

FINAL ASSIGNMENT REPORT

Major: INFORMATION TECHNOLOGY

Course code: CMP186

Course name: Tools and Environments for Software Development

Lecturer's name: MR. NGUYEN VAN TAN

Student's name:

- | | | |
|-------------------------|----------------|-----------------|
| 1. HUYNH THANH DANH | ID: 2180602030 | Class: 21DTHQA2 |
| 2. LE QUOC HUNG | ID: 2182500349 | Class: 21DTHQA2 |
| 3. PHAM PHI DAN | ID: 2180602037 | Class: 21DTHQA2 |
| 4. NGUYEN DOAN DONG ANH | ID: 2180603310 | Class: 21DTHQA2 |

Ho Chi Minh City

2025

CONTENTS

COMMENTS	5
ACKNOWLEDGEMENTS	6
Chapter 1. Introduction to the System.....	7
1.1 Introduction.....	7
1.2 Features and Benefits.....	7
1.3 Detailed Survey Notes	7
1.4 Background and Objectives.....	7
1.5 Key Features	8
Book Catalog Management:	8
Book Search:	8
Borrowing/Returning Management:.....	8
Member Management:.....	8
Reports and Statistics:	8
OPAC (Online Public Access Catalogue) Interface:.....	8
1.6 Technical Implementation	9
1.7 Advantages Over Manual Systems.....	9
1.8 Challenges and Future Development.....	11
1.9 Conclusion	11
1.10 Project Overview	12
Library Management System Project Abstract:	12
Existing System:	12
Proposed System:	12
Features:	13
Technologies and Products Used:	13

Modules Overview:	13
Screenshots:	14
System Configuration:	15
Hardware Configuration:	15
Software Configuration:	15
Chapter 2. Git and GitHub Overview	16
2.1 What is Git?	16
2.2 What is a Version Control System (VCS)?	16
2.3 How Does a VCS Benefit Developers?	17
2.4 How Does Git Work?	18
2.5 What Are the Benefits of Using Git?	19
2.6 Essential Git Commands	24
2.7 Best Practices for Using Git Effectively	25
2.8 What is GitHub?	27
2.9 How Does GitHub Work?	28
2.10 Git vs. GitHub: What's the Difference?	30
Step 3: Install Git on Your Computer	35
Step 4: Connect Git to GitHub	35
Step 5: Initialize a Git Repository on Your Computer	35
2.11 Conclusion	36
Chapter 3. Apply Git/GitHub to Projects	37
3.1 Git, GitHub and real application demo images	37
3.2 Git Configuration	41
3.3 Git, GitHub and Applying it in Practice:	46
Basic Git Commands	46

Commands Used (From the Log):.....	46
Practical Application and Workflow:.....	47
Additional Basic Git Commands (Not in the Log but Essential):.....	48
Real-World Applications of Git	48
Personal Software Development:	48
Team Collaboration:	49
DevOps and CI/CD (Continuous Integration/Continuous Deployment):	49
Open Source Contributions:	49
3.4 Conclusion	50
Reference	51

COMMENTS

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

Teacher

(Sign and print full name)

ACKNOWLEDGEMENTS

Dear Mr. Nguyen Van Tan,

We, Group 4, would like to express our sincere and deepest gratitude. Throughout the course of this subject, we not only learned valuable professional knowledge about tools and environments for software development but were also inspired and motivated by you to self-learn and explore.

Your enthusiastic and dedicated guidance helped us overcome the initial difficulties when approaching Git and GitHub. Thanks to you, we have gained a clearer understanding of teamwork, task allocation, and project management effectively. We have also become more confident in working on large-scale projects and contributing to open-source projects.

We sincerely thank you for your informative lectures, your sincere advice, and the enthusiastic support you have given us. The knowledge and experience we have gained from you will be a valuable asset to help us succeed in our future careers.

We wish you good health, happiness, and success in your noble career of educating people.

Ho Chi Minh, City

2025

Chapter 1. Introduction to the System

1.1 Introduction

The library management system is designed to assist librarians and users in educational institutions, such as schools and universities, in efficiently managing library activities. Compared to manual methods using pen and paper, this system stores data in a computer database, reducing time and effort while minimizing errors. This project, implemented using Java with MS Access, is an ideal choice for final-year students, as it not only enhances programming skills but also strengthens personal portfolios.

1.2 Features and Benefits

The system offers several key features, including:

Searching for books by title or author, displaying relevant results instantly.

Managing book categories, adding/deleting/editing details such as title, author, and publisher.

Recording and tracking borrowing/return transactions, including calculating late return fines.

Generating reports on book inventory, borrowed books, and revenue from fines.

The benefits include time savings, increased efficiency for librarians, and easier book searching for students. Notably, the system helps reduce paperwork and labor costs while improving the user experience with mobile access and email reminders.

1.3 Detailed Survey Notes

The library management system, developed using Java with an MS Access database, is a technological solution designed to improve library efficiency in educational institutions, particularly schools and universities. Below is a detailed analysis based on reliable sources, covering features, technical implementation, benefits, and challenges to provide a comprehensive view of this project.

1.4 Background and Objectives

Traditional library management relies on manual methods, using pen and paper to record details such as book inventory, categories, titles, and student borrowing/returning

records. This approach is time-consuming, labor-intensive, and prone to errors. This project aims to address these challenges by automating operations through a computerized database, enhancing library efficiency.

The main objectives include:

Automating the recording and management of books, members, and borrowing/return transactions.

Providing a user-friendly interface for quick book searches without manual effort.

Ensuring accuracy and reducing errors compared to manual methods.

This project is well-suited for final-year students, not only for its practical application but also for enhancing personal portfolios, particularly in Java programming and database management.

1.5 Key Features

Based on research from sources like GeeksforGeeks Library Management System, this system offers the following features:

Book Catalog Management: Add, delete, and edit book details, including title, author, publisher, publication date, price, and purchase date.

Book Search: Users can enter a book title or author, and the system will display relevant books, saving time compared to manual searching. This is one of the most notable features, especially useful for students and faculty.

Borrowing/Returning Management: Records book borrowings, return dates, and calculates late return fines based on book categories and membership types.

Member Management: Add, delete, and edit student/faculty information, including borrowing history and account status.

Reports and Statistics: Generate reports on book inventory, borrowed books, overdue books, and revenue from fines, aiding management decisions.

OPAC (Online Public Access Catalogue) Interface: Allows users to search books online, check account status, and reserve books, enhancing user experience.

Additionally, the system can integrate barcode or RFID technology for book scanning during borrowing/returning, as suggested by MasterSoft Library Management, increasing efficiency and accountability.

1.6 Technical Implementation

The system is developed using Java due to its platform independence and object-oriented features, making it suitable for multi-platform applications (Windows, Linux, macOS, etc.). MS Access is chosen as the database for its ease of use, lack of complex server requirements, and suitability for small to medium-sized libraries.

According to Code With C Library Management System, the technical implementation includes:

Using **JDBC-ODBC bridge** or the **UCanAccess library** to connect Java with MS Access, ensuring communication between the application and the database.

The **user interface** can be built with Java Swing for a standalone application or JSP for a web version, supported by tools like **NetBeans 5.0** and **Tomcat 5.5**.

Minimum hardware requirements: Pentium III 630MHz processor, 128MB RAM, 20GB hard drive; software: Windows NT/98/XP OS, JRE, and MS Access.

However, MS Access has scalability limitations when handling large amounts of data, which can cause performance slowdowns, especially for libraries with thousands of books and transactions.

1.7 Advantages Over Manual Systems

Compared to traditional manual methods, this system offers numerous advantages, as highlighted by Entab Library Management Benefits:

Advantage	Description
Continuous Updates	Keeps book and journal records up to date, allowing easy additions and deletions.
Increased Student Engagement	Students can log in, browse, and find books, enhancing library usage.

Advantage	Description
Record Maintenance	Stores borrower details, return dates, and calculates fines based on book categories.
Web Interface	Allows students to check borrowing history, renewals, return dates, and fines, improving connectivity.
Academic Growth Support	Online library management fosters academic growth and school development.
Easy Inventory Management	Facilitates library inventory checks with efficient and accessible software records.
Dynamic Reports	Optimizes performance with reports, charts, and graphs, aiding decision-making.
Error Reduction	Automated data processing minimizes human errors and improves usability.
Innovation	Students can search, write, upload media, manage emails, and collaborate via chat and social networks.
Fully Customizable	Adapts to educational institution needs, ensuring fast and reliable data handling.
Cost Savings	Reduces costs with cloud libraries, digitization, and mobile access, eliminating paper-based methods.
High Security	Regular updates ensure a secure user database, with minimal risk of system failures (e.g., Etna system).

Advantage	Description
Mobile Access	Flexible access via mobile apps, supporting both online and offline usage.
Improved Reporting & Monitoring	Automated record updates support efficient reservations, user tracking, and material circulation management.

These benefits not only improve librarian efficiency but also support students and faculty in academic research.

1.8 Challenges and Future Development

Despite its advantages, MS Access has scalability limitations, particularly when handling large amounts of data, which may cause performance issues. Other challenges include:

Difficulty in integrating with larger systems or those requiring high security.

Lack of built-in mobile support, requiring additional development for a mobile application.

In the future, the system can be upgraded by:

Migrating to a more robust database like **MySQL or PostgreSQL** for better scalability.

Integrating a mobile application to allow users to access the system anytime, anywhere, with email or SMS notifications for return dates.

Adding **data analytics features**, such as book borrowing trends by category, to support library management decisions.

1.9 Conclusion

The library management system, developed using Java and MS Access, is an efficient solution for educational libraries, automating and optimizing operations. With features like quick book searches, borrowing/return management, and report generation,

along with benefits such as cost savings and increased efficiency, this project is ideal for final-year students. However, scalability limitations should be considered, and future upgrades should be planned to accommodate growth.

1.10 Project Overview

Library Management System Project Abstract:

LMS is a software application useful for librarians in any educational institution for management and handling of typical operations in the library. It manages data through computerized system in a database which eradicates the tediousness in searching a particular book in the entire library.

Existing System:

The existing [library management system](#) is very traditional and manual. Most of the things such as record of students who've issued the book, no. of books in the library, fine of [students](#), etc. are recorded using pen-paper system. This traditional system requires a lot of time and manpower for performing simple operations in the library.

The drawbacks of existing system are:

Record keeping is paper-based.

The system is not computerized.

There's high risk of data mis-management.

Students face a lot of problem while searching a book.

Chances of book theft from the library is high.

A lot of paperwork and manpower is to be allocated for simple library management operations.

The existing system is very inefficient and unreliable.

Proposed System:

The proposed library management software is developed after the analysis of issues, problems, and drawbacks of the existing system. Here, different modules have been assigned for managing and organizing different [tasks](#) in a library. Using this software,

librarian can store information of all the books, with author name, faculty, etc., available in the library. Only one person can handle the entire library.

The main feature of this system is that all the books available in the library can be displayed in a list so that students need not roam through the entire library to find a book. Additionally, the application effectively maintains the details of users/students to whom books have been issued; it also records the issued date and return date.

Features:

Add, modify and delete book details into the database. Search feature for finding book availability in library [stock](#). Add students records upon issue of a book. Record issue date, return date, and fine (penalty). Payment system/feature allows the librarian to calculate payment details and print bill.

Technologies and Products Used:

NetBeans 5.0

Tomcat 5.5 embedded in NetBeans

Oracle10g Express Edition

Jboss 4.0.3

Java Server Pages (JSP)

Java Beans

Enterprise Java Beans

Modules Overview:

Insertion Module: With the help of this module, librarian can insert information and details of users along with book details and time and date of book issue.

Updating Module: This module allows modification to the existing records and details in the library management system database.

Report Generation Module: This module allows task such as printing payment details, stock details of books, etc.

Screenshots:

Add Books

BOOK INFORMATION

Add a new book:

The book subject: JAVA

The book title: Completer Referencr

The name of the Author(s): Kerty Syera

The name of the Publisher: Com tech

Copyright for the book: Author

The edition number: 8

The number of Pages: 504

ISBN for the book: 45622

The number of copies: 5

The name of the Library: Public Library TVM

Shelf No 35

Insert the Information

Exit

Add New Books

Books


THE LIST FOR THE BOOKS

Books:

[print the books](#)

Boo...	Subject	Title	Author	Publisher	Copyright	Edition	Pages	NumberOfBooks	ISBN	Library	Availble	Sh
4	Java, Swing	Java Swing	Salah	Thubarti Co.	2002	2	1230	4	1-8877-904-X	Main	<input checked="" type="checkbox"/>	
5	PHP	PHP 4.0	Salah	Thubarti Co.	2000	3	860	2	1-8877-905-X	Main	<input checked="" type="checkbox"/>	
6	HTML	Using HTML	Salah	Thubarti Co.	1999	1	450	4	1-8877-906-X	Main	<input checked="" type="checkbox"/>	
7	Java, JSP	More JavaServer Pages	Salah	Thubarti Co.	2001	1	980	2	1-8877-907-X	Main	<input checked="" type="checkbox"/>	
11	Assembly	Introduction To Assembly L...	Salah	Thubarti Co.	1998	2	640	1	1-9988-908-X	Main	<input checked="" type="checkbox"/>	
15	Java	Programming in Java	Balaguru	Tata	1998	3	1230	135	1-1997-904-X	Main	<input checked="" type="checkbox"/>	
16	java	bel.java	balaguru	tata	1989	1	230	7	12-34	main	<input checked="" type="checkbox"/>	

List of All Books

Available Books										
THE LIST FOR THE AVAILABLE BOOKS										
Available books:										
 print the books										
BookID	Subject	Title	Author	Publisher	Copyright	Edition	Pages	ISBN	Library	S
4	Java, Swing	Java Swing	Salah	Thubaiti Co.	2002	2	1230	1-8877-904-X	Main	
5	PHP	PHP 4.0	Salah	Thubaiti Co.	2000	3	860	1-8877-905-X	Main	
6	HTML	Using HTML	Salah	Thubaiti Co.	1999	1	450	1-8877-906-X	Main	
7	Java, JSP	More JavaServer Pages	Salah	Thubaiti Co.	2001	1	980	1-8877-907-X	Main	
11	Assembly	Introduction To Assembly Lan...	Salah	Thubaiti Co.	1998	2	640	1-9988-908-X	Main	
15	Java	Programming in Java	Balaguru	Tata	1998	3	1230	1-1997-904-X	Main	
16	java	bal.java	balaguru	tata	1989	1	230	12-34	main	

Available Books

System Configuration:

Hardware Configuration:

Processor: Pentium III 630MHz

RAM: 128MB

Hard Disk: 20GB

Monitor: 15" Color Monitor

Keyboard: 122 keys

Software Configuration:

Operating System: Windows NT/ Windows 98/ Windows XP

Language: Java 2 Runtime Environment

Database: MS Access

Chapter 2. Git and GitHub Overview

2.1 What is Git?

Git is a powerful and widely adopted Distributed Version Control System (DVCS), celebrated as the de facto standard for managing source code and project versions in the software development community. Unlike centralized version control systems that rely on a single server to store all changes, Git distributes the full repository—including the entire history of modifications—to every developer's local machine. This decentralized architecture empowers individuals and teams to work independently, collaborate seamlessly, and maintain robust version control, even in offline environments. Originally created by Linus Torvalds in 2005 to support Linux kernel development, Git has since evolved into a cornerstone tool for projects of all sizes, from solo endeavors to massive open-source collaborations.

Git's popularity stems from its speed, flexibility, and ability to handle complex workflows. Each developer's local repository acts as a self-contained unit, containing not just the current state of the project but also every change ever made, complete with metadata like timestamps, authors, and commit messages. This design ensures that developers can experiment, track progress, and recover from mistakes without relying on constant server connectivity, making Git a versatile and resilient solution for modern software engineering.

2.2 What is a Version Control System (VCS)?

A Version Control System (VCS) is a software framework designed to systematically track, manage, and store changes to a project's files—most commonly source code—over time. It acts as a digital ledger, preserving every modification in a structured manner and enabling developers to navigate the evolution of their work. VCS tools are essential in software development, where files are frequently updated, shared, and iterated upon by multiple contributors. By maintaining a comprehensive history of changes, a VCS allows developers to create distinct versions of a project, each representing a snapshot or milestone in its development lifecycle.

At its core, a VCS operates by storing all project files in a repository—a centralized or distributed storage location. Developers can clone this repository to their local machines,

make changes independently, and commit those changes to create new versions. These updates can then be pushed to a shared server, where other team members with appropriate permissions can pull the latest revisions or even clone the entire repository anew. Each version in the VCS includes critical details: the altered content, the date and time of the change, the identity of the contributor, and often a note explaining the purpose or context of the modification. This timeline-like structure allows developers to review past states, compare differences, and revert to earlier versions if necessary.

VCS systems come in two main flavors: Centralized Version Control Systems (CVCS), like Subversion, which rely on a single server, and Distributed Version Control Systems (DVCS), like Git, which distribute the full repository to every user. The distributed model, exemplified by Git, offers greater autonomy and fault tolerance, as the loss of a server doesn't jeopardize the entire project history.

2.3 How Does a VCS Benefit Developers?

The adoption of a VCS brings transformative advantages to software development workflows:

Comprehensive History Tracking: Every change—whether an addition, deletion, or modification—is logged, creating a detailed archive. This allows developers to revisit past versions, understand the evolution of the codebase, or roll back to a stable state if a bug emerges.

Simplified Code Sharing: A VCS enables developers to share their work effortlessly, either publicly for open-source projects or privately within a restricted group. Repositories can be hosted on platforms like GitHub, GitLab, or Bitbucket, facilitating global collaboration.

Change Accountability: By recording who made each change and why (via commit messages), a VCS fosters transparency and accountability, especially in team settings where multiple contributors are involved.

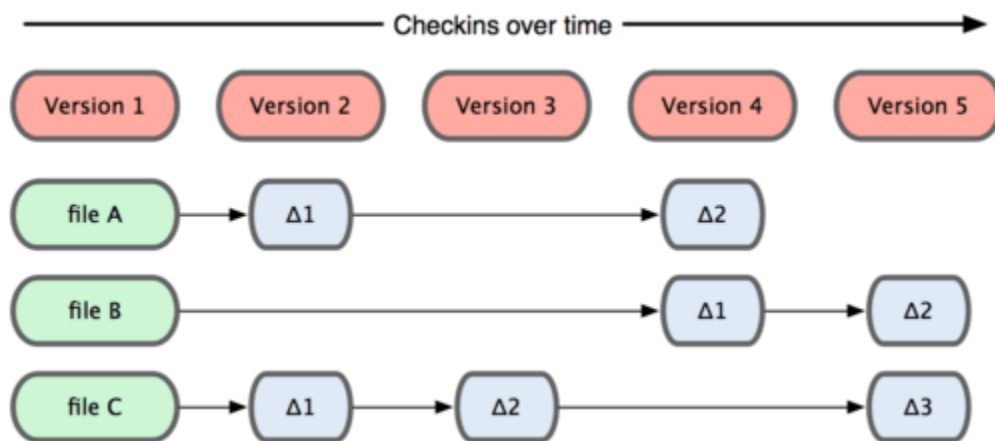
Error Recovery: Mistakes happen—whether it's a buggy feature or an accidental deletion. A VCS provides a safety net, allowing developers to revert to a previous version or undo specific changes without losing progress.

Parallel Development: Multiple developers can work on different aspects of a project simultaneously, with the VCS managing how these contributions are integrated, reducing conflicts and duplication of effort.

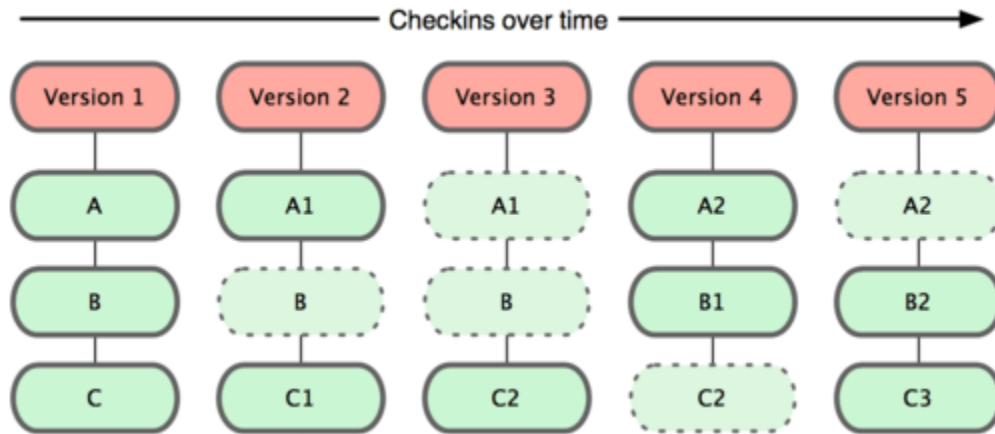
For Git specifically, its distributed nature amplifies these benefits. Every clone of a Git repository is a fully functional backup, complete with the entire change history, making it exceptionally resilient and adaptable to diverse development scenarios.

2.4 How Does Git Work?

Git’s operational model sets it apart from other VCS tools, such as Subversion (SVN), CVS, Perforce, or Bazaar, through its innovative approach to data management. Traditional VCS systems typically store information as a sequence of file-based differences, or “deltas,” tracking incremental changes to each file over time. For example, if a file is modified, these systems record only what changed since the last version, building a chain of deltas. This approach, while space-efficient for small projects, can become cumbersome as repositories grow or when complex branching is involved.



Git, however, takes a radically different stance. Instead of focusing on deltas, Git treats its data as a series of snapshots. Each time a developer commits changes, Git captures a complete picture of the project’s state at that moment—a snapshot of every file in the repository. To optimize storage, Git doesn’t redundantly save unchanged files; it creates a reference (or link) to the identical file from a previous snapshot. This snapshot-based system allows Git to efficiently manage large projects while providing rapid access to any point in the project’s history.



Here's a step-by-step look at Git's workflow:

Local Modifications: A developer edits files in their working directory.

Staging Changes: Using `git add`, changes are moved to the staging area (index), preparing them for the next commit.

Committing: With `git commit`, Git creates a snapshot of the staged changes, permanently storing it in the local repository with a unique identifier (a SHA-1 hash).

Synchronization: Changes can be pushed to a remote repository (`git push`) or pulled from it (`git pull`) to share with others.

This snapshot model underpins Git's strengths, such as its ability to handle branching and merging with ease. Unlike delta-based systems, which may struggle to reconcile divergent changes, Git's snapshots provide a clear, holistic view of the project at every commit, simplifying conflict resolution and version comparison.

Visual Reference: Diagrams from StackOverflow contrast Git's snapshot approach with the delta-based storage of traditional VCS tools, highlighting the conceptual leap.

Git's design also imbues it with filesystem-like qualities. It's not just a VCS but a lightweight, self-contained system with powerful tools for manipulating data, making it a versatile platform for version control and beyond.

2.5 What Are the Benefits of Using Git?

Git's unique features translate into tangible benefits that enhance development workflows:

Team Collaboration:

In projects with multiple contributors, Git prevents code conflicts by isolating work in branches, allowing developers to merge their efforts seamlessly.

Adaptability to Change:

Software requirements often shift mid-project. Git's versioning and branching capabilities let developers revert to earlier states or pivot to new directions without disrupting the main codebase.

Robust Branching:

Git's lightweight branching model supports parallel development—think feature experimentation, bug fixes, or release preparation—all within the same repository.

Speed and Simplicity:

Git's commands are fast and intuitive, reducing the overhead of version control tasks.

Seamless Merging:

Combining branches is straightforward, thanks to Git's snapshot system, which minimizes merge conflicts and simplifies integration.

Offline Productivity:

Developers can clone a repository and work anywhere, committing changes locally without needing an internet connection until they're ready to sync.

Effortless Deployment:

Git streamlines deployment by allowing precise control over which changes are pushed to production environments, reducing deployment errors.

Security and Integrity:

Every commit is checksummed with a SHA-1 hash, ensuring data integrity and making tampering detectable.

These advantages make Git indispensable for modern software projects, from small scripts to enterprise-scale applications.

Key Git Terminology

To wield Git effectively, developers must grasp its foundational concepts and vocabulary:

Branch:

A separate line of development within a repository, enabling isolated work on features, fixes, or experiments.

Commit:

A recorded snapshot of changes, capturing the state of the project at a specific moment, tagged with a message and metadata.

Checkout:

The process of switching to a different branch or restoring a specific commit's state (e.g., `git checkout feature-branch`).

Fetch:

Retrieves updates from a remote repository without merging them, allowing inspection before integration.

Fork:

A personal copy of a repository, often used for contributing to open-source projects without altering the original.

Head:

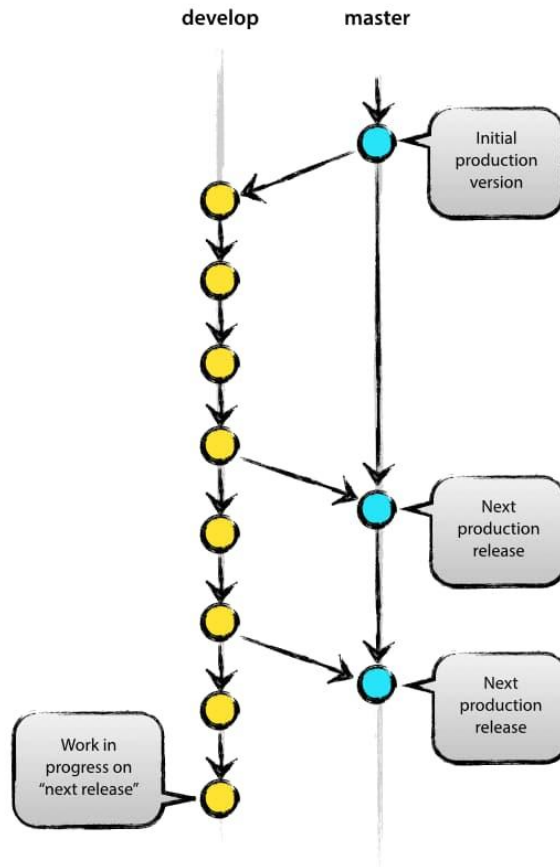
The most recent commit on the current branch, representing the active working state.

Index (Staging Area):

A buffer where changes are prepared before committing, offering flexibility to refine what's included.

Master (or Main):

The primary branch, traditionally the stable, production-ready version of the project.

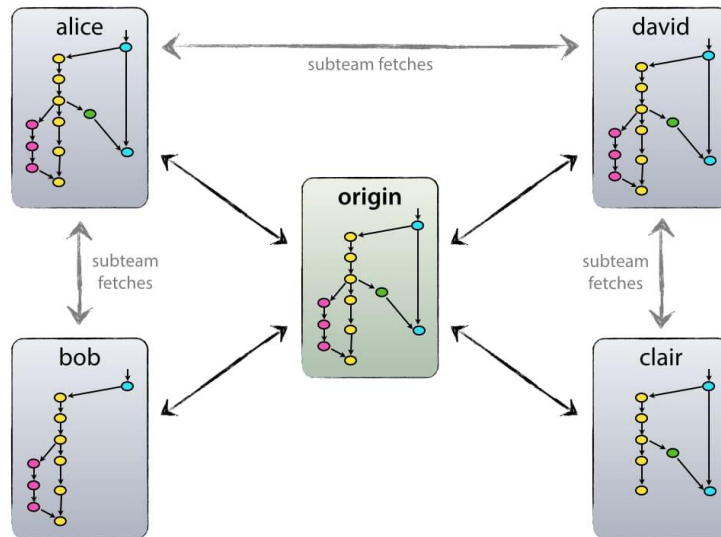


Merge:

Integrates changes from one branch into another, combining parallel efforts.

Origin:

The default alias for the remote repository from which the local copy was cloned.



Pull:

Fetches and merges remote changes into the local branch, keeping it up to date.

Push:

Sends local commits to a remote repository, sharing updates with others.

Rebase:

Rewrites commit history by moving or combining branches, creating a cleaner timeline.

Remote:

A repository hosted elsewhere (e.g., on a server) that syncs with the local copy.

Repository:

The container for all project files, branches, tags, and history.

Stash:

Temporarily shelves uncommitted changes, freeing the working directory for other tasks.

Tags:

Labels for significant commits (e.g., v1.0), available as lightweight pointers or annotated objects with additional metadata.

Upstream:

The destination for pushed changes, typically the main branch or origin repository.

2.6 Essential Git Commands

Git's power lies in its command-line interface. Below are key commands with detailed explanations:

`git config`:

Configures user identity for commits (e.g., `git config --global user.name "Jane Doe"` or `git config --global user.email "jane@example.com"`).

`git init`:

Initializes a new Git repository in the current directory, creating a `.git` folder to store version data.

`git clone`:

Downloads a remote repository to the local machine (e.g., `git clone https://github.com/user/repo.git`).

`git status`:

Shows the status of the working directory and staging area, highlighting modified, staged, or untracked files.

`git add`:

Stages changes for the next commit (e.g., `git add file.txt` or `git add .` for all changes).

`git commit`:

Saves staged changes as a snapshot with a descriptive message (e.g., `git commit -m "Implement login feature"`).

`git push`:

Uploads local commits to a remote repository (e.g., `git push origin main`).

`git pull`:

Fetches and merges updates from a remote branch (e.g., `git pull origin main`).

git branch:

Lists existing branches or creates new ones (e.g., `git branch` or `git branch feature-x`).

git checkout:

Switches branches or restores files (e.g., `git checkout feature-x` or `git checkout -b new-branch` to create and switch).

git merge:

Combines changes from one branch into another (e.g., `git merge feature-x` while on `main`).

git reset:

Unstages changes or reverts commits (e.g., `git reset HEAD file.txt` to unstage a file).

git stash:

Shelves uncommitted changes for later (e.g., `git stash` and `git stash pop` to retrieve).

git remote:

Manages connections to remote repositories (e.g., `git remote add origin <url>` or `git remote -v` to list).

2.7 Best Practices for Using Git Effectively

To harness Git's full potential in daily workflows, consider these expanded recommendations:

Leverage Cheat Sheets:

With dozens of commands, keeping a reference handy is invaluable. Explore resources like [Git Cheat Sheet](#), [Git Explorer](#), or [Roger Dudler's Guide](#).

Commit Frequently and Granularly:

Break changes into small, logical commits (e.g., one for a bug fix, another for a new feature). This simplifies debugging, merging, and collaboration.

Test Before Committing:

Never commit untested code. Run unit tests, integration tests, or manual checks to ensure stability before sharing with the team.

Write Meaningful Commit Messages:

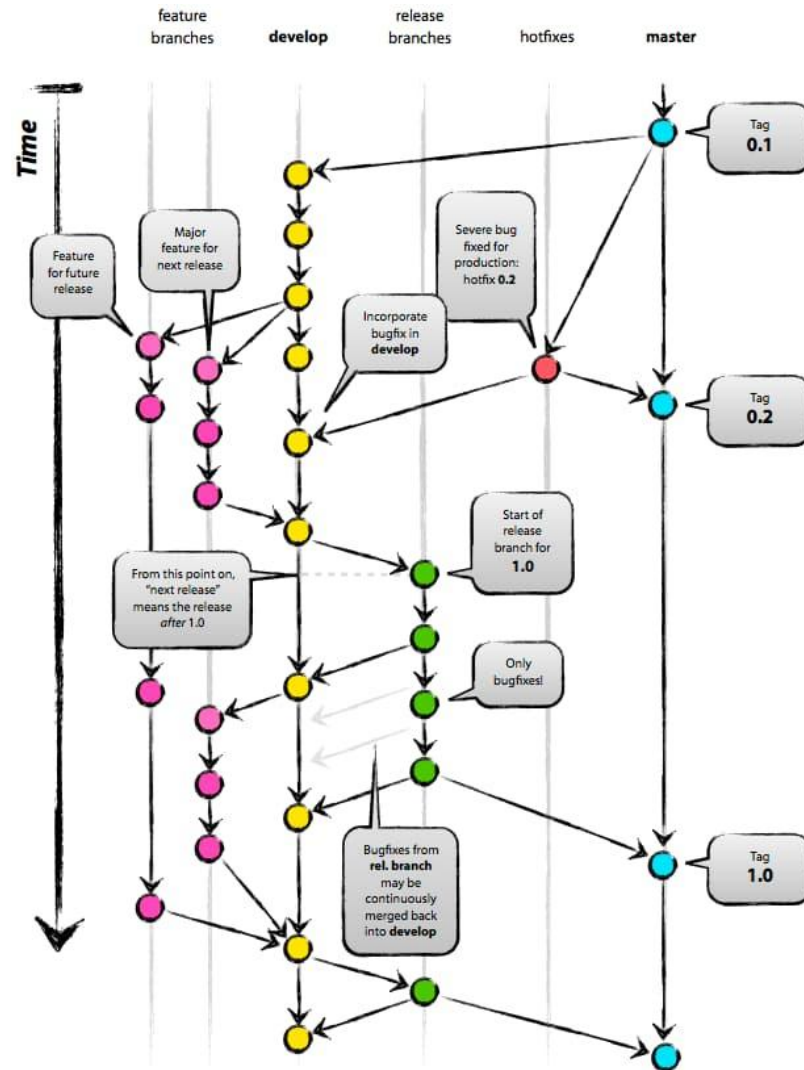
Use clear, detailed messages (e.g., “Fix null pointer exception in login module” instead of “Fix bug”). Include context to aid future readers.

Experiment with Branches:

Create branches for every feature, fix, or experiment. This keeps the main branch clean and allows risk-free exploration.

Adopt a Git Workflow:

Standardize team practices with workflows like Gitflow (feature branches, releases) or GitHub Flow (simple branching for continuous deployment).



Review Changes Before Pushing:

Use git diff or git log to inspect commits, ensuring only intended changes are shared.

Backup with Remotes:

Regularly push to a remote repository to safeguard against local data loss.

2.8 What is GitHub?

GitHub is a unique social network tailored for developers, serving as a comprehensive platform for project management, source code storage, version tracking, and collaboration on software projects. It enables programmers to clone source code from repositories, acting as a public repository hosting service where anyone can create an account and establish their own repositories to work on personal or collaborative projects.



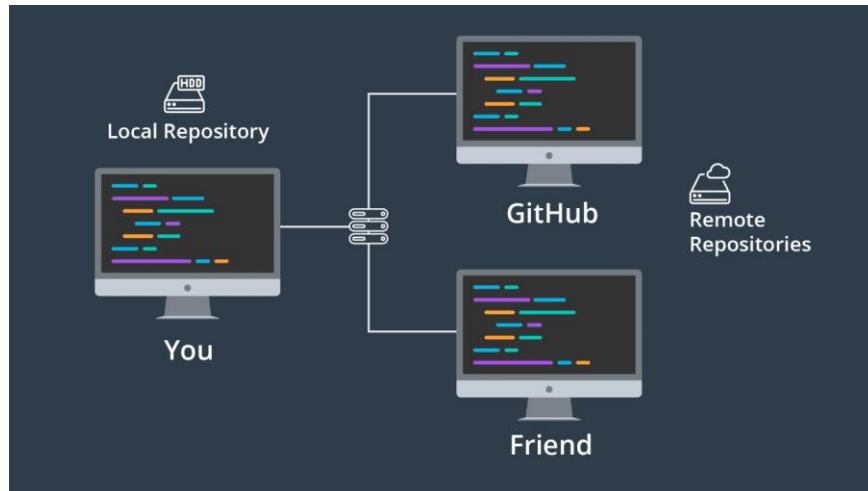
Beyond its role as a storage solution, GitHub is a globally recognized service that provides Git-based repositories for software development. It encompasses all the essential features of Git while enhancing them with social networking tools, allowing developers to connect, share, and learn from one another. GitHub offers two tiers: a free version, widely used by individuals for storing source code, and a paid version, often utilized by businesses to improve team management, access control, and project security.

GitHub introduces social networking features such as feeds, followers, and network graphs, enabling developers to gain insights and expertise by reviewing commit histories. While a code comment explains a specific snippet, a commit message on GitHub describes the actions taken on the source code, providing clarity and context for every change.

GitHub has become a cornerstone of the open-source community. Alongside LinkedIn, it serves as a modern alternative to a traditional resume. Recruiters frequently examine GitHub profiles to assess a candidate's coding skills, making proficiency in Git and GitHub a mandatory requirement for job applicants in today's tech landscape.

2.9 How Does GitHub Work?

GitHub empowers developers to create accounts, upload files, and manage programming projects efficiently. Most software development involves teams, whether co-located or remote, working synchronously or asynchronously. This diversity poses collaboration challenges, but GitHub simplifies the process with its robust features.



Centralized Code and Documentation

GitHub centralizes all source code and project documentation within repositories. This ensures that contributors have seamless access to necessary resources, minimizing information gaps. Each repository typically includes detailed guidelines, helping participants understand the project's goals and rules.

Conflict Management

Programming is more than just writing code—it requires creativity and adaptability. For instance, two developers might work on separate code segments that need to function together harmoniously. A change in one part could inadvertently break another or cause unexpected behavior. GitHub addresses this by displaying changes from multiple contributors before they are pushed to the main branch, allowing teams to identify and resolve potential conflicts early.

Version Tracking and Recovery

One of GitHub's standout features is its version control system, built on Git technology. It tracks every change through commits, enabling developers to revert to previous versions if errors arise or to review who made specific changes and when. This capability is invaluable for maintaining project integrity and debugging.

A Social Network for Developers

GitHub doubles as the largest and most user-friendly social platform for developers, offering core features like wikis, issue tracking, statistics, project renaming, and user-

specific namespaces. Developers can “watch” projects to monitor others’ progress or “follow” users to stay updated on their activities. There are two primary ways to engage with GitHub: creating your own projects or contributing to existing ones by forking a repository, making changes, and submitting pull requests for review.

2.10 Git vs. GitHub: What’s the Difference?

While GitHub is built on Git, the two are distinct. Git is an open-source version control system that tracks code changes locally, usable without GitHub. GitHub, however, is a cloud-based platform that extends Git’s functionality with a user-friendly web interface, remote storage, collaboration tools, and bug tracking. It enhances Git by facilitating teamwork and remote collaboration.



Key Git Concepts to Understand

git: The command-line prefix for Git operations.

branch: A separate line of development, allowing multiple versions to diverge and evolve independently.

commit: A snapshot of changes in the project’s work tree.

clone: Copying a repository from a Git-based system (e.g., GitHub, GitLab) to your local machine for further development.

fork: Duplicating someone else’s repository to your GitHub account, treating it as your own.

repository: A storage space for project data and source code.

tag: A marker for a specific commit, useful for managing numerous commits.

remote: A repository hosted on a Git server, managed similarly to local branches.

diff: A comparison showing differences between the current version and another.

.gitignore: A file specifying which files or directories to exclude from being pushed to the server.

A Brief History of GitHub

GitHub was developed using Ruby on Rails and Erlang by Tom Preston-Werner, Chris Wanstrath, and PJ Hyett. Officially launched in April 2008, it grew rapidly. By March 2018, GitHub had become the world's largest code-hosting service, boasting over 25 million users and 80 million project repositories. It remains an indispensable tool for the open-source and developer communities globally.

Benefits of GitHub for Developers

Simplified Source Code Management

When you create a repository, GitHub stores all its source code, allowing you to review your work through commit messages. Multiple developers can collaborate on the same repository, with visibility into who committed what. Projects can evolve across multiple branches, which are eventually merged into the main branch (often called “master”) to complete the project.

Version Change Tracking

Tracking revisions in a multi-developer project can be complex—who changed what, when, and where are the files stored? GitHub solves this by maintaining a history of all changes pushed to the repository, similar to version histories in Microsoft Word or Google Drive, ensuring no prior work is lost.

Markdown Support

GitHub supports Markdown, a lightweight text formatting language for the web. With Markdown, developers can format text (e.g., bold, italics), add images, and create lists using simple symbols like # or *. It's widely used in GitHub for wikis, issue comments, pull requests, and files with .md or .markdown extensions.

Proving Your Developer Credentials

While a well-crafted resume is essential, source code is the ultimate proof of your skills. A GitHub profile in your CV can set you apart from other candidates, acting as a “lie detector” for recruiters to distinguish genuine developers from pretenders. A strong GitHub presence, with contributions to the community or personal projects, significantly boosts your credibility and employability.

Skill Improvement and Bug Tracking

GitHub is a learning goldmine, with countless open-source projects, contributors, and daily commits. By studying these, developers can refine their coding skills. Its built-in “bug tracking” feature simplifies identifying and fixing issues, with dashboards that filter and organize data by popularity, update time, or relevance—earning it high praise in the developer community.


A Treasure Trove of Resources

The “Explore” feature lets you discover open-source projects aligned with your preferred technologies. GitHub’s code search works across projects and websites, enhanced by strong SEO, making it easy to find publicly shared code snippets.

GitHub Actions

GitHub Actions enable automated workflows on its servers, responding to repository events or performing tasks. For example, a tool like “Autotagger” can automatically generate tags when a package.json version changes—a small but impactful feature that saves maintainers time and effort.

In summary, GitHub is more than a code repository—it’s a dynamic ecosystem that empowers developers to collaborate, learn, and showcase their talents, making it an essential tool in the modern programming world.



GitHub Action

Autotagger

1.0.2 Latest version

Use latest version

Autotag

This action will read a `package.json` file and compare the `version` attribute to the project's known tags. If a corresponding tag does not exist, it will be created.

Usage

The following is an example `.github/main.workflow` that will execute when a `push` to the `master` branch occurs.

```

name: My Workflow
on:
  push:
    branches:
      - master
jobs:
  build:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@master
      - uses: butlerlogic/action-autotag@stable
    with:
      GITHUB_TOKEN: "${{ secrets.GITHUB_TOKEN }}"

```

To make this work, the workflow must have the checkout action *before* the autotag action.

This order is important!

```

- uses: actions/checkout@master
- uses: butlerlogic/action-autotag@1.0.0


```

Stars

★ Unstar

1

Contributors



Categories

Publishing

Deployment

Links

ButlerLogic/action-autotag

Open issues 1

Pull requests 1

Report abuse

Autotagger is not certified by GitHub. It is provided by a third-party and is governed by separate terms of service, privacy policy, and support documentation.

GitHub Package Registry

This package registry allows developers to maintain their own distribution registries, including npm, Docker, Maven, NuGet, and RubyGems. Don't hesitate to create a GitHub account right away. Build your own projects and share them with others, or feel free to fork an open-source project. Create pull requests or issues if you find bugs or need support.

bash

npm

Docker

Maven

NuGet

RubyGems

```

$ npm login --registry=https://npm.pkg.github.com --scope=@phanatic
Successfully logged in.

$ npm publish
Package published

```

Read developer docs →

Expand Your Network

Meet new developers: Thousands of developers worldwide are part of GitHub's vast network, sharing their experiences and brilliant ideas.

Share your own expertise: Git allows users to share code, text snippets, or any information with other developers. You can use it to exchange text, and gists work like Git repositories, enabling you to branch and update versions.

Getting Started with GitHub

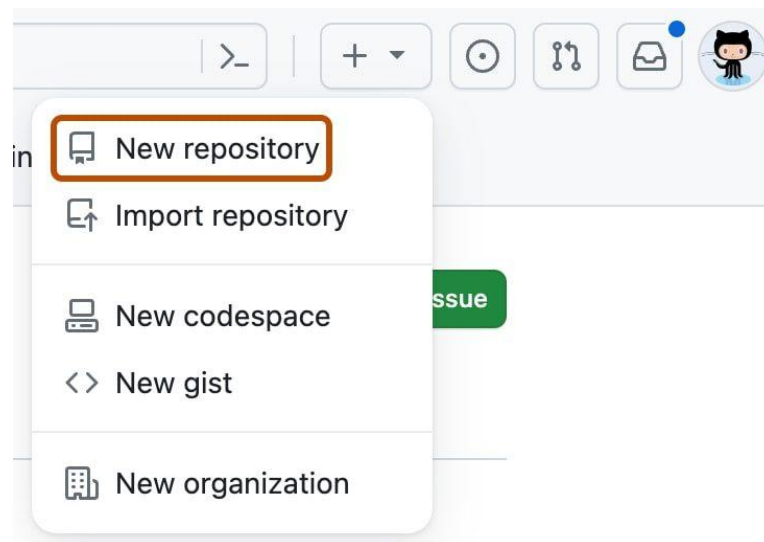
Here are the basic steps to begin using GitHub:

Step 1: Create a GitHub Account
Visit GitHub and sign up for a free account.

Step 2: Create a New Repository:
A repository is where you store your code and project-related files. To create a new repository:

Click the plus (+) icon in the top-right corner and select "New repository."

Fill in the required details like name, description, and privacy settings (public or private).



Step 3: Install Git on Your Computer

You'll need Git installed to interact with GitHub from the command line. Download Git from [Git downloads](#) and install it.

Step 4: Connect Git to GitHub

Open a terminal or command prompt and configure Git with your GitHub account details:

```
git config --global user.name "Your GitHub Username"
git config --global user.email "Your GitHub Email"
```

Step 5: Initialize a Git Repository on Your Computer

Navigate to your project folder and initialize a Git repository by running:

```
git init
```

Step 6: Add Your Code to the Repository

Add your files to the repository with:

```
git add
```

Step 7: Commit Your Code

Create a commit to save the state of your code:

```
git commit -m "First commit"
```

Step 8: Connect Your Local Repository to GitHub

Link your local repository to your GitHub repository:

```
git remote add origin https://github.com/username/repository-name.git
git push -u origin master
```

Step 9: Use GitHub Desktop (Optional)

If you prefer not to use the command line, GitHub Desktop is a user-friendly GUI application. Download and install it from GitHub Desktop.

Step 10: Create and Manage Branches

Branches let you work on new features or fixes without affecting the main codebase. To create a new branch:

```
git checkout -b feature-branch
```

Step 11: Push Changes to GitHub

After completing your changes, push them to your GitHub repository:

```
git push origin feature-branch
```

2.11 Conclusion

Git is a version control system for managing code changes, while GitHub is a platform enhancing Git with collaboration tools, popular in Vietnam for project management. Together, they support efficient development, with Git handling technical control and GitHub adding social and hosting features, making them essential for modern software teams.

Chapter 3. Apply Git/GitHub to Projects

3.1 Git, GitHub and real application demo images

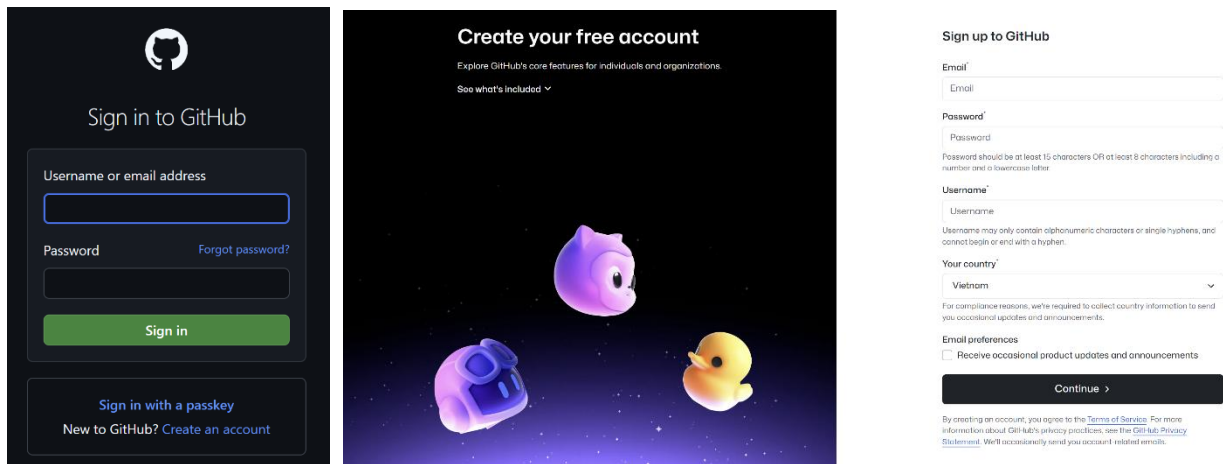


Figure 1. Sign-in and Sign-up

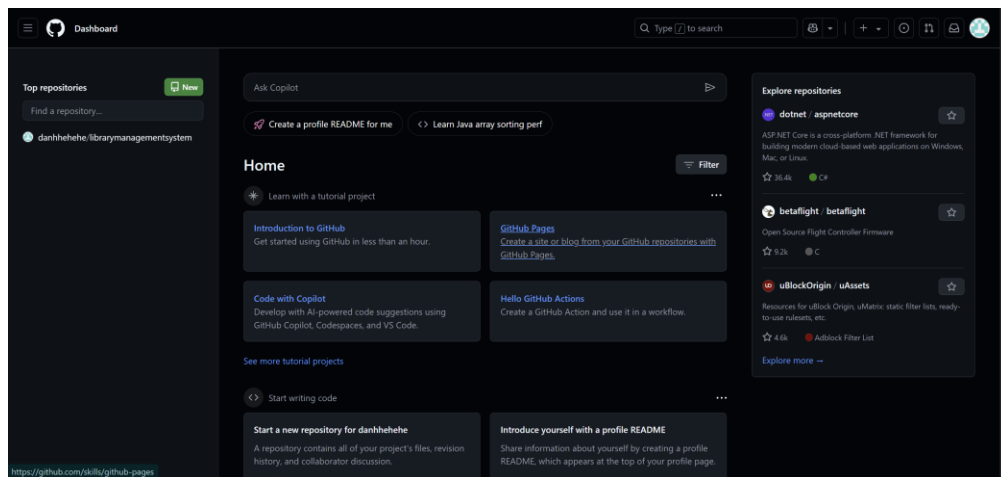


Figure 2. Home Page GitHub After logging in, you will be directed to GitHub Home page.

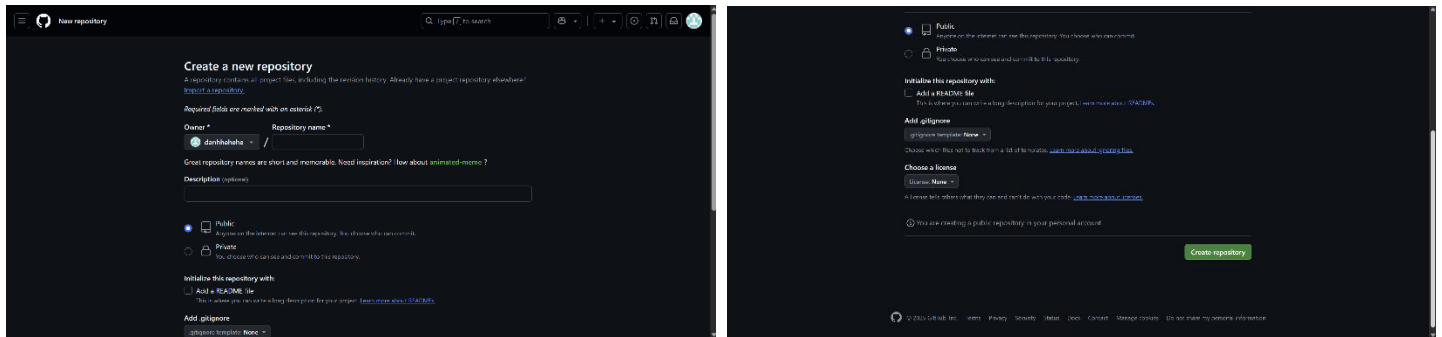


Figure 3. Create a new repository

When you click “New” on the home page to the “Create a new repository” interface on GitHub, you need to do the following steps to create a new repository:

Fill in the repository information:

Owner:

This field is usually pre-filled with your username or organization.

Make sure it is correct.

Repository name I will enter “librarymanagementsystem” you can enter the name of your project.

Description is a short description of your repository (optional).

Public or Private can be Public or Private here I will choose the default Public

Initialize the repository (optional):

Add a README file I will enter Downloaded from www.codewithc.com Thank you for using Code with C! to README. This is a text file that explains your project. It is highly recommended to add this file to make it easier for others to understand the repository.

Add .gitignore and some others I will leave as default, please fill in if required.

After filling in all the information, click the "Create repository" button. GitHub will create a new repository for you.

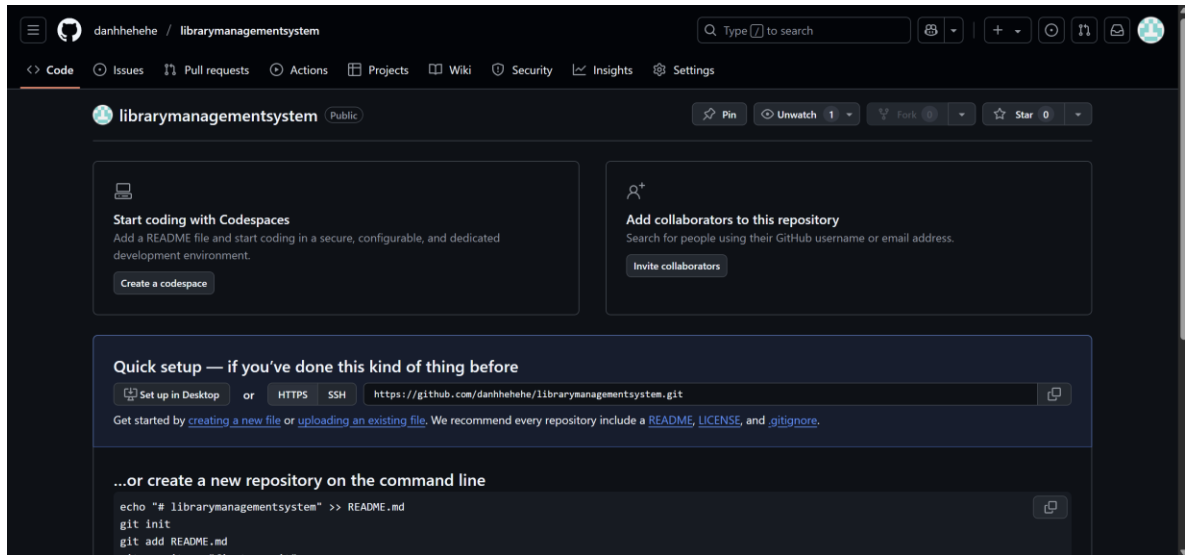


Figure 4. Quick setup new repository for you.

This interface includes settings if you have not used it before

Functional buttons:

"Pin": Pin the repository to the top of the page.

"Unwatch": Unfollow the repository.

"Star 0": Star (favorite) the repository (no one has starred it yet).

"Start coding with Codespaces" section:

Encourages the use of Codespaces to write code directly in the browser.

"Create a codespace" button: Create a Codespace environment.

"Add collaborators to this repository" section:

Allows you to add collaborators to the repository.

"Invite collaborators" button: Invite others to collaborate.

"Quick setup - if you've done this kind of thing before" section:

Provides options for quickly setting up the repository:

"Set up in Desktop": Set up on GitHub Desktop.

"HTTPS" or "SSH": Copy the repository URL (HTTPS or SSH).

The guide starts by creating a new file or uploading an existing one.

It is recommended to add README, LICENSE, and .gitignore files.

The "...or create a new repository on the command line" section:

Provides Git commands to create a repository on the command line:

`echo "# librarymanagementsystem" >> README.md` (creates a README file).

`git init` (initializes a Git repository).

`git add README.md` (adds a README file to a commit).

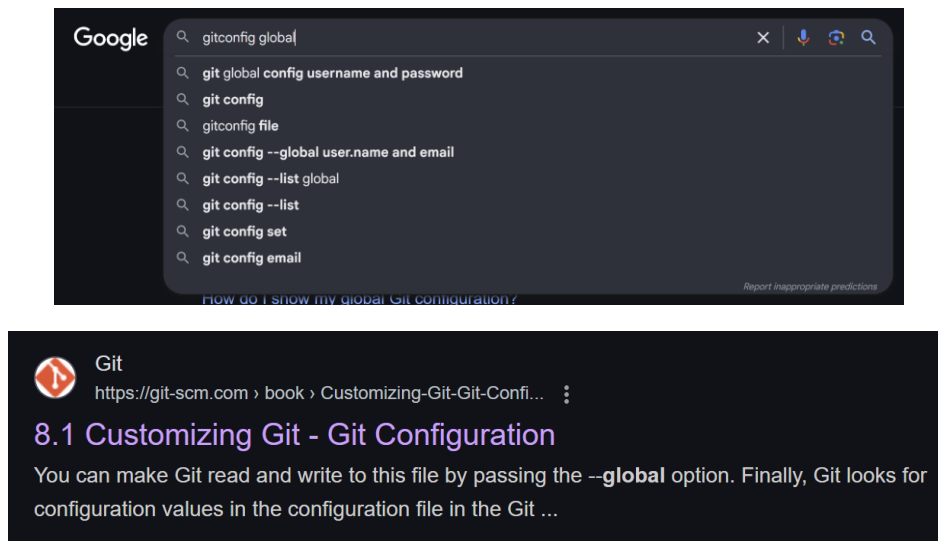


Figure 5. Customizing Git - Git Configuration

Please select a browser to access with the phrase "gitconfig global".

Then select 8.1 Customizing Git - Git Configuration.

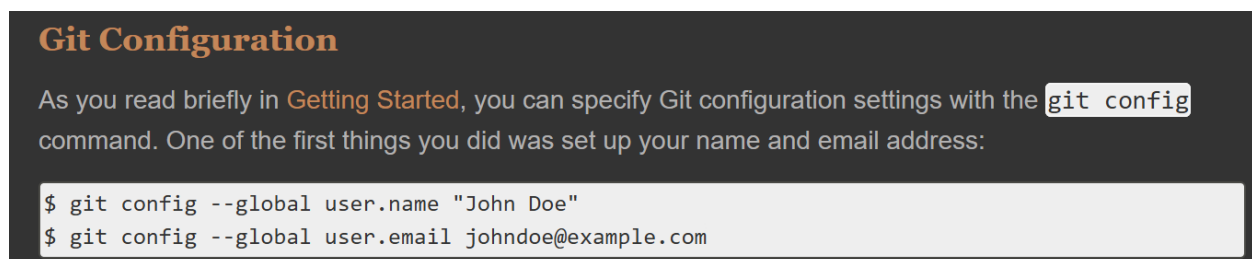


Figure 6. Git Configuration

Now let's follow the steps to apply git and github to your project.

3.2 Git Configuration

As you read briefly in [Getting Started](#), you can specify Git configuration settings with the `git config` command. One of the first things you did was set up your name and email address:

```
HUYNH THANH DANH@DESKTOP-TKL5MOV MINGW64 ~  
$ git config --global user.name "danhhehehe"  
  
HUYNH THANH DANH@DESKTOP-TKL5MOV MINGW64 ~  
$ git config --global user.email huynhtoan166714@gmail.com  
  
HUYNH THANH DANH@DESKTOP-TKL5MOV MINGW64 ~  
$
```

```
HUYNH THANH DANH@DESKTOP-TKL5MOV MINGW64 ~  
$ git config --list  
diff.astextplain.textconv=astextplain  
filter.lfs.clean=git-lfs clean -- %f  
filter.lfs.smudge=git-lfs smudge -- %f  
filter.lfs.process=git-lfs filter-process  
filter.lfs.required=true  
http.sslbackend=schannel  
core.autocrlf=true  
core.fscache=true  
core.symlinks=false  
pull.rebase=false  
credential.helper=manager  
credential.https://dev.azure.com.usehttppath=true  
init.defaultbranch=master  
user.name=danhhehehe  
user.email=huynhtoan166714@gmail.com  
  
HUYNH THANH DANH@DESKTOP-TKL5MOV MINGW64 ~  
$ |
```

Figure 7. Configure your Git name and email to use for all Git repositories on this computer.

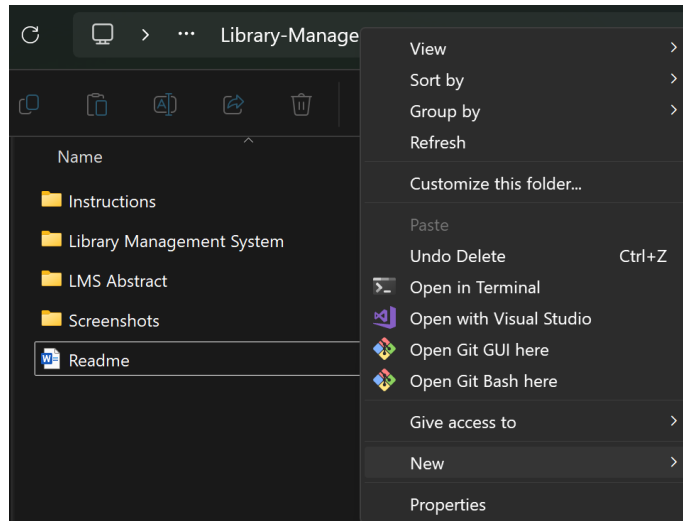


Figure 8. Right click and select "Open Git Bash here"

```
HUYNH THANH DANH@DESKTOP-TKL5MOV MINGW64 /d/Tools and Environments for Software
Development/Library-Management-System-Java-Project (master)
$ git init
Reinitialized existing Git repository in D:/Tools and Environments for Software
Development/Library-Management-System-Java-Project/.git/

HUYNH THANH DANH@DESKTOP-TKL5MOV MINGW64 /d/Tools and Environments for Software
Development/Library-Management-System-Java-Project (master)
$
```

Name	Date modified	Type	Size
.git	4/3/2025 4:58 AM	File folder	
Instructions	5/18/2015 10:31 AM	File folder	
Library Management System	5/18/2015 10:23 AM	File folder	
LMS Abstract	5/18/2015 10:59 AM	File folder	
Screenshots	4/3/2025 3:03 AM	File folder	
Readme	5/2/2015 11:34 PM	Microsoft Word Doc...	5 KB

Figure 9. The Git repository already contains the project "Library-Management-System-Java-Project".

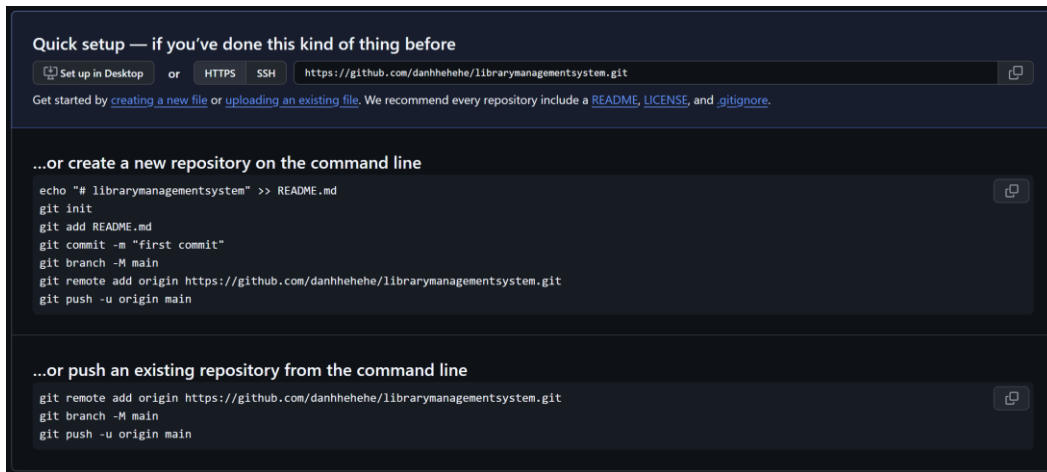


Figure 10. This is a detailed page with instructions on setting up a Git repository on GitHub.

```
HUYNH THANH DANH@DESKTOP-TKL5MOV MINGW64 /d/Tools and Environments for Software Development/Library-Management-System-Java-Project (master)
$ git remote add origin https://github.com/danhhehehe/librarymanagementsystem.git
HUYNH THANH DANH@DESKTOP-TKL5MOV MINGW64 /d/Tools and Environments for Software Development/Library-Management-System-Java-Project (master)
$
```

Figure 11. Add an "origin" remote to your Git repository.

```
HUYNH THANH DANH@DESKTOP-TKL5MOV MINGW64 /d/Tools and Environments for Software Development/Library-Management-System-Java-Project (master)
$ git remote -v
origin https://github.com/danhhehehe/librarymanagementsystem.git (fetch)
origin https://github.com/danhhehehe/librarymanagementsystem.git (push)
HUYNH THANH DANH@DESKTOP-TKL5MOV MINGW64 /d/Tools and Environments for Software Development/Library-Management-System-Java-Project (master)
$ |
```

Figure 12. Configure the "origin" remote to point to the Git repository on GitHub.

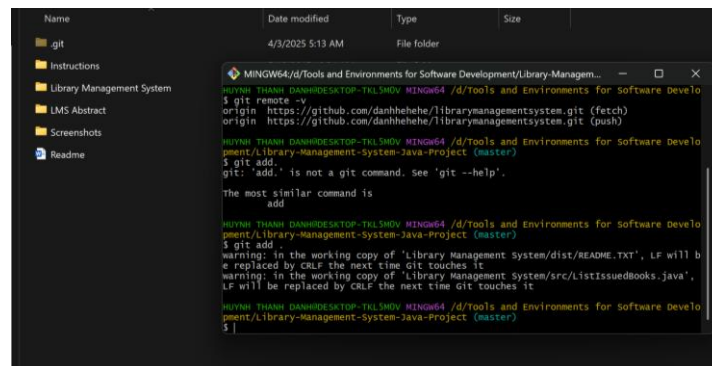


Figure 13. Adds all changes in the current directory to the "staging area", preparing you for making a commit.

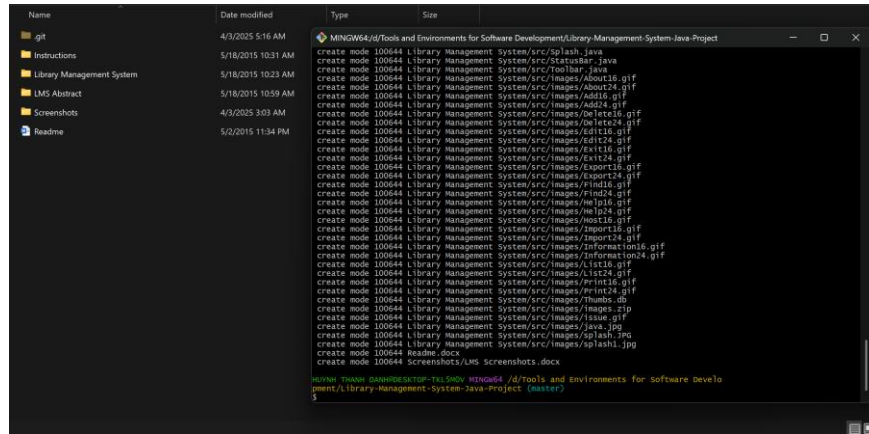


Figure 14. Create the first commit, marking the beginning of the project.

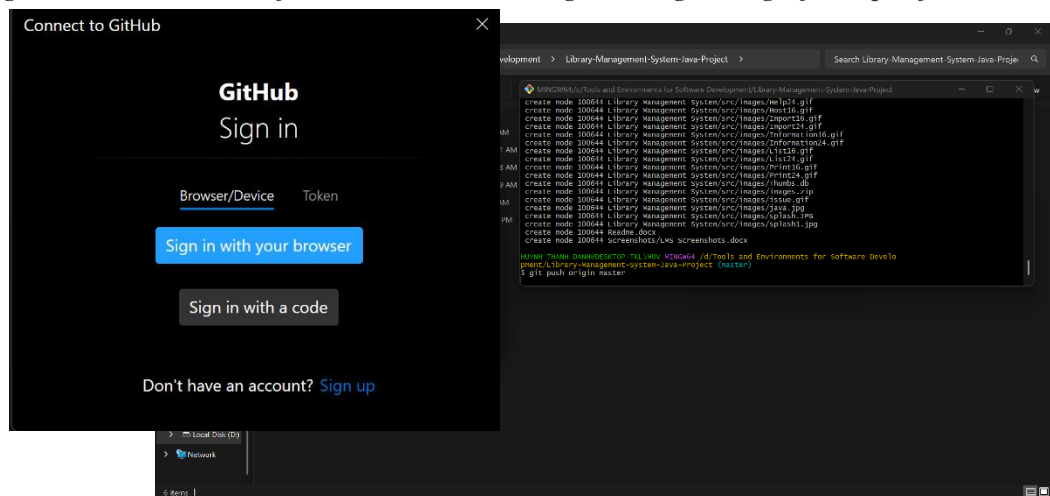


Figure 15. Add new files to the "staging area" and push them to the GitHub repository and the user needs to log in via browser to GitHub to be able to push commits to the repository.

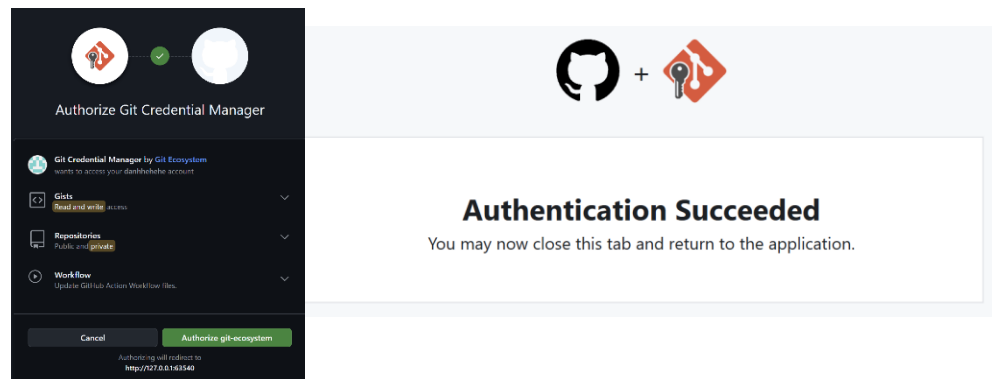


Figure 16. Connection interface

```
HUYNH THANH DANH@DESKTOP-TKL5M0V MINGW64 /d/Tools and Environments for Software Development/Library-Management-System-Java-Project (master)
$ git push origin master
info: please complete authentication in your browser...
Enumerating objects: 195, done.
Counting objects: 100% (195/195), done.
Delta compression using up to 20 threads
Compressing objects: 100% (193/193), done.
Writing objects: 100% (195/195), 870.32 KiB | 13.39 MiB/s, done.
Total 195 (delta 59), reused 0 (delta 0), pack-reused 0 (from 0)
remote: Resolving deltas: 100% (59/59), done.
To https://github.com/danhhehe/librarymanagementsystem.git
* [new branch]      master -> master
```

Figure 17. Successfully pushed commits from local "master" branch to "master" branch on GitHub repository

Name	Date modified	Type	Size
hooks	4/3/2025 4:57 AM	File folder	
info	4/3/2025 4:57 AM	File folder	
logs	4/3/2025 5:16 AM	File folder	
objects	4/3/2025 5:16 AM	File folder	
refs	4/3/2025 4:57 AM	File folder	
COMMIT_EDITMSG	4/3/2025 5:16 AM	File	1 KB
config	4/3/2025 5:10 AM	File	1 KB
description	4/3/2025 4:57 AM	File	1 KB
HEAD	4/3/2025 4:57 AM	File	1 KB
index	4/3/2025 5:16 AM	File	26 KB

Figure 18. Basic structure of the .git directory in a standard Git repository.

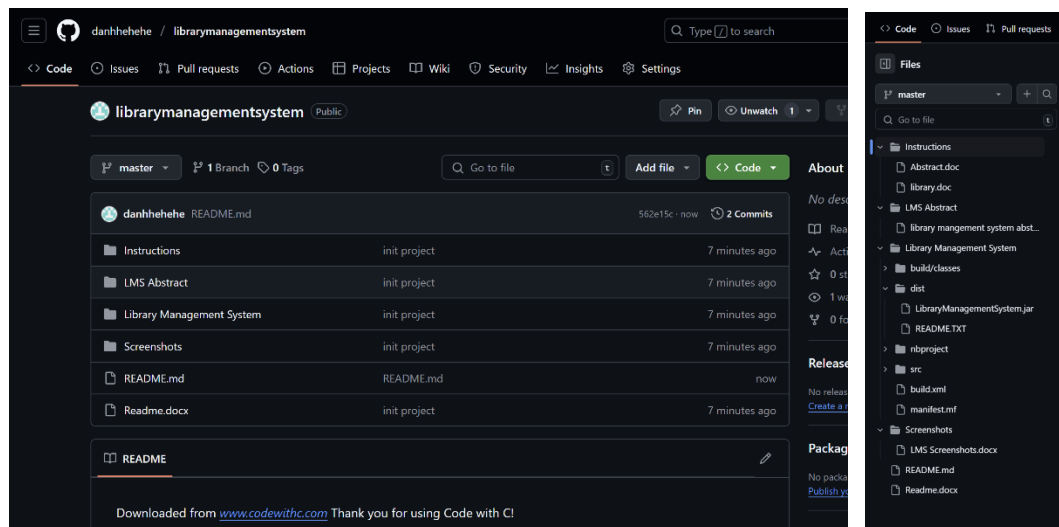


Figure 19. It contains the basic files and folders of the "Library Management System" project.

3.3 Git, GitHub and Applying it in Practice:

Basic Git Commands

We'll categorize and explain the basic Git commands used in the provided log, and then discuss their practical application.

Commands Used (From the Log):

`git init:`

Function: Initializes a new Git repository in the current directory.

Application:

Used at the start of a new project to turn the project directory into a Git repository.

In the log, it was used (or re-used) to ensure the project was under Git version control.

`git remote add origin <repository_url>:`

Function: Adds a remote repository (usually on GitHub or GitLab) to your local Git repository.

Application:

Connects your local project to a remote repository, enabling you to push and pull changes.

`origin` is the standard name for the main remote repository.

`git remote -v:`

Function: Lists the configured remote repositories, showing their names and URLs.

Application:

Used to verify that the remote repository has been added correctly.

Helps to confirm the URL of the remote.

`git add <files> or git add .:`

Function: Adds changes in the working directory to the staging area.

Application:

`git add <files>` adds specific files.

`git add .` adds all changes (new, modified, deleted) in the current directory and subdirectories.

Prepares changes for the next commit.

In the log, it was used to stage all the files of the project.

`git commit -m "<commit_message>":`

Function: Creates a new commit with the staged changes.

Application:

Saves a snapshot of the staged changes in the Git history.

The `-m` option allows you to add a commit message directly in the command.

Commit messages should be clear and concise, describing the changes made.

In the log, it was used to make the initial commit of the project.

`git push origin master:`

Function: Pushes the commits from the local master branch to the remote origin repository's master branch.

Application:

Uploads your local commits to the remote repository, making them accessible to others.

Used to synchronize local and remote repositories.

In the log, it was used to push the initial commit to the GitHub repository.

Practical Application and Workflow:

Start a new project:

`git init` to initialize a Git repository.

Write your code.

Connect to a remote repository:

`git remote add origin <repository_url>` to link your local repository to a remote one (e.g., on GitHub).

Make changes:

Edit, create, or delete files.

Stage changes:

`git add <files>` or `git add .` to add changes to the staging area.

Commit changes:

`git commit -m "Descriptive commit message"` to create a commit.

Push changes:

`git push origin master` to upload commits to the remote repository.

Repeat:

Continue making changes, staging, committing, and pushing.

Additional Basic Git Commands (Not in the Log but Essential):

git status: Shows the status of the working directory and staging area.

git log: Displays the commit history.

git pull origin master: Fetches and merges changes from the remote master branch into your local master branch.

Real-World Applications of Git

Git's versatility makes it indispensable across various software development scenarios:

Personal Software Development:

Git enables developers to track changes in their projects, revert to previous versions, and experiment with new features without risking the stability of the main codebase.

It facilitates efficient management of individual projects, allowing for organized development and easy maintenance.

Even for solo projects, Git's branching capabilities allow developers to explore different ideas without affecting the stable version of their code.

Team Collaboration:

Platforms like GitHub and GitLab leverage Git to provide robust collaboration tools.

Multiple developers can work on the same project simultaneously, with Git managing the merging of changes and resolving conflicts.

Pull requests and code reviews facilitate a structured and transparent development process, ensuring code quality and consistency.

Git enables teams to maintain a shared, centralized repository, streamlining communication and coordination.

DevOps and CI/CD (Continuous Integration/Continuous Deployment):

Git is a cornerstone of DevOps practices, serving as the source code repository for automated build, test, and deployment pipelines.

CI/CD tools integrate with Git to trigger automated processes whenever changes are pushed to the repository.

This automation ensures rapid feedback on code changes, enabling faster and more reliable software releases.

Git's branching model supports different deployment environments (e.g., development, staging, production).

Open Sources Contributions:

GitHub has become the central hub for open-source software development.

Git allows developers to contribute to open-source projects by forking repositories, making changes, and submitting pull requests.

This fosters a collaborative environment where developers can share knowledge, improve existing software, and contribute to the community.

Understanding Git is essential for anyone wanting to participate in the open-source ecosystem.

3.4 Conclusion

By understanding how to use Git and GitHub, through lessons that Huynh Thanh Danh and the group absorbed, self-taught, and received enthusiastic guidance from teacher Nguyen Van Tan, we confidently work on large-scale projects. We contribute to open-source projects. And we understand more clearly the importance of teamwork and how crucial task allocation is. After using Git and GitHub, we have witnessed the revolution in modern software development, becoming indispensable tools for both individual developers and large teams. Their impact extends beyond simple version control, influencing how software is developed, tested, and deployed. Working with Git and GitHub is easier than traditional methods, achieving higher productivity when managing projects. Git and GitHub have fostered a vibrant open-source community, promoting collaboration and free knowledge sharing.

Reference

1. Code With C. (n.d.). *Library Management System Java Project*. Retrieved from <https://www.codewithc.com/library-management-system-java-project>
2. Entab. (n.d.). *Library Management Software Benefits*. Retrieved from <https://www.entab.in/library-management-software.html>
3. GeeksforGeeks. (n.d.). *Library Management System*. Retrieved from <https://www.geeksforgeeks.org/library-management-system>
4. MasterSoft. (n.d.). *Library Management Software*. Retrieved from <https://www.iitms.co.in/library-management-software>
5. Git SCM. (n.d.). *About Version Control*. Retrieved from <https://git-scm.com/book/en/v2/Getting-Started-About-Version-Control>
6. GitHub. (n.d.). *About Page - Build Software Better, Together*. Retrieved from <https://github.com/about>
7. TechTarget. (n.d.). *What Is GitHub? Definition and Explanation*. Retrieved from <https://www.techtarget.com/whatis/definition/GitHub>
8. GeeksforGeeks. (n.d.). *What is Git Version Control?*. Retrieved from <https://www.geeksforgeeks.org/version-control-system>
9. How-To Geek. (n.d.). *What Is GitHub and What Do Geeks Use It For?*. Retrieved from <https://www.howtogeek.com/whatisgithub>
10. GitHub. (n.d.). *Features - Build and Ship Software*. Retrieved from <https://github.com/features>
11. GitHub Blog. (n.d.). *7 Unique Software Collaboration Features in GitHub Discussions*. Retrieved from <https://github.blog>
12. Atlassian. (n.d.). *Git Tutorial - What is Version Control?*. Retrieved from <https://www.atlassian.com/git/tutorials/what-is-version-control>
13. TopDev. (n.d.). *GitHub là gì?*. Retrieved from <https://topdev.vn/blog/github-la-gi>
14. Viblo. (n.d.). *Phân biệt về Git và GitHub*. Retrieved from <https://viblo.asia/p/phan-biet-git-va-github>
15. Unitop. (n.d.). *Git và GitHub*. Retrieved from <https://unitop.vn/git-va-github>

16. 200Lab Blog. (n.d.). *Git là gì?*. Retrieved from <https://200lab.io/blog/git-la-gi>
17. ThachPham. (n.d.). *Git và GitHub là gì?*. Retrieved from <https://thachpham.com/tools/git-va-github.html>
18. HocChudong. (n.d.). *GitHub for Sysadmin*. Retrieved from <https://hocchudong.com/github-for-sysadmin>
19. Got It Vietnam. (n.d.). *Git và GitHub*. Retrieved from <https://medium.com/@gotitvietnam/git-va-github>