# Contribution Guide

If you want to contribute to a Cherow repo, please use a GitHub pull request. This is the fastest way for us to evaluate your code and to merge it into the code base. Please don't file an issue with snippets of code. Doing so means that we need to manually merge the changes in and update any appropriate tests. That decreases the likelihood that your code is going to get included timely. Please use pull requests.

Please do \*\*not\*\* create a pull request without reading this guide first. Failure to do so may result in the \*\*rejection\*\* of the pull request.

And also keep in mind that performance is the main priority. The "fastest" way to solve an issue, isn't the fastest one, but the slowest one!

## Set up a development environment

If you'd like to work on a pull request and you've never submitted code before, follow these steps:

### Step 1: Install Node.js

Go to <https://nodejs.org/> to download and install the latest stable version for your operating system.

Most of the installers come with [npm](https://www.npmjs.com/) already installed, but if for some reason it doesn't work on your system, you can install it manually using the instructions on the site.

### Step 2: Install yarn & lerna

```bash

npm i yarn lerna -g

```

### Step 3: Fork and checkout your own Cherow repository

Go to <https://github.com/cherow/cherow> and click the "Fork" button. Follow the [GitHub documentation](https://help.github.com/articles/fork-a-repo) for forking and cloning.

Once you've cloned the repository, run `yarn install` to get all the necessary dependencies:

```bash

$ cd cherow

$ yarn install

$ lerna bootstrap

```

You must be connected to the Internet for this step to work. You'll see a lot of utilities being downloaded.

### Step 4: Add the upstream source

The \*upstream source\* is the main Cherow repository that active development happens on. While you won't have push access to upstream, you will have pull access, allowing you to pull in the latest code whenever you want.

To add the upstream source for Cherow, run the following in your repository:

```

git remote add upstream git@github.com:cherow/cherow.git

```

Now, the remote `upstream` points to the upstream source.

### Step 5: Run the tests

Running the tests is the best way to ensure you have correctly set up your development environment. Make sure you're in the `cherow` directory and run:

```bash

$ lerna run test

```

The testing takes a few minutes to complete. If some tests fail, that likely means one or more parts of the environment setup didn't complete correctly. The upstream tests always pass.

After that, you're ready to start working on code.

## Working with Code

The process of submitting a pull request is fairly straightforward and generally follows the same pattern each time:

1. [Create a new branch](#step1)

2. [Make your changes](#step2)

3. [Rebase onto upstream](#step3)

4. [Run the tests](#step4)

5. [Double check your submission](#step5)

6. [Push your changes](#step6)

7. [Submit the pull request](#step7)

Details about each step are found below.

### Step 1: Create a new branch<a name="step1"></a>

The first step to sending a pull request is to create a new branch in your Cherow fork. Give the branch a descriptive name that briefly explains what you are trying to fix, for example:

```

$ git checkout -b issue1234

```

You should do all of your development for the issue in this branch.

\*\*Note:\*\* Do not combine fixes for multiple issues into one branch. Use a separate branch for each issue you're working on.

### Step 2: Make your changes<a name="step2"></a>

<!-- TODO: code conventions-->

Make the changes to the code and tests, following the [code conventions](./code-conventions.md) as you go. Once you have finished, commit the changes to your branch:

```

$ git add -A

$ git commit

```

Our commit message follows [Conventional Commits](https://conventionalcommits.org/), format is as follows:

```

<type>[optional scope]: <description>

[optional body]

[optional footer]

```

The commit message format is important because these messages are used to create a changelog for each release. The tag and issue number help to create more consistent and useful changelogs.

### Step 3: Rebase onto upstream<a name="step3"></a>

Before you send the pull request, be sure to rebase onto the upstream source. This ensures your code is running on the latest available code.

```

git fetch upstream

git rebase upstream/master

```

### Step 4: Run the tests<a name="step4"></a>

After rebasing, be sure to run all of the tests once again to make sure nothing broke:

```

$ lerna run test

```

If any of the tests fail, update your code until all tests pass.

### Step 5: Double check your submission<a name="step5"></a>

With your code ready to go, this is a good time to double-check your submission to make sure it follows our conventions. Here are the things to check:

\* Make sure your commit is formatted correctly.

\* The pull request must have a description. The description should explain what you did and how its effects can be seen.

\* The commit message is properly formatted.

\* The change introduces no functional regression. Be sure to run `lerna run test` to verify your changes before submitting a pull request.

\* Make separate pull requests for unrelated changes. Large pull requests with multiple unrelated changes may be closed without merging.

\* All changes must be accompanied by tests, even if the feature you're working on previously had no tests.

\* All user-facing changes must be accompanied by appropriate documentation.

<!-- \* Follow the [Code Conventions](../code-conventions.html). -->

### Step 6: Push your changes<a name="step6"></a>

Next, push your changes to your clone:

```

git push origin issue1234

```

If you are unable to push because some references are old, do a forced push instead:

```

git push -f origin issue1234

```

### Step 7: Send the pull request<a name="step7"></a>

Now you're ready to send the pull request. Go to your Cherow fork and then follow the [GitHub documentation](https://help.github.com/articles/creating-a-pull-request) on how to send a pull request.

## Following Up

Once your pull request is sent, it's time for the team to review it. As such, please make sure to:

1. Monitor the status of the Travis CI build for your pull request. If it fails, please investigate why. We cannot merge pull requests that fail Travis for any reason.

1. Respond to comments left on the pull request from team members. Remember, we want to help you land your code, so please be receptive to our feedback.

1. We may ask you to make changes, rebase, or squash your commits.

### Updating the Commit Message

If your commit message is in the incorrect format, you'll be asked to update it. You can do so via:

```

$ git commit --amend

```

This will open up your editor so you can make changes. After that, you'll need to do a forced push to your branch:

```

$ git push origin issue1234 -f

```

### Updating the Code

If we ask you to make code changes, there's no need to close the pull request and create a new one. Just go back to the branch on your fork and make your changes. Then, when you're ready, you can add your changes into the branch:

```

$ git add -A

$ git commit

$ git push origin issue1234

```

When updating the code, it's usually better to add additional commits to your branch rather than amending the original commit, because reviewers can easily tell which changes were made in response to a particular review. When we merge pull requests, we will squash all the commits from your branch into a single commit on the `master` branch.

### Rebasing

If your code is out-of-date, we might ask you to rebase. That means we want you to apply your changes on top of the latest upstream code. Make sure you have set up a development environment and then you can rebase using these commands:

```

$ git fetch upstream

$ git rebase upstream/master

```

You might find that there are merge conflicts when you attempt to rebase. Please [resolve the conflicts](https://help.github.com/articles/resolving-merge-conflicts-after-a-git-rebase/) and then do a forced push to your branch:

```

$ git push origin issue1234 -f

```