Packaging Python Projects

Christian Külker

2024-06-17

Contents

This document briefly describes the process of packaging Python projects. It follows the Python Software Foundation's Packaging Tutorial, which uses setuptools. **Setuptools** is a collection of extensions to the Python distribution utilities that allow you to more easily build and distribute Python distributions, especially those that have dependencies on other packages. For details on how to use the setuptools, see the [setuptools documentation].

1 Example Test Package

1.1 Create Account

Because TestPyPI has a separate database from the live **PyPI**, you'll need a separate user account specifically for **TestPyPI**. [Create an account at test.pypi.org to register an account with a valid email address.

Go to the Token Section of your account and create a new API token; do not restrict it to a specific project, as you are creating a new project.

Since we will be using `twine`, copy the API token to `$HOME/.pypirc`. However, this will not prevent you from being asked for it. So it may be useless, but mention it in the token section of your test account.

```
[pypi]
  username = __token__
  password = pypi-..............
```

1.2 Create A Project

We create the project structure:

```
mkdir -p packaging_tutorial/example_pkg/tests
touch packaging_tutorial/example_pkg/__init__.py
touch packaging_tutorial/{LICENSE,README.md,setup.py}
cd packaging_tutorial
echo -e "# Example Package\n\nThis is a simple example package.">README.md
vim setup.py
```

The contents of `setup.py`, replace `YOUR-USERNAME-HERE` with your username. And probably update others as well, like the URL to your repository and your email address. (This is a guess, the tutorial does not give any information about this).

```
import setuptools

with open("README.md", "r") as fh:
    long_description = fh.read()

setuptools.setup(
    name="example-pkg-YOUR-USERNAME-HERE", # Replace with your own username
    version="0.0.1",
    author="Example Author",
    author_email="author@example.com",
    description="A small example package",
    long_description=long_description,
    long_description_content_type="text/markdown",
    url="https://github.com/pypa/sampleproject",
    packages=setuptools.find_packages(),
    classifiers=[
        "Programming Language :: Python :: 3",
        "License :: OSI Approved :: MIT License",
        "Operating System :: OS Independent",
    ],
    python_requires='>=3.6',
)
```

1.3 Install Dependencies

Although it is recommended to install the latest version of `setuptool` and `wheel`, if you want to maintain the security management supported by your Linux distribution, you will probably prefer the package manager.

```
aptitude install python3-setuptools python3-wheel twine python3-pip
```

1.4 Build The Project

Build distribution

```
python3 setup.py sdist bdist_wheel
running sdist
running egg_info
creating example_pkg_ckuelker.egg-info
writing example_pkg_ckuelker.egg-info/PKG-INFO
writing dependency_links to
 ↪  example_pkg_ckuelker.egg-info/dependency_links.txt
writing top-level names to example_pkg_ckuelker.egg-info/top_level.txt
writing manifest file 'example_pkg_ckuelker.egg-info/SOURCES.txt'
reading manifest file 'example_pkg_ckuelker.egg-info/SOURCES.txt'
writing manifest file 'example_pkg_ckuelker.egg-info/SOURCES.txt'
running check
creating example-pkg-ckuelker-0.0.1
creating example-pkg-ckuelker-0.0.1/example_pkg
creating example-pkg-ckuelker-0.0.1/example_pkg_ckuelker.egg-info
copying files to example-pkg-ckuelker-0.0.1...
copying README.md -> example-pkg-ckuelker-0.0.1
copying setup.py -> example-pkg-ckuelker-0.0.1
copying example_pkg/__init__.py -> example-pkg-ckuelker-0.0.1/example_pkg
copying example_pkg_ckuelker.egg-info/PKG-INFO ->
 ↪  example-pkg-ckuelker-0.0.1/\
example_pkg_ckuelker.egg-info
copying example_pkg_ckuelker.egg-info/SOURCES.txt -> \
example-pkg-ckuelker-0.0.1/example_pkg_ckuelker.egg-info
copying example_pkg_ckuelker.egg-info/dependency_links.txt -> \
example-pkg-ckuelker-0.0.1/example_pkg_ckuelker.egg-info
copying example_pkg_ckuelker.egg-info/top_level.txt -> \
example-pkg-ckuelker-0.0.1/example_pkg_ckuelker.egg-info
Writing example-pkg-ckuelker-0.0.1/setup.cfg
creating dist
Creating tar archive
removing 'example-pkg-ckuelker-0.0.1' (and everything under it)
running bdist_wheel
running build
running build_py
creating build
creating build/lib
creating build/lib/example_pkg
copying example_pkg/__init__.py -> build/lib/example_pkg
installing to build/bdist.linux-x86_64/wheel
running install
running install_lib
creating build/bdist.linux-x86_64
creating build/bdist.linux-x86_64/wheel
creating build/bdist.linux-x86_64/wheel/example_pkg
copying build/lib/example_pkg/__init__.py ->
 ↪  build/bdist.linux-x86_64/wheel/\
example_pkg
running install_egg_info
Copying example_pkg_ckuelker.egg-info to build/bdist.linux-x86_64/wheel/\
example_pkg_ckuelker-0.0.1.egg-info
running install_scripts
adding license file "LICENSE" (matched pattern "LICEN[CS]E*")
creating
 ↪  build/bdist.linux-x86_64/wheel/example_pkg_ckuelker-0.0.1.dist-info/\
WHEEL
creating 'dist/example_pkg_ckuelker-0.0.1-py3-none-any.whl' and adding \
'build/bdist.linux-x86_64/wheel' to it
adding 'example_pkg/__init__.py'
adding 'example_pkg_ckuelker-0.0.1.dist-info/LICENSE'
adding 'example_pkg_ckuelker-0.0.1.dist-info/METADATA'
adding 'example_pkg_ckuelker-0.0.1.dist-info/WHEEL'
adding 'example_pkg_ckuelker-0.0.1.dist-info/top_level.txt'
adding 'example_pkg_ckuelker-0.0.1.dist-info/RECORD'
removing build/bdist.linux-x86_64/wheel
$ tree dist
dist
├── example_pkg_ckuelker-0.0.1-py3-none-any.whl
└── example-pkg-ckuelker-0.0.1.tar.gz
```

Altogether the project tree looks like this.

```
$ tree
.
├── build
│   ├── bdist.linux-x86_64
│   └── lib
│       └── example_pkg
│           └── __init__.py
├── dist
│   ├── example_pkg_ckuelker-0.0.1-py3-none-any.whl
│   └── example-pkg-ckuelker-0.0.1.tar.gz
├── example_pkg
│   ├── __init__.py
│   └── tests
├── example_pkg_ckuelker.egg-info
│   ├── dependency_links.txt
│   ├── PKG-INFO
│   ├── SOURCES.txt
│   └── top_level.txt
├── LICENSE
├── README.md
└── setup.py
```

1.5 Upload The Project

Upload the distribution (using username `__token__` and the token for password):

```
python3 -m twine upload --repository testpypi dist/*
Enter your username: __token__
/usr/lib/python3/dist-packages/twine/utils.py:238: UserWarning: Failed to
↪  open
```

```
keyring: org.freedesktop.DBus.Error.NoReply: Did not receive a reply.
 ↪  Possible
causes include: the remote application did not send a reply, the message
 ↪  bus
security policy blocked the reply, the reply timeout expired, or the
 ↪  network
connection was broken..
  warnings.warn(str(exc))
Enter your password:
Uploading distributions to https://test.pypi.org/legacy/
Uploading example_pkg_ckuelker-0.0.1-py3-none-any.whl
100%|█████████████████████████████████████| 16.7k/16.7k [00:01<00:00,
 ↪  10.2kB/s]
Uploading example-pkg-ckuelker-0.0.1.tar.gz
100%|█████████████████████████████████████| 4.29k/4.29k [00:00<00:00,
 ↪  4.89kB/s]
```

The package should be visible at `https://test.pypi.org/project/example-pkg-YOUR-USERNAME-HERE`

1.6 Install The Project

It can be installed via `virtualenv` and `pip` (in this case `python2`):

```
virtualenv /tmp/python-packaging-tutorial-example-package
Running virtualenv with interpreter /usr/bin/python2
New python executable in
 ↪  /tmp/python-packaging-tutorial-example-package/bin/\
python2
Also creating executable in
 ↪  /tmp/python-packaging-tutorial-example-package/\
bin/python
Installing setuptools, pkg_resources, pip, wheel...done.
source /tmp/python-packaging-tutorial-example-package/bin/activate
```

For `zsh`, you know that the environment is active when your `zsh` prompt is messed up because it prints (DIR) in front of your prompt.

Or you use `virtalenv` with `python3`

```
virtualenv -p /usr/bin/python3
 ↪  /tmp/python-packaging-tutorial-example-package
Already using interpreter /usr/bin/python3
Using base prefix '/usr'
```

```
New python executable in
 ↪  /tmp/python-packaging-tutorial-example-package/bin/\
python3
Also creating executable in
 ↪  /tmp/python-packaging-tutorial-example-package/\
bin/python
Installing setuptools, pkg_resources, pip, wheel...done.
source /tmp/python-packaging-tutorial-example-package/bin/activate
```

Why `virtualenv` is complaining about 'Already …' is unclear, since it surely used `python2` before. Does this software have some attitude issues?

Install the new test software:

```
python3 -m pip install --index-url https://test.pypi.org/simple/ --no-deps
 ↪  \
example-pkg-YOUR-USERNAME-HERE
Looking in indexes: https://test.pypi.org/simple/
Collecting example-pkg-ckuelker
  Downloading https://test-files.pythonhosted.org/packages/bb/83/\
  db02a09e5963f60e3ef79e9c491d9aaf772d3e9a60d7b628d5434e9d6f26/\
  example_pkg_ckuelker-0.0.1-py3-none-any.whl (13 kB)
Installing collected packages: example-pkg-ckuelker
Successfully installed example-pkg-ckuelker-0.0.1
```

Test the installation with the command line.

```
python3
import example_pkg
```

Since the example package does nothing, there is nothing more to say about this. However, one could add a non-trivial package to test.pypi.org to better understand the process.

2 Real Project

Real projects should be tested on test.pypi.org, but at some point you might want to upload it to the real pypi.org. Here are just a few small differences to the sample project section above.

- Chose a unique name for your project (you should not add your user name to the project name)
- Create an account on pypi.org (not on the test.pypi.org)

- Use `twine upload dist/*` without `--repository` option
- Test installation with `pip3 install` (No `--index-url`, no `--no-deps`)

3 Excursus On Python3 vs PyPy

```
aptitude search setuptools
p   pypy-setuptools                    - PyPy Distutils Enhancements
p   pypy-setuptools-scm                - blessed package to manage your
                                         versions by scm tags for PyPy
p   python-setuptools                  - Python Distutils Enhancements
p   python-setuptools-doc              - Python Distutils Enhancements
                                         (documentation)
p   python-setuptools-git             - plugin for setuptools that enables
 ↪  git
                                         integration
p   python-setuptools-scm              - blessed package to manage your
                                        versions by scm tags for Python 2
v   python2.7-setuptools-git           -
i A python3-setuptools                 - Python3 Distutils Enhancements
p   python3-setuptools-git            - plugin for setuptools that enables
 ↪  git
                                         integration
p   python3-setuptools-scm             - blessed package to manage your
                                        versions by scm tags for Python 3
p   python3-setuptools-scm-git-archive - Plugin for setuptools_scm to add
                                         support for git archives
```

From a distance, Debian offers 2 packages: `python3-setuptools` and `pypy-setuptools`. However, they seem to be the same.

```
aptitude show pypy-setuptools
Package: pypy-setuptools
Version: 40.8.0-1
State: not installed
Multi-Arch: foreign
Priority: optional
Section: python
Maintainer: Matthias Klose <doko@debian.org>
Architecture: all
Uncompressed Size: 1,353 k
Depends: pypy-pkg-resources (= 40.8.0-1), pypy
Suggests: python-setuptools-doc
```

```
Description: PyPy Distutils Enhancements
 Extensions to the python-distutils for large or complex distributions.
Homepage: https://pypi.python.org/pypi/setuptools
```

```
aptitude show python3-setuptools
Package: python3-setuptools
Version: 40.8.0-1
State: installed
Automatically installed: yes
Multi-Arch: foreign
Priority: optional
Section: python
Maintainer: Matthias Klose <doko@debian.org>
Architecture: all
Uncompressed Size: 1,353 k
Depends: python3-pkg-resources (= 40.8.0-1), python3-distutils,
 ↪  python3:any
Suggests: python-setuptools-doc
Description: Python3 Distutils Enhancements
Extensions to the python-distutils for large or complex distributions.
Homepage: https://pypi.python.org/pypi/setuptools
```

4 Local Package Installs and Testing

Consider the following small Flask application called boxbrainiac. It has the following structure. Note that unlike the previous example, the `tests` test directory is at the project level.

```
boxbrainiac-dev  # = PACKAGE_DEV
.
├── boxbrainiac
│   ├── config.py
│   ├── debug.py
│   ├── env.py
│   ├── error.py
│   ├── exception.py
│   ├── git.py
│   ├── __init__.py
│   ├── main.py
│   ├── static
│   │   └── css
│   │       └── default.css
│   ├── store.py
│   ├── templates
│   │   ├── edit_view.html
│   │   ├── error_view.html
│   │   ├── input_view.html
│   │   ├── list_view.html
│   │   └── search_view.html
│   ├── util.py
│   └── version.py
├── MANIFEST.in
├── setup.py
└── tests
    ├── __init__.py
    ├── test_config.py
    ├── test_debug.py
    ├── test_env.py
    ├── test_exception.py
    ├── test_git.py
    ├── test_main.py
    ├── test_store.py
    ├── test_template.py
    └── test_util.py
```

4.1 Test a Python3 Package With pip3 Inside venv

First, we need to create a new virtual environment to isolate our dependencies from the rest of our system. This is done using the `venv` module in Python.

```
cd /tmp
python3 -m venv test_env
cd test_env
source bin/activate # We assume the rest in venv
export PYTHONDONTWRITEBYTECODE=1 # prevent __pycache__
```

Next, we install wheel to make sure we can build the project into a Python wheel file, a kind of built-package format for Python.

```
pip3 install wheel # And probably others
# We assume the package to be in ~/PACKAGE_DEV
cd PACKAGE_DEV
python3 setup.py sdist bdist_wheel
```

This creates two different types of distributable package formats: `sdist` (a source distribution format encapsulated in a tar file) and `bdist_wheel` (a built distribution format encapsulated in a `*.whl` file) and many other files.

```
├── boxbrainiac.egg-info
│   ├── dependency_links.txt
│   ├── entry_points.txt
│   ├── PKG-INFO
│   ├── requires.txt
│   ├── SOURCES.txt
│   └── top_level.txt
├── build
│   ├── bdist.linux-x86_64
│   └── lib
│       └── boxbrainiac
│           ├── config.py
│           ├── debug.py
│           ├── env.py
│           ├── error.py
│           ├── exception.py
│           ├── git.py
│           ├── __init__.py
│           ├── main.py
│           ├── store.py
│           ├── templates
│           │   ├── edit_view.html
│           │   ├── error_view.html
│           │   ├── input_view.html
│           │   ├── list_view.html
│           │   └── search_view.html
│           ├── util.py
│           └── version.py
├── dist
│   ├── boxbrainiac-0.1.0-py3-none-any.whl
│   └── boxbrainiac-0.1.0.tar.gz
```

Some packages can run tests without installation (like Perl modules), in other cases you need to install the package before you can test it. Unlike Perl, there is an uninstall target, so if the tests are negative, the software can be removed. For the moment we assume that we do not need to install it.

4.1.1 Install Test Environment

This will install test dependencies as well as all dependencies and the application.

Bash

```
pip3 install .[test]
```

Zsh

```
pip3 install .\[test\]
```

4.1.2 Run The Tests

To run the tests, we use Python's built-in `unittest` module. This module provides a rich set of tools for constructing and executing tests, and can be run directly from the command line to test single or multiple modules.

```
cd PACKAGE_DEV
python3 -m unittest tests/test_util.py # One test
python3 -m unittest tests/test_*.py    # All tests
```

The last command automatically detects and executes all test cases found in the tests directory.

4.1.3 Install Package Inside venv

Before installing our package, we need to make sure we're in a virtual environment (`venv`). A virtual environment is a self-contained directory tree that contains a Python installation and a number of additional packages. This helps to keep dependencies required by different projects separate by creating isolated spaces for them. In this case, it can be understood if the `(test_env)` is displayed after the shell prompt.

Install via tar archive:

```
tar tvzf dist/boxbrainiac-0.1.0.tar.gz # Check the content
pip3 install dist/boxbrainiac-0.1.0.tar.gz # Install
```

Install via `*.whl` file:

```
pip3 install dist/boxbrainiac-0.1.0-py3-none-any.whl
```

Check if it is installed:

```
pip3 list|grep boxbrainiac
boxbrainiac (0.1.0)
```

Uninstall:

```
pip3 uninstall boxbrainiac
```

4.1.4 Conclusion

By following this guide, you can effectively test your Python projects and ensure that they are easy to distribute and install. Remember, always use a virtual environment when working with Python projects to avoid many common problems related to project dependencies. For more advanced uses and best practices regarding Python packaging, see the official Python Packaging User Guide https://packaging.python.org/en/latest/.

5 History

Version	Date	Notes
0.1.4	2024-06-17	Change verbatim environment for trees (PDF)
0.1.3	2023-05-16	Fix some errors like names or python3, pip3
0.1.2	2023-05-10	Improve writing, Add Test a Python3 Package With …
0.1.1	2022-05-26	+History, Change shell to bash
0.1.0	2020-05-16	Initial release

6 Disclaimer of Warranty

7 Limitation of Liability