# Contributing

Among billions other things you can do with your time, you choose to spend it on our little extension, how extraordinary!? Thank you!

vscode-jest started as a simple extension and grew quite a bit in the last 12 months with many amazing contributions from the community! As the community and code base continue to expand, we decided to restructure the contribution page to make it easier and more fun to contribute:

1. [Code of Conduct](#code\_of\_conduct): the standard open source contribution guideline.

1. [Our Philosophy](#our\_philosophy): outline what guided our decision for feature expansion and issue/PR discussions.

1. [How to Contribute](#how\_to\_contribute): tips on how to setup your local vscode-jest development environment; as well as test and debug it before submission.

## Code of Conduct

We adopt the standard open source [contributor covenant](https://www.contributor-covenant.org/version/1/4/code-of-conduct.html).

\_(If you have contributed to any open source project before, you already knew it; otherwise, we recommend a quick read above.)\_

## Our Philosophy

As the functionality grows, inevitably, so is the complexity. In order for this little extension continues to be stable and useful, as well as pleasant and fun to work with for the current and future contributors, we sometimes had to make hard trade-off choices between functionality and complexity. The following principles help to guide these decisions...

### Principle \#1: be a good extension

Sure, but what does that really mean?

- \*\*utilize vscode platform as much as possible\*\*, i.e. do not reinvent the wheel; do not force users to learn our way of doing the same thing. A good extension should be \*intuitive\*, i.e. what they learned from the platform should just apply in our extension.

- \*\*fix root causes\*\*: it is tempting to fix issues in the extension when the root cause is actually in vscode, jest or other dependent packages. It is almost always better to address them in their corresponding packages, 1) it benefit more people, an open source spirit 2) it keeps vscode-jest clean and focus on our core competency.

- \*\*being adaptive to user environment\*\*: js development environment is dynamic and complex, while we work hard to make the tool easy to use, there is no illusion that we can cover every use case, configuration, or framework used. When facing introducing extra logic and external assumptions for retrieving user-environment specific info, we favor configuration based solution. Not only users can easily adapt vscode-jest to their specific environments, with the precision and performance we can never match; vscode-jest can stay clean and simple.

### Principle \#2: be a good developer tool

- \*\*do not block\*\*: as a developer, I think we all have encountered the frustrating experience that the tool supposed to boost our productivity became the very thing to block our progress. Think about the failure scenario, can user unblock themselves if my code failed?

- \*\*help users to help themselves\*\*: show status, meaningful error messages, instructions on how they can fix or customize are sometimes more important than codes. Do not under estimate the importance of documentation, feel free to submit document changes along with your PR.

- \*\*be courteous of all users' time\*\*: every feature, however useful for those who use it, could be a burden for others that don't need it. Consider costly operation on-demand, deferred until needed, options to turn on/off etc.

### Principle \#3: be easy to contribute

There are a lot of things we can do in this area, and this document is part of it. We welcome your ideas on how we can continue to improve.

But maybe more importantly, every contributor can play a critical role here. Nobody enjoys working with spaghetti code that breaks easily and difficult to read. While it is fun and exciting when adding new logic, we should all be mindful that it will impact other contributors long after our commits:

- all future contributors need to read our code to understand how the system works. So keep the code simple and short, and yes comments are welcome ;-)

- once the code is in, it will need to be maintained. Therefore, keep the state as tight as possible, don't create a class variable that only used in a single method; avoid assumptions about user environments, package locations, etc; less is less, and it is often better than more ;-)

- safe guard your logic with tests so the future contributors will not need to worry about their change might accidentally break yours. Don't underestimate the power of search-and-replace or fat-finger effect ;-)

## How to contribute

### Repository Setup

The extension is in two parts, one is \_this\_ repo. It contains all the VS Code specific work.

```js

git clone https://github.com/jest-community/vscode-jest

cd vscode-jest

yarn install

code .

```

The other part is inside the [jest-editor-support](https://github.com/jest-community/jest-editor-support)".

It's very possible that you're going to want to make changes inside here, if you're doing something that touches the test runner process or file parsers. To get that set up to work I'd recommend doing this:

```

# set up jest

cd ..

git clone https://github.com/jest-community/jest-editor-support.git

cd jest-editor-support

yarn install

# link jest-editor-support

yarn link

# set up vscode-jest to use the real jest-editor-support

cd ../

cd vscode-jest

yarn link jest-editor-support

# go back and start the jest build watcher

cd ../

cd jest-editor-support

yarn build --watch

```

With that installed, you want to use a custom `jest-editor-support` by going into `cd jest-editor-support` and running `yarn link`.

Go back to vscode-jest, and do one more `yarn link "jest-editor-support"` and now you're using those files directly from master of `jest-editor-support`.

As `jest-editor-support` requires running through Babel, you can run the Babel watcher using the command `yarn build --watch` inside the `jest-editor-support` root directory.

Yeah, it's a bit of a process, but we'll be sharing code with the nuclide team and that's worth it IMO.

### Testing and Debugging

Be kind to the reviewers and future contributors, please make sure you pass the following tests before submitting the PR:

\*\*1. unit tests\*\*

Make sure `yarn lint`, `yarn test` and `yarn vscode:prepublish` all work.

\*\*2. integration tests\*\*

There are two debugging launch configurations defined in `.vscode/launch.json`:

\* Debug Tests

\* Launch Extension

To debug the extension, [change the launch configuration](https://code.visualstudio.com/docs/editor/debugging#\_launch-configurations) to \*\*Launch Extension\*\* and start debugging.

\*\*3. eat our own dog food\*\*

The ultimate test is to actually use it in our real day-to-day working environment for a while. There are multiple ways to do this:

- by command line: `code --extensionDevelopmentPath=your-local-vscode-jest`

- by environment variable: `CODE\_EXTENSIONS\_PATH`

- by symlink:

Here is a mac example:

```

$ cd ~/.vscode/extensions

$ mv Orta.vscode-jest-2.7.0 vscode.orig

$ ln -s Orta.vscode-jest-2.7.0 your-local-vscode-jest

```

restore vscode.orig when you are done testing.