SciWork 2020 Packaging Tutorial

Mar 03, 2020

Contents

This tutorial walks you through how to package a simple Python project. It will show you how to add the necessary files and structure to create the package, how to build the package, and how to upload it to a package index (such as PyPI).

CHAPTER 1

Set up a Development Environment

Please have the followings ready before you start:

- Any Python installation with pip available.
- Conda from either Miniconda or Anaconda.

Please arrive early to the tutorial if you are not sure whether you have all things set up. The lecturer will help get things ready.

1.1 Install development tools

Run the following command to install/update the needed development tools.

Anaconda:

```
conda install conda-build
```

On Windows:

```
py -3 -m pip install --upgrade pip setuptools wheel twine
```

On Mac or Linux:

```
python3 -m pip install --upgrade pip setuptools wheel twine
```

Note: For brevity, we will use `py -3` from here on. Please substitute it to `python3` yourself as needed, if you used `python3` for the above command.

1.2 Register an account on Test PyPI

Register an account at https://test.pypi.org/account/register/.

Follow the steps to complete registration. You will also need to **verify your email address** to be able to upload packages.

1.3 Put code in a directory

This tutorial uses a simple project named `sampleproject`. The complete source is available at https://github.com/pypa/sampleproject.git. We will be creating the project from scratch, but you can use the repository as a reference if anything is not working.

If you already have a project that you want to package up, *and feel confident* to do so, simply replace the `sample` directory with the one you want to use, and replace the metadata as appropriate.

We start with the following file structure:

```
sampleproject/
    sample/
        __init__.py
```

All of the commands in this tutorial will need to be run within the top-level directory just created. Be sure to `cd sampleproject` into the project directory to run following commands successfully.

Note: Put some code in `__init__.py` so it is easy to test. For example:

```
def greet():
    print("Hello!")
```

And test it like this:

```
$ py -3 -c "import sample; sample.greet()"
Hello!
```

CHAPTER 2

Describe the project

Create some additional files in the project root directory, alongside the `sample` directory. You should end up with the following structure:

```
sampleproject-YOUR-USERNAME/
    sample/
        __init__.py
    pyproject.toml
    README.md
    setup.cfg
    setup.py
```

2.1 Creating README.md

A README file is essential for people (including yourself in the future) to understand what the project is doing. You can put some text inside this file to describe the project, using the GitHub-flavored Markdown syntax. Put in the following content if you are not sure what to write:

```
# Example Package

This is a simple example package. You can use
GitHub-flavored Markdown to write your content.
```

2.2 Creating pyproject.toml

Add the following content:

This file describes how we want to package the project. We will be using Setuptools, a packaging tool maintained by the Python Packaging Authority (PyPA).

2.3 Creating setup.cfg

This file tells Setuptools about the project (such as the name and version), as well as which code files to include.

Change the `name` value to include your username (for example, `sampleproject-uranusjr`), to make the proejct name unique, and does not conflict with other people when you upload it. Or you can use any name you like (as long as it is not already registered—search on the PyPI to find out).

```
[metadata]
name = sampleproject-YOUR-USERNAME
version = 0.0.1
author = Example Author
author_email = author@example.com
description = A small example package
long_description = file: README.md
long_description_content_type = text/markdown
url = https://github.com/pypa/sampleproject
license = MIT
classifier =
    Programming Language :: Python :: 3
    License :: OSI Approved :: MIT License
    Operating System :: OS Independent

[options]
packages = find:
python_requires = >=3.6
```

Here is a run-down of the configuration:

- `name` is the distribution name of your package. This can be any name as long as only contains letters, numbers, `_`, and `-`. It also must not already be taken on the PyPI. **Be sure to update this with your username**, as this ensures you won't try to upload a package with the same name as one which already exists when you upload the package.
- `version` is the package version. See PEP 440 for more details on versions.
- `author` and `author_email` are used to identify the author of the package.
- `description` is a short, one-sentence summary of the package.
- `long_description` is a detailed description of the package. This is shown on the package detail package on the PyPI. Here, we tell Setuptools to use the content of `README.md`, which is a common pattern.
- `long_description_content_type` tells the index what type of markup is used for the long description. In this case, it's Markdown.
- `url` is the URL for the homepage of the project. For many projects, this will just be a link to GitHub, GitLab, Bitbucket, or similar code hosting service.
- `license` describes what license this project uses—Here we use MIT, but there are many more choices available. Omit this field if you do not intend your code to be open source.
- `classifiers` gives the index and pip some additional metadata about your package. In this case, the package described as
  - Only compatible with Python 3
  - Uses the MIT license
  - Does not care about operating systems

  A complete list of classifiers can be found at https://pypi.org/classifiers/.

- `packages` is a list of all Python code that should be included in the *distribution package*. Instead of listing each package manually, we can use the *find derivative* to automatically discover all packages and subpackages. This is also a common pattern unless you have an uncommon project layout.
- `python_requires` descibes what versions of Python this project is compatible with. Here, we only allow the project to be installed on Python 3.6 or later.

The keys listed above is a relatively minimal set, and there a a few more you can specify. Visit the documentation on setup.cfg to find a comprehensive list of available configurations.

2.4 Creating setup.py

`setup.py` is a script to call Setuptools. It can be used to include custom logic to build the project, but we are using all defaults here. Simply put:

```
import setuptools
setuptools.setup()
```

CHAPTER 3

Distribute the Project

With the metadata declared, we can now ask Setuptools to generate *distribution packages*, which are archives containing files to be installed on the target machine.

3.1 Build the distributions

Use the following command to build distributions:

```
py -3 setup.py sdist bdist_wheel
```

This command should output a lot of text. Once completed, it should generate two files in the dist directory:

```
dist/
    sampleproject_YOUR_USERNAME-0.0.1-py3-non3-any.whl
    sampleproject-YOUR-USERNAME-0.0.1.tar.gz
```

The `tar.gz` file is a *source distribution* (sdist), and the `.whl` file is a *built distribution* in the *wheel* format. Newer pip versions prefer to install built distributions, but will fall back to sdist if there are no built distributions compatible with the user's platform. In this case, our example package is compatible with Python on any platform, so only one built distribution is needed.

Warning: If you made mistakes in your `setup.cfg` and want to change it, make sure to *delete previously-generated metadata* before running `setup.py` again.

On Windows:

```
rmdir /q /s build dist
del sampleproject_YOUR_USERNAME.egg-info
```

On macOS and Linux:

```
rm -rf build dist sampleproject_YOUR_USERNAME.egg-info
```

3.2 Upload the distributions

Now we can upload the files to the package index. For demostration purposes, we will be uploading to *Test PyPI*. This is a separate instance of the package index, intended for testing and experimentation. This is great for things like this tutorial, where we don't necessarily want to upload "for real."

Make sure to *Register an account on Test PyPI* before you continue with the tutorial.

Use the following command to upload the distributions built in the previous section:

```
py -3 -m twine upload --repository-url https://test.pypi.org/legacy/ dist/*
```

Enter your username and password when prompted.

Once the command finished successfully, your package should be viewable on Test PyPI at e.g. https://test.pypi.org/project/sampleproject-YOUR-USERNAME.

Congratulations, you have successfully packaged and distributed a Python project!

3.3 Test package installation

The package uploaded to Test PyPI can be installed with pip, like you would any other package, by explicitly specifying the "index" to install from:

```
py -3 -m pip install --index-url https://test.pypi.org/simple sampleproject-YOUR-
↪USERNAME
```

Now try to use it by running `py -3`:

```
>>> import sample
>>> sample.greet()
Hello!
```

3.4 Optional: Release your package to PyPI.org

By uploading the package to PyPI.org (instead of Test PyPI), you are telling the world the package is ready for download. The steps are simple—Register an accout on PyPI.org, build the distributions like before, and change the repository URL in the upload command:

```
py -3 -m twine upload --repository-url https://pypi.org/legacy/ dist/*
```

The only difference is to use `pypi.org` (removing the `test.` part). That's it! Now people can install your package directly:

```
py -3 -m pip install sampleproject-YOUR-USERNAME
```

There are some caveats though:

- Files uploaded to PyPI are immutable. More especifically, although deletion is possible, your cannot *re-upload* a file. So be extra careful before you release to PyPI. The only way to override a mistake on PyPI is to release a new version.
- Following the previous point, it is usually a good idea to delete the `build`, `dist`, and `.egg-info` directories every time you want to release a new version, to make sure the files are all in the newest version.

- Accounts and packages on Test PyPI and (real) PyPI are all distinct. You do not automatically own a package on PyPI by uploading it to Test PyPI, and vice versa.

CHAPTER 4

Distribute a Conda Package

As with Setuptools, we also need to provide both some *description* and *build instruction* for a Conda package. More specifically, three files are needed:

- `meta.yaml`
- `build.sh`
- `blt.bat`

Put them in the project root (alongside with other metadata files such as `README.md`, `pyproject.toml`).

4.1 Creating meta.yaml

This is used to describe the package.

```
package:
  name: sampleproject-YOUR-USERNAME
  version: "0.0.1"
about:
  home: https://github.com/pypa/sampleproject
  license: MIT
  summary: A small example package
source:
  fn: sampleproject-YOUR-USERNAME-0.0.1.tar.gz
  url: https://test.pypi.org/packages/source/s/sampleproject-YOUR-USERNAME/
↪sampleproject-YOUR-USERNAME-0.0.1.tar.gz
  md5: "39180d64b5021c5399a2568fe2103cd2"

requirements:
  build:
    - pip >=19
    - python >=3.6
  run:
    - python >=3.6
```

Note: Remember to replace `YOUR-USERNAME` with your own name.

Warning: Make sure to quote the `version` and `md5` fields! Otherwise YAML may interpret them as numbers. Also, the extension is `yaml`, not `yml`.

Here, we instruct Conda about basic project information (`package` and `about` sections), where to download it (`source`), and what other packages are needed in order to build and run it.

Unlike PyPI, Conda does not allow uploading source code (only built files), so you need to upload your code somewhere else, and use `meta.yaml` to tell Conda where it is.

Here, we are re-using our data on Test PyPI (or the real PyPI, if you eventually release the package to the public). The package URL is predictable, and based on your package name (the first `/s/` part is the package name's first letter).

The MD5 value is used by Conda to make sure it downloads the correct file. It can be calculated using a local command.

On Windows:

```
fciv -md5 dist\sampleproject-YOUR-USERNAME-0.0.1.tar.gz
```

On macOS:

```
md5 dist/sampleproject-YOUR-USERNAME-0.0.1.tar.gz
```

On Linux:

```
md5sum dist/sampleproject-YOUR-USERNAME-0.0.1.tar.gz
```

Alternatively, you can also view this on your PyPI project page at https://test.pypi.org/project/sampleproject-YOUR-USERNAME/#files. Click on the **View** button beside the `.tar.gz` file, and copy the hash digest field on the **MD5** row.

4.2 Creating build scripts

Conda packages use two files to build: `build.sh` and `bld.bat`. These two are basically the same thing, but one for installing on Windows, and one for on Linux and macOS. It is highly recommended you provide *both* if possible (unless you really don't want the package to be installed on a platform).

`build.sh`:

```
"$PYTHON" -m pip install --no-deps .
```

`bld.bat`:

```
"%PYTHON%" -m pip install --no-deps .
exit %errorlevel%
```

4.3 Building the package

In the project root (where `meta.yaml` is), use `conda build` to generate a Conda package:

```
conda build .
```

This will generates a lot of text. When it finished, you should see something like the following near the end:

```
TEST START: ~/miniconda/conda-bld/linux-64/sampleproject-YOUR-USERNAME-0.0.1-py38_0.
↪tar.bz2
Nothing to test for: ~/miniconda/conda-bld/linux-64/sampleproject-YOUR-USERNAME-0.0.1-
↪py38_0.tar.bz2
# Automatic uploading is disabled
# If you want to upload package(s) to anaconda.org later, type:

anaconda upload ~/miniconda/conda-bld/linux-64/sampleproject-YOUR-USERNAME-0.0.1-py38_
↪0.tar.bz2
```

This means that the package has been built successfully. The `TEST START` and `anaconda upload` lines also shows the location of the package; in this case it is:

```
~/miniconda/conda-bld/linux-64/sampleproject-YOUR-USERNAME-0.0.1-py38_0.tar.bz2
```

Save this path somewhere; it will be useful in the following (optional) sections.

4.4 Testing the Conda package

You can try installing the newly built package into your Conda environment:

```
conda install --use-local sampleproject-YOUR-USERNAME
```

The `--use-local` flag instructs Conda to install the local conda-build channel on your computer, rather than the default Anaconda or conda-forge.

If installed successfully, the package will be present in `conda list`.

4.5 Optional: Building for a different Python version

The package built above was against Python 3.8 (the file name contains a `py38` part). Conda by default builds packages for the version of Python installed in the root environment. To build packages for other versions of Python, use the `--python` flag followed by a version:

```
conda build --python 3.6 sampleproject-YOUR-USERNAME
```

Notice that the file printed at the end of the output would change to reflect the requested version of Python. When you `conda install`, it will look in the package directory for the file that matches your current Python version.

4.6 Optional: Converting a package for use on all platforms

Similar to the Python version, Conda by default requires you to build the package on each platform, to make sure the uploaded files are always correctly built. We built against `linux-64` (Linux, 64-bit) in the above example. Sometimes, however, you are *very* sure the package works on all platforms. Conda can convert the package for you in this case:

```
conda convert --platform all ~/miniconda/conda-bld/linux-64/sampleproject-YOUR-
↪USERNAME-0.0.1-py38_0.tar.bz2
```

Replace the final path with your package location saved above.

4.7 Optional: Uploading to Anaconda.org

After converting your files for use on other platforms, you may choose to upload your files to Anaconda.org. You will need to register an account (sign up) on the website first.

Make sure you have the Anaconda Client installed:

```
conda install anaconda-client
```

And log in (sign in) into your registered Anaconda account:

```
anaconda login
```

After that, you can use the command to upload packages:

```
anaconda upload ~/miniconda/conda-bld/linux-64/sampleproject-YOUR-USERNAME-0.0.1-py38_
↪0.tar.bz2
```

Replace the final path with your package location saved above. Packages built against another Python, or converted by `conda convert` can also be uploaded in the same way, to make your package support multiple Python versions and platforms.

CHAPTER 5

Distribute Private Projects

We have been talking about how to release your package to the public, and it is a common misconception that package publishing is only for open source.

While it is true that Python package managers are more focused on working with open source (because they are part of it), they are also used by a lot of people to publish packages in private settings, so various projects in a company can easily share code, deploy software, and manage version upgrades.

5.1 Custom Conda channels

In Conda, you can choose to install packages from a specific *channel*, which means Conda will look at that location for the package. Anaconda for Enterprise offers a private channel service (i.e. publish packages that only authorised people can download), but it is also easy to run a channel server yourself, either with HTTP(S), or even with a shared disk that's accessible with `file://`.

A Conda channel is structured like this:

```
channel/
   linux-64/
       ... packages
   linux-32/
       ... packages
   osx-64/
       ... packages
   win-64/
       ... packages
   win-32/
       ... packages
```

The root directory can be any directory on disk, and can take any name, but here we use `channel` as an example. Each directory inside root represents a platform to support; you can omit any you do not plan to support. Package files (like our previous `sampleproject-YOUR-USERNAME-0.0.1-py38_0.tar.bz2`) are put inside each directory to make them downloadable.

Run the following command to generate a repository listing:

```
conda index channel/
```

This will generate a file `repodata.json` in each repository directory, which Conda uses to get the metadata for the packages in the channel.

Note: Re-run `conda index` each time you add or modify anything in the channel (e.g. release a new package version), to keep the metadata up-to-date.

To install from the custom index, use the following syntax:

```
conda install --channel=file://path/to/channel/ sampleproject-YOUR-USERNAME
```

You can also set up the channel in your `.condarc` to avoid specifying the channel every time.

5.2 "Find-links" pages

TODO...

5.3 devpi

TODO...

CHAPTER 6

Further Guides

TODO...

CHAPTER 7

Indices and tables

- genindex
- modindex
- search