# Contributing to Prettier

To get up and running, install the dependencies and run the tests:

```bash

yarn

yarn lint:eslint

yarn test

```

Here's what you need to know about the tests:

- The tests use [Jest snapshots](https://facebook.github.io/jest/docs/en/snapshot-testing.html).

- You can make changes and run `jest -u` (or `yarn test -u`) to update the snapshots. Then run `git diff` to take a look at what changed. Always update the snapshots when opening a PR.

- You can run `AST\_COMPARE=1 DEEP\_COMPARE=1 jest` for a more robust test run.

- `AST\_COMPARE` That formats each file, re-parses it, and compares the new AST with the original one and makes sure they are semantically equivalent.

- `DEEP\_COMPARE` That formats each file, then formats the output again, and checks that the second output is the same as the first.

- Each test folder has a `jsfmt.spec.js` that runs the tests. For JavaScript files, generally you can just put `run\_spec(\_\_dirname, ["babel", "flow", "typescript"]);` there. This will verify that the output using each parser is the same. You can also pass options as the third argument, like this: `run\_spec(\_\_dirname, ["babel"], { trailingComma: "es5" });`

- `tests/flow-repo/` contains the Flow test suite, and is not supposed to be edited by hand. To update it, clone the Flow repo next to the Prettier repo and run: `node scripts/sync-flow-tests.js ../flow/tests/`.

- If you would like to debug prettier locally, you can either debug it in Node or the browser. The easiest way to debug it in Node, is to create a local test file with some example code you want formatted and either run it in an editor like VS Code or run it directly via `./bin/prettier.js <your\_test\_file>`. The easiest way to debug it in the browser is to build Prettier's website locally (see `website/README.md`).

Run `yarn lint:eslint --fix` to automatically format files.

If you can, take look at [commands.md](commands.md) and check out [Wadler's paper](http://homepages.inf.ed.ac.uk/wadler/papers/prettier/prettier.pdf) to understand how Prettier works.

If you want to know more about Prettier's GitHub labels, see the [Issue Labels](https://github.com/prettier/prettier/wiki/Issue-Labels) page on the Wiki.

## Performance

If you're contributing a performance improvement, the following Prettier CLI options can help:

- `--debug-repeat N` uses a naïve loop to repeat the formatting `N` times and measures the average run duration. It can be useful to highlight hot functions in the profiler. The measurements are printed at the debug log level, use `--loglevel debug` to see them.

- `--debug-benchmark` uses [`benchmark`](https://npm.im/benchmark) module to produce statistically significant duration measurements. The measurements are printed at the debug log level, use `--loglevel debug` to see them.

For convenience, the following commands for profiling are available via `package.json` `scripts`.

\_Unfortunately, [`yarn` simply appends passed arguments to commands, cannot reference them by name](https://github.com/yarnpkg/yarn/issues/5207), so we have to use inline environment variables to pass them.\_

- `PERF\_FILE=<filename> PERF\_REPEAT=[number-of-repetitions:1000] yarn perf:repeat` starts the naïve loop. See the CLI output for when the measurements finish, and stop profiling at that moment.

- `PERF\_FILE=<filename> PERF\_REPEAT=[number-of-repetitions:1000] yarn perf:repeat-inspect` starts the naïve loop with `node --inspect-brk` flag that pauses execution and waits for Chromium/Chrome/Node Inspector to attach. Open [`chrome://inspect`](chrome://inspect), select the process to inspect, and activate the CPU Profiler, this will unpause execution. See the CLI output for when the measurements finish, and stop the CPU Profiler at that moment to avoid collecting more data than needed.

- `PERF\_FILE=<filename> yarn perf:benchmark` starts the `benchmark`-powered measurements. See the CLI output for when the measurements finish.

In the above commands:

- `yarn && yarn build` ensures the compiler-optimized version of Prettier is built prior to launching it. Prettier's own environment checks are defaulted to production and removed during the build. The build output is cached, so a rebuild will happen only if the source code changes.

- `NODE\_ENV=production` ensures Prettier and its dependencies run in production mode.

- `node --inspect-brk` pauses the script execution until Inspector is connected to the Node process.

- `--loglevel debug` ensures the `--debug-repeat` or `--debug-benchmark` measurements are printed to `stderr`.

- `> /dev/null` ensures the formatted output is discarded.

In addition to the options above, you can use [`node --prof` and `node --prof-process`](https://nodejs.org/en/docs/guides/simple-profiling/), as well as `node --trace-opt --trace-deopt`, to get more advanced performance insights.