# Development Setup

This project uses a monorepo setup that requires using [Yarn](https://yarnpkg.com) because it relies on [Yarn workspaces](https://yarnpkg.com/blog/2017/08/02/introducing-workspaces/).

```sh

# install dependencies

yarn

```

## Dependency architecture

In this monorepo, `vue-cli-plugin-styleguidist` depends on `vue-styleguidist` which in turn depends on `vue-docgen-api`. So if you change something in vue-docgen-api, a new version of each module is going to be delivered. Whereas if you only change something only in the plugin, only the plugin will be delivered.

## Testing Tips

If you run `yarn test`, yarn will run eslint linting, then test vue-styleguidist and vue-docgen-api then test the plugin. Finally it will format all your files using prettier.

The full test suite is rather slow, because it has a number of e2e tests that perform full webpack builds of actual projects. To narrow down the tests needed to run during development, you can pass the test script a regex to match test filenames:

```sh

yarn test:jest <filenameRegex>

```

Note the regex matches against full paths relative to the project root, so for example if you want to test all the docgen tests `packages/vue-docgen-api`, you can simply run:

```sh

yarn test:jest vue-docgen-api

```

You can also pass `--watch` to run tests in watch mode.

Note that `jest --onlyChanged` isn't always accurate because some tests spawn child processes.

To test the cli plugin, everything has to be run in sequence. vue-styleguidist always uses the same http port (8080) and one cannot have 2 instances running at the same time. To run test for the plugin just run this:

```sh

yarn test:plugin

```

## The Scripts

Simplify running and maintaining the examples, compile all packages, test and showcase.

### yarn compile

This command will compile `vue-docgen-api` typescript, then transpile `vue-styleguidist/src` with babel It is never run automatically so if you make any changes to the react components in `src`, don't forget to compile again.

`yarn compile:watch` will launch the babel compilation of the `rsg-components` folder in watch mode.

My favoured developing setup: 2 terminal windows, one with `yarn compile:watch` the other with `yarn start`.

### yarn start

Allows you to run the examples against your local version of styleguidist. An extra parameter allows you to choose which one. By default, it will run `basic`

```sh

yarn start customised

```

This command will run the `customised` example. It will look for `examples/customised/styleguide.config.js` and run styleguidist on it. You can omit the last param and do just `yarn start` to run the basic example.

### yarn build

Same command as `yarn start` but will start the basic example of styleguidist to see what the final bundle is going to look like.

## Commit Message Guidelines

We have very precise rules over how our git commit messages can be formatted. This leads to \*\*more readable messages\*\* that are easy to follow when looking through the \*\*project history\*\*. But also, we use the git commit messages to \*\*generate the Vue-Styleguidist change log\*\*. Helper script `npx git-cz` provides a command-line based wizard to format commit messages easily. If you want to be complete and specify a scope, you will need to read through.

If you are using VSCode, check out the [commitizen plugin](https://marketplace.visualstudio.com/items?itemName=KnisterPeter.vscode-commitizen)

## Commit Message Format

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\*\*:

```text

<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 than 100 characters! This allows the message to be easier to read on GitHub as well as in various git tools.

### 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:

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

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

- \*\*docs\*\*: Documentation only changes

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

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

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

- \*\*test\*\*: Adding missing tests

- \*\*chore\*\*: Changes to the build process or auxiliary tools and libraries such as documentation generation

### Scope

The scope could be anything specifying place of the commit change.

The valid scopes are:

- `docgen` when you touch the vue-docgen-api

- `compiler` for the in-browser compiler

- `plugin` will update the vue-cli-plugin

- `cli` if you have changed the markdown generator

The scopes are not controlled so you can misspell with only consequences that you will have 2 entries in the changelog instead of one.

If you don't specify a scope, it is assumed that you only touched vue-styleguidist.

### Subject

The subject contains succinct description of the change:

- use the imperative, present tense: "change" not "changed" nor "changes"

- don't capitalize 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.