Commit Message Guidelines

Good commit messages are essential for maintaining a clean and understandable project history. They help team members understand what changes were made, why they were necessary, and how they affect the codebase. Clear commit practices improve collaboration, simplify debugging, and make code reviews more efficient.

General Best Practices: A good commit message should be short, descriptive, and focused on a single change. Each commit should represent one logical unit of work, avoiding the combination of unrelated changes. Messages should be written in the present tense and explain what the commit does rather than what was done. Keeping commits small and meaningful makes it easier to track changes and revert them if necessary.

Commit Message Structure: A commonly followed convention is to start commit messages with a type prefix such as feat, fix, or refactor. This prefix provides quick insight into the nature of the change. The prefix is followed by a concise summary that clearly explains the purpose of the commit.

feat (Feature Commits): Feature commits are used when adding new functionality to the application. They should clearly describe the new behavior introduced. Example: *feat: add user authentication using JWT* or *feat: implement dark mode toggle*.

fix (Bug Fix Commits): Fix commits are used to resolve bugs or unexpected behavior. The message should briefly explain what was broken and how it was corrected. Example: *fix: resolve login crash on invalid credentials* or *fix: correct calculation error in checkout total*.

refactor (Code Improvement Commits): Refactor commits are used when improving code structure without changing functionality. These commits focus on readability, performance, or maintainability. Example: *refactor: simplify API service logic* or *refactor: remove duplicate validation code*.

Conclusion: Following proper commit message guidelines ensures a clean and professional project history. Using clear prefixes such as feat, fix, and refactor helps developers quickly understand changes and improves long-term maintainability of the codebase, especially in team-based projects.