# Contributing in @webpack-contrib

We'd always love contributions to further improve the webpack / webpack-contrib ecosystem!

Here are the guidelines we'd like you to follow:

- [Questions and Problems](#question)

- [Issues and Bugs](#issue)

- [Feature Requests](#feature)

- [Pull Request Submission Guidelines](#submit-pr)

- [Commit Message Conventions](#commit)

## <a name="question"></a> Got a Question or Problem?

Please submit support requests and questions to StackOverflow using the tag [[webpack]](http://stackoverflow.com/tags/webpack).

StackOverflow is better suited for this kind of support though you may also inquire in [Webpack Gitter](https://gitter.im/webpack/webpack).

The issue tracker is for bug reports and feature discussions.

## <a name="issue"></a> Found an Issue or Bug?

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.

We want to fix all the issues as soon as possible, but before fixing a bug we need to reproduce and confirm it. In order to reproduce bugs, we ask that you to provide a minimal reproduction scenario (github repo or failing test case). Having a live, reproducible scenario gives us a wealth of important information without going back & forth to you with additional questions like:

- version of Webpack used

- version of the loader / plugin you are creating a bug report for

- the use-case that fails

A minimal reproduce scenario allows us to quickly confirm a bug (or point out config problems) as well as confirm that we are fixing the right problem.

We will be insisting on a minimal reproduce scenario in order to save maintainers time and ultimately be able to fix more bugs. We understand that sometimes it might be hard to extract essentials bits of code from a larger code-base but we really need to isolate the problem before we can fix it.

Unfortunately, we are not able to investigate / fix bugs without a minimal reproduction, so if we don't hear back from you we are going to close an issue that doesn't have enough info to be reproduced.

## <a name="feature"></a> Feature Requests?

You can \_request\_ a new feature by creating an issue on Github.

If you would like to \_implement\_ a new feature, please submit an issue with a proposal for your work `first`, to be sure that particular makes sense for the project.

## <a name="submit-pr"></a> Pull Request Submission Guidelines

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

- Search Github for an open or closed PR that relates to your submission. You don't want to duplicate effort.

- 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.

- Fill out our `Pull Request Template`. Your pull request will not be considered if it is ignored.

- Please sign the `Contributor License Agreement (CLA)` when a pull request is opened. We cannot accept your pull request without this. Make sure you sign with the primary email address associated with your local / github account.

## <a name="commit"></a> Webpack Contrib Commit Conventions

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 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.

Examples:

```

docs(readme): update install instructions

```

```

fix: refer to the `entrypoint` instead of the first `module`

```

### 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:

- \*\*build\*\*: Changes that affect the build system or external dependencies (example scopes: babel, npm)

- \*\*chore\*\*: Changes that fall outside of build / docs that do not effect source code (example scopes: package, defaults)

- \*\*ci\*\*: Changes to our CI configuration files and scripts (example scopes: circleci, travis)

- \*\*docs\*\*: Documentation only changes (example scopes: readme, changelog)

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

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

- \*\*perf\*\*: A code change that improves performance

- \*\*refactor\*\*: A code change that neither fixes a bug nor adds a feature

- \*\*revert\*\*: Used when reverting a committed change

- \*\*style\*\*: Changes that do not affect the meaning of the code (white-space, formatting, missing semi-colons)

- \*\*test\*\*: Addition of or updates to Jest tests

### Scope

The scope is subjective & depends on the `type` see above. A good example would be a change to a particular class / module.

### 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\*\*.

\*\*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.

Example

```

BREAKING CHANGE: Updates to `Chunk.mapModules`.

This release is not backwards compatible with `Webpack 2.x` due to breaking changes in webpack/webpack#4764

Migration: see webpack/webpack#5225

```

## Testing Your Pull Request

You may have the need to test your changes in a real-world project or dependent

module. Thankfully, Github provides a means to do this. Add a dependency to the

`package.json` for such a project as follows:

```json

{

"devDependencies": {

"mini-css-extract-plugin": "webpack-contrib/mini-css-extract-plugin#{id}/head"

}

}

```

Where `{id}` is the # ID of your Pull Request.

## Contributor License Agreement

When submitting your contribution, a CLA (Contributor License Agreement) bot will come by to verify that you signed the [CLA](https://cla.js.foundation/webpack-contrib/mini-css-extract-plugin).

If it is your first time, it will link you to the right place to sign it.

However, if you have committed your contributions using an email that is not the same as your email used on GitHub, the CLA bot can't accept your contribution.

Run `git config user.email` to see your Git email, and verify it with [your GitHub email](https://github.com/settings/emails).

## Thanks

For your interest, time, understanding, and for following this simple guide.