# Contributing

We encourage any form of contribution, whether that be issues, comments, or pull requests. If you are going to be submitting a PR, there are a few things we would appreciate that you do to keep the codebase clean:

\* \*\*Write tests.\*\* We try as close to 100% code coverage as possible on this repo so any new code that gets written should have accompanying tests.

\* \*\*Follow the linter.\*\* We use our [ESLint configuration with Airbnb JavaScript Styleguide](https://github.com/airbnb/javascript), and we run `npm run lint` in our Travis builds.

\* \*\*Ask questions if you aren't sure.\*\* If you have any questions while implementing a fix or feature, feel free to create an issue and ask us. We're happy to help!

## <a name="submit"></a> Submission Guidelines

### <a name="submit-issue"></a> Submitting an Issue

Before you submit an issue, please search the issue tracker, maybe an issue for your problem already exists and the discussion might inform you of workarounds readily available.

### <a name="submit-pr"></a> Submitting a Pull Request (PR)

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

1. Search [GitHub](https://github.com/amejiarosario/dsa.js/pulls) for an open or closed PR

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

1. Be sure that an issue describes the problem you're fixing, or documents the design for the feature you'd like to add.

Discussing the design up front helps to ensure that we're ready to accept your work.

1. Fork the `amejiarosario/dsa.js` repo.

1. Make your changes in a new git branch:

```shell

git checkout -b my-fix-branch master

```

1. Create your patch, \*\*including appropriate test cases\*\*.

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

1. Commit your changes using a descriptive commit message that follows our

[commit message conventions](#commit). 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.

1. Push your branch to GitHub:

```shell

git push origin my-fix-branch

```

1. In GitHub, send a pull request to `dsa.js: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

```

## <a name="commit"></a> Commit Message Guidelines

We have some guidelines 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\*\*. But also,

we use the git commit messages to \*\*generate the change log\*\*.

### 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>

```

Example of a commit with header, body and footer:

```

fix(linked-list): insert in the middle bug

When inserting an item on the middle of a linked list one reference was not being updated properly.

Fixes: #8

```

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.

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

```

docs(changelog): update changelog to beta.5

```

```

fix(release): need to depend on latest rxjs and zone.js

The version in our package.json gets copied to the one we publish, and users need the latest of these.

```

### 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:

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

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

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

### Scope

The scope should be the name of the npm package affected (as perceived by the person reading the changelog generated from commit messages.

The following is an example of supported scopes:

\* \*\*list\*\*

\* \*\*map\*\*

\* \*\*tree\*\*

\* \*\*graph\*\*

\* \*\*sorting\*\*

\* etc.

### Subject

The subject contains a succinct description of the change:

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

\* don't capitalize the 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\*\*.

```

Closes #234

```

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

Examples of breaking changes include:

\* removal or redefinition of existing API arguments

\* changing return values

\* removing or modifying existing properties on an options argument

\* adding or removing errors

\* altering expected timing of an event

\* changing the side effects of using a particular API

## Generating Changelog

We use these three sections in changelog: new features, bug fixes, breaking changes.

List of all subjects (first lines in commit message) since last release:

```sh

git log <last tag> HEAD --pretty=format:%s

# example

git log 1.1.0..HEAD --pretty=format:%s

```

New features in this release

```sh

git log <last release> HEAD --grep feat

# examples

git log 1.2.0..HEAD --pretty=format:"- %s [commit](https://github.com/amejiarosario/dsa.js/commit/%H)" --grep "BREAKING CHANGE:"

git log 1.2.0..HEAD --pretty=format:"- %s [commit](https://github.com/amejiarosario/dsa.js/commit/%H)" --grep "^feat\S\*:"

git log 1.2.0..HEAD --pretty=format:"- %s [commit](https://github.com/amejiarosario/dsa.js/commit/%H)" --grep "^fix\S\*:"

```

<!-- Examples -->

<!-- https://github.com/nodejs/node/blob/v12.0.0/COLLABORATOR\_GUIDE.md -->

<!-- https://github.com/nodejs/node/blob/v12.0.0/doc/guides/writing-and-running-benchmarks.md -->