# Contributing to IxJS

[Read and abide by the Code of Conduct](CODE\_OF\_CONDUCT.md)! Even if you don't read it,

it still applies to you. Ignorance is not an exemption.

Contents

<!-- START doctoc generated TOC please keep comment here to allow auto update -->

<!-- DON'T EDIT THIS SECTION, INSTEAD RE-RUN doctoc TO UPDATE -->

- [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)

- [Unit Tests](#unit-tests)

- [CI Tests](#ci-tests)

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

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

- [Revert](#revert)

- [Type](#type)

- [Scope](#scope)

- [Subject](#subject)

- [Body](#body)

- [Footer](#footer)

<!-- END doctoc generated TOC please keep comment here to allow auto update -->

(This document is a work and progress and is subject to change)

## Submitting a Pull Request (PR)

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

\* Search [GitHub](https://github.com/ReactiveX/IxJS/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.

\* Run the micro and macro performance tests against your feature branch and compare against master

to ensure performance wasn't changed for the worse.

\* Commit your changes using a descriptive commit message that follows our

[commit message guidelines](#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 `IxJS:master`.

\* If we suggest changes then:

\* Make the required updates.

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

\* Re-run performance tests to make sure your changes didn't hurt performance.

\* 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

- Please use proper types and generics throughout your code.

- 2 space indentation only

- favor readability over terseness

(TBD): For now try to follow the style that exists elsewhere in the source, and use your best judgment.

## Unit Tests

Unit tests are located under the [spec directory](/spec). Unit tests are written using [tape](https://github.com/substack/tape)

Each operator under test must be in its own file to cover the following cases:

- Never

- Empty

- Single/Multiple Values

- Error in the sequence

If the operator accepts a function as an argument from the user/developer (for example `filter(fn)` or `zip(a, fn)`),

then it must cover the following cases:

- Success with all values in the callback

- Success with the context, if any allowed in the operator signature

- If an error is thrown

### CI Tests

- Using [Travis](https://travis-ci.org/) on your forked version of IxJS will allow running CI tests on that fork before submitting a PR to master

- Simply create a `Travis` account and add your fork as a new project

- As master runs both of these tests per each check in, it'd be welcome to setup those test before creating your PR

## Commit Message Guidelines

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\*\*. But also,

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

`Observable`, `Subject`, `switchMap`, 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.