# Team Git Workflow Guide

## Introduction

To streamline our collaboration and ensure high-quality code, we’re implementing a professional Git workflow. This document outlines our branching strategy, commit message conventions, pull request requirements, code review process, and merging practices. Please follow these guidelines closely, as they’ll help us maintain a clean, efficient codebase and make working together smoother.

## 1. Branching Strategy

We’re adopting a structured branching model to organize our work and keep our code history readable and accessible. Here’s the approach we’ll use:

* Main Branch (`main`): Holds production-ready code only. Code merged here should be fully tested and reviewed.
* Develop Branch (`develop`): Contains the latest, completed features and is a step before production.
* Feature Branches: For new features or tasks. Name feature branches based on the task or issue, like `feature/user-auth` or `feature/payment-gateway`.
* Hotfix Branches: For urgent fixes to `main`. Use names like `hotfix/login-issue`.

## 2. Commit Message Guidelines

Consistent, descriptive commit messages help everyone understand what’s been done. We’ll use the \*\*Conventional Commits\*\* style, which follows this format:  
  
type(scope): summary  
  
Types include:

* feat: New feature (e.g., `feat(auth): add login validation`)
* fix: Bug fix (e.g., `fix(profile): resolve profile image upload error`)
* docs: Documentation update
* style: Formatting or styling changes (no code changes)
* refactor: Code improvements without feature/fix changes
* test: Adding or updating tests
* chore: Miscellaneous tasks (build tasks, config changes)

Scope should be the feature or module affected. Summary should be short and specific.

## 3. Pull Requests (PR)

Pull Requests are a critical part of our workflow, ensuring that code is reviewed and approved before it reaches production. Here’s what to include:

* Title: Clear and descriptive, e.g., `Implement user registration and authentication`
* Description: Explain the changes, why they were made, and any relevant details.  
   - Include references to any issues or tasks.  
   - Highlight any dependencies or additional setup needed.
* Checklist: Use this checklist before submitting:  
   - [ ] Code is tested and functioning as expected.  
   - [ ] Documentation is updated if needed.  
   - [ ] Linting is checked and passed.
* Labels: Apply relevant labels to clarify the type of change (`feature`, `bugfix`, `urgent`, etc.).

## 4. Code Review Process

Code reviews help us maintain quality, share knowledge, and catch issues early. Here’s how we’ll handle reviews:

* Assign Reviewers: Each PR should have at least one or two assigned reviewers. Reviewers check for:
* - Functionality: Does it meet the requirement?
* - Code Quality: Is it clean, readable, and efficient?
* - Adherence to guidelines: Does it follow our code style and commit conventions?
* Approval: PRs require at least one approval before merging.
* Continuous Integration (CI): Ensure all automated tests pass before requesting reviews.

## 5. Merge Strategy

Our merging strategy is designed to keep our history clean and understandable.

* Squash and Merge: For most PRs, we’ll squash all commits into a single commit. This helps maintain a simple, clean history.
* Rebase and Merge: Use this if you need to keep individual commits but maintain a linear history.
* Merge Commit: Only used for merging larger feature branches or releases to `develop` or `main` if there’s a need to keep detailed history.

## 6. General Tips for Effective Git Usage

* Sync Regularly: Pull changes from `develop` frequently to avoid large conflicts later.
* Small Commits: Commit changes often, focusing on small, manageable updates. This makes code review easier.
* Use Descriptive Branch Names: Naming branches by task or feature helps us know what’s being worked on at a glance.

## 7. Documentation and Training Resources

We’ll keep this workflow documented in our team’s README and project wiki. Please check back here if you need a refresher, and feel free to share any questions or suggestions to make this workflow work better for everyone.

By following these guidelines, we’ll keep our codebase organized, minimize conflicts, and ensure that everyone is working within a consistent and professional structure. Let’s work together to make this a smooth and productive workflow!