# Contributing to JupyterHub

Welcome! As a [Jupyter](https://jupyter.org) project,

you can follow the [Jupyter contributor guide](https://jupyter.readthedocs.io/en/latest/contributor/content-contributor.html).

Make sure to also follow [Project Jupyter's Code of Conduct](https://github.com/jupyter/governance/blob/master/conduct/code\_of\_conduct.md)

for a friendly and welcoming collaborative environment.

## Setting up a development environment

<!--

https://jupyterhub.readthedocs.io/en/stable/contributing/setup.html

contains a lot of the same information. Should we merge the docs and

just have this page link to that one?

-->

JupyterHub requires Python >= 3.5 and nodejs.

As a Python project, a development install of JupyterHub follows standard practices for the basics (steps 1-2).

1. clone the repo

```bash

git clone https://github.com/jupyterhub/jupyterhub

```

2. do a development install with pip

```bash

cd jupyterhub

python3 -m pip install --editable .

```

3. install the development requirements,

which include things like testing tools

```bash

python3 -m pip install -r dev-requirements.txt

```

4. install configurable-http-proxy with npm:

```bash

npm install -g configurable-http-proxy

```

5. set up pre-commit hooks for automatic code formatting, etc.

```bash

pre-commit install

```

You can also invoke the pre-commit hook manually at any time with

```bash

pre-commit run

```

## Contributing

JupyterHub has adopted automatic code formatting so you shouldn't

need to worry too much about your code style.

As long as your code is valid,

the pre-commit hook should take care of how it should look.

You can invoke the pre-commit hook by hand at any time with:

```bash

pre-commit run

```

which should run any autoformatting on your code

and tell you about any errors it couldn't fix automatically.

You may also install [black integration](https://github.com/psf/black#editor-integration)

into your text editor to format code automatically.

If you have already committed files before setting up the pre-commit

hook with `pre-commit install`, you can fix everything up using

`pre-commit run --all-files`. You need to make the fixing commit

yourself after that.

## Testing

It's a good idea to write tests to exercise any new features,

or that trigger any bugs that you have fixed to catch regressions.

You can run the tests with:

```bash

pytest -v

```

in the repo directory. If you want to just run certain tests,

check out the [pytest docs](https://pytest.readthedocs.io/en/latest/usage.html)

for how pytest can be called.

For instance, to test only spawner-related things in the REST API:

```bash

pytest -v -k spawn jupyterhub/tests/test\_api.py

```

The tests live in `jupyterhub/tests` and are organized roughly into:

1. `test\_api.py` tests the REST API

2. `test\_pages.py` tests loading the HTML pages

and other collections of tests for different components.

When writing a new test, there should usually be a test of

similar functionality already written and related tests should

be added nearby.

The fixtures live in `jupyterhub/tests/conftest.py`. There are

fixtures that can be used for JupyterHub components, such as:

- `app`: an instance of JupyterHub with mocked parts

- `auth\_state\_enabled`: enables persisting auth\_state (like authentication tokens)

- `db`: a sqlite in-memory DB session

- `io\_loop`: a Tornado event loop

- `event\_loop`: a new asyncio event loop

- `user`: creates a new temporary user

- `admin\_user`: creates a new temporary admin user

- single user servers

- `cleanup\_after`: allows cleanup of single user servers between tests

- mocked service

- `MockServiceSpawner`: a spawner that mocks services for testing with a short poll interval

- `mockservice`: mocked service with no external service url

- `mockservice\_url`: mocked service with a url to test external services

And fixtures to add functionality or spawning behavior:

- `admin\_access`: grants admin access

- `no\_patience`: sets slow-spawning timeouts to zero

- `slow\_spawn`: enables the SlowSpawner (a spawner that takes a few seconds to start)

- `never\_spawn`: enables the NeverSpawner (a spawner that will never start)

- `bad\_spawn`: enables the BadSpawner (a spawner that fails immediately)

- `slow\_bad\_spawn`: enables the SlowBadSpawner (a spawner that fails after a short delay)

To read more about fixtures check out the

[pytest docs](https://docs.pytest.org/en/latest/fixture.html)

for how to use the existing fixtures, and how to create new ones.

When in doubt, feel free to [ask](https://gitter.im/jupyterhub/jupyterhub).