# Contributing

Contributions are always welcome. If you don't know how you can help, you can check [issues](https://github.com/zloirock/core-js/issues) or ask @zloirock.

## How to add a new polyfill

- The polyfill implementation should be added to the [`packages/core-js/modules`](./packages/core-js/modules) directory.

- Any shared helpers should be added to the [`packages/core-js/internals`](./packages/core-js/internals) directory.

- If the implementation for the `pure` version differs from the global version, add it to [`packages/core-js-pure/override`](./packages/core-js-pure/override) directory. The rest parts of `core-js-pure` will be copied from `core-js` package.

- For export the polyfill, in almost all cases use `internals/export` helper.

- Add feature detection of the polyfill to [`tests/compat/tests.js`](./tests/compat/tests.js) and compatibility data to [`packages/core-js-compat/src/data.js`](./packages/core-js-compat/src/data.js) and [`packages/core-js-compat/src/modules-by-versions.js`](./packages/core-js-compat/src/modules-by-versions.js) (this data also used for getting the default list of polyfills at bundling).

- Add it to entry points where it's required: directories [`packages/core-js/features`](./packages/core-js/features), [`packages/core-js/es`](./packages/core-js/es), [`packages/core-js/proposals`](./packages/core-js/proposals), [`packages/core-js/stage`](./packages/core-js/stage) and [`packages/core-js/web`](./packages/core-js/web).

- Add unit tests to [`tests/tests`](./tests/tests) and [`tests/pure`](./tests/pure).

- Add tests of entry points to [`tests/commonjs.js`](./tests/commonjs).

- Add documentation to [README.md](./README.md).

## Style and standards

The coding style should follow our [`.eslintrc`](./.eslintrc.js). You can test it by calling [`npm run lint`](#testing). Different places have different syntax and standard library limitations:

- Polyfill implementations should use only ES3 syntax and standard library. Polyfills should not use another polyfill from the global namespace.

- In unit tests should be used modern syntax with our [minimalistic Babel config](./babel.config.js). Unit tests for the `pure` version should not use any modern standard library features.

- In building tools and tests, performed in Node.js, should be used only available in Node.js 4 syntax and standard library.

File names should be in the kebab-case. Name of files with polyfills should follow naming convention `namespace.subnamespase-where-required.feature-name`, for example, `esnext.promise.try`. Top-level namespace could be `es` for stable ECMAScript features, `esnext` for ECMAScript proposals and `web` for other web standards.

## Testing

Before testing, you should install dependencies:

```

$ npm i

```

You can run all tests by

```

$ npm run test

```

You can run parts of the test case separately:

- Linting:

```

$ npm run lint

```

- Global version unit tests:

```

$ npm run unit-tests

```

- `pure` version unit tests:

```

$ npm run unit-tests-pure

```

- [Promises/A+ test case](https://github.com/promises-aplus/promises-tests):

```

$ npm run promises-tests

```

- [ECMAScript `Observable` test case](https://github.com/tc39/proposal-observable):

```

$ npm run observables-tests

```

- CommonJS entry points tests:

```

$ npm run commonjs-tests

```

If you want to run tests in a certain browser at first you should build packages and test bundles:

```

$ npm run build

```

- For running the global version of the unit test case use this file:

```

tests/tests.html

```

- For running the pure version of the unit test case use this file:

```

tests/pure.html

```

- Before running [Promises/A+ test case](https://github.com/promises-aplus/promises-tests) in the browser you should bundle it:

```

$ npm run bundle-promises-tests

```

and after that use this file:

```

tests/promises-aplus.html

```

## Updating `core-js-compat` data

For updating `core-js-compat` data:

- Clone `core-js` repo.

- If you wanna add new data for a browser, run in this browser `tests/compat/index.html` and you will see which `core-js` modules required for this browser.

- If you wanna add new data for Node.js, run `tests/compat/node-runner.js` in required Node.js version and you will see results in the console.

- After getting this data, add it to [`packages/core-js-compat/src/data.js`](./packages/core-js-compat/src/data.js).

- If you wanna add new mapping (for example, add a new iOS Safari version based on Safari or Node.js based on Chrome), add it to [`packages/core-js-compat/src/mapping.js`](./packages/core-js-compat/src/mapping.js).

- Add a pull request to `core-js` repo.