# Contributing to semantic-release

? Thanks for contributing to \*\*semantic-release\*\*! ?

As a contributor, here are the guidelines we would like you to follow:

- [Code of conduct](#code-of-conduct)

- [How can I contribute?](#how-can-i-contribute)

- [Using the issue tracker](#using-the-issue-tracker)

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

- [Coding rules](#coding-rules)

- [Working with the code](#working-with-the-code)

We also recommend that you read [How to Contribute to Open Source](https://opensource.guide/how-to-contribute).

## Code of conduct

Help us keep \*\*semantic-release\*\* open and inclusive. Please read and follow our [Code of conduct](CODE\_OF\_CONDUCT.md).

## How can I contribute?

### Improve documentation

As a \*\*semantic-release\*\* user, you are the perfect candidate to help us improve our documentation: typo corrections, clarifications, more examples, new [recipes](docs/recipes/README.md), etc. Take a look at the [documentation issues that need help](https://github.com/issues?utf8=%E2%9C%93&q=is%3Aopen+is%3Aissue+user%3Asemantic-release+archived%3Afalse+label%3A%22help+wanted%22+label%3Adocs+).

Please follow the [Documentation guidelines](#documentation).

### Give feedback on issues

Some issues are created without information requested in the [Bug report guideline](#bug-report). Help make them easier to resolve by adding any relevant information.

Issues with the [design label](https://github.com/issues?q=is%3Aopen+is%3Aissue+user%3Asemantic-release+archived%3Afalse+label%3Adesign) are meant to discuss the implementation of new features. Participating in the discussion is a good opportunity to get involved and influence the future direction of \*\*semantic-release\*\*.

### Fix bugs and implement features

Confirmed bugs and ready-to-implement features are marked with the [help wanted label](https://github.com/issues?utf8=%E2%9C%93&q=is%3Aopen+is%3Aissue+user%3Asemantic-release+archived%3Afalse+label%3A%22help+wanted%22). Post a comment on an issue to indicate you would like to work on it and to request help from the [@semantic-release/maintainers](https://github.com/orgs/semantic-release/teams/contributors) and the community.

## Using the issue tracker

The issue tracker is the channel for [bug reports](#bug-report), [features requests](#feature-request) and [submitting pull requests](#submitting-a-pull-request) only. Please use the [Support](docs/support/README.md) and [Get help](README.md#get-help) sections for support, troubleshooting and questions.

Before opening an issue or a Pull Request, please use the [GitHub issue search](https://github.com/issues?utf8=%E2%9C%93&q=user%3Asemantic-release) to make sure the bug or feature request hasn't been already reported or fixed.

### Bug report

A good bug report shouldn't leave others needing to chase you for more information. Please try to be as detailed as possible in your report and fill the information requested in the [Bug report template](https://github.com/semantic-release/semantic-release/issues/new?template=bug-report.md).

### Feature request

Feature requests are welcome, but take a moment to find out whether your idea fits with the scope and aims of the project. It's up to you to make a strong case to convince the project's developers of the merits of this feature. Please provide as much detail and context as possible and fill the information requested in the [Feature request template](https://github.com/semantic-release/semantic-release/issues/new?template=feature-request.md).

### New plugin request

[Plugins](docs/usage/plugins.md) are a great way to extend \*\*semantic-release\*\* capabilities, integrate with other systems and support new project type. Please provide as much detail and context as possible and fill the information requested in the [New plugin request template](https://github.com/semantic-release/semantic-release/issues/new?template=plugin-request.md).

## Submitting a Pull Request

Good pull requests, whether patches, improvements, or new features, are a fantastic help. They should remain focused in scope and avoid containing unrelated commits.

\*\*Please ask first\*\* before embarking on any significant pull requests (e.g. implementing features, refactoring code), otherwise you risk spending a lot of time working on something that the project's developers might not want to merge into the project.

If you have never created a pull request before, welcome ? ?. [Here is a great tutorial](https://opensource.guide/how-to-contribute/#opening-a-pull-request) on how to send one :)

Here is a summary of the steps to follow:

1. [Set up the workspace](#set-up-the-workspace)

2. If you cloned a while ago, get the latest changes from upstream and update dependencies:

```bash

$ git checkout master

$ git pull upstream master

$ rm -rf node\_modules

$ npm install

```

3. Create a new topic branch (off the main project development branch) to contain your feature, change, or fix:

```bash

$ git checkout -b <topic-branch-name>

```

4. Make your code changes, following the [Coding rules](#coding-rules)

5. Push your topic branch up to your fork:

```bash

$ git push origin <topic-branch-name>

```

6. [Open a Pull Request](https://help.github.com/articles/creating-a-pull-request/#creating-the-pull-request) with a clear title and description.

\*\*Tips\*\*:

- For ambitious tasks, open a Pull Request as soon as possible with the `[WIP]` prefix in the title, in order to get feedback and help from the community.

- [Allow semantic-release maintainers to make changes to your Pull Request branch](https://help.github.com/articles/allowing-changes-to-a-pull-request-branch-created-from-a-fork). This way, we can rebase it and make some minor changes if necessary. All changes we make will be done in new commit and we'll ask for your approval before merging them.

## Coding rules

### Source code

To ensure consistency and quality throughout the source code, all code modifications must have:

- No [linting](#lint) errors

- A [test](#tests) for every possible case introduced by your code change

- \*\*100%\*\* test coverage

- [Valid commit message(s)](#commit-message-guidelines)

- Documentation for new features

- Updated documentation for modified features

### Documentation

To ensure consistency and quality, all documentation modifications must:

- Refer to brand in [bold](https://help.github.com/articles/basic-writing-and-formatting-syntax/#styling-text) with proper capitalization, i.e. \*\*GitHub\*\*, \*\*semantic-release\*\*, \*\*npm\*\*

- Prefer [tables](https://help.github.com/articles/organizing-information-with-tables) over [lists](https://help.github.com/articles/basic-writing-and-formatting-syntax/#lists) when listing key values, i.e. List of options with their description

- Use [links](https://help.github.com/articles/basic-writing-and-formatting-syntax/#links) when you are referring to:

- a \*\*semantic-release\*\* concept described somewhere else in the documentation, i.e. How to [contribute](CONTRIBUTING.md)

- a third-party product/brand/service, i.e. Integrate with [GitHub](https://github.com)

- an external concept or feature, i.e. Create a [GitHub release](https://help.github.com/articles/creating-releases)

- a package or module, i.e. The [`@semantic-release/github`](https://github.com/semantic-release/github) module

- Use the the [single backtick `code` quoting](https://help.github.com/articles/basic-writing-and-formatting-syntax/#quoting-code) for:

- commands inside sentences, i.e. the `semantic-release` command

- programming language keywords, i.e. `function`, `async`, `String`

- packages or modules, i.e. The [`@semantic-release/github`](https://github.com/semantic-release/github) module

- Use the the [triple backtick `code` formatting](https://help.github.com/articles/creating-and-highlighting-code-blocks) for:

- code examples

- configuration examples

- sequence of command lines

### Commit message guidelines

#### Atomic commits

If possible, make [atomic commits](https://en.wikipedia.org/wiki/Atomic\_commit), which means:

- a commit should contain exactly one self-contained functional change

- a functional change should be contained in exactly one commit

- a commit should not create an inconsistent state (such as test errors, linting errors, partial fix, feature with documentation etc...)

A complex feature can be broken down into multiple commits as long as each one maintains a consistent state and consists of a self-contained change.

#### 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\*\*:

```commit

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

<BLANK LINE>

<body>

<BLANK LINE>

<footer>

```

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

The \*\*footer\*\* can contain a [closing reference to an issue](https://help.github.com/articles/closing-issues-via-commit-messages).

#### 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

The type must be one of the following:

| Type | Description |

|--------------|-------------------------------------------------------------------------------------------------------------|

| \*\*build\*\* | Changes that affect the build system or external dependencies (example scopes: gulp, broccoli, npm) |

| \*\*ci\*\* | Changes to our CI configuration files and scripts (example scopes: Travis, Circle, BrowserStack, SauceLabs) |

| \*\*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 |

| \*\*style\*\* | Changes that do not affect the meaning of the code (white-space, formatting, missing semi-colons, etc) |

| \*\*test\*\* | Adding missing tests or correcting existing tests |

#### 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.

#### Examples

```commit

`fix(pencil): stop graphite breaking when too much pressure applied`

```

```commit

`feat(pencil): add 'graphiteWidth' option`

Fix #42

```

```commit

perf(pencil): remove graphiteWidth option`

BREAKING CHANGE: The graphiteWidth option has been removed.

The default graphite width of 10mm is always used for performance reasons.

```

## Working with the code

### Set up the workspace

[Fork](https://guides.github.com/activities/forking/#fork) the project, [clone](https://guides.github.com/activities/forking/#clone) your fork, configure the remotes and install the dependencies:

```bash

# Clone your fork of the repo into the current directory

$ git clone https://github.com/semantic-release/<repo-name>

# Navigate to the newly cloned directory

$ cd <repo-name>

# Assign the original repo to a remote called "upstream"

$ git remote add upstream https://github.com/semantic-release/<repo-name>

# Install the dependencies

$ npm install

```

### Lint

All the [semantic-release](https://github.com/semantic-release) repositories use [XO](https://github.com/sindresorhus/xo) for linting and [Prettier](https://prettier.io) for formatting. Prettier formatting will be automatically verified and fixed by XO.

Before pushing your code changes make sure there are no linting errors with `npm run lint`.

\*\*Tips\*\*:

- Most linting errors can be automatically fixed with `npm run lint -- --fix`.

- Install the [XO plugin](https://github.com/sindresorhus/xo#editor-plugins) for your editor to see linting errors directly in your editor and automatically fix them on save.

### Tests

Running the integration test requires you to install [Docker](https://docs.docker.com/engine/installation) on your machine.

All the [semantic-release](https://github.com/semantic-release) repositories use [AVA](https://github.com/avajs/ava) for writing and running tests.

Before pushing your code changes make sure all \*\*tests pass\*\* and the \*\*coverage is 100%\*\*:

```bash

$ npm run test

```

\*\*Tips:\*\* During development you can:

- run only a subset of test files with `ava <glob>`, for example `ava test/mytestfile.test.js`

- run in watch mode with `ava -w` to automatically run a test file when you modify it

- run only the test you are working on by adding [`.only` to the test definition](https://github.com/avajs/ava#running-specific-tests)

### Commits

All the [semantic-release](https://github.com/semantic-release) repositories use [Commitizen](https://github.com/commitizen/cz-cli) to help you create [valid commit messages](#commit-message-guidelines).

After staging your changes with `git add`, run `npm run cm` to start the interactive commit message CLI.