Cheatography

# QK Conventional Commits Cheat Sheet

by Marco Ponton (mponton) via cheatography.com/173169/cs/36383/

## General Format

```
<ty pe> [op tional scope][!]:
<de scr ipt ion>
<BLANK LINE>
[optional body]
<BLANK LINE>
[optional footer(s)]
```

## type

See `Types` on the right.

## scope

The `scope` is OPTIONAL but if present is must immediately follow the `type` and be in parenthesis (e.g. `(auth)`). The scope further qualifies the change. For example, "`fix: enforce pw min length check`" could become "`fix(auth): enforce pw min length check`" to point out the fix was done in the `auth` section of the codebase.

## exclamation (!)

An exclamation `!` mark can be used before the colon to indicate a breaking change without having to provide a footer. This correlates with MAJOR in Semantic Versioning. Example: `fix!: remove insecure login option`.

## description

Subject/summary of the change. See also `Good Commit Message` on the right.

## body

An optional more complete description of the changes. Usually you should use this further describe the reasons and implications of a change.

## footer(s)

You can add optional footers. In the context of BitBucket and CHANGELOGS, the only useful footer is `BREAKING CHANGE`. It *MUST* be present if you introduce a breaking change and should include a description of what will break. Example: `BREAKING CHANGE: --insecure flag will now produce an error`

## Types (cont)

| | |
|---|---|
| dev | **QK**: Commit worth mentioning during active development (included in changelogs) |
| docs | Commit that affects the documentation |
| feat | Commit that introduces a new feature to the codebase (this correlates with MINOR in Semantic Versioning). |
| fix | Commit that patches a bug in your codebase (this correlates with PATCH in Semantic Versioning). |
| perf | Commit that improves performance |
| refactor | Commit that neither fixes a bug nor adds a feature |
| revert | Commit that reverts a previous commit |
| style | Commit that does not affect the meaning of the code |
| test | Commit that adds or updates tests |
| wip | **QK**: Work In Progress (ignored in changelogs) |

## Tips

Follow structure, spacing and use exact types. `fix:foobar` ⚠ != `fix: foobar` ✔ , `doc` ⚠ != `docs` ✔

Don't use a `scope` unless meaningful as it reduce the length of the `descri ption`

Limit yourself to the valid types documented here. Anything else will be ignored in changelogs.

## Good Commit Message

## git Good Commits

Commits with types other than `dev` and `wip` should make up a single unit of work (change). For example, a `fix` commit should fix a single issue, not many.

Commit often. Do not wait to have 20 modified files with different changes left and right. It will be harder to extract proper commits from all the changes and you will end up committing unrelated changes together.

Make sure you test your code before committing it, but *particularly* before *pushing* it. It is easy to amend a commit or rebase locally, harder to keep a clean history after having pushed bad code to BitBucket.

Keep your history clean. It is OK and desirable to *push* code often to prevent data loss, but work on your own branch and rebase before merging to a published branch. Nobody likes to see 13 `wip: something` messages in the commit log.

| Types | |
|---|---|
| build | Commit that affects the build system or external dependencies |
| chore | Other commit that do not modify source or test files |
| ci | Commit that affect the CI configuration files and scripts |

Use imperative verb form (e.g. `add foo bar` not `added foo bar`).

Write commit summary in lowercase letters.

Review your message for typos before committing.

Commit summary (which is the complete `type(s cope): descri ptiline`) *should* be limited to 50 characters, or 72 characters at most

The `body` can contain as many lines as needed but each should be at most 72 characters.

By **Marco Ponton** (mponton)
cheatography.com/mponton/

Not published yet.
Last updated 11th January, 2023.