Conventional Commits Cheat Sheet

The Conventional Commits specification is a lightweight convention on top of commit messages. It provides an easy set of rules for creating an explicit commit history; which makes it easier to write automated tools on top of. This convention dovetails with SemVer, by describing the features, fixes, and breaking changes made in commit messages.

The commit message should be structured as follows:

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

[optional body]

[optional footer(s)]
```

The commit contains the following structural elements, to communicate intent to the consumers of your library:

1. **fix:** a commit of the *type* fix patches a bug in your codebase (this correlates with PATCH in Semantic Versioning).
2. **feat:** a commit of the *type* feat introduces a new feature to the codebase (this correlates with MINOR in Semantic Versioning).
3. **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*.
4. *types* other than fix: and feat: are allowed, as defined in @commitlint/config-conventional
5. *footers* other than BREAKING CHANGE: <description> may be provided and follow a convention similar to git trailer format.
6. A scope may be provided to a commit's type, to provide additional contextual information and is contained within parenthesis, e.g., feat(parser): add ability to parse arrays.

Examples

```
docs(changelog): update changelog to beta.5
```

```
fix(release): need to depend on latest rxjs and zone.js

The version in our package.json gets copied to the one we publish,
and users need the latest of these.
```

type(scope): subject
BODY
FOOTER

TYPE

build: Changes that affect the build system or external dependencies (example scopes: maven, gradle, npm, yarn)

chore: Changes that do not fall in any other category, should contain a scope for further context.

ci: Changes to our CI configuration files and scripts (example scopes: GitLab, CircleCI, GitHub Workflows)

docs: Documentation only changes

feat: A new feature

fix: A bug fix

perf: A code change that improves performance

refactor: A code change that neither fixes a bug nor adds a feature

revert: A code change that reverts previous changes or configuration.

style: Changes that do not affect the meaning of the code (white-space, formatting, missing semicolons, etc)

test: Adding tests or correcting existing tests

SCOPE

A scope is provided in parentheses after a type. It is a phrase describing the package or parts of the code affected by the changes. For example "(ui) or (core)".

SUBJECT

The subject contains a short description of the applied changes.

- use the imperative, present tense: "change" not "changed" nor "changes"
- don't capitalize the first letter
- no dot (.) at the end

BODY (optional)

Just as in the **subject**, use the imperative, present tense: "change" not "changed" nor "changes". The body should include the motivation for the change and contrast this with previous behavior.

Used to indicate breaking changes, if so must start with "BREAKING CHANGE"

FOOTER (optional)

Used to reference issues affected by the code changes. For example "resolves #10".

Can also be used to indicate breaking changes by starting with "BREAKING CHANGE"