# Contributing

## How do I... <a name="toc"></a>

- [Use This Guide](#introduction)?

- Make Something? ???????

- [Project Setup](#project-setup)

- [Contribute Documentation](#contribute-documentation)

- [Contribute Code](#contribute-code)

- Manage Something ?????

- [Provide Support on Issues](#provide-support-on-issues)

- [Review Pull Requests](#review-pull-requests)

- [Join the Project Team](#join-the-project-team)

## Introduction

Thank you so much for your interest in contributing!. All types of contributions are encouraged and valued. See the [table of contents](#toc) for different ways to help and details about how this project handles them!?

The [Project Team](#join-the-project-team) looks forward to your contributions. ???

## Project Setup

So you wanna contribute some code! That's great! This project uses GitHub Pull Requests to manage contributions, so [read up on how to fork a GitHub project and file a PR](https://guides.github.com/activities/forking) if you've never done it before.

If this seems like a lot or you aren't able to do all this setup, you might also be able to [edit the files directly](https://help.github.com/articles/editing-files-in-another-user-s-repository/) without having to do any of this setup. Yes, [even code](#contribute-code).

If you want to go the usual route and run the project locally, though:

- [Install Node.js](https://nodejs.org/en/download/)

- [Install Yarn](https://yarnpkg.com)

- [Fork the project](https://guides.github.com/activities/forking/#fork)

Then in your terminal:

- Add coc.nvim to your vim's rtp by `set runtimepath^=/path/to/coc.nvim`

- `cd path/to/your/coc.nvim`

- `yarn install`

- Install [coc-tsserver](https://github.com/neoclide/coc-tsserver) by

`:CocInstall coc-tsserver` in your vim

- Install [coc-tslint-plugin](https://github.com/neoclide/coc-tslint-plugin) by

`:CocInstall coc-tslint-plugin` in your vim.

And you should be ready to go!

## Contribute Documentation

Documentation is a super important, critical part of this project. Docs are how we keep track of what we're doing, how, and why. It's how we stay on the same page about our policies. And it's how we tell others everything they need in order to be able to use this project -- or contribute to it. So thank you in advance.

Documentation contributions of any size are welcome! Feel free to file a PR even if you're just rewording a sentence to be more clear, or fixing a spelling mistake!

To contribute documentation:

- [Set up the project](#project-setup).

- Edit or add any relevant documentation.

- Make sure your changes are formatted correctly and consistently with the rest of the documentation.

- Re-read what you wrote, and run a spellchecker on it to make sure you didn't miss anything.

- In your commit message(s), begin the first line with `docs:`. For example: `docs: Adding a doc contrib section to CONTRIBUTING.md`.

- Write clear, concise commit message(s) using [conventional-changelog format](https://github.com/conventional-changelog/conventional-changelog-angular/blob/master/convention.md). Documentation commits should use `docs(<component>): <message>`.

- Go to https://github.com/neoclide/coc.nvim/pulls and open a new pull request with your changes.

- If your PR is connected to an open issue, add a line in your PR's description that says `Fixes: #123`, where `#123` is the number of the issue you're fixing.

## Contribute Code

We like code commits a lot! They're super handy, and they keep the project going and doing the work it needs to do to be useful to others.

Code contributions of just about any size are acceptable!

The main difference between code contributions and documentation contributions is that contributing code requires inclusion of relevant tests for the code being added or changed. Contributions without accompanying tests will be held off until a test is added, unless the maintainers consider the specific tests to be either impossible, or way too much of a burden for such a contribution.

To contribute code:

- [Set up the project](#project-setup).

- Make any necessary changes to the source code.

- Include any [additional documentation](#contribute-documentation) the changes might need.

- Make sure the code doesn't have lint issue by command `yarn lint` in your

terminal.

- Write tests that verify that your contribution works as expected when necessary.

- Make sure all tests passed by command `yarn jest` in your terminal.

- Write clear, concise commit message(s) using [conventional-changelog format](https://github.com/conventional-changelog/conventional-changelog-angular/blob/master/convention.md).

- Dependency updates, additions, or removals must be in individual commits, and the message must use the format: `<prefix>(deps): PKG@VERSION`, where `<prefix>` is any of the usual `conventional-changelog` prefixes, at your discretion.

- Go to https://github.com/neoclide/coc.nvim/pulls and open a new pull request with your changes.

- If your PR is connected to an open issue, add a line in your PR's description that says `Fixes: #123`, where `#123` is the number of the issue you're fixing.

Once you've filed the PR:

- Barring special circumstances, maintainers will not review PRs until all checks pass (Travis, AppVeyor, etc).

- One or more maintainers will use GitHub's review feature to review your PR.

- If the maintainer asks for any changes, edit your changes, push, and ask for another review. Additional tags (such as `needs-tests`) will be added depending on the review.

- If the maintainer decides to pass on your PR, they will thank you for the contribution and explain why they won't be accepting the changes. That's ok! We still really appreciate you taking the time to do it, and we don't take that lightly. ?

- If your PR gets accepted, it will be marked as such, and merged into the `latest` branch soon after. Your contribution will be distributed to the masses next time the maintainers [tag a release](#tag-a-release)

## Provide Support on Issues

[Needs Collaborator](#join-the-project-team): none

Helping out other users with their questions is a really awesome way of contributing to any community. It's not uncommon for most of the issues on an open source projects being support-related questions by users trying to understand something they ran into, or find their way around a known bug.

Sometimes, the `support` label will be added to things that turn out to actually be other things, like bugs or feature requests. In that case, suss out the details with the person who filed the original issue, add a comment explaining what the bug is, and change the label from `support` to `bug` or `feature`. If you can't do this yourself, @mention a maintainer so they can do it.

In order to help other folks out with their questions:

- Go to the issue tracker and [filter open issues by the `support` label](https://github.com/neoclide/coc.nvim/issues?q=is%3Aopen+is%3Aissue+label%3Asupport).

- Read through the list until you find something that you're familiar enough with to give an answer to.

- Respond to the issue with whatever details are needed to clarify the question, or get more details about what's going on.

- Once the discussion wraps up and things are clarified, either close the issue, or ask the original issue filer (or a maintainer) to close it for you.

Some notes on picking up support issues:

- Avoid responding to issues you don't know you can answer accurately.

- As much as possible, try to refer to past issues with accepted answers. Link to them from your replies with the `#123` format.

- Be kind and patient with users -- often, folks who have run into confusing things might be upset or impatient. This is ok. Try to understand where they're coming from, and if you're too uncomfortable with the tone, feel free to stay away or withdraw from the issue. (note: if the user is outright hostile or is violating the CoC, [refer to the Code of Conduct](CODE\_OF\_CONDUCT.md) to resolve the conflict).

## Review Pull Requests

[Needs Collaborator](#join-the-project-team): Issue Tracker

While anyone can comment on a PR, add feedback, etc, PRs are only \_approved\_ by team members with Issue Tracker or higher permissions.

PR reviews use [GitHub's own review feature](https://help.github.com/articles/about-pull-request-reviews/), which manages comments, approval, and review iteration.

Some notes:

- You may ask for minor changes ("nitpicks"), but consider whether they are really blockers to merging: try to err on the side of "approve, with comments".

- \_ALL PULL REQUESTS\_ should be covered by a test: either by a previously-failing test, an existing test that covers the entire functionality of the submitted code, or new tests to verify any new/changed behavior. All tests must also pass and follow established conventions. Test coverage should not drop, unless the specific case is considered reasonable by maintainers.

- Please make sure you're familiar with the code or documentation being updated, unless it's a minor change (spellchecking, minor formatting, etc). You may @mention another project member who you think is better suited for the review, but still provide a non-approving review of your own.

- Be extra kind: people who submit code/doc contributions are putting themselves in a pretty vulnerable position, and have put time and care into what they've done (even if that's not obvious to you!) -- always respond with respect, be understanding, but don't feel like you need to sacrifice your standards for their sake, either. Just don't be a jerk about it?

## Join the Project Team

### Ways to Join

There are many ways to contribute! Most of them don't require any official status unless otherwise noted. That said, there's a couple of positions that grant special repository abilities, and this section describes how they're granted and what they do.

All of the below positions are granted based on the project team's needs, as well as their consensus opinion about whether they would like to work with the person and think that they would fit well into that position. The process is relatively informal, and it's likely that people who express interest in participating can just be granted the permissions they'd like.

You can spot a collaborator on the repo by looking for the `[Collaborator]` or `[Owner]` tags next to their names.

| Permission | Description |

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

| Issue Tracker | Granted to contributors who express a strong interest in spending time on the project's issue tracker. These tasks are mainly [labeling issues](#label-issues), [cleaning up old ones](#clean-up-issues-and-prs), and [reviewing pull requests](#review-pull-requests), as well as all the usual things non-team-member contributors can do. Issue handlers should not merge pull requests, tag releases, or directly commit code themselves: that should still be done through the usual pull request process. Becoming an Issue Handler means the project team trusts you to understand enough of the team's process and context to implement it on the issue tracker. |

| Committer | Granted to contributors who want to handle the actual pull request merges, tagging new versions, etc. Committers should have a good level of familiarity with the codebase, and enough context to understand the implications of various changes, as well as a good sense of the will and expectations of the project team. |

| Admin/Owner | Granted to people ultimately responsible for the project, its community, etc. |