**Conventional Commits**

A specification for adding human and machine readable meaning to commit messages

Conventional Commits 1.0.0

The commit message should be structured as follows:

<type>[optional scope]: <description>

[optional body]

[optional footer(s)]

**Importance of Conventional Commits**

* Automatically generating CHANGELOGs.
* Automatically determining a semantic version bump (based on the types of commits landed).
* Communicating the nature of changes to teammates, the public, and other stakeholders.
* Triggering build and publish processes.
* Making it easier for people to contribute to your projects, by allowing them to explore a more structured commit history.

Semantic Versioning is all about versioning API.

The syntax is universally known:

<MAJOR>.<MINOR>.<PATCH>

MAJOR Introduce a new backward-incompatible change 1.0.0 → 2.0.0

MINOR Introduce a new backward-compatible change 1.0.0 → 1.1.0

PATCH Fix a bug while maintaining backward-compatibility 1.0.0 → 1.0.1

Rules of a great Git commit message

1. **Specify the type of commit**:

* **fix:** a commit of the *type* fix patches a bug in your codebase (this correlates with PATCH in Semantic Versioning).
* **feat:** a commit of the *type* feat introduces a new feature to the codebase (this correlates with MINOR in Semantic Versioning).
* **BREAKING CHANGE:** a commit that has a footer BREAKING CHANGE:, or appends a ! after the type/scope, introduces a breaking API change (correlating with MAJOR in Semantic Versioning). A BREAKING CHANGE can be part of commits of any *type*.
* **style**: Feature and updates related to styling
* **refactor**: Refactoring a specific section of the codebase
* **test**: Everything related to testing
* **docs**: Everything related to documentation
* **chore**: Regular code maintenance.[ You can also use emojis to represent commit types]

1. [**Separate subject from body with a blank line**](https://cbea.ms/git-commit/#separate)

Firstly, not every commit requires both a subject and a body. Sometimes a single line is fine, especially when the change is so simple that no further context is necessary. For example:

$ git commit -m "Fix typo in introduction to user guide"

However, when a commit merits a bit of explanation and context, you need to write a body.

$ git commit -m “Subject” -m “Description….”

The first -m option is the subject (short description), and the next is the extended description (body).

1. [**Limit the subject line to 50 characters**](https://cbea.ms/git-commit/#limit-50)

50 characters is not a hard limit, just a rule of thumb. Keeping subject lines at this length ensures that they are readable, and forces the author to think for a moment about the most concise way to explain what’s going on.

1. [**Capitalize the subject line**](https://cbea.ms/git-commit/#capitalize)

This is as simple as it sounds. Begin all subject lines with a capital letter.

For example:

* Modified the app file

Instead of:

* ~~modified the app file~~

1. [**Do not end the subject line with a period**](https://cbea.ms/git-commit/#end)

Trailing punctuation is unnecessary in subject lines. Besides, space is precious when you’re trying to keep them to [50 chars or less](https://cbea.ms/posts/git-commit/#limit-50).

Example:

* Open the pod bay doors

Instead of:

* ~~Open the pod bay doors.~~

6. [**Use the imperative mood in the subject line**](https://cbea.ms/git-commit/#imperative)

*Imperative mood* just means “spoken or written as if giving a command or instruction”. A few examples:

"Fix bug" and not "Fixed bug" or "Fixes bug."

7. [**Wrap the body at 72 characters**](https://cbea.ms/git-commit/#wrap-72)

8. [**Use the body to explain what and why vs. how**](https://cbea.ms/git-commit/#why-not-how)