Maintenance

Release 0.0.0

August 16, 2016

Contents

These are notes on maintaining the various open-source projects I distribute - while mostly as a way of making various tasks easier for me, these notes will hopefully be useful to other people and projects.

CHAPTER 1

Deployment

1.1 Development environments

When installing packages for Python, Ruby or other similar languages, always avoid installing packages alongside packages installed with the systems package manager.

1.1.1 Python

Use pipsi to install python tools, and virtualenv/tox to install project dependencies. If it's necessary to install packages into a global namespace, use `pip install --user`. v and virtualenvwrapper are useful tools to manage a set of Python virtual environments.

Python 2.7 is a required package on Debian and RHEL systems, and often comes with python modules installed via the system's package manager. **It's a really bad idea to install python packages that aren't managed by the system package manager into the system's Python installation.** In particular, pip may break things on some versions of Debian/Ubuntu due to the way the Debian packages are distributed.

1.1.2 Ruby

Use `gem install --user-install` to install gems into your home directory instead of the systems folders. rvm can be used to install and manage multiple Ruby versions. bundler can be used to manage project dependencies.

Uninstall **all** Ruby gems with:

```
gem list | cut -d" " -f1 | xargs sudo gem uninstall -aIx
```

Pages on specific languages or topics contain detailed information and explanations. They are usually followed by a checklist page which provides minimal notes on steps to take in a project or package's workflow.

CHAPTER 2

Projects

This section contains general notes on publishing all open-source projects.

2.1 Essentials

All projects should have:

- A known maintainer
- Basic installation and usage instructions
- A style guide assigned

2.2 License

Code projects should be licensed under the MIT license:

Written documentation or other creative work should be licensed under the Creative Commons Attribution-ShareAlike 4.0 International License.

CHAPTER 3

Rust

3.1 Packaging

Follow the Cargo guide. Nice and simple.

3.1.1 Readme

```
# PROJECT [![](https://img.shields.io/github/tag/borntyping/PROJECT.svg)](https://github.com/borntypi

Project description

* [Source on GitHub](https://github.com/borntyping/PROJECT)
* [Packages on Crates.io](https://crates.io/crates/PROJECT)
* [Builds on Travis CI](https://travis-ci.org/borntyping/PROJECT)

Usage
-----

Run the program using `cargo`:

    cargo run

Licence
-------

`PROJECT` is licenced under the [MIT Licence](http://opensource.org/licenses/MIT).

Authors
-------

Written by [Sam Clements](sam@borntyping.co.uk).
```

3.2 Testing

3.2.1 Test annotations

Code can be annotated to mark it as a test.

3.2.2 Cargo

Run tests with *cargo test*.

3.2.3 Travis-CI configuration

For a project using Cargo, a *.travis.yml* is very simple:

```
language: rust
```

3.3 Checklist

3.3.1 Creating a new package

Package structure *cargo new <project>*

3.3.2 Releasing a package

Run tests *cargo test*

Update metadata Change version number in *Cargo.toml*

Tag release `git-tag-version` *or* `git tag -a "v0.1.0" -m "Version 0.1.0"`

Package upload *cargo publish*

3.3.3 Documentation

Build documentation *cargo doc*

Build and open documentation *cargo doc –open*

CHAPTER 4

Python

4.1 Packaging Python modules

The Python Packaging User Guide has excellent guidance on packaging Python projects - in particular, use the information on packaging and distributing projects, and the suggested tools. Python projects that provide a package should include `setup.py` and `setup.cfg` configuration files:

```
from setuptools import setup

setup(
    name='<NAME>',
    version='0.0.0',

    description='A short description of the package',
    long_description=open('README.md').read(),

    author='Sam Clements',
    author_email='sam@borntyping.co.uk',
    url='https://github.com/borntyping/python-<NAME>',
    license='MIT License',

    packages=[
        '<NAME>'
    ],

    classifiers=[
        'Development Status :: 3 - Alpha',
        'License :: OSI Approved :: MIT License',
        'Programming Language :: Python',
        'Programming Language :: Python :: 2',
        'Programming Language :: Python :: 2.6',
        'Programming Language :: Python :: 2.7',
        'Programming Language :: Python :: 3',
        'Programming Language :: Python :: 3.3',
        'Programming Language :: Python :: 3.4'
    ]
)
```

Notes on specific metadata items:

- Version numbers should follow the semantic versioning specification.
- PyPI provides a list of available package classifiers. - Python 3 only projects should use `Programming`

```
Language :: Python :: 3 :: Only.
```

- Python projects should use the MIT License (see Projects).

Packages that run on Python 2 and 3 should ensure they have a `setup.cfg` file that configures `bdist_wheel` to mark the wheel as 'universal' (otherwise, the wheel will declare that it is only for the version of Python it was built on).

```
[bdist_wheel]
universal=1
```

4.1.1 PyPi

A new python package can be registered to PyPI with:

```
python setup.py register
```

Python packages can generally be released with:

```
python setup.py sdist bdist_wheel upload
```

Tox can be used to automate package releases, by including a test environment that runs the release commands:

```
[testenv:release]
commands=python setup.py sdist bdist_wheel upload
deps=wheel
```

In the unusual case that a package supplies metadata depending on which Python version it is running on (see complex_setup_example), you'll need to run *bdist_wheel* for each of those Python versions. While source distributions run *setup.py* when *installing* a package, built distributions run *setup.py* when creating the package).

```
[testenv:release]
commands=
    python2 setup.py sdist upload
    python2 setup.py bdist_wheel upload
    python3 setup.py bdist_wheel upload
deps=wheel
```

4.1.2 Readme

README files for Python projects should use Markdown (see this commit). The README should include links to the GitHub repository, any packages on PyPi,builds on Travis-CI and documentation on Read The Docs (which is especially useful when the README is shown in multiple places).

```
# EXAMPLE [![](https://img.shields.io/pypi/v/EXAMPLE.svg)](https://warehouse.python.org/project/EXAMP

A short description of the project.

* [Source on GitHub](https://github.com/borntyping/EXAMPLE)
* [Packages on PyPI](https://warehouse.python.org/project/EXAMPLE/)
* [Documentation on Read the Docs](https://EXAMPLE.readthedocs.org/en/latest/)
* [Builds on Travis CI](https://travis-ci.org/borntyping/EXAMPLE)

Usage
-----

Run `EXAMPLE --help` for a list of availible subcommands.
```

```
Installation
------------

```bash
pip install EXAMPLE
```

Licence

EXAMPLE is licenced under the [MIT Licence](http://opensource.org/licenses/MIT).

Authors

Written by [Sam Clements](sam@borntyping.co.uk).
```

You will need to create a `MANIFEST.in` file so that the README is included in the package:

```
include README.md
```

README files for older Python projects using reStructuredText can use the rst-lint tool for checking the validity of a README file.

```
example
=======

.. image:: https://img.shields.io/pypi/v/example.svg?style=flat-square
    :target: https://warehouse.python.org/project/example/
    :alt: example on PyPI

.. image:: https://img.shields.io/pypi/l/example.svg?style=flat-square
    :target: https://warehouse.python.org/project/example/
    :alt: example on PyPI

.. image:: https://readthedocs.org/projects/example/badge/?version=latest&style=flat-square
    :target: https://example.readthedocs.org/en/latest/
    :alt: Documentation for example on Read The Docs

.. image:: https://img.shields.io/travis/borntyping/example/master.svg?style=flat-square
    :target: https://travis-ci.org/borntyping/example
    :alt: Travis-CI build status for example

.. image:: https://img.shields.io/github/issues/borntyping/example.svg?style=flat-square
    :target: https://github.com/borntyping/example/issues
    :alt: GitHub issues for example

|

Short description of the project.

* `Source on GitHub <https://github.com/borntyping/example>`_
* `Documentation on Read the Docs <https://example.readthedocs.org/en/latest/>`_
* `Packages on PyPI <https://warehouse.python.org/project/example/>`_
* `Builds on Travis CI <https://travis-ci.org/borntyping/example>`_
```

4.2 Testing Python projects

- Test runners: Tox, Travis-CI
- Testing: pytest, mock
- Style checker: flake8, flake8-docstrings

Use Tox to run tests across multiple versions of Python. Tox installs the package into a virtualenv for each environment and runs the test command inside that.

Most projects should use pytest, a testing runner and framework that is much easier to use that the `unittest` module.

A basic `tox.ini` file to use pytest should look like this:

```
[tox]
minversion=1.8.0
envlist=py{26,27,33,34},lint,docs

[testenv]
commands=py.test {posargs} <MODULE>
deps=pytest

[pytest]
addopts=-q
```

Configuring Travis-CI to run Tox properly is slightly verbose, but means local and remote tests run in an almost identical fashion. The `.travis.yml` file should include each tox environment in it's matrix (see the *Configuration* section below for a full example):

```
language: python
env:
  - TOXENV=py26
  - TOXENV=py27
  - TOXENV=py33
  - TOXENV=py34
  - TOXENV=lint
  - TOXENV=docs
install:
  - pip install tox
script:
  - tox
```

4.2.1 Style and lint checking

Use flake8 to run style and lint checks, which collects pyflakes, pep8 and mccabe. The flake8-docstrings module extends it to include pep257. As it reads it's configuration from `tox.ini`, it can be added to Tox very easily.

```
[testenv:lint]
commands=flake8 <MODULE>
basepython=python2.7
deps=
    flake8
    flake8_docstrings

[flake8]
ignore=D102,D203
```

This ignores the following messages:

```
D102: Function docstring missing
D203: Expected 1 blank line *before* class docstring, found 0
```

4.2.2 Configuration

A full Tox configuration file for a standard python package should run tests on each supported version of Python, run style checkers, and include environments to build documentation and release the package (see the Python Packaging page). This configuration allows packages to be built and tested without needing to install any dependancies other than Tox.

```
[tox]
minversion=1.8.0
envlist=py{26,27,33,34},lint,docs

[testenv]
commands=py.test {posargs} <MODULE>
deps=pytest

[pytest]
addopts=-q

[testenv:lint]
commands=flake8 <MODULE>
basepython=python2.7
deps=
    flake8
    flake8_docstrings

[flake8]
ignore=D102,D203

[testenv:docs]
commands=sphinx-build -qE docs/ docs/_build/
deps=
    sphinx
    sphinx_rtd_theme

[testenv:release]
commands=python setup.py sdist bdist_wheel upload
deps=wheel
```

Travis-CI should be configured to run all environments (excluding releasing the package), though style and documentation environments should be allowed to fail. Including `sudo: false` will use Travis-CI's container based build system.

```
language: python
env:
  - TOXENV=py26
  - TOXENV=py27
  - TOXENV=py33
  - TOXENV=py34
  - TOXENV=lint
  - TOXENV=docs
install:
  - pip install tox
```

```
script:
  - tox
matrix:
  allow_failures:
    - env: TOXENV=lint
    - env: TOXENV=docs
sudo: false
```

4.3 Packaging checklist

4.3.1 Creating a new package

Package metadata Create `setup.py` and `setup.cfg`

Testing metadata Create `tox.ini` and `travis.yml`

Package registration `python setup.py register`

4.3.2 Releasing a package

Run tests `tox`

Update metadata Update version numbers in `setup.py` and `__init__.py`

Tag release `git-tag-version` *or* `git tag -a "v0.1.0" -m "Version 0.1.0"`

Package upload `tox -e release` or `python setup.py sdist bdist_wheel upload`

4.3.3 Documenation

Build Sphinx documenation `sphinx-build docs docs/_build`

Open Sphinx documenation `xdg-open docs/_build/index.html`

The status page links to most of the open-source projects I have published and displays their current status. This includes number of open issues, build status and package information.

CHAPTER 5

Status

This page links to most of the open-source projects I have published and displays their current status. It should list all projects from my github profile, though projects from borntyping-sandbox are not listed.

5.1 Python packages

These packages are actively maintained, though several are feature complete and have no plans for futher improvements. You can view a list of all packages owned or maintained by me on PyPI.

Package	Version	GitHub Issues	CI status	Documentation
clack				
colorlog				
csdl-unoffical				
dice				
mannhunter				
riemann-client				
sous-chef				
ssh-run				
supermann				
tg				
v				
watch-fs				

5.2 Chef cookbooks

Project	Version	GitHub Issues	CI status
alternate_search			
ruby			

5.3 Rust packages

Rust packages are currently only used for my final year university project, and are not really supported or ready yet.

Project	Version	GitHub Issues	CI status
cyborg			
cyborg-demo			
mutiny			
psutil			

5.4 Personal projects

These projects are published under open-source licences, but are primarily for my own use.

Project	GitHub Issues	CI status
maintenance		
deployment		
borntyping.github.io		

5.5 Deprecated projects

These projects are very old, but I can't bring myself to get rid of them. Most of the code is awful. *They are not supported or maintained.*

Package
chef-ladder
this-is-a-game
nus
django-kaos
django-kaos-avatar
enginesheddata

5.5.1 Deprecated python packages

Bugfixes are published for these Python packages, but no further work is planned. `diceroll` and `spotter` have been replaced by `dice` and `watch-fs` respectively, while `argumented` and `infix` shouldn't be used.

Package	Version	GitHub Issues	CI status	Documentation
argumented				
diceroll				
infix				
spotter				