# Contribution guidelines

## Table of Contents

- [Getting started](#getting-started)

- [Language](#language)

- [For native English speakers](#for-native-english-speakers)

- [Code of Conduct](#code-of-conduct)

- [How can I help?](#how-can-i-help)

- [Documentation](#documentation)

- [Issues](#issues)

- [Submitting an issue](#submitting-an-issue)

- [Feedback](#feedback)

- [Code](#code)

- [Dev environment](#dev-environment)

- [Commiting](#commiting)

- [Skipping building process](#skipping-building-process)

- [Why all these rules?](#why-all-these-rules)

- [Submitting a pull request](#submitting-a-pull-request)

## Getting started

First off, we would like to thank you for taking the time to contribute and make this a better project!

Here we have a set of instructions and guidelines to reduce misunderstandings and make the process of contributing to `unform` as smooth as possible.

We hope this guide makes the contribution process clear and answers any questions you may have.

### Language

Please, while contributing or interacting in any way in this project, refrain from using any language other than \*\*English\*\*.

#### For native English speakers

Try to use simple words and sentences. Don't make fun of non-native English speakers if you find something wrong about the way they express themselves.

Try to encourage newcomers to express their opinions, and make them comfortable enough to do so.

### Code of Conduct

We expect that project participants to adhere to our Code of Conduct. You can check the [full text](CODE\_OF\_CONDUCT.md) so that you may understand the kind of conduct we are expecting and what actions will and will not be tolerated.

By participating in this project, you agree to abide by its terms.

## How can I help?

Here are some ways you can help along with some guidelines.

### Documentation

As a user of `unform`, you're the perfect candidate to help us improve our documentation!

Typos, errors, lack of examples and/or explanation and so on, are just some examples of things that could be fixed and/or improved.

You could even make improvements to this guide! :)

While documenting, try to keep things simple and clear.

### Issues

Some issues are created with missing information, without a template, not reproducible, or plain

invalid.

You can make them easier to understand and resolve.

#### Submitting an issue

- Please search for similar issues before opening a new one;

- Use one of the corresponding issue templates;

- Use a clear and descriptive title;

- Include as much information as possible by filling out the provided issue

template;

- Most of the time, the best way to report an issue is a failing test proving it.

### Feedback

The more feedback the better! We're always looking for more suggestions and opinions on discussions. That's a good opportunity to influence the future direction of this tool.

This includes submitting an enhancement suggestion, including completely new features and minor improvements to existing functionality.

The [`question`](https://github.com/Rocketseat/unform/labels/question)

and

[`rfc`](https://github.com/Rocketseat/unform/labels/rfc)

labels are a good place to find ongoing discussions.

### Code

You can use issue labels to discover issues you could help out with:

- [`bug` issues](https://github.com/Rocketseat/unform/labels/bug)

are known bugs we'd like to fix;

- [`enhancement` issues](https://github.com/Rocketseat/unform/labels/enhancement)

are features we're open to include.

The

[`help wanted`](https://github.com/Rocketseat/unform/labels/help%20wanted)

and

[`good first issue`](https://github.com/Rocketseat/unform/labels/good%20first%20issue)

labels are especially useful.

When you see an issue that is already assigned, please check to see if there isn't someone working on it already (maybe try asking in the issue). This is to prevent unnecessary work for everyone involved.

#### Dev environment

When developing, prefer using \*\*Node\*\* ? 8 and \*\*yarn\*\*. Writing code with the latest stable Node versions allows us to use newer developer tools.

After [cloning the repository](https://help.github.com/articles/cloning-a-repository/), run `yarn` to install dependencies.

A summary of the scripts:

- To run the example use `yarn dev:example`;

- The `yarn dev:start` command will build the lib and watch files in bundle and rebuild on changes;

- Running `yarn dev` will run both `dev:example` and `dev:start`;

- Use `yarn test` to run the test suite (powered by [Jest](https://facebook.github.io/jest/));

- `yarn coveralls` can't be used locally, this is only used to provide test coverage statistics to [Coveralls](https://coveralls.io);

- For code coverage locally, you can run `yarn test --coverage`.

- `yarn build` will build the lib using [Rollup](https://rollupjs.org/guide/en);

This project uses [Prettier](http://prettier.io/) for code formatting. Consider installing an [editor plugin](https://prettier.io/docs/en/editors.html) for the best experience, but code will also be formatted with a precommit script (using [lint-staged](https://github.com/okonet/lint-staged)).

## Commiting

A commit message can consists of a \*\*header\*\*, \*\*body\*\* and \*\*footer\*\*. The header is the only mandatory part and consists of a type and a subject. The body is used to fully describe the change. The footer is the place to reference any issues or pull requests related to the commit. That said, we end with a template like this:

```

<type>: <subject>

[optional body]

[optional footer]

```

To ensure that a commit is valid, easy to read, and changelog-ready, we have a hook that lints the commit message before allowing a commit to pass. This linter verifies the following:

- The header (first line) is the only mandatory part of the commit message;

- The body and footer are both optional but its use is highly encouraged;

- The header should contains:

- A type:

- Must be lowercase;

- Must be one of:

- \*\*chore\*\*: A change that neither fix a bug nor adds a feature;

- \*\*ci\*\*: A CI change;

- \*\*docs\*\*: A documentation change or fix;

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

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

- \*\*test\*\*: A test-related change.

- A subject:

- Must be capitalized;

- Must be limited to 50 characters or less;

- Must omit any trailing punctuation.

- The body:

- Must have a leading blank line;

- Each line must be limited to 72 characters or less.

- The footer:

- Must have a leading blank line;

- Each line must be limited to 72 characters or less;

- If your commit is about documentation or meta files, please add the tag \*\*[skip ci]\*\* to skip the building process.

- If needed, reference to issues and pull requests must be made here in the last line.

You also should follow these general guidelines when committing:

- Use the present tense ("Add feature" not "Added feature");

- Use the imperative mood ("Move cursor to..." not "Moves cursor to...");

- Try to answer the following questions:

- Why is this change necessary?

- How does it address the issue?

- What side effects (if any) does this change may have?

Example of a commit message:

```

type: Commit message style guide for Git

The first line of a commit message serves as a summary. When displayed

on the web, it's often styled as a heading, and in emails, it's

typically used as the subject. As such, you should specify a "type" and

a "subject". The type must be lowercase and one of: chore, ci, docs,

feat, fix, test. For the subject you'll need capitalize it and

omit any trailing punctuation. Aim for about 50 characters, give or

take, otherwise it may be painfully truncated in some contexts. Write

it, along with the rest of your message, in the present tense and

imperative mood: "Fix bug" and not "Fixed bug" or "Fixes bug".

Consistent wording makes it easier to mentally process a list of

commits.

Oftentimes a subject by itself is sufficient. When it's not, add a

blank line (this is important) followed by one or more paragraphs hard

wrapped to 72 characters. Git is strongly opinionated that the author

is responsible for line breaks; if you omit them, command line tooling

will show it as one extremely long unwrapped line. Fortunately, most

text editors are capable of automating this.

Issues and pull request can be referenced on the footer: #3 #12

```

### Skipping building process

By default, Travis CI automatically runs the building process whenever you push changes. If your commit is about documentation or meta files, you can override this behavior by adding a \*\*[skip ci]\*\* tag anywhere in a commits \*\*footer\*\*. This not only skips the marked commit, but also \*\*all other commits\*\* in the push.

### Why all these rules?

We try to enforce these rules for the following reasons:

- Automatically generating changelog;

- Communicating in a better way the nature of changes;

- Triggering build and publish processes;

- Automatically determining a semantic version bump (based on the types of commits);

- Making it easier for people to contribute, by allowing them to explore a more structured commit history.

## Submitting a pull request

Before submitting a pull request, please make sure the following is done:

- [Fork](https://help.github.com/en/articles/fork-a-repo) the repository and create your branch from `master`.

- Example: `feature/my-awesome-feature` or `fix/annoying-bug`;

- Run `yarn` in the repository root;

- If youve fixed a bug or added code that should be tested, \*\*add tests\*\*;

- Ensure the test suite passes;

- Ensure your commit is validated;