# Stylus contributing guidelines

Thank you for wanting to contribute to Stylus!

You can find below our guidelines for contribution, explaining how to send [pull requests](#pull-requests), [report issues](#filling-bugs) and [ask questions](#asking-questions), as well as which [workflow](#workflow) we're using while developing Stylus.

## Maintainers

Current maintainers of Stylus are:

- Roman Komarov ([@kizu](https://github.com/kizu)),

- Mikhail Korepanov ([@panya](https://github.com/panya)).

- Valeriy Chupurnov ([@xdan](https://github.com/xdan)).

If you'll have any questions, feel free to mention us or use emails from our profiles to contact us.

## Code of Conduct

Please note that this project is released with a [Contributor Code of Conduct](Code\_of\_Conduct.md). By participating in this project you agree to abide by its terms.

## How you can help

You're welcome to:

- send pull requests;

- report bugs;

- ask questions;

- fix existing issues;

- suggest new features and enhancements;

- write, rewrite, fix and enhance docs;

- contribute in other ways if you'd like.

### Pull-requests

If you fixed or added something useful to the project, you can send a pull-request. It will be reviewed by a maintainer and accepted, or commented for rework, or declined.

#### Before submitting a PR:

1. Make sure you have tests for your modifications.

2. Run npm test locally to catch any errors.

#### Why did you close my pull request or issue?

Nothing is worse than a project with hundreds of stale issues. To keep things orderly, the maintainers try to close/resolve issues as quickly as possible.

#### PR/Issue closing criteria

We'll close your PR or issue if:

1. It's a duplicate of an existing issue.

2. Outside of the scope of the project.

3. The bug is not reproducible.

4. You are unresponsive after a few days.

5. The feature request introduces too much complexity (or too many edge cases) to the tool

- We weigh a request's complexity with the value it brings to the community.

Please do not take offense if your ticket is closed. We're only trying to keep the number of issues manageable.

### Filing bugs

If you found an error, typo, or any other flaw in the project, please report it using [GitHub Issues](https://github.com/stylus/stylus/issues). Try searching the issues to see if there is an existing report of your bug, and if you'd find it, you could bump it by adding your test case there.

When it comes to bugs, the more details you provide, the easier it is to reproduce the issue and the faster it could be fixed.

The best case would be if you'd provide a minimal reproducible test case illustrating a bug. For most cases just a code snippet would be enough, for more complex cases you can create gists or even test repos on GitHub — we would be glad to look into any problems you'll have with Stylus.

### Asking questions

GitHub issues is not the best place for asking questions like “why my code won't work” or “is there a way to do X in Stylus”, but we are constantly monitoring the [stylus tag at StackOverflow](http://stackoverflow.com/unanswered/tagged/stylus), so feel free to ask there! It would make it easier for other people to get answers and to keep GitHub Issues for bugs and feature requests.

### Fixing existing issues

If you'd like to work on an existing issue, just leave a comment on the issue saying that you'll work on a PR fixing it.

Also, check out the [c: Easy](https://github.com/stylus/stylus/labels/c%3A%20Easy) label for a set of beginner-friendly tickets, as this is the easiest way to start contributing to Stylus.

### Proposing features

If you've got an idea for a new feature, file an issue providing some details on your idea. Try searching the issues to see if there is an existing proposal for your feature and feel free to bump it by providing your use case or explaining why this feature is important for you.

We should note that not everything should be done as a “Stylus feature”, some features better be a Stylus plug-ins, some could be much faster implemented using a post-processor, some are just not in the scope of the project.

\* \* \*

## Workflow

This section describes the workflow we use for Stylus releases, the naming of the branches and the meaning behind them.

### Branches

#### Permanent branches

The following branches should always be there. Do not fork them directly, always create a new branch for your Pull Requests.

- `master`. The code in this branch should always be equal to the latest version that was published in npm.

- `dev`. This is a branch for coldfixes — both code and documentation. When you're fixing something, it would make sense to send a PR to this branch and not to the `master` — this would make our job a bit easier.

The code in this branch should always be backwards compatible with `master` — it should only introduce fixes, changes to documentation and other similar things like those, so at every given moment we could create a patch release from it.

- `gh-pages`. This is a branch for running Stylus' site. It shouldn't contain any actual docs, instead we would add `dev` branch to it as a submodule. This would mean the docs are always up to date with the current published release and storing the docs with the code would mean it would be easy to write the docs alongside the code.

- `client`. This branch is a clent-side fork of Stylus that is used for interactive examples on our site. The client version shouldn't be used to serve CSS in production. The branch is updated manually with new release at the moment, in future we could think on automatizing this.

#### Temporarily branches

- `issue-NNN`. If you're working on a fix for an issue, you can use this naming. This would make it easy to understand which issue is affected by your code. You can optionally include a postfix with a short description of the problem, for example `issue-1289-broken-mqs`.

- `feature-…`. Any new feature should be initially be a feature-branch. Such branches won't be merged into `master` or `dev` branches directly. The naming would work basically the same as the `issue-…`, but you can omit the issue's number as there couldn't be one issue covering the feature, or you're working on some refactoring.

- `rc-…`. Any new feature release should be at first compiled into a release candidate branch. For example, `rc-0.43` would be a branch for a coming `0.43.0` release. We would merge feature branches and Pull Requests that add new features to the rc-branch, then we test all the changes together, writing tests and docs for those new features and when everything is ready, we increase the version number in `package.json`, then merge the rc-branch into `dev` and `master`.

### Releasing workflow

We follow [semver](http://semver.org/). We're in `0.x` at the moment, however, as Stylus is already widely used, we don't introduce backwards-incompatible changes to our minor releases.

Each minor release should be first compiled into `rc-`branch. Minor release \*should not\* have fixes in it, as patch-release should be published before a minor one if there are fixes. This would deliver the fixes to the people using the fixed minor, but `x` at patch version.

Patch releases don't need their own `rc` branches, as they could be released from the `dev` branch.

### Adding tests

First you want to make sure to run the below commands

```

npm install

# for a more verbose output you can install mocha at a global level

npm install mocha -g

```

Then at the root of the project you can run `npm test` or `mocha` to execute all tests. If you need to add or edit tests, they are located in the `test/cases` directory.

Each `.styl` file has a corresponding `.css` file. The `.styl` is the mock, and the `.css` is the expected result.

\* \* \*

This document is inspired my many other Contributing.md files, a notable example is [JSCS' Contributing Guide](https://github.com/jscs-dev/node-jscs/blob/master/CONTRIBUTING.md).