# Contribution guidelines

Thanks for taking a look at our contribution guidelines. This is a small

open-source project and we’re always looking to get more people actively

involved in its development and direction, even if you can’t send code!

<!-- vim-markdown-toc GFM -->

- [How to contribute](#how-to-contribute)

- [Bugs & feature requests](#bugs--feature-requests)

- [Code or documentation](#code-or-documentation)

- [Contributing updates](#contributing-updates)

- [Getting started](#getting-started)

- [Dev scripts](#dev-scripts)

- [Code style](#code-style)

- [Writing self-tests](#writing-self-tests)

- [Commit format](#commit-format)

- [Opening a PR](#opening-a-pr)

- [Notes for maintainers](#notes-for-maintainers)

<!-- vim-markdown-toc -->

## How to contribute

### Bugs & feature requests

For bugs, please

[open a ticket](https://github.com/theintern/intern/issues/new), providing:

- a description of the problem

- reproduction steps

- expected result

- actual result

- environment information (Intern version, Node version, browser, OS)

It’s very hard for us to solve your problem without all of this information.

For feature requests, open a ticket describing what you’d like to see and we’ll

try to figure out how it can happen! We (and all the other Intern users) would

really appreciate it if you could also pitch in to actually implement the

feature (maybe with some help from us?).

Note that we prefer to keep the issue tracker focused on development tasks. If

you have questions about using Intern or writing tests, please see our

[Help](docs/help.md) doc.

### Code or documentation

If you want to get involved with the sexy, sexy world of testing software, but

aren’t sure where to start, come

[talk to us on Gitter](https://gitter.im/theintern/intern) or look through the

[issues](https://github.com/theintern/intern/issues) for something that piques

your interest. The development team is always happy to provide guidance to new

contributors!

If you’re not a coder (or you just don’t want to write code), we can still

really use your help in other areas, like improving documentation, performing

marketing and outreach, or [helping other users](docs/help.md), so get in touch

if you’d be willing to help in any way!

Like most open source projects, we require everyone to sign a contributor

license agreement before we can accept any pull requests. You’ll be asked to

sign the CLA when you open your first PR.

## Contributing updates

We ask that any updates use the same style as the existing code and

documentation. Intern installs a pre-commit hook that runs our linting and

styling tools whenever you commit, so that \_should\_ be automatically taken care

of. If possible and appropriate, updated tests should also be a part of

code-based pull requests. (If you’re having trouble writing tests, we can help

you with them!)

### Getting started

The first step in creating an update is making sure you can build Intern and run

the self tests. All of Intern’s build and test processes are handled through

`npm` scripts.

1. Fork this repository and clone your fork

2. In the repo, run `npm ci` to install development packages

3. Run `npm test` to run the Node-based self-tests

4. Run `npm run test:chrome` to run the self-tests in Chrome (“edge”, “firefox”,

and “safari” are also available)

> ? Intern requires at least Node 10.0.0 to build and run.

Assuming everything is working, create a new branch in your repo for your work.

This is what you’ll eventually use to open a PR.

### Dev scripts

Intern uses a number of `npm` scripts to manage the development process. The

most commonly used ones are:

- `clean` - clean up build artifacts

- `build` - build the Intern package and the API documentation data

- `start` - start an Intern test server for use with the browser client

- `test` - build Intern and run Node-based unit tests

- `test:<browser>` - run unit and functional tests in `<browser>` (chrome, edge,

firefox, safari)

### Code style

Intern uses [prettier](https://prettier.io), so formatting issues should be

automatically taken care of.

> ?Prettier doesn’t (yet) reflow comments; those should be wrapped at 80

> characters. Long lines for URLs are fine.

Intern uses [eslint](https://eslint.org) to enforce code style rules (no unused

imports, prefer `const` where applicable, etc.). Some issues will be

automatically fixed by eslint when the pre-commit hook is run, but other issues

may require manual intervention (and will cause a commit to fail).

### Writing self-tests

Tests are in the `tests` directory, and the test config is the `intern.json`

file in the project root. Tests are organized by type in `unit/`, `functional/`,

`integreation/`, and `benchmark/` directories. The directory structure within

each type should mirror the main `src/` directory structure. For example, unit

tests for `src/core/lib/executors/Executor` should go in

`tests/unit/core/lib/executors/Executor`.

While most of the existing tests use the “object” interface (`registerSuite`),

new tests should use the “bdd” interface (`describe` and `it`). Suite names

should generally indicate what module is being tested.

```ts

import { describe, it } from 'src/core/lib/interfaces/bdd';

describe('core/lib/someModule', () => {

it('should do something', () => { ... };

it('should do something else', () => { ... };

});

```

### Commit format

Commit messages should follow the

[conventional commits](https://www.conventionalcommits.org/en/v1.0.0/#summary)

standard. Intern uses [commitlint](https://commitlint.js.org/#/) to check that

messages follow the format. Basically, they should look like:

```

<type>[optional scope]: <description>

[optional body]

[optional footer(s)]

```

A couple of examples:

```

fix: fix instrumentation bug in Node < 10.16.0

resolves #1111

```

```

feat: switch to native Promises for async APIs

- All async APIs now use native Promises -- no more Task or

CancellablePromise

- The cancellation API is mostly new. Command chains still provide a

`cancel` method. Tests, suites, and exector runs can be cancelled by

calling `cancel` on the test, suite, or executor.

references #1018

BREAKING CHANGE: async APIs now return native Promises, and

the cancellation API is entirely new

```

The `type` field should be one of:

- `feat` - adds a feature

- `fix` - fixes a bug

- `docs` - only updates documentation

- `chore` - other changes that don’t modify source or test files

- `test` - updates to self tests

- `style` - only formatting changes

- `refactor` - a code update that does fix a bug or add a feature

- `build` - updates to the build system or external dependencies

- `ci` - changes to CI scripts or config

- `revert` - revert a previous commit

- `perf` - a code change to improve performance

The `scope` field can indicate the general scope of the commit (cli, webdriver,

tunnels, etc.).

> ?? If a commit introduces a breaking change, it should have a footer section

> that starts with `BREAKING CHANGE:`.

### Opening a PR

When you feel like your work is finished, it’s time to open a PR! (You can also

open a draft PR before your work is done if you’d like early reviews.)

1. Fetch any updates from the main Intern repo, and rebase your branch on

current `master`.

2. Clean up the commit history in your branch. Ideally, each commit in a PR

should do something meaningful (add a feature, fix a bug, update a doc file,

etc.). You can use `git rebase -i` to rearrange and squash commits as needed.

3. Push your changes to your fork of Intern and then open a PR. Reference the

issue the PR is addressing in the initial PR comment, something like

`resolves #123`.

## Notes for maintainers

- Please try to provide helpful feedback when reviewing contributions.

- When in doubt, ask for a second review; don’t commit code that smells wrong

just because it exists.

- Clean up PRs (or encourage the contributor to do so) before merging. PRs

should contain a few meaningful commits. Streams of WIP commit messages should

be squashed.

- Put `[ci skip]` at the end of commit messages for commits that do not modify

any code (README changes, etc.).