# How to contribute to ESPnet

If you are interested in contributing to ESPnet, your contributions will fall into three categories:

1. If you want to propose a new feature and implement it, please post about your intended feature at the issues,

or you can contact Shinji Watanabe <shinjiw@ieee.org> or other main developers.

We shall discuss the design and implementation.

Once we agree that the plan looks good, go ahead and implement it.

You can find ongoing major development plans at https://github.com/espnet/espnet/milestones

2. If you want to implement a minor feature or bug-fix for an issue, please first take a look at

the existing issues (https://github.com/espnet/espnet/pulls) and/or pull requests (https://github.com/espnet/espnet/pulls).

Pick an issue and comment on the task that you want to work on this feature.

If you need more context on a particular issue, please ask us and then we shall provide more information.

We also welcome if you find some bugs during your actual use of ESPnet and make a PR to fix them.

3. ESPnet provides and maintains a lot of reproducible examples similar to Kaldi (called `recipe`).

The recipe creation/update/bug-fix is one of our major development items, and we really encourage

you to work on it.

When you port a Kaldi recipe to ESPnet, see https://github.com/espnet/espnet/wiki/How-to-port-the-Kaldi-recipe-to-the-ESPnet-recipe%3F

We also encourage you to report your results with your detailed environmental info and upload the model for the reproducibility

(e.g., see https://github.com/espnet/espnet/blob/master/egs/tedlium2/asr1/RESULTS.md).

To make a report for `RESULTS.md`

- execute `show\_result.sh` at a recipe main directory (where `run.sh` is located), as follows.

You'll get environmental information and the evaluation result of each experiments in a markdown format.

```

$ show\_result.sh

```

- execute `pack\_model.sh` at a recipe main directory as follows. You'll get model information in a markdown format

```

$ pack\_model.sh --lm <language model> <tr\_conf> <dec\_conf> <cmvn> <e2e>

```

- `pack\_model.sh` also produces a packed espnet model (`model.tar.gz`). If you upload this model to somewhere with a download link,

please put the link information to `RESULTS.md`.

- please contact Shinji Watanabe <shinjiw@ieee.org> if you want a web storage to put your model files.

Once you finish implementing a feature or bugfix, please send a Pull Request to https://github.com/espnet/espnet

If you are not familiar with creating a Pull Request, here are some guides:

- http://stackoverflow.com/questions/14680711/how-to-do-a-github-pull-request

- https://help.github.com/articles/creating-a-pull-request/

## Version policy and development branches

We basically maintain the `master` and `v.0.X.0` branches for our major developments.

1. We will keep the first version digit `0` until we have some super major changes in the project organization level.

2. The second version digit will be updated when we have major updates including new functions and refactoring, and

their related bug fix and recipe changes.

This version update will be done roughly at every half year so far (but it depends on the development plan).

This is developed at the `v.0.X.0` branch to avoid confusions in the `master` branch.

3. The third version digit will be updated when we fix serious bugs or accumulate some minor changes including

recipe related changes periodically (every two months or so).

This is developed at the `master` branch, and these changes are also reflected to the `v.0.X.0` branch frequently.

## Unit testing

ESPnet's testing is located under `test/`. You can install additional packages for testing as follows:

``` console

$ cd <espnet\_root>

$ pip install -e ".[test]"

```

### python

Then you can run the entire test suite using [flake8](http://flake8.pycqa.org/en/latest/), [autopep8](https://github.com/hhatto/autopep8) and [pytest](https://docs.pytest.org/en/latest/) with [coverage](https://pytest-cov.readthedocs.io/en/latest/reporting.html) by

``` console

./ci/test\_python.sh

```

To create new test file. write functions named like `def test\_yyy(...)` in files like `test\_xxx.py` under `test/`.

[Pytest](https://docs.pytest.org/en/latest/) will automatically test them.

You can find pytest fixtures in `test/conftest.py`. [They finalize unit tests.](https://docs.pytest.org/en/latest/fixture.html#using-fixtures-from-classes-modules-or-projects)

### bash scripts

You can also test the scripts in `utils` with [bats-core](https://github.com/bats-core/bats-core) and [shellcheck](https://github.com/koalaman/shellcheck).

To test:

``` console

./ci/test\_bash.sh

```

### Configuration files

- [setup.cfg](setup.cfg) configures pytest and flake8.

- [.travis.yml](.travis.yml) configures Travis-CI.

- [.circleci/config.yml](.circleci/config.yml) configures Circle-CI.

## Writing new tools

You can place your new tools under

- `espnet/bin`: heavy and large (e.g., neural network related) core tools.

- `utils`: lightweight self-contained python/bash scripts.

For `utils` scripts, do not forget to add test scripts under `test\_utils`.

### Python tools guideline

To generate doc, do not forget `def get\_parser(): -> ArgumentParser` in the main file.

```python

#!/usr/bin/env python3

# Copyright XXX

# Apache 2.0 (http://www.apache.org/licenses/LICENSE-2.0)

import argparse

# NOTE: do not forget this

def get\_parser():

parser = argparse.ArgumentParser(

description="awsome tool", # DO NOT forget this

)

...

return parser

if \_\_name\_\_ == '\_\_main\_\_':

args = get\_parser().parse\_args()

...

```

### Bash tools guideline

To generate doc, support `--help` to show its usage. If you use Kaldi's `utils/parse\_option.sh`, define `help\_message="Usage: $0 ..."`.

## Writing documentation

See [doc](doc/README.md).

## Adding pretrained models

Pack your trained models using `utils/pack\_model.sh` and upload it [here](https://drive.google.com/open?id=1k9RRyc06Zl0mM2A7mi-hxNiNMFb\_YzTF) (You require permission).

Add the shared link to `utils/recog\_wav.sh` or `utils/synth\_wav.sh` as follows:

```sh

"tedlium.demo") share\_url="https://drive.google.com/open?id=1UqIY6WJMZ4sxNxSugUqp3mrGb3j6h7xe" ;;

```

The model name is arbitrary for now.

## On CI failure

### Travis CI

1. read log from PR checks > details

### Circle CI

1. read log from PR checks > details

2. turn on Rerun workflow > Rerun job with SSH

3. open your local terminal and `ssh -p xxx xxx` (check circle ci log for the exact address)

4. try anything you can to pass the CI

### Codecov

1. write more tests to increase coverage

2. explain to reviewers why you can't increase coverage