Conventional Commits

Types

`feat`	Features	A new feature
`fix`	Bug Fixes	A bug fix
`docs`	Documentation	Documentation only changes
`style`	Styles	Changes that do not affect the meaning of the code (white-space, formatting, missing semi-colons, etc)
`refactor`	Code Refactoring	A code change that neither fixes a bug nor adds a feature
`perf`	Performance Improvements	A code change that improves performance
`test`	Tests	Adding missing tests or correcting existing tests
`build`	Builds	Changes that affect the build system or external dependencies (example scopes: gulp, broccoli, npm)
`ci`	Continuous Integrations	Changes to our CI configuration files and scripts (example scopes: Travis, Circle, BrowserStack, SauceLabs)
`chore`	Chores	Other changes that don't modify src or test files
`revert`	Reverts	Reverts a previous commit

Commit Message Structure

```
<type>[optional scope]: <description>

[optional body]

[optional footer]
```

Specification

The key words “MUST”, “MUST NOT”, “REQUIRED”, “SHALL”, “SHALL NOT”, “SHOULD”, “SHOULD NOT”, “RECOMMENDED”, “MAY”, and “OPTIONAL” in this document are to be interpreted as described in RFC 2119.

1. Commits MUST be prefixed with a type, which consists of a noun, feat, fix, etc., followed by a colon and a space.
2. The type feat MUST be used when a commit adds a new feature to your application or library.
3. The type fix MUST be used when a commit represents a bug fix for your application.
4. An optional scope MAY be provided after a type. A scope is a phrase describing a section of the codebase enclosed in parenthesis, e.g., fix(parser):
5. A description MUST immediately follow the type/scope prefix. The description is a short description of the code changes, e.g., fix: array parsing issue when multiple spaces were contained in string.
6. A longer commit body MAY be provided after the short description, providing additional contextual information about the code changes. The body MUST begin one blank line after the description.
7. A footer MAY be provided one blank line after the body. The footer SHOULD contain additional issue references about the code changes (such as the issues it fixes, e.g., Fixes #13).
8. Breaking changes MUST be indicated in the footer AND by appending a ! after the type/scope. A BREAKING CHANGE introduces a breaking API change (correlating with MAJOR in Semantic Versioning). A BREAKING CHANGE can be part of commits of any type.
9. A description MUST be provided after the BREAKING CHANGE:, describing what has changed about the API, e.g., BREAKING CHANGE: environment variables now take precedence over config files.
10. The footer MUST only contain BREAKING CHANGE, external links, issue references, and other meta-information.
11. Types other than feat and fix MAY be used in your commit messages.