Contributing to Pipenv

======================

If you\'re reading this, you\'re probably interested in contributing to

Pipenv. Thank you very much! Open source projects live-and-die based on

the support they receive from others, and the fact that you\'re even

considering contributing to the Pipenv project is \*very\* generous of

you.

This document lays out guidelines and advice for contributing to this

project. If you\'re thinking of contributing, please start by reading

this document and getting a feel for how contributing to this project

works. If you have any questions, feel free to reach out to either [Dan

Ryan](https://github.com/techalchemy), [Tzu-ping

Chung](https://github.com/uranusjr), or [Nate

Prewitt](https://github.com/nateprewitt), the primary maintainers.

The guide is split into sections based on the type of contribution

you\'re thinking of making, with a section that covers general

guidelines for all contributors.

General Guidelines

------------------

### Be Cordial

> \*\*Be cordial or be on your way\*\*. \*---Kenneth Reitz\*

Pipenv has one very important rule governing all forms of contribution,

including reporting bugs or requesting features. This golden rule is

\"[be cordial or be on your

way](https://www.kennethreitz.org/essays/be-cordial-or-be-on-your-way)\".

\*\*All contributions are welcome\*\*, as long as everyone involved is

treated with respect.

### Get Early Feedback {#early-feedback}

If you are contributing, do not feel the need to sit on your

contribution until it is perfectly polished and complete. It helps

everyone involved for you to seek feedback as early as you possibly can.

Submitting an early, unfinished version of your contribution for

feedback in no way prejudices your chances of getting that contribution

accepted, and can save you from putting a lot of work into a

contribution that is not suitable for the project.

### Contribution Suitability

Our project maintainers have the last word on whether or not a

contribution is suitable for Pipenv. All contributions will be

considered carefully, but from time to time, contributions will be

rejected because they do not suit the current goals or needs of the

project.

If your contribution is rejected, don\'t despair! As long as you

followed these guidelines, you will have a much better chance of getting

your next contribution accepted.

Questions

---------

The GitHub issue tracker is for \*bug reports\* and \*feature requests\*.

Please do not use it to ask questions about how to use Pipenv. These

questions should instead be directed to [Stack

Overflow](https://stackoverflow.com/). Make sure that your question is

tagged with the `pipenv` tag when asking it on Stack Overflow, to ensure

that it is answered promptly and accurately.

Code Contributions

------------------

### Steps for Submitting Code

When contributing code, you\'ll want to follow this checklist:

1. Understand our [development

philosophy](https://pipenv.pypa.io/en/latest/dev/philosophy/).

2. Fork the repository on GitHub.

3. Set up your `dev-setup`{.interpreted-text role="ref"}

4. Run the tests (`testing`{.interpreted-text role="ref"}) to confirm

they all pass on your system. If they don\'t, you\'ll need to

investigate why they fail. If you\'re unable to diagnose this

yourself, raise it as a bug report by following the guidelines in

this document: `bug-reports`{.interpreted-text role="ref"}.

5. Write tests that demonstrate your bug or feature. Ensure that they

fail.

6. Make your change.

7. Run the entire test suite again, confirming that all tests pass

\*including the ones you just added\*.

8. Send a GitHub Pull Request to the main repository\'s `master`

branch. GitHub Pull Requests are the expected method of code

collaboration on this project.

The following sub-sections go into more detail on some of the points

above.

### Development Setup {#dev-setup}

To get your development environment setup, run:

``` {.sh}

pip install -e .

pipenv install --dev

```

This will install the repo version of Pipenv and then install the

development dependencies. Once that has completed, you can start

developing.

The repo version of Pipenv must be installed over other global versions

to resolve conflicts with the `pipenv` folder being implicitly added to

`sys.path`. See

[pypa/pipenv\#2557](https://github.com/pypa/pipenv/issues/2557) for more

details.

### Testing

Tests are written in `pytest` style and can be run very simply:

``` {.sh}

pytest

```

This will run all Pipenv tests, which can take awhile. To run a subset

of the tests, the standard pytest filters are available, such as:

- provide a directory or file: `pytest tests/unit` or

`pytest tests/unit/test\_cmdparse.py`

- provide a keyword expression:

`pytest -k test\_lock\_editable\_vcs\_without\_install`

- provide a nodeid: `pytest tests/unit/test\_cmdparse.py::test\_parse`

- provide a test marker: `pytest -m lock`

### Code Review

Contributions will not be merged until they\'ve been code reviewed. You

should implement any code review feedback unless you strongly object to

it. In the event that you object to the code review feedback, you should

make your case clearly and calmly. If, after doing so, the feedback is

judged to still apply, you must either apply the feedback or withdraw

your contribution.

### Package Index

To speed up testing, tests that rely on a package index for locking and

installing use a local server that contains vendored packages in the

`tests/pypi` directory. Each vendored package should have it\'s own

folder containing the necessary releases. When adding a release for a

package, it is easiest to use either the `.tar.gz` or universal wheels

(ex: `py2.py3-none`). If a `.tar.gz` or universal wheel is not

available, add wheels for all available architectures and platforms.

Documentation Contributions

---------------------------

Documentation improvements are always welcome! The documentation files

live in the `docs/` directory of the codebase. They\'re written in

[reStructuredText](http://docutils.sourceforge.net/rst.html), and use

[Sphinx](http://sphinx-doc.org/index.html) to generate the full suite of

documentation.

When contributing documentation, please do your best to follow the style

of the documentation files. This means a soft-limit of 79 characters

wide in your text files and a semi-formal, yet friendly and

approachable, prose style.

When presenting Python code, use single-quoted strings (`'hello'`

instead of `"hello"`).

Bug Reports

-----------

Bug reports are hugely important! They are recorded as [GitHub

issues](https://github.com/pypa/pipenv/issues). Please be aware of the

following things when filing bug reports:

1. Avoid raising duplicate issues. \*Please\* use the GitHub issue search

feature to check whether your bug report or feature request has been

mentioned in the past. Duplicate bug reports and feature requests

are a huge maintenance burden on the limited resources of the

project. If it is clear from your report that you would have

struggled to find the original, that\'s ok, but if searching for a

selection of words in your issue title would have found the

duplicate then the issue will likely be closed extremely abruptly.

2. When filing bug reports about exceptions or tracebacks, please

include the \*complete\* traceback. Partial tracebacks, or just the

exception text, are not helpful. Issues that do not contain complete

tracebacks may be closed without warning.

3. Make sure you provide a suitable amount of information to work with.

This means you should provide:

- Guidance on \*\*how to reproduce the issue\*\*. Ideally, this should

be a \*small\* code sample that can be run immediately by the

maintainers. Failing that, let us know what you\'re doing, how

often it happens, what environment you\'re using, etc. Be

thorough: it prevents us needing to ask further questions.

- Tell us \*\*what you expected to happen\*\*. When we run your

example code, what are we expecting to happen? What does

\"success\" look like for your code?

- Tell us \*\*what actually happens\*\*. It\'s not helpful for you to

say \"it doesn\'t work\" or \"it fails\". Tell us \*how\* it

fails: do you get an exception? A hang? The packages installed

seem incorrect? How was the actual result different from your

expected result?

- Tell us \*\*what version of Pipenv you\'re using\*\*, and \*\*how you

installed it\*\*. Different versions of Pipenv behave differently

and have different bugs, and some distributors of Pipenv ship

patches on top of the code we supply.

If you do not provide all of these things, it will take us much

longer to fix your problem. If we ask you to clarify these and you

never respond, we will close your issue without fixing it.

Run the tests

-------------

Three ways of running the tests are as follows:

1. `make test` (which uses `docker`)

2. `./run-tests.sh` or `run-tests.bat`

3. Using pipenv:

``` {.console}

$ git clone https://github.com/pypa/pipenv.git

$ cd pipenv

$ git submodule sync && git submodule update --init --recursive

$ pipenv install --dev

$ pipenv run pytest

```

For the last two, it is important that your environment is setup

correctly, and this may take some work, for example, on a specific Mac

installation, the following steps may be needed:

# Make sure the tests can access github

if [ "$SSH\_AGENT\_PID" = "" ]

then

eval `ssh-agent`

ssh-add

fi

# Use unix like utilities, installed with brew,

# e.g. brew install coreutils

for d in /usr/local/opt/\*/libexec/gnubin /usr/local/opt/python/libexec/bin

do

[[ ":$PATH:" != \*":$d:"\* ]] && PATH="$d:${PATH}"

done

export PATH

# PIP\_FIND\_LINKS currently breaks test\_uninstall.py

unset PIP\_FIND\_LINKS