---

id: contributing

title: Contributing Guideline

---

You can contribute to textlint in various ways:

- Reporting issues

- Submitting pull request which fixes known issues, improve documentation, or implement new feature

- Creating an enhancement request or suggestions

- Writing your own rules or plugins

- Writing an article about textlint on your blog

Although we are accepting pull request, we recommend creating an issue first as it enables us to discuss your proposal before you put significant effort into it.

## Reporting New Issues or Feature Suggestions

Please [create a new issue](https://github.com/textlint/textlint/issues/new) on GitHub if you found a bug, got a question, or had an idea for improvement. All work on textlint happens on GitHub in English.

As described at the [`ISSUE\_TEMPLATE.md`](https://github.com/textlint/textlint/blob/master/.github/ISSUE\_TEMPLATE.md), please include following information when reporting a bug:

- What version of textlint are you using? (`textlint -v`)

- What file type (Markdown, plain text, etc.) are you using?

- What did you do? Please include the actual source code causing the issue.

- What did you expect to happen?

- What actually happened? Please include the actual, raw output from textlint. (`textlint --debug <options> ...`)

Creating a new repository that can reproduce the issue helps us understand your situation. [This repository](https://github.com/azu/textlint-isssue78) for example.

## Development Workflow

Here you can see how to get the source of textlint, build it, and run tests locally. We use [GitHub Flow](https://guides.github.com/introduction/flow/) as development workflow.

If you want to contribute to textlint, please check issues labeled [`good first issue`](https://github.com/textlint/textlint/issues?q=is%3Aissue+is%3Aopen+label%3A%22good+first+issue%22). These issues are suitable for your first contribution.

### Installing Prerequisites

Please install following development prerequisites. You also need a [GitHub](https://github.com/) account for contribution.

- [Git](https://git-scm.com/)

- [Node.js](https://nodejs.org/en/) -- we tend to use latest stable Node.js although textlint supports >= 6.0.0

- [Yarn](https://classic.yarnpkg.com/en/) -- textlint supports [npm](https://www.npmjs.com/get-npm) >= 2.0.0, but for development purpose, we chose Yarn as package manager

- Text editor

- Terminal emulator

### Cloning Copy of textlint

Forking a repository allows you to work with textlint codebase without special permission to the textlint repository itself.

1. Navigate to [textlint](https://github.com/textlint/textlint/) repository

2. In the top–right corner of the page, click \*\*Fork\*\* button

3. Create a clone of the fork locally in your terminal:

```sh

$ git clone --recursive https://github.com/YOUR\_ACCOUNT/textlint YOUR\_FORKED\_REPOSITORY

```

See [Fork A Repo: GitHub Help](https://help.github.com/articles/fork-a-repo/) for further detail.

### Building textlint

After getting your clone, you can start playing with textlint.

1. Change directory to your clone:

```sh

$ cd YOUR\_FORKED\_REPOSITORY

```

2. Install dependencies and build packages:

```sh

$ yarn install

```

3. Install all of textlint packages' dependencies:

```sh

$ yarn bootstrap

```

4. Building website:

```sh

$ yarn website

```

Under the hood, textlint uses [Lerna](https://lerna.js.org/) to manage multiple packages:

- `packages/\*`

- `packages/@textlint/\*`

- `examples/\*`

- `scripts/release`

- `test/integration-test`

- `website`

If you are new to Lerna, it seems to add another layer of complexity but it's simpler than you think; you can edit codes, run tests, commit changes, etc. as usual in most cases.

Note that `yarn install` also builds a codebase, you can manually build by running `yarn build`. We have separate task `yarn website` since code and documentation have different life cycle.

### Creating a Branch for Your Work

Before adding changes to your clone, please create a new branch (typically called \_feature branch\_). Changes made to feature branch don't affect or corrupt `master` branch so you will feel safe. In Git, creating a branch is easy and fast:

```sh

$ git checkout -b your-new-feature

```

### Making Changes

You have your feature branch with working textlint then it's time to start making changes! Edit codes with text editor of your choice and add commits as you work on. Please don't forget to add or modify test cases and documents according to your changes.

### Note: TypeScript's Project References

This monorepo use [Project References](https://www.typescriptlang.org/docs/handbook/project-references.html) for faster building.

- [What is a Project Reference?](https://www.typescriptlang.org/docs/handbook/project-references.html#what-is-a-project-reference)

You can update Project References from `package.json`'s dependencies.

```

yarn run update:projectReferences

```

Also, You can check Project References if it is correct.

```

yarn run test:projectReferences

```

#### Coding Guideline

##### Languages

While working with your idea, please use:

- [TypeScript](https://www.typescriptlang.org/) for new codes and tests

- [GitHub flavored Markdown](https://github.github.com/gfm/) for documentation

We are migrating entire codes to TypeScript.

##### Linting and Style

This repository uses [ESLint](https://eslint.org/) for JavaScript linter and [Prettier](https://prettier.io/) for code formatter. We use [`lint-staged`](https://www.npmjs.com/package/lint-staged) and [`husky`](https://www.npmjs.com/package/husky) to make coding style consistent before commit, but if you have your own [Git hooks](https://git-scm.com/book/gr/v2/Customizing-Git-Git-Hooks) locally, these setup doesn't work. In such case, please run ESLint and Prettier manually as below after making changes.

- Run ESLint:

```sh

$ yarn eslint

```

- Run ESLint with [`--fix`](https://eslint.org/docs/user-guide/command-line-interface#--fix) feature to fix some wrong style automatically:

```sh

$ yarn eslint:fix

```

- Run Prettier to reformat code:

```sh

$ yarn prettier

```

##### Commit Message Format

We use [Angular Convention](https://github.com/conventional-changelog/conventional-changelog/tree/master/packages/conventional-changelog-angular) for commit message.

In order to make repository history clean, please use the following guideline as possible as you can. It also enables us creating comprehensive changelog semi–automatically.

```

component commit title

commit type / /

\ | |

feat(rule-context): add template url parameter to events

(a blank line)

body -> The `src` (i.e. the url of the template to load) is now provided to the

`$includeContentRequested`, `$includeContentLoaded` and `$includeContentError`

events.

referenced -> Closes #8453

issues Ref. #8454

```

- commit type:

- `docs`: create or update document

- `feat`: add new feature

- `fix`: fix a bug

- `style`: change formatting

- `perf`: performance related change

- `test`: update on tests

- `chore`: house–keeping

- `refactor`: refactoring related change

- component: package or file name

- commit title:

- Limit to 50 characters including commit type and component (as possible as you can)

- Do not capitalize first character

- Do not end with a period

- Use imperative mood, in present tense; commit title should always be able to complete the following sentence:

- If applied, this commit will \_commit title here\_

- body:

- Separate from subject with a blank line

- Wrap texts at 72 characters

- Explain \_what\_ and \_why\_, not \_how\_

- [GitHub flavored Markdown](https://github.github.com/gfm/) is ok to use

- Start with `BREAKING CHANGE: ` when you made significant change in the commit (see versioning section below).

Example commit message:

```

test(textlint-formatter): check types while testing

- Add strict type check option to `ts-node` to make sure future

interface changes will be took into account while running test.

- Update test case for interface changes made at #430.

Closes #448.

```

Please see [Commit Message Format](https://github.com/conventional-changelog/conventional-changelog/tree/master/packages/conventional-changelog-angular) and [How to Write a Git Commit Message](https://chris.beams.io/posts/git-commit/) for detail.

##### Versioning

We care version number while releasing packages to npm registry so you should not modify `version` field of `package.json`. For the record, we use [Semantic Versioning](https://semver.org/).

- Patch release (intended to not break your lint build)

- A bug fix to the CLI or core (including formatters)

- Improvements to documentation

- Non-user-facing changes such as refactoring

- Re-releasing after a failed release (i.e., publishing a release that doesn't work for anyone)

- Minor release (might break your lint build)

- A new option

- An existing rule is deprecated

- A new CLI capability is created

- New public API are added (new classes, new methods, new arguments to existing methods, etc.)

- It might break type interface(`.d.ts`)

- A new formatter is created

- Major release (break your lint build)

- A new option to an existing rule that results in ESLint reporting more errors by default

- An existing formatter is removed

- Part of the public API is removed or changed in an incompatible way

### Running Tests

We have four type of tests. You should run at least \*\*unit test\*\* or \*\*documentation test\*\* according to your type of changes before submitting a pull request.

All tests should be run at the top directory of your fork.

#### Unit Test

Run tests under [`packages/`](https://github.com/textlint/textlint/tree/master/packages):

```sh

$ yarn test

```

While developing, it would be good to run package level unit test since it will run faster:

```sh

$ cd packages/PACKAGE

$ yarn test

```

#### Example Test

Run [`examples`](https://github.com/textlint/textlint/tree/master/examples) as test:

```sh

$ yarn test:examples

```

#### Integration Test

Run tests with [real–world documents](https://github.com/textlint/textlint/tree/master/test/integration-test):

```sh

$ git submodule update --init

$ yarn test:integration

```

#### Documentation Test

Run textlint to [`docs/`](https://github.com/textlint/textlint/tree/master/docs), [`.github/`](https://github.com/textlint/textlint/tree/master/.github), and [README](https://github.com/textlint/textlint/blob/master/README.md) (we are dog–fooding!):

```sh

$ yarn test:docs

```

#### All Tests

Also you can run all of the above by:

```sh

$ yarn test:all

```

### Pushing the Commit and Opening a Pull Request

After finishing your changes and unit tests or documentation test ran fine, you can push your feature branch to GitHub. Please see [GitHub Help](https://help.github.com/articles/pushing-to-a-remote/) for detail but typically, run `git push` at your terminal.

```sh

$ git push origin feature-branch

```

Then follow another [GitHub Help](https://help.github.com/articles/creating-a-pull-request-from-a-fork/) to create a pull request.

### Working with Reviews (if any)

Once a pull request has been created, it will initiate continuous integration builds and we can work on your changes. You can push additional commits to your fork according to feedback.

### Merging

After all participants on pull request are satisfied to the changes, we will merge your code into the textlint master branch. Yay!

## Release Flow

A Maintainer release new version of textlint by following way.

1. Checkout release branch

```

# checkout release branch like "release-2019-10-10"

git checkout -b "release-$(date '+%Y-%m-%d')"

# push current branch

git push origin HEAD -u

```

2. Version up and Update CHANGELOG

```

# bump vesrion and update changelog

npm run versionup

## npm run versionup:{patch,minor,major}

# bump version and update changelog and update GitHub release notes

## GH\_TOKEN="${GITHUB\_TOKEN}" npm run versionup -- --create-release=github

# push the changes to release branch

git push

```

Tips: Copy changelogs from CHANGELOG.md

```

## Collect changelog in the versionup

yarn run collect-changelog # pbcopy

```

3. Create a Pull Request and Review the release

- If you need, write blog posts in website/blog

4. Publish to npm

```

npm run release

```