Git & GitHub for Absolute Beginners

A Step-by-Step Introduction to Version Control and Collaboration

Preface

Welcome to your journey into the world of GitHub and version control! Whether you're a complete beginner taking your first steps into programming, a student working on projects, or a professional looking to modernize your workflow, this book will guide you from absolute zero to confidently using GitHub for your projects.

Why This Book Exists

In today's digital world, **GitHub has become the backbone of software development and collaboration**. It's where millions of developers store their code, contribute to open-source projects, and work together on everything from simple scripts to complex applications. Yet for many beginners, GitHub can seem intimidating—a maze of unfamiliar terms like "repositories," "commits," and "pull requests."

This book exists to bridge that gap. We believe that **everyone should have access to GitHub's powerful collaboration tools**, regardless of their technical background. By the end of this journey, you'll not only understand how to use GitHub effectively but also appreciate why it has revolutionized how people work together on digital projects.

What You'll Learn

This book takes a **hands-on, step-by-step approach** to mastering GitHub and its underlying technology, Git. You'll start by understanding the fundamental concepts of version control—why it matters and how it solves real problems that every developer faces. From there, you'll learn the essential distinction between Git and GitHub, setting the foundation for everything that follows.

The first half of the book focuses on **Git fundamentals**—the local version control system that powers GitHub. You'll learn to track changes, make commits, work with branches, and understand your project's history. These skills are crucial because they form the foundation of everything you'll do on GitHub.

The second half shifts focus to **GitHub itself**—where the real magic of collaboration happens. You'll discover how to connect your local Git repositories to GitHub, collaborate with others, create and review pull requests, and participate in the global development community. By the end, you'll have all the tools you need to contribute to projects, manage your own repositories, and work effectively in team environments.

How This Book Is Different

Unlike other technical books that assume prior knowledge, **this guide truly starts from the beginning**. Every concept is explained in plain English, with practical examples that relate to real-world scenarios. We've included comprehensive appendices with cheat sheets, common error explanations, workflow diagrams, and practice exercises—everything you need to reinforce your learning.

The book follows a **progressive learning structure**: each chapter builds on the previous one, ensuring you develop a solid foundation before moving to more

advanced topics. You'll never encounter a concept that hasn't been properly introduced and explained.

Who This Book Is For

This book is designed for **absolute beginners** who want to master GitHub. You might be:

- A new programmer learning your first programming language
- A student working on coding assignments or projects
- A professional transitioning into a technical role
- Someone interested in contributing to open-source projects
- A team member looking to improve collaboration workflows

No prior experience with Git or GitHub is required—just curiosity and willingness to learn.

Acknowledgments

This book exists thanks to the countless developers who have shared their knowledge through blog posts, tutorials, and community forums. Special appreciation goes to the GitHub team for creating such an accessible platform that has democratized software collaboration, and to the Git community for building the robust version control system that makes it all possible.

How to Use This Book

Start from Chapter 1 and work through each chapter sequentially—the concepts build on each other deliberately. Practice the examples as you read, and don't hesitate to experiment beyond what's shown. The appendices are designed as reference materials you can return to whenever needed.

Your GitHub journey starts now. Let's begin building the skills that will transform how you work with code and collaborate with others.

Happy coding!

Nico Brandt

Table of Contents

Chapter 1: What Version Control Is (Explained Simply)

Introduction: The Problem Every Developer Faces

Picture this scenario: You're working on an important project, whether it's a website, a mobile application, or even a simple document. You've spent hours crafting the perfect code, and everything works beautifully. Then, in a moment of inspiration, you decide to add a new feature. You modify several files, restructure some code, and suddenly nothing works anymore. Panic sets in as you realize you can't remember exactly what you changed, and you don't have a backup of your working version.

This nightmare scenario happens to developers, writers, designers, and anyone who works with digital files every single day. It's a universal problem that has plagued creative professionals since the dawn of computing. Before version control systems like GitHub existed, people tried various desperate measures to solve this problem. Some would create folders with names like "Project_Final," "Project_Final_v2," "Project_Final_ACTUALLY_FINAL," and "Project_Final_USE_THIS_ONE." Others would email themselves copies of their work or save files with timestamps in their names.

These makeshift solutions were clunky, unreliable, and often led to more confusion than clarity. Imagine trying to collaborate with a team using these methods.

You'd end up with dozens of versions floating around, no clear understanding of which version was the most recent, and no way to track who made what changes or when. The chaos was overwhelming, and productivity suffered tremendously.

This is where version control systems, particularly GitHub, revolutionized how we work with digital files and collaborate with others. GitHub didn't just solve these problems; it transformed the entire landscape of software development and digital collaboration. Understanding what version control is and how GitHub implements it is fundamental to modern development practices.

Understanding Version Control: The Digital Time Machine

Version control is essentially a sophisticated system that tracks and manages changes to files over time. Think of it as a digital time machine that allows you to travel back to any point in your project's history, see exactly what changed, who made the changes, and why those changes were made. It's like having a detailed diary of your project's evolution, complete with the ability to undo mistakes and explore different paths of development.

At its core, version control solves several critical problems that plague anyone working with digital files. First, it provides a complete history of your project. Every time you make a change and save it to the version control system, it creates a snapshot of your entire project at that moment. This snapshot, called a "commit" in GitHub terminology, includes not just the files themselves, but also metadata about when the change was made, who made it, and a description of what was changed.

The beauty of this system lies in its granularity and precision. Unlike simply saving different versions of files manually, version control systems like GitHub track changes at the line level within files. This means that if you change a single word in

a 1000-line document, the system knows exactly which word was changed, when it was changed, and can show you the before and after states with surgical precision.

GitHub takes this concept and extends it into the cloud, making it accessible from anywhere in the world and adding powerful collaboration features. When you use GitHub, you're not just getting version control for your local files; you're getting a complete platform for managing projects, collaborating with others, and sharing your work with the global development community.

The fundamental principle behind GitHub's version control is that every change is tracked, every version is preserved, and everything can be recovered. This creates a safety net that allows developers to experiment fearlessly, knowing that they can always return to a previous working state if something goes wrong. It's this confidence that enables innovation and rapid development in the software industry.

The Evolution Problem: How Projects Grow and Change

To truly appreciate the power of GitHub's version control system, it's important to understand how projects naturally evolve and the challenges this evolution creates. Every project, whether it's a simple website or a complex software application, goes through a lifecycle of growth, refinement, and adaptation.

In the beginning, a project might start as a simple idea implemented in a few files. As the project grows, new features are added, existing functionality is improved, bugs are discovered and fixed, and the codebase becomes increasingly complex. This growth is not linear or predictable. Sometimes you need to try different approaches to solve a problem, sometimes you need to remove features that

aren't working, and sometimes you need to completely restructure your code for better organization.

Without version control, managing this evolution becomes increasingly difficult. Consider a web development project that starts with a basic HTML page and a simple CSS file. Over time, you might add JavaScript functionality, integrate with external APIs, implement user authentication, add a database layer, and optimize for mobile devices. Each of these additions involves modifying existing files and creating new ones.

Now imagine that after implementing user authentication, you discover that it's causing performance issues on mobile devices. Without version control, you would need to manually identify and remove all the authentication-related code, hoping that you don't accidentally remove something important or break other functionality in the process. With GitHub's version control, you can simply revert to the state before authentication was added, or you can create a separate branch to experiment with different authentication approaches while keeping your main codebase stable.

GitHub's version control system excels at managing this complexity because it treats your project as a living entity with a complete evolutionary history. Every decision point, every experiment, every bug fix becomes part of the permanent record. This historical perspective is invaluable not just for recovering from mistakes, but also for understanding how and why your project evolved the way it did.

The branching and merging capabilities of GitHub allow you to explore multiple evolutionary paths simultaneously. You can create a branch to experiment with a new feature while continuing to fix bugs in your main codebase. If the experimental feature works out, you can merge it back into the main branch. If it doesn't, you can simply abandon the branch without affecting your stable code. This flexibility is crucial for managing the unpredictable nature of project evolution.

GitHub's Approach to Version Control

GitHub has revolutionized version control by making it accessible, collaborative, and integrated with modern development workflows. While Git, the underlying technology that powers GitHub, was created by Linus Torvalds for managing the Linux kernel development, GitHub transformed this powerful but complex tool into a user-friendly platform that anyone can use.

The genius of GitHub lies in its approach to distributed version control. Unlike older centralized systems where there was a single server containing the project's history, GitHub allows every developer to have a complete copy of the project's entire history on their local machine. This distributed approach has several profound advantages that make GitHub particularly powerful for modern development.

First, it means that every copy of your project is a complete backup. If GitHub's servers were to disappear tomorrow (which is highly unlikely), every developer who has cloned your repository would have a complete copy of your project's history. This redundancy provides an unprecedented level of data security and availability.

Second, the distributed nature of GitHub allows for incredibly flexible collaboration patterns. Developers can work offline, making commits to their local repository, and then synchronize their changes with GitHub when they have internet access. This is particularly valuable for developers who travel frequently or work in areas with unreliable internet connections.

GitHub's web-based interface makes version control accessible to people who might be intimidated by command-line tools. While power users can still use Git's command-line interface for maximum flexibility, GitHub provides intuitive web interfaces for most common operations. You can create repositories, upload files,

make commits, create branches, and merge changes all through your web browser.

The platform also adds powerful collaboration features that go beyond basic version control. Pull requests, for example, allow developers to propose changes to a project and have those changes reviewed by other team members before they're merged into the main codebase. This review process helps maintain code quality and ensures that all team members are aware of changes being made to the project.

GitHub's issue tracking system integrates seamlessly with version control, allowing you to link specific commits to bug reports or feature requests. This creates a complete audit trail that connects every change in your code to the business reasons for making that change. Project managers and stakeholders can see not just what changed, but why it changed and how it relates to the project's goals.

Real-World Scenarios: When Version Control Saves the Day

To illustrate the practical value of GitHub's version control system, let's explore several real-world scenarios where version control capabilities prove invaluable. These scenarios demonstrate not just the technical benefits, but the peace of mind and confidence that comes from having a robust version control system backing up your work.

Scenario 1: The Accidental Deletion

Sarah is working on a e-commerce website with hundreds of product pages. She's been developing the site for three months, and it's nearly ready for launch. While

cleaning up some old files, she accidentally deletes the main product catalog component that took weeks to develop. In a panic, she realizes that her local backup is two weeks old and doesn't include several important features that were added recently.

With GitHub's version control, this scenario transforms from a disaster into a minor inconvenience. Sarah can simply check the repository history, identify the commit where the file was deleted, and restore it with a few clicks. Not only does she get the file back, but she gets it back with all the recent improvements intact. The entire recovery process takes less than five minutes, and she's back to work with no data loss whatsoever.

Scenario 2: The Team Collaboration Challenge

A development team of five people is working on a mobile application. Each developer is responsible for different features: user authentication, payment processing, social media integration, data synchronization, and user interface design. Without version control, coordinating changes between team members would be a nightmare of email attachments, shared folders, and constant confusion about who has the latest version of which files.

GitHub's version control system transforms this potential chaos into a smooth, coordinated effort. Each developer works on their own branch, implementing their features independently. When features are ready, they create pull requests that allow other team members to review the code before it's merged into the main branch. The system automatically handles merging changes from different developers, and when conflicts occur, it provides tools to resolve them systematically.

The team can see exactly what each member is working on, track progress on different features, and ensure that everyone is always working with the most up-to-date version of the shared codebase. Communication improves because develop-

ers can comment on specific lines of code, suggest improvements, and discuss implementation details directly within the context of the code itself.

Scenario 3: The Performance Optimization Gone Wrong

Mike is working on optimizing the performance of a web application. The site has been running slowly, and he's identified several areas where improvements can be made. He spends an entire weekend refactoring the database queries, optimizing image loading, and restructuring the CSS files. The changes are extensive, touching dozens of files throughout the project.

When he tests the optimized version on Monday morning, he discovers that while the site is indeed faster, several features are now broken. Users can't log in, the shopping cart doesn't work properly, and the search functionality returns incorrect results. The optimization introduced subtle bugs that are difficult to identify and fix quickly.

Without version control, Mike would face the daunting task of manually identifying and reversing all his optimization changes while trying to preserve the performance improvements that did work. With GitHub's version control, he has several elegant options. He can revert the entire project to the state before he started optimization, giving him a working baseline to start from. Alternatively, he can use GitHub's powerful diff tools to examine each change he made, identifying which specific modifications caused the problems and reverting only those changes while keeping the beneficial optimizations.

Even better, he can create a new branch from the pre-optimization state and selectively apply his improvements one at a time, testing each change to ensure it doesn't break existing functionality. This systematic approach allows him to achieve

his performance goals while maintaining the stability and reliability of the application.

The Collaborative Power of GitHub

One of GitHub's most transformative aspects is how it enables collaboration at a scale that was previously impossible. Traditional methods of collaboration on software projects were limited by geographical constraints, time zones, and the complexity of coordinating changes between multiple contributors. GitHub eliminates these barriers and creates opportunities for global collaboration that have fundamentally changed how software is developed.

The platform's collaboration features extend far beyond simple file sharing. When you host a project on GitHub, you're creating a space where developers from around the world can discover your work, contribute improvements, report bugs, and suggest new features. This open collaboration model has led to the creation of countless open-source projects that benefit millions of users worldwide.

GitHub's pull request system is particularly elegant in how it handles contributions from external developers. When someone wants to contribute to your project, they create a fork (their own copy) of your repository, make their changes in their fork, and then submit a pull request asking you to incorporate their changes into your main project. This process allows you to review proposed changes, discuss them with the contributor, request modifications if needed, and ultimately decide whether to accept the contribution.

The review process built into GitHub's collaboration workflow helps maintain code quality and consistency across projects. Multiple team members can review proposed changes, leave comments on specific lines of code, suggest improvements, and ensure that new code follows the project's standards and conventions.

This collaborative review process often results in better code than any individual developer could produce working alone.

GitHub's issue tracking system creates a structured way for users to report bugs, request features, and discuss project direction. Issues can be labeled, assigned to specific developers, linked to milestones, and referenced in commit messages. This creates a comprehensive record of not just what changed in your code, but why it changed and how those changes relate to user needs and project goals.

The platform also provides powerful project management tools that integrate seamlessly with version control. Project boards allow teams to organize work using methodologies like Kanban or Scrum, tracking the progress of features from initial concept through implementation and deployment. These tools provide visibility into project status for both technical team members and non-technical stakeholders.

Key Concepts and Terminology

Understanding GitHub requires familiarity with several key concepts and terms that form the foundation of version control thinking. These concepts might seem abstract at first, but they represent powerful ideas that make sophisticated project management possible.

A **repository** (often shortened to "repo") is the fundamental unit of organization in GitHub. It's a container that holds all the files for a project, along with the complete history of changes made to those files. Think of a repository as a project folder that remembers everything that ever happened to it. When you create a new project on GitHub, you're creating a new repository to house that project.

A **commit** represents a specific point in your project's history. When you make changes to your files and commit them to GitHub, you're creating a permanent

snapshot of your project at that moment. Each commit includes the actual file changes, metadata about when and who made the changes, and a commit message describing what was changed and why. Commits are the building blocks of your project's history.

Branches allow you to work on different versions of your project simultaneously. The main branch (usually called "main" or "master") represents the stable, production-ready version of your project. When you want to add a new feature or experiment with changes, you create a new branch. This gives you a separate workspace where you can make changes without affecting the main branch. If your changes work out, you can merge them back into the main branch. If they don't, you can simply delete the experimental branch.

A **merge** is the process of combining changes from one branch into another. When you've finished working on a feature in a separate branch, you merge those changes back into the main branch. GitHub provides tools to handle merges automatically when possible, and to help resolve conflicts when the same parts of files have been changed in different ways on different branches.

Pull requests are GitHub's mechanism for proposing changes to a repository. When you want to contribute changes to a project (whether it's your own project or someone else's), you create a pull request that shows what changes you want to make and allows others to review and discuss those changes before they're merged into the main codebase.

A **fork** is your own copy of someone else's repository. When you fork a repository, you get a complete copy of the project that you can modify without affecting the original. Forks are commonly used in open-source development, where contributors fork a project, make improvements in their fork, and then submit pull requests to have their improvements incorporated into the original project.

Cloning is the process of creating a local copy of a GitHub repository on your computer. When you clone a repository, you download all the project files and the

complete history of changes. This allows you to work on the project locally, make commits to your local copy, and then push your changes back to GitHub when you're ready to share them.

Basic GitHub Operations and Commands

While GitHub provides user-friendly web interfaces for many operations, understanding the underlying Git commands helps you appreciate what's happening behind the scenes and gives you more power and flexibility when working with your repositories.

Repository Creation and Setup

Creating a new repository on GitHub is straightforward through the web interface, but understanding the process helps you make informed decisions about how to structure your projects. When you create a new repository, you'll need to choose a name, decide whether it should be public or private, and optionally initialize it with a README file, .gitignore file, and license.

The repository name should be descriptive and follow common naming conventions. Most developers use lowercase letters with hyphens to separate words, like "my-awesome-project" or "portfolio-website". The decision between public and private repositories depends on whether you want your code to be visible to everyone on the internet or restricted to specific collaborators.

Initializing your repository with a README file is generally recommended because it provides a place to describe what your project does, how to install and use it, and any other important information. The .gitignore file tells Git which files to ig-

nore when tracking changes, which is important for excluding temporary files, build artifacts, and sensitive information like passwords or API keys.

Making Your First Commit

Once you have a repository, the next step is adding files and making your first commit. If you're working through GitHub's web interface, you can upload files directly or create new files using the online editor. Each time you save changes, you're prompted to write a commit message describing what you changed.

Good commit messages are crucial for maintaining a useful project history. They should be concise but descriptive, explaining not just what changed but why it changed. For example, "Fix login bug that prevented users with long passwords from authenticating" is much more useful than "Fixed bug" or "Updated login.js".

When working locally with Git commands, the process involves several steps:

```
git add filename.txt
git commit -m "Add new feature for user authentication"
git push origin main
```

The `git add` command stages files for commit, telling Git which changes you want to include in the next commit. The `git commit` command creates the actual commit with your staged changes and the provided message. The `git push` command uploads your local commits to GitHub, making them visible to collaborators and backing them up in the cloud.

Understanding File Status and Changes

GitHub and Git track files in several different states, and understanding these states helps you manage your changes effectively. Files can be untracked (not yet added

to version control), modified (changed since the last commit), staged (marked for inclusion in the next commit), or committed (saved to the repository history).

The `git status` command shows you the current state of your files, which ones have been modified, which ones are staged for commit, and which ones are untracked. This command is invaluable for understanding what changes you're about to commit and ensuring you don't accidentally include files you didn't mean to modify.

The `git diff` command shows you exactly what changed in your files since the last commit. This is particularly useful for reviewing your changes before committing them, ensuring that you understand exactly what you're about to save to the repository history.

Branching and Merging Workflows

Branching is one of GitHub's most powerful features, but it can be confusing for beginners. The key is to think of branches as parallel universes where you can experiment with changes without affecting your main codebase.

Creating a new branch is simple:

```
git checkout -b new-feature
```

This command creates a new branch called "new-feature" and switches to it. Now any changes you make will be isolated to this branch, leaving your main branch untouched.

When you're satisfied with the changes in your branch, you can merge them back into the main branch:

```
git checkout main
git merge new-feature
```

GitHub's web interface provides tools for creating and managing branches without using command-line tools, making this powerful feature accessible to developers who prefer graphical interfaces.

Practical Examples and Exercises

To solidify your understanding of GitHub's version control concepts, let's work through some practical examples that demonstrate how these tools work in real-world scenarios.

Exercise 1: Creating Your First Repository

Start by creating a new repository on GitHub for a simple personal website. This exercise will walk you through the entire process from repository creation to making your first commits.

1. Log into your GitHub account and click the "New repository" button
2. Name your repository "my-personal-website"
3. Add a description like "A simple personal website to showcase my projects"
4. Make the repository public so others can see your work
5. Initialize with a README file
6. Create the repository

Now you have a repository with a single file (README.md). Edit this file through GitHub's web interface to include information about yourself and what you plan to build. When you save your changes, you'll be prompted to write a commit mes-

sage. Use something descriptive like "Update README with personal information and project goals."

Exercise 2: Adding Content and Tracking Changes

Create a simple HTML file for your personal website directly in GitHub's web interface:

1. Click "Create new file" in your repository
2. Name the file "index.html"
3. Add basic HTML structure with your name and a brief introduction
4. Commit the file with the message "Add basic HTML structure for homepage"

Now modify the HTML file to add more content, perhaps a list of your skills or projects. Notice how GitHub shows you exactly what changed when you commit these modifications. The green lines show what you added, and red lines would show what you removed.

Exercise 3: Working with Branches

Practice the branching workflow by adding a new feature to your website:

1. Create a new branch called "add-contact-form"
2. In this branch, create a new file called "contact.html" with a simple contact form
3. Modify your index.html file to include a link to the contact page
4. Commit these changes to the branch
5. Create a pull request to merge your branch back into main

6. Review the pull request to see all the changes you made
7. Merge the pull request

This exercise demonstrates how branches allow you to work on new features without disrupting your main codebase, and how pull requests provide a structured way to review and integrate changes.

Exercise 4: Collaborating with Others

If you have friends or colleagues who also use GitHub, practice the collaboration workflow:

1. Have someone fork your repository
2. Ask them to make improvements to your website (fix typos, improve styling, add content)
3. Have them submit a pull request with their changes
4. Review their pull request, leave comments on specific lines if you have suggestions
5. Merge their changes if you're satisfied with them

This exercise shows how GitHub enables collaboration between people who might never meet in person, creating opportunities for global cooperation on projects.

Common Pitfalls and How to Avoid Them

As you begin working with GitHub, you'll likely encounter some common challenges that trip up many beginners. Understanding these pitfalls and how to avoid them will save you time and frustration as you develop your version control skills.

Commit Message Quality

One of the most common mistakes beginners make is writing poor commit messages. Messages like "fixed stuff," "updates," or "changes" provide no useful information about what actually changed or why. Good commit messages are essential for maintaining a useful project history that you and your collaborators can understand months or years later.

Develop the habit of writing commit messages that complete the sentence "This commit will..." For example, "This commit will fix the login bug that prevented users with special characters in their passwords from authenticating" becomes "Fix login bug for passwords with special characters."

Committing Too Much or Too Little

Another common mistake is committing changes that are too large or too small. Commits that change hundreds of files and implement multiple features make it difficult to understand what changed and why. On the other hand, commits that change only a single character or fix a trivial typo can clutter the project history with noise.

Aim for commits that represent a single logical change to your project. If you find yourself struggling to write a clear commit message, it might be because your

commit is trying to do too many things at once. Consider breaking large changes into smaller, more focused commits.

Ignoring Branch Strategy

Many beginners work exclusively on the main branch, missing out on one of GitHub's most powerful features. Working directly on main means that your repository is always in an unstable state while you're developing new features, and it makes collaboration more difficult.

Develop the habit of creating branches for new features, bug fixes, and experiments. This keeps your main branch stable and deployable while giving you the freedom to experiment and iterate on new ideas.

Not Using .gitignore Files

Beginners often commit files that shouldn't be tracked by version control, such as temporary files, build artifacts, or files containing sensitive information like passwords or API keys. This clutters the repository and can create security risks.

Learn to use .gitignore files to tell Git which files and directories to ignore. GitHub provides templates for common programming languages and frameworks that include appropriate .gitignore rules for most projects.

Conclusion: Your Journey into Version Control

Understanding version control through GitHub opens up a world of possibilities for managing your projects and collaborating with others. The concepts we've cov-

ered in this chapter form the foundation for everything else you'll learn about GitHub and modern software development practices.

Version control is not just a technical tool; it's a mindset that encourages experimentation, collaboration, and continuous improvement. When you know that every change is tracked and every version is preserved, you gain the confidence to try new approaches, refactor code for better organization, and accept contributions from others without fear of breaking your project.

The collaborative aspects of GitHub have transformed how software is developed, moving from isolated individual efforts to global communities working together on shared goals. Open-source projects hosted on GitHub demonstrate the incredible things that can be accomplished when talented people from around the world can easily contribute their skills to common projects.

As you continue your journey with GitHub, remember that version control is a skill that improves with practice. The concepts might seem abstract at first, but as you work on real projects and encounter real challenges, you'll develop an intuitive understanding of when and how to use GitHub's various features effectively.

The investment you make in learning version control will pay dividends throughout your career. Whether you're working on personal projects, collaborating with a small team, or contributing to large open-source initiatives, the skills you develop with GitHub will make you a more effective and confident developer.

In the next chapter, we'll dive deeper into Git fundamentals and explore how the underlying technology works, giving you a more complete understanding of the tools and concepts that make GitHub so powerful. We'll also start working with actual repositories and practicing the workflows that professional developers use every day.

The journey from version control novice to expert is rewarding and transformative. Every commit you make, every branch you create, and every pull request you

submit builds your understanding and confidence. Welcome to the world of version control, and welcome to GitHub.