**Commit Message Types and Git Flow Branch Naming**

**Git commit message types:**

* **feat**: Introducing new features or significant improvements.

## 📌 1. feat – New Feature

**Use when:** Adding a new user-facing feature.

**Example:**

feat(profile): add ability to update profile picture

* **fix**: Bug fixes that resolve issues in your code.

## fix – Bug Fix

**Use when:** Fixing a known issue or unexpected behavior.

**Example:**

fix(auth): resolve login crash on empty password

* **docs**: Updates or additions to documentation.

## 3. docs – Documentation Change

**Use when:** Updating or fixing documentation only.

**Example:**

docs(readme): update installation instructions

* **style**: Cosmetic changes that don't affect code functionality (like formatting).

## 4. style – Code Style Fixes (No logic change)

**Use when:** Making formatting changes (indentation, spaces, semicolons).

**Example:**

style(navbar): fix indentation and spacing issues

* **refactor**: Code changes that neither fix a bug nor add a feature but improve structure.

## 5. refactor – Code Refactoring (No new feature or bug fix)

**Use when:** Restructuring code for readability or performance without changing behavior.

**Example:**

refactor(user-service): simplify user role check logic

* **test**: Everything about testing - adding or fixing tests.

## 7. test – Tests Added or Modified

**Use when:** Adding, fixing, or improving tests.

**Example:**

test(payment): add unit tests for refund flow

* **chore**: Routine tasks or updates to the build process.

## chore – Maintenance Work

**Use when:** Updating dependencies, build process, or non-functional code.

**Example:**

chore(deps): upgrade axios to v1.4.0

* **perf**: Enhancements that improve performance.

## 6. perf – Performance Improvements

**Use when:** Improving performance without changing functionality.

**Example:**

perf(image-loader): cache images to reduce load time

* **ci**: Modifications related to CI/CD processes.

**Use when:** Updating CI/CD scripts or pipelines.

**Example:**

ci(github-actions): add lint check to PR workflow

* **build**: Changes affecting the build system or external dependencies.

## 9. build – Build System or Package Changes

**Use when:** Changing build configuration (Webpack, Gradle, etc.).

**Example:**

build(ci): update GitHub Actions node version to 18

* **revert**: Undoing previous changes.

## 11. revert – Reverting a Previous Commit

**Use when:** Reverting a change.

**Example:**

revert: feat(login): add remember me feature

**Streamlining Workflow with Git Flow Branch Naming Conventions**

**Git Flow** is a branching strategy that helps manage and structure various phases of a project. It's a framework that utilizes specific branch types, each with its distinct role and naming convention:

* **master**: The source of truth for production-ready states.
* **develop**: The integration branch for development work.
* **feature/**: For developing new features, e.g., feature/add-login.
* **release/**: Preparing a new production release, e.g., release/1.2.0.
* **hotfix/**: Quick fixes for the production version, e.g., hotfix/critical-login-bug.
* **support/**: Long-term support for older versions, e.g., support/1.x.
* **bugfix/**: For specific bug fixes (optional), e.g., bugfix/login-error.