# Contributing to fast-check

? First off, thanks for taking the time to contribute! ?

The following is a set of guidelines for contributing to fast-check and its packages.

These are mostly guidelines, not rules.

Use your best judgment, and feel free to propose changes to this document in a pull request.

\*\*Feel free to contribute, ask questions, report bugs and issue pull requests\*\*

## How Can I Contribute?

### Asking questions

Before asking questions, please double-check you can not find your answer in one of the examples provided or in the documentation of the project:

- [Documentation](https://github.com/dubzzz/fast-check/blob/master/README.md)

- [Examples provided inside the project](https://github.com/dubzzz/fast-check/tree/master/example)

- [Examples of properties](https://github.com/dubzzz/fast-check-examples)

- [Example: fuzzing a REST API](https://github.com/dubzzz/fuzz-rest-api)

If nothing answered your question, please do not hesitate to [create a new issue in GitHub](https://github.com/dubzzz/fast-check/issues).

### Reporting bugs

You should report bugs using [create a new issue in GitHub](https://github.com/dubzzz/fast-check/issues).

### Issuing pull requests

#### Getting started

In order to start playing with the code locally you must run the following set of commands:

```bash

git clone https://github.com/dubzzz/fast-check.git && cd fast-check

yarn

yarn prebuild #generate missing implementations: tuple and properties

yarn build #compile the code in ./src, build the ./lib content

```

Once done, everything is ready for you to start working on the code.

#### Code style

Code style standard is enforced by Prettier.

Once done with your development you can check it follow the recommended code style by running `yarn format:check` or run autofixes with `yarn format:fix`.

You should also check for linting by running `yarn lint:check`.

#### Travis CI integration

All pull requests will trigger Travis CI builds.

It ensures that the pull request follow the code style of the project and do not break existing tests.

#### Update your PR

If you plan to update your PR with either a fix for the tests or change following code reviews please directly commit your new commit in your branch, PR will get updated automatically.

Before your fix:

```

--\*---> master on dubzzz/fast-check

\

#1 branch-pr on your fork

```

After your fix:

```

--\*---> master on dubzzz/fast-check

\

#1 --- #2 branch-pr on your fork

```

#### Resync PR with master

Ideally to resync your branch with master prefer a merge of master branch into your PR branch. It has the advantage to preserves the commit history on GitHub PR (contrary to rebase and force push).

### Examples

#### Adding a new arbitrary

?? \*Create a feature request\*

Before adding any new arbitrary into fast-check please make sure to fill a `Feature request` to justify the need for such arbitrary.

?? \*Code the arbitrary\*

All the arbitraries defined by fast-check are available in `src/check/arbitrary`.

Create a new file for the new one if it does not fit into the existing ones.

?? \*Test the arbitrary\*

Most of the newly added arbitraries will just be a combination of existing ones (mostly mapping from one entry to another).

We expect a quite minimal amount of tests to be added as most of the logic depends on the built-in blocks.

- \*Unit-test\* - in `test/unit/check/arbitrary`

```js

import { myArb } from '../../../../src/check/arbitrary/MyArbitrary';

import \* as genericHelper from './generic/GenericArbitraryHelper';

describe('MyArbitrary', () => {

describe('myArb', () => {

// genericHelper.isValidArbitrary is repsonsible to ensure that the arbitrary is valid

// and fulfill the minimum requirements asked by fast-check

genericHelper.isValidArbitrary((settings) => myArb(settings), {

isValidValue: (g: MyArbGeneratedType, settings) => isValidMyArbOutput(g),

seedGenerator: anArbitraryProducingSettingsExpectedByMyArb // optional field

});

});

});

```

- No regression test - in `test/e2e/NoRegression.spec.ts`

Then run `yarn e2e -- -u` locally to update the snapshot file. The `NoRegression` spec is supposed to prevent unwanted breaking changes to be included in a future release of fast-check by taking a snapshot of the current output and enforcing it does not change over time (except if needed).

- Legacy support test - in `test/legacy/main.js`

The `legacy` spec is responsible to check that most of the arbitraries provided by fast-check are working fine on very old releases of node.

?? \*Document the arbitrary\*

- Provide a minimal JSDoc on top of your new arbitrary and use the `/\*\* @internal \*/` tag to hide internals - otherwise they would get published into the generated documentation

- Add the arbitrary into the list of Built-in Arbitraries - see https://github.com/dubzzz/fast-check/blob/master/documentation/1-Guides/Arbitraries.md