pull Requests & Merge Conflicts

Building on the **my-blog-practice** repo you initialized in Case Study 1, you'll work on two parallel style enhancements to the same CSS file, open draft and regular pull requests, handle a merge conflict, and then practice fast-forward and squash merges.

1. Prepare your styling file
    - In your local clone, ensure you have an `assets/styles` folder (or create it), and a CSS file named `main.css` inside.
2. Create and push two feature branches
    - From `main`, create a branch called `feature/subscribe-button`
        - In `main.css`, add rules for a "Subscribe" button (e.g. background, padding, hover state).
        - Commit and push to origin.
    - Return to `main`, then create `feature/dark-theme`
        - In the *same* `main.css`, add or override rules for a dark-mode color scheme (e.g. dark background, light text).

        - Commit and push to origin.

3. Draft and convert your first pull request

    - On GitHub, open a **draft** pull request from `feature/subscribe-button` into `main`.
    - When you're ready for feedback, click **"Ready for review"** to convert it into a regular PR.

4. Merge the first PR, then handle a conflict in the second

    - After review, merge the subscribe-button PR via the web UI.
    - Next, open the PR for `feature/dark-theme`. GitHub will report a merge conflict in `main.css`.
    - Follow GitHub's instructions to pull `feature/dark-theme` locally, merge `main` into it, resolve the conflict by combining both your button and dark-theme rules, commit the resolution, and push.
    - Return to GitHub and complete the merge once the conflict is cleared.

5. Fast-forward and squash merges from the CLI

    - **Fast-forward merge**:
        - Locally, create a short-lived branch (e.g. `quick-fix/header-margin`), make a trivial tweak to `main.css`, commit, then switch back to `main` and merge without a new merge commit (`git merge quick-fix/header-margin`). Push `main`.
    - **Squash merge**:
        - Suppose you have a longer-lived branch `feature/footer-widgets` with multiple commits. Use the GitHub CLI or web UI to perform a squash merge (`gh pr merge --squash --delete-branch`) so that all those commits become a single commit on `main`.

Case Study 3: Issues, Project Boards & Community Features

Continuing in the same repo, this study demonstrates how to track work, manage milestones, document your project, and engage via discussions.

- Create three GitHub Issues in `my-blog-practice`: label one as `bug`, one as `enhancement`, and one as `question`, assigning each to yourself.
- Set up a **Project board** named “Sprint Board” with columns **To do**, **In progress**, and **Done**. Add your Issues as cards, then move at least one card through all three columns.
- Define a Milestone called “Sprint 1” with a due date one week away and attach two of your Issues to it.
- In the repository’s **Wiki**, add an **Installation** page that explains how to clone the repo and start working. Link to this page from your README.
- Enable **Discussions** in the repository settings, start a thread titled “How to contribute?”, and post an answer to your own question.
- (If your repo belongs to an organization) create a team with **Write** access, then invite a collaborator with **Read**-only permissions.