Conventional Commits

“A specification for adding human and machine readable meaning to commit messages.” – ConventionalCommits.org

Why?

- clear communication by following a standard
- most commonly used standard for commit messages
- be a better developer

Requirements

- communicating the nature of changes to teammates, the public, and other stakeholders
- determine next semantic version semver bump
- automate things
  - CHANGELOG (think `git log --online HEAD..last-release`)
  - trigger a release, deployment
- reference ticketing-system requirements and information

We are not perfect

We’re all learning to become better developers every day. So don’t expect perfect commit messages right after this talk but try to get better every day.

Reading commit-messages, release-notes and CHANGELOGs from open-source projects helps to get better.

Message Structure

Basic structural elements of a commit message where components of the first line (enclosed by `<>`) are required and the other ones are optional (`[]`):

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

[optional body]

[optional footer(s)]
```

Description

- Mandatory
- Active Voice (prefer "add", "remove" over "adds", "added", "removed")
- Short message (up to 72 chars)
- present tense

Type

Indicates the most fitting type of change that is committed. There's a fixed list that also could be extended if required.

Type

Most common ones:

- feat: implementations or additions to new features
- fix: defect fixing
- refactor: changes to existing code, but does not alter or change existing behavior in the product.
- test: add missing tests, remove tests, fix assertions

... to be continued

Type (Continued)

- style: updates or reformats the style of the source code, but does not otherwise change the product implementation.
- docs: adds, updates, or revises documentation
- chore: normal house-keeping commits, usually used as catch-all
- build: alterations of the build system, updates to external dependencies, tooling
- ci: updates to build workflow files use in CI environment
- revert: reverted commits

Type: Semver Bump

The types of messages that would trigger a semver bump

- `feat:` a feature-release: `0.x.0`
- `fix:` Trigger a patch-release: `0.0.x`
- `BREAKING CHANGE` / `!`: major-release: `x.0.0`

Excercise

```
fc489bb fix(scan-processing): check forbidden code before method check
40b2ec4 chore: wip
0c390d8 ci: use infrastructure makefile for prod deployment
d6ff3cc fix(log)!: redact cookie & authorization, request named req
eca6217 style: moves lambda source dir to handler
688750e style: fix eslint
be092cb build: removes depcheck as it's not working with workspaces
134e769 docs: replace migration plan steps with todo list
b460697 feat: add ScanEnhancer in replacement for nest js service
```

What would be the type of the next release?

A) PATCH **B)** FEAT **C)** BREAKING

Scopes

Scopes are usually defined in a separate process and should be re-used.

As there are synonyms possible it makes sense to provide a list of acceptable scopes.

Examples

```
fix(authentification): case-insensitive check email
```

```
docs(api): add documentation for user-login
```

```
ci(prod): add lint step to workflow
```

Breaking Changes

Communicate changes the would break other components or integrations

Indicate breaking changes that require manual adjustment of integration by adding a `!` after the "type"

```
feat!: adds new login method
```

or add `BREAKING CHANGE:` in the body or footer, followed by space or two newlines and a description

```
BREAKING CHANGE: This is a breaking change description which can examplain what happened on how to migrate
```

Footer

Additional Metadata

The footer contains additional, meta information following the git-trailer format which is is a basic definition list:

```
Refs: SQ-1283
Refs: SQ-1297
Signed-off-by: Bob <bob@example.com>
Co-authored-by: NAME <NAME@EXAMPLE.COM>
Co-authored-by: AUTHOR-NAME <ANOTHER-NAME@EXAMPLE.COM>"
```

Log messages can display these meta informations:

```
git log --format="%h %s %(trailers:key=Refs,valueonly)"
```

More To Read & Learn

- Conventional Commits Guidelines
- Why should I write good commit messages?
- Bad Commit Messages Hall of Fame
- Conventional Commits: A Better Way

Thanks for listening!

Please give feedback & ask questions!