# Contributors Guidelines to PySyft

## Getting Started

### Learn Git and Github

All our development is done using Git and Github. If you're not too familiar with Git and Github, start by reviewing this guide. <https://guides.github.com/activities/hello-world/>

### Slack

A great first place to join the Community is the Slack channel <http://slack.openmined.org>.

### Issues

On <https://github.com/OpenMined/PySyft/issues> you can find all open Issues. You can find a detailed explanation on how to work with issues below under [Issue Allocation](#issue-allocation).

## Setup

### Forking a Repository

To contribute to PySyft you will need to fork the OpenMind/PySyft repository.

Then you can work risk-free on your fork.

You will just need to fork once. After that you can call `git fetch upstream` and `git pull 'branch-name'` before you do your local changes to get the remote changes and be up-to-date

### Setting up Pre-Commit Hook

PySyft uses the python package `pre-commit` to make sure the correct formatting (black & flake) is applied.

You can install it via `pip install -r pip-dep/requirements\_dev.txt` or directly doing `pip install pre-commit`

Then you just need to call `pre-commit install`

This can all also be done by running `make install\_hooks`

### Syncing a Forked Repository

To sync your fork with the OpenMined/PySyft repository please see this [Guide](https://help.github.com/articles/syncing-a-fork/) on how to sync your fork.

### Installing PySyft after Cloning Repository

To install the development version of the package, once the `dev` version of the requirements have been satisified, one should:

1. Follow the instructions as laid out in [INSTALLATION.md](https://github.com/OpenMined/PySyft/blob/master/INSTALLATION.md) to complete the installation process.

2. Make a clone of PySyft repo on one's local machine at the terminal

3. Set up the pre-commit hook as described above in [Setting up Pre-Commit Hook](#Setting-up-Pre-Commit-Hook)

4. Do the following two steps:

```bash

cd PySyft

pip install -e .

```

NOTE: If you are using a virtual environment, please be sure to use the correct executable for `pip` or `python` instead.

### Deploying Workers

You can follow along [this example](./examples/deploy\_workers/deploy-and-connect.ipynb) to learn how to deploy PySyft workers and start playing around.

## Contributing

### Beginner Issues

If you are new to the project and want to get into the code, we recommend picking an issue with the label "good first issue". These issues should only require general programming knowledge and little to none insights into the project.

### Issue Allocation

Each issue someone is currently working on should have an assignee. If you want to contribute to an issue someone else is already working on please make sure to get in contact with that person via slack or github and organize yourself.

If you want to work on an open issue, please post a comment telling that you will work on that issue, we will assign you as the assignee then.

\*\*Caution\*\*: We try our best to keep the assignee up-to-date, but as we are all humans with our own schedule delays are possible, so make sure to check the comments once before you start working on an issue even when no one is assigned to it.

### Writing Test Cases

Always make sure to create the necessary tests and keep test coverage at 100%. You can always ask for help in slack or via github if you don't feel confident about your tests.

We aim to have a 100% test coverage, and the GitHub Actions CI will fail if the coverage is below this value. You can evaluate your coverage using the following commands.

```bash

coverage run --omit=\*/venv/\*,setup.py,.eggs/\* setup.py test

coverage report --fail-under 100 -m

```

PySyft is using `pytest` to execute the test cases.

#### Parametrize your Test Cases

Sometimes you want to test functions that hold multiple arguments, which again can have multiple values. To test this, please parametrize your tests.

Example:

```python

@pytest.mark.parametrize(

"compress, compressScheme", [(True, "lz4"), (False, "lz4")]

)

def test\_hooked\_tensor(self, compress, compressScheme):

TorchHook(torch)

t = Tensor(numpy.random.random((100, 100)))

t\_serialized = serialize(t, compress=compress, compressScheme=compressScheme)

t\_serialized\_deserialized = deserialize(

t\_serialized, compressed=compress, compressScheme=compressScheme

)

assert (t == t\_serialized\_deserialized).all()

```

### Process for Serde Protocol Changes

Constants related to PySyft Serde protocol are located in separate repository: [OpenMined/syft-proto](https://github.com/OpenMined/syft-proto).

All classes that need to be serialized have to be listed in the [`proto.json`](https://github.com/OpenMined/syft-proto/blob/master/proto.json) file and have unique code value.

Updating lists of \_simplifiers and detailers\_ in `syft/serde/native\_serde.py`, `syft/serde/serde.py`, `syft/serde/torch\_serde.py`

or renaming/moving related classes can make unit tests fail because `proto.json` won't be in sync with PySyft code anymore.

Use following process:

1. Fork [OpenMined/syft-proto](https://github.com/OpenMined/syft-proto) and create new branch.

2. In your PySyft branch, update `pip-deps/requirements.txt` file to have `git+git://github.com/<your\_account>/syft-proto@<branch>#egg=syft-proto` instead of `syft-proto>=\*`.

3. Make required changes in your PySyft and syft-proto branches. [`helpers/update\_types.py`](https://github.com/OpenMined/syft-proto/blob/master/helpers/update\_types.py) can help update `proto.json` automatically.

4. Create PRs in PySyft and syft-proto repos.

5. PRs should pass CI checks.

6. After syft-proto PR is merged, new version of syft-proto will be published automatically. You can look up new version [in PyPI

](https://pypi.org/project/syft-proto/#history).

7. Before merging PySyft PR, update `pip-deps/requirements.txt` to revert from `git+git://github.com/<your\_account>/syft-proto@<branch>#egg=syft-proto` to `syft-proto>=<new version>`.

### Documentation and Codestyle

To ensure code quality and make sure other people can understand your changes, you have to document your code. For documentation we are using the Google Python Style Rules which can be found [here](https://github.com/google/styleguide/blob/gh-pages/pyguide.md). A well wrote example can we viewed [here](https://sphinxcontrib-napoleon.readthedocs.io/en/latest/example\_google.html).

You documentation should not describe the obvious, but explain what's the intention behind the code and how you tried to realize your intention.

You should also document non self-explanatory code fragments e.g. complicated for-loops. Again please do not just describe what each line is doing but also explain the idea behind the code fragment and why you decided to use that exact solution.

#### Imports

For better merge compatibility each import is within a separate line. Multiple imports from one package are written in one line each.

Example:

```python

from syft.serde import serialize

from syft.serde import deserialize

```

#### Generating Documentation

```bash

sphinx-apidoc -f -o docs/modules/ syft/

```

#### Type Checking

The codebase contains [static type hints](https://docs.python.org/3/library/typing.html) for code clarity and catching errors prior to runtime. If you're adding type hints, please run the static type checker to ensure the type annotations you added are correct via:

```bash

mypy syft

```

Due to issue [#2323](https://github.com/OpenMined/PySyft/issues/2323) you can ignore existing type issues found by mypy.

### Keep it DRY (Don't repeat yourself)

As with any software project, it's important to keep the amount of code to a minimum, so keep code duplication to a minimum!

### Contributing a notebook and adding it to the CI system

If you are contributing a notebook, please ensure you install the requirements for testing notebooks locally. `pip install -r pip-dep/requirements\_notebooks.txt`.

Also please add tests for it in the `tests/notebook/test\_notebooks.py` file. There are plenty of examples, for questions about the notebook tests please feel free to reference https://github.com/fdroessler.

### Creating a Pull Request

At any point in time you can create a pull request, so others can see your changes and give you feedback.

Please create all pull requests to the `master` branch.

If your PR is still work in progress and not ready to be merged please add a `[WIP]` at the start of the title.

Example:`[WIP] Serialization of PointerTensor`

### Check CI and Wait for Reviews

After each commit GitHub Actions will check your new code against the formatting guidelines (should not cause any problems when you setup your pre-commit hook) and execute the tests to check if the test coverage is high enough.

We will only merge PRs that pass the GitHub Actions checks.

If your check fails, don't worry, you will still be able to make changes and make your code pass the checks.