
**Missing Semester - Git Workflow and Best Practices (Week 6)**

# Workflow and Best Practices

Welcome to today's lesson on Git Workflow and Best Practices! Git is a powerful version control system that allows multiple developers to collaborate on a project efficiently. However, without a proper workflow and adherence to best practices, managing a project with Git can become challenging.

In this lesson, we will explore various Git workflows and learn about the best practices that can help streamline your development process. Whether you are a beginner or an experienced developer, understanding Git workflows and best practices is essential for maintaining a clean and organized codebase, preventing conflicts, and ensuring smooth collaboration.

By the end of this lesson, you will have a solid understanding of different Git workflows, such as the centralized workflow, feature branch workflow, and Gitflow, and how to choose the most suitable one for your project. We will also cover essential best practices, including commit guidelines, branch naming conventions, and code review processes.

So, let's dive in and discover the world of Git workflows and best practices to enhance your development experience and make your projects more efficient!# Git Workflow and Best Practices

Git is a powerful version control system that allows developers to track changes in their codebase and collaborate with others. To make the most out of Git, it is important to follow a well-defined workflow and adhere to

best practices. This lesson will cover some commonly used Git workflows and provide guidelines for best practices.

## Git Workflow

### 1. Centralized Workflow

The centralized workflow is a simple and straightforward approach suitable for small teams or solo developers. In this workflow, there is a single central repository that serves as the source of truth. Developers clone this repository, make changes locally, and push their changes back to the central repository.

The steps involved in the centralized workflow are as follows:

1. Clone the central repository to your local machine.
2. Create a new branch for your feature or bug fix.
3. Make changes to the code in your local branch.
4. Commit your changes with descriptive commit messages.
5. Push your branch to the central repository.
6. Create a pull request to merge your changes into the main branch.
7. Review and discuss the changes with other team members.
8. Once approved, merge the pull request and delete the branch.

### 2. Feature Branch Workflow

The feature branch workflow is a popular Git workflow that encourages collaboration and parallel development. In this workflow, each new feature or bug fix is developed in a dedicated branch. Once the feature is complete, it is merged back into the main branch.

The steps involved in the feature branch workflow are as follows:

1. Clone the central repository to your local machine.
2. Create a new branch for your feature or bug fix.
3. Make changes to the code in your local branch.
4. Commit your changes with descriptive commit messages.

5. Push your branch to the central repository.
6. Create a pull request to merge your changes into the main branch.
7. Review and discuss the changes with other team members.
8. Once approved, merge the pull request and delete the branch.

**3. Gitflow Workflow**

The Gitflow workflow is a more complex branching model suitable for larger teams and projects. It provides a structured approach to managing feature development, releases, and hotfixes. This workflow involves two main branches: `master` and `develop`.

The steps involved in the Gitflow workflow are as follows:

1. Clone the central repository to your local machine.
2. Create a new branch for your feature or bug fix from the `develop` branch.
3. Make changes to the code in your local branch.
4. Commit your changes with descriptive commit messages.
5. Push your branch to the central repository.
6. Create a pull request to merge your changes into the `develop` branch.
7. Review and discuss the changes with other team members.
8. Once approved, merge the pull request into the `develop` branch.
9. Periodically, create a release branch from the `develop` branch for stable releases.
10. Perform any necessary bug fixes in the release branch.
11. Merge the release branch into both `develop` and `master` branches.
12. Tag the `master` branch with a version number.
13. Delete the release branch.
14. For hotfixes, create a new branch from the `master` branch, make the necessary changes, and merge it back into both `develop` and `master` branches.

## Best Practices

1. **Commit Frequently**: Make small, logical commits with descriptive commit messages. This helps in tracking changes and understanding the purpose of each commit.
2. **Use Branches**: Avoid making changes directly in the main branch. Instead, create a new branch for each feature or bug fix. This allows for parallel development and easier collaboration.
3. **Pull Before Push**: Always pull the latest changes from the central repository before pushing your changes. This helps in avoiding conflicts and ensures that you are working with the most up-to-date codebase.
4. **Review Code**: Encourage code reviews within your team. Reviewing code helps in catching bugs, improving code quality, and sharing knowledge among team members.
5. **Use Meaningful Branch and Tag Names**: Choose descriptive names for branches and tags to make it easier to understand their purpose and history.
6. **Keep the Repository Clean**: Remove unnecessary files, branches, and tags from the repository to keep it clean and organized.
7. **Backup and Remote Repositories**: Regularly backup your local repository and consider using remote repositories (e.g., GitHub, GitLab) for additional backup and collaboration features.

By following these workflows and best practices, you can effectively utilize Git for version control and collaboration in your development projects.

## Resources and Exercises for Further Studies.

1. [BLOG] Read this guide on how to properly get started with creating projects - https://git-scm.com/book/en/v2/Appendix-C%3A-Git-Commands-Getting-and-Creating-Projects
2. [BLOG] How to create an end to end Github project? - https://docs.github.com/en/get-started/using-github/github-flow