# Contributor Guide

## Getting Started

Glad that you decided to make your contribution in Dash, to set up your development environment, run the following commands:

```bash

# in your working directory

$ git clone https://github.com/plotly/dash

$ cd dash

$ python3 -m venv .venv/dev

# activate the virtualenv (on windows .venv\dev\scripts\activate)

$ . .venv/dev/bin/activate

# install dash and dependencies

$ pip install -e .[testing,dev] # in some shells you need \ to escape []

$ cd dash-renderer

# build renderer bundles, this will build all bundles from source code

# the only true source of npm version is defined in package.json

$ npm install

$ npm run build # or `renderer build`

# install dash-renderer for development

$ pip install -e .

# build and install components used in tests

$ cd .. # should be back in dash/ root directory

$ npm install

$ npm run setup-tests

# you should see both dash and dash-renderer are pointed to local source repos

$ pip list | grep dash

```

### Dash-Renderer Beginner Guide

`Dash Renderer` began as a separate repository. It was merged into the main `Dash` repository as part of the 1.0 release. It is the common frontend for all Dash backends (\*\*R\*\* and \*\*Python\*\*), and manages React Component layout and backend event handling.

If you want to contribute or simply dig deeper into Dash, we encourage you to play and taste it. This is the most efficient way to learn and understand everything under the hood.

For contributors with a primarily \*\*Python\*\* or \*\*R\*\* background, this section might help you understand more details about developing and debugging in JavaScript world.

As of Dash 1.2, the renderer bundle and its peer dependencies can be packed and generated from the source code. The `dash-renderer\package.json` file is the one version of the truth for dash renderer version and npm dependencies. A build tool `renderer`, which is a tiny Python script installed by Dash as a command-line tool, has a few commands:

1. `renderer clean` deletes all the previously generated assets by this same tool.

2. `renderer npm` installs all the npm modules using this `package.json` files. Note that the `package-lock.json` file is the computed reference product for the versions defined with tilde(~) or caret(^) syntax in npm.

3. `renderer bundles` parses the locked version JSON, copies all the peer dependencies into dash\_renderer folder, bundles the renderer assets, and generates an `\_\_init\_\_.py` to map all the resources. There are also a list of helpful `scripts` property defined in `package.json` you might need to do some handy tasks like linting, syntax format with prettier, etc.

4. `renderer digest` computes the content hash of each asset in `dash\_renderer` folder, prints out the result in logs, and dumps into a JSON file `digest.json`. Use this when you have a doubt about the current assets in `dash\_renderer`, and compare it with previous result in one shot by this command.

5. `renderer build` runs 1, 2, 3, 4 in sequence as a complete build process from scratch.

6. `renderer build local` runs the same order as in 5 and also generates source maps for debugging purposes. You also need to install dash-renderer with editable mode: `pip install -e .`.

When a change in renderer code doesn't reflect in your browser as expected, this could be: confused bundle generation, caching issue in a browser, Python package not in `editable` mode, etc. The new tool reduces the risk of bundle assets by adding the digest to help compare asset changes.

## Python 2 And 3 Compatibility

Writing Python 2/3 compatible code might be a challenging task for contributors used to working on one particular version, especially new learners who start directly with Python 3.

We use `python-future` as our tool to mainly write Python 3 code and make it back-compatible to Python 2.7 (the only Python 2 version Dash supports). Please refer to [this list of idioms](https://python-future.org/compatible\_idioms.html "https://python-future.org/compatible\_idioms.html") for more details on working with `python-future`.

## Git

Use the [GitHub flow](https://guides.github.com/introduction/flow/) when proposing contributions to this repository (i.e. create a feature branch and submit a PR against the default branch).

### Organize your commits

For pull request with notable file changes or a big feature development, we highly recommend to organize the commits in a logical manner, so it

- makes a code review experience much more pleasant

- facilitates a possible cherry picking with granular commits

\*an intuitive [example](https://github.com/plotly/dash-core-components/pull/548) is worth a thousand words.\*

#### Git Desktop

Git command veterans might argue that a simple terminal and a cherry switch keyboard is the most elegant solution. But in general, a desktop tool makes the task easier.

1. <https://www.gitkraken.com/git-client>

2. <https://desktop.github.com/>

### Emoji

Plotlyers love to use emoji as an effective communication medium for

\*\*Commit Messages\*\*

Emojis make the commit messages :cherry\_blossom:. If you have no idea about what to add ? Here is a nice [cheatsheet](https://gitmoji.carloscuesta.me/) and just be creative!

\*\*Code Review Comments\*\*

- :dancer: `:dancer:` - used to indicate you can merge! Equivalent to GitHub's :squirrel:

- :cow2: `:cow2:` cow tip - minor coding style or code flow point

- :tiger2: `:tiger2:` testing tiger - something needs more tests, or tests need to be improved

- :snake: `:snake:` security snake - known or suspected security flaw

- :goat: `:goat:` grammar goat

- :smile\_cat: `:smile\_cat:` happy cat - for bits of code that you really like!

- :dolls: `:dolls:` documentation dolls

- :pill: `:pill:` performance enhancing pill

- :hocho: `:hocho:` removal of large chunks of code (obsolete stuff, or feature removals)

- :bug: `:bug:` - a bug of some kind. 8 legged or 6. Sometimes poisonous.

- :camel: :palm\_tree: `:camel:` `:palm\_tree:` - The Don't Repeat Yourself (DRY) camel or palm tree.

- :space\_invader: `:space\_invader:` - Too much space or too little.

- :spaghetti: `:spaghetti:` - copy-pasta, used to signal code that was copy-pasted without being updated

### Coding Style

We use `flake8`, `pylint`, and [`black`](https://black.readthedocs.io/en/stable/) for linting. please refer to the relevant steps in `.circleci/config.yml`.

## Tests

Our tests use Google Chrome via Selenium. You will need to install [ChromeDriver](https://chromedriver.chromium.org/getting-started) matching the version of Chrome installed on your system. Here are some helpful tips for [Mac](https://www.kenst.com/2015/03/installing-chromedriver-on-mac-osx/) and [Windows](http://jonathansoma.com/lede/foundations-2018/classes/selenium/selenium-windows-install/).

We use [pytest](https://docs.pytest.org/en/latest/) as our test automation framework, plus [jest](https://jestjs.io/) for a few renderer unit tests. You can `npm run test` to run them all, but this command simply runs `pytest` with no arguments, then `cd dash-renderer && npm run test` for the renderer unit tests.

Most of the time, however, you will want to just run a few relevant tests and let CI run the whole suite. `pytest` lets you specify a directory or file to run tests from (eg `pytest tests/unit`) or a part of the test case name using `-k` - for example `pytest -k cbcx004` will run a single test, or `pytest -k cbcx` will run that whole file. See the [testing tutorial](https://dash.plotly.com/testing) to learn about the test case ID convention we use.

### Unit Tests

For simple API changes, please add adequate unit tests under `/tests/unit`

Note: \*You might find out that we have more integration tests than unit tests in Dash. This doesn't mean unit tests are not important, the [test pyramid](https://martinfowler.com/articles/practical-test-pyramid.html) is still valid. Dash project has its unique traits which needs more integration coverage than typical software project, another reason was that dash was a quick prototype crafted by chris in a lovely montreal summer.\*

### Integration Tests

We introduced the `dash.testing` feature in [Dash 1.0](https://community.plotly.com/t/announcing-dash-testing/24868). It makes writing a Dash integration test much easier. Please read the [tutorial](https://dash.plotly.com/testing) and add relevant integration tests with any new features or bug fixes.

## Financial Contributions

Dash, and many of Plotly's open source products, have been funded through direct sponsorship by companies. [Get in touch] about funding feature additions, consulting, or custom app development.

[Dash Core Components]: https://dash.plotly.com/dash-core-components

[Dash HTML Components]: https://github.com/plotly/dash-html-components

[write your own components]: https://dash.plotly.com/plugins

[Dash Component Boilerplate]: https://github.com/plotly/dash-component-boilerplate

[issues]: https://github.com/plotly/dash-core-components/issues

[GitHub flow]: https://guides.github.com/introduction/flow/

[semantic versioning]: https://semver.org/

[Dash Community Forum]: https://community.plotly.com/c/dash

[Get in touch]: https://plotly.com/products/consulting-and-oem

[Documentation]: https://github.com/orgs/plotly/projects/8

[Dash Docs]: https://github.com/plotly/dash-docs