# Contributing

## Table of contents

- [Setting Up the Environment](#setting-up-the-environment)

- [Submitting a Pull Request (PR)](#submitting-a-pull-request-pr)

- [After your pull request is merged](#after-your-pull-request-is-merged)

- [Coding Style Guidelines](#coding-style-guidelines)

- [Commit Message Guidelines](#commit-message-guidelines)

- [Commit Message Format](#commit-message-format)

- [Revert](#revert)

- [Type](#type)

- [Scope](#scope)

- [Subject](#subject)

- [Body](#body)

- [Footer](#footer)

## Setting Up the Environment

1. Run `pnpm install` in the root of the repository to install all dependencies.

2. For compiling all projects, run `pnpm run compile` in the root of the repository. To run a task that will recompile the projects on change, run `pnpm run watch`.

3. In order to run all the tests in the repository, run `pnpm run test-master`. You may also run tests of specific projects by running `pnpm test` inside a project's directory or using `pnpm --filter <project name> test`.

## Submitting a Pull Request (PR)

Before you submit your Pull Request (PR) consider the following guidelines:

- Search [GitHub](https://github.com/pnpm/pnpm/pulls) for an open or closed PR

that relates to your submission. You don't want to duplicate effort.

- Make your changes in a new git branch:

```shell

git checkout -b my-fix-branch master

```

- Create your patch, following [code style guidelines](#coding-style-guidelines), and \*\*including appropriate test cases\*\*.

- Run the full test suite and ensure that all tests pass.

- Commit your changes using a descriptive commit message that follows our

[commit message conventions](#commit-message-guidelines). Adherence to these conventions

is necessary because release notes are automatically generated from these messages.

```shell

git commit -a

```

Note: the optional commit `-a` command line option will automatically "add" and "rm" edited files.

- Push your branch to GitHub:

```shell

git push origin my-fix-branch

```

- In GitHub, send a pull request to `pnpm:master`.

- If we suggest changes then:

- Make the required updates.

- Re-run the test suites to ensure tests are still passing.

- Rebase your branch and force push to your GitHub repository (this will update your Pull Request):

```shell

git rebase master -i

git push -f

```

That's it! Thank you for your contribution!

### After your pull request is merged

After your pull request is merged, you can safely delete your branch and pull the changes

from the main (upstream) repository:

- Delete the remote branch on GitHub either through the GitHub web UI or your local shell as follows:

```shell

git push origin --delete my-fix-branch

```

- Check out the master branch:

```shell

git checkout master -f

```

- Delete the local branch:

```shell

git branch -D my-fix-branch

```

- Update your master with the latest upstream version:

```shell

git pull --ff upstream master

```

## Coding Style Guidelines

[![js-standard-style](https://cdn.rawgit.com/feross/standard/master/badge.svg)](https://github.com/feross/standard)

Use the [Standard Style](https://github.com/feross/standard).

## Commit Message Guidelines

[![Commitizen friendly](https://img.shields.io/badge/commitizen-friendly-brightgreen.svg)](http://commitizen.github.io/cz-cli/)

We have very precise rules over how our git commit messages can be formatted. This leads to \*\*more

readable messages\*\* that are easy to follow when looking through the \*\*project history\*\*. Helper script `npm run commit`

provides command line based wizard to format commit message easily.

### Commit Message Format

Each commit message consists of a \*\*header\*\*, a \*\*body\*\* and a \*\*footer\*\*. The header has a special

format that includes a \*\*type\*\*, a \*\*scope\*\* and a \*\*subject\*\*:

<type>(<scope>): <subject>

<BLANK LINE>

<body>

<BLANK LINE>

<footer>

The \*\*header\*\* is mandatory and the \*\*scope\*\* of the header is optional.

Any line of the commit message cannot be longer 100 characters! This allows the message to be easier

to read on GitHub as well as in various git tools.

#### Revert

If the commit reverts a previous commit, it should begin with `revert:`, followed by the header of the reverted commit. In the body it should say: `This reverts commit <hash>.`, where the hash is the SHA of the commit being reverted.

#### Type

Must be one of the following:

- \*\*feat\*\*: A new feature

- \*\*fix\*\*: A bug fix

- \*\*docs\*\*: Documentation only changes

- \*\*style\*\*: Changes that do not affect the meaning of the code (white-space, formatting, missing

semi-colons, etc)

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

- \*\*perf\*\*: A code change that improves performance

- \*\*test\*\*: Adding missing tests

- \*\*chore\*\*: Changes to the build process or auxiliary tools and libraries such as documentation

generation

#### Scope

The scope could be anything specifying place of the commit change. For example

`plugin-example`, `render-md`, etc.

#### Subject

The subject contains succinct description of the change:

- use the imperative, present tense: "change" not "changed" nor "changes"

- don't capitalize first letter

- no dot (.) at the end

#### Body

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.

#### Footer

The footer should contain any information about \*\*Breaking Changes\*\* and is also the place to

reference GitHub issues that this commit \*\*Closes\*\*.

\*\*Breaking Changes\*\* should start with the word `BREAKING CHANGE:` with a space or two newlines. The rest of the commit message is then used for this.