ISSN: 2755-0109

Journal of Media & Management

SCIENTIFIC Research and Community

Review Article

Open Access

Comprehensive Study of Git and Github & Implementing Them as Learning Objectives in Modern Education

Harika Sanugommula

Independent Researcher

ABSTRACT

Git and GitHub are pivotal in modern software development, providing developers with robust tools for version control and collaboration. Git, a distributed version control system, allows multiple developers to work on code simultaneously while maintaining a comprehensive history of changes. GitHub, built on Git, enhances this functionality by offering a platform for hosting repositories, facilitating collaboration, and integrating various development tools. This paper explores the architecture of Git, the features of Git & GitHub, common workflows, how to integrate Git & GitHub, best practices along with mainly implementing Git & GitHub in modern day ed

***Corresponding author**

Harika Sanugommula, Independent Researcher.

Received: November 15, 2022; **Accepted:** November 22, 2022; **Published:** November 29, 2022

Keywords: Git, GitHub, Version Control, Collaboration, Software Development

Introduction

In the rapidly evolving field of software development, effective version control and collaboration are essential for success. Git, created by Linus Torvalds in 2005, is a distributed version control system that allows developers to manage code changes efficiently. GitHub, launched in 2008, leverages Git's capabilities by providing a user friendly platform for hosting and sharing repositories. This paper examines the architecture of Git, the features of GitHub, common workflows, and best practices to optimize development processes.

Git Architecture

Distributed Version Control: Each developer has a complete copy of the repository, enabling offline access and faster operations.
Snapshots: Git tracks changes as snapshots, allowing users to view the entire project state at any commit.

Features on Git & GitHub

Git and GitHub offer essential tools for version control and collaboration, each with unique features that support developers in managing code effectively. Core features of Git include its powerful branching and merging capabilities, allowing developers to create separate branches for different features or bug fixes, which can then be merged back into the main branch when ready. This promotes parallel development, making it easier for multiple team members to work on various aspects of a project simultaneously. Git also includes a staging area, a temporary space where developers can prepare changes before committing them to the repository. This feature allows for selective inclusion of changes, ensuring that only finalized updates are committed. Additionally, Git maintains a comprehensive commit history that logs all changes made to a project, providing a chronological record that can be used to review or revert to previous versions as needed.

GitHub builds on Git's functionality by offering a range of collaborative features through a centralized platform. It provides repository hosting, making it easy to store, share, and manage Git repositories. GitHub also includes robust collaboration tools like pull requests, issues, and project boards, which streamline team communication and enable developers to discuss and review changes before merging them into the main codebase. Furthermore, GitHub supports integration with CI/CD (Continuous Integration and Continuous Deployment) tools, enabling teams to automate testing, deployment, and other workflows directly within the platform. This integration enhances efficiency by allowing code to be automatically tested and deployed after every change, ensuring a consistent and reliable development pipeline. Together, Git and GitHub provide a powerful suite of tools for version control and collaborative coding, promoting both individual productivity and seamless teamwork.

Git	GitHub
1. It is a software	1. It is a service
2. It is installed locally on the system	2. It is hosted on Web
3. It is a command line tool	3. It provides a graphical interface
4. It is a tool to manage different versions of edits, made to files in a git repository	4. It is a space to upload a copy of the Git repository
5. It provides functionalities like Version Control System Source Code Management	5. It provides functionalities of Git like VCS, Source Code Management as well as adding few of its own features

Source: https://medium.com/edureka/gitvsgithub67c511d09d3e

Common Workflows

- **Centralized Workflow:** All changes are made in a central repository, suitable for small teams.
- **Feature Branch Workflow:** Each new feature is developed in its own branch, promoting isolation and easier code review.
- **Fork and Pull Request Workflow:** Common in opensource projects, where developers fork repositories, make changes, and submit pull requests for review.

Why and how to Integrate Git & GitHub?
Integrating Git and GitHub is necessary to enhance collaboration among developers, streamline version control, and provide remote access to code. It facilitates code reviews through pull requests, offers backup and redundancy, and integrates with CI/CD tools for automated testing and deployment. Additionally, it supports project management with issue tracking and allows teams to maintain organized workflows, making it essential for efficient software development. The integration of Git and GitHub occurs through a series of steps that facilitate seamless collaboration and version control in software development.

Setting Up a GitHub Repository
Create a New Repository: Users start by creating a new repository on GitHub, which serves as a central location for their project.
Initialize with a README: Optionally, you can initialize the repository with a README file to describe the project.

Cloning the Repository
Developers clone the GitHub repository to their local machines using the following command: git clone <repositoryurl>
This command downloads a copy of the repository, allowing developers to work on it locally.

Making Changes Locally
Creating Branches: Developers create branches for new features or bug fixes using:
git checkout b featurebranch
Committing Changes: After making changes, developer's stage and commit those changes:
git add <file>
git commit m "Description of changes"

Pushing Changes to GitHub
Once local changes are committed, developers push their branches back to the GitHub repository: git push origin featurebranch
This updates the remote repository with the new commits, making them accessible to collaborators.

Creating a Pull Request
After pushing changes, developers can initiate a pull request (PR) on GitHub to merge their changes into the main branch. This is typically done through the GitHub web interface, where they can describe the changes and request reviews.

Code Review and Collaboration
Team members can review the pull request, comment on specific lines of code, and suggest changes. This collaborative review process helps maintain code quality. Once the PR is approved, it can be merged into the main branch via GitHub.

Merging Changes
After approval, the changes are merged into the main branch using the GitHub interface. This can involve either a fastforward merge or a merge commit, depending on the project's workflow.

Continuous Integration/Deployment (CI/CD)
Many teams integrate CI/CD tools with GitHub to automate testing and deployment processes. For example, when a pull request is created or updated, automated tests can run to ensure that the code changes do not introduce new issues. Tools like GitHub Actions can be configured to deploy the code automatically after successful tests.

Syncing with the Main Branch
Developers frequently sync their branches with the main branch to incorporate the latest changes. This can be done by pulling updates: git pull origin main

A Few Commonly used Commands
1. Git init it initializes an existing directory as a Git repository.
2. Git status displays the status of our working directory.
3. Git log it shows all commits in the current branch's history.
4. Git rm (name of file) delete's the file from project and stage the removal for commit
5. Git commit we can create a new commit from changes added to the staging area.
6. Git checkout by this command we can switch to another branch and check it out into our working directory

Best Practices
Adopting best practices in version control is essential for efficient and effective collaborative development. One key practice is frequent commits, where developers commit changes regularly rather than waiting until larger changes accumulate. Frequent commits capture incremental changes, making it easier to understand the evolution of a project and enabling developers to pinpoint issues when debugging. This approach supports greater transparency, as each change is recorded in the project's history, which can significantly simplify project management, especially in larger teams.

Another crucial best practice is using descriptive commit messages. Each commit should come with a clear, informative message detailing the purpose of the change. Descriptive messages not only help the developer who made the changes but also make it easier for others reviewing the project to understand the context and intent behind each update. Effective messages may include information on what was added, removed, or fixed, giving a snapshot of the codebase's progress and enabling smoother team collaboration.

Lastly, it is essential to regularly sync with remote repositories. Frequent synchronization with the central repository ensures that developers are working with the latest version of the project, helping to prevent conflicts that might arise if multiple people are working on the same files. Regular syncing allows teams to pull in updates and integrate them into their local workspaces, facilitating a smoother and more cohesive development workflow. Together, these best practices frequent commits, clear commit messages, and regular syncing create a structured, communicative, and conflictfree environment, which is especially important for collaborative and largescale projects.

Implementing Git & GitHub in Modern Day Education
Incorporating Git and GitHub into modern education offers students valuable, realworld experience in version control and collaborative development, skills that are essential in today's technology centrist world. Git, a widely used distributed version control system, enables students to manage project changes systematically, track their progress over time, and resolve conflicts in collaborative projects. GitHub, as a platform for hosting and sharing Git repositories, complements these skills by providing an environment for teamwork, where multiple contributors can engage in code reviews, project sharing, and collaborative coding. Together, these tools are pivotal in teaching students how to work effectively within teams on largescale projects and contribute to opensource initiatives.

To successfully integrate Git and GitHub into educational programs, strategies for implementation might include embedding Git/GitHub projects directly into the curriculum. Courses that involve programming or collaborative assignments can incorporate these tools in practical exercises, ensuring that students gain handson experience. Schools and universities can also host workshops and offer online resources to provide foundational skills in Git and GitHub, giving students opportunities to develop confidence in using these tools outside a traditional classroom setting.

The benefits of using Git and GitHub for students are farreaching. Students acquire core skills in version control and software development practices that are in high demand in the job market. Additionally, Git and GitHub projects promote active learning and engagement, encouraging students to explore coding, project management, and collaborative skills in an interactive manner. This exposure to realworld development tools prepares students for future careers in technology, engineering, and other fields where these competencies are crucial.

Conclusion

Git and GitHub have transformed the software development landscape, providing powerful tools for version control and collaboration. By understanding Git's architecture and leveraging GitHub's features, development teams can enhance their workflows, improve code quality, and foster effective collaboration. Adopting best practices ensures that projects are managed efficiently, leading to successful software development outcomes.

References

1. S Chacon, B Straub, Pro Git, Apress, 2014.
2. J Smart (2015) "Version Control with Git," IEEE Software 32: 104107.
3. A Beasley (2016) "The Benefits of Using Git and GitHub in Software Development," Journal of Software Engineering and Applications 9: 2534.
4. M M H S Rahman, R Reaz (2018) "An Introduction to Git and GitHub," International Journal of Computer Applications 179: 1117.
5. "GIT CHEAT SHEET" (2021) Github. https://education.github.com/gitcheatsheeteducation.pdf.