Python Packaging for Beginners

(Presentation & Reference)

David J. Lambert
www.LinkedIn.com/in/DavidJLambert

BayPIGgies, March 22, 2018

Posted at https://github.com/David-J-Lambert/PyPkg

Giving Your Code to Someone (1/2)

The Hard Way (for them):

1. Throw N python files over cube wall
2. “Lunch time, gotta run!”

No README or instructions

No idea if other libraries required

What versions of other libraries?

Giving Your Code to Someone (2/2)

The Easy Way (for them):

One file containing everything

This file's type is widely known,

with widely known methods

to install code.

File contains all needed information

That is packaging.

Package ≠ Executable ≠ Script

Package:

- File to transport code
- Wheel file

Script:

- `print("Hello World!")`

Executable:

- Dropbox
- bittorrent client

Other Things Called Package, Confusing!

"Distribution" package: wheel, egg ⟸ THIS ONE

NOT THESE

"Import" package: `pickle`, as in `import pickle`

Linux Distro Package: `rpm`, `yum`, `apt`

But Linux distro packages can install Python programs

Two Types of Distribution Packages

Source Distribution:
includes source code

Built Distribution:
binaries, no source code

Quick History of Python Packaging

In the beginning, no distribution packages.

Had to use “The Hard Way”

Had to know Python directory structure

Present-day directory structure (2.x, 3.x) …

Files & Directories in **Windows**

Windows File/Directory	
Python	`%Home%\python.exe`
Pip	`%Home%\Scripts\pip3.exe`
Python Standard Library	`%Home%\Lib\`
Installed Packages	`%Home%\Lib\site-packages\`

Installed By	%Home%
Admin	`C:\Program Files\Python36\` `C:\Program Files (x86)\Python36-32\`
Non-Admin	`C:\Users\<User>\AppData\Local\Programs\Python36\` `C:\Users\<User>\AppData\Local\Programs\Python36-32\`

Folders end with "\"

Files & Directories in **Linux & OSX**

Linux & OSX File/Directory		
Python		`$Home/bin/python3.6`
Pip		`$Home/bin/pip3`
Python Standard Library		`$Home/lib[64]/python3.6/`
Installed Packages	OSX & most Linux	`$Home/lib[64]/python3.6/site-packages/`
	Debian-based linux distros (like Ubuntu)	`$Home/lib/python3.6/dist-packages/`

OS	$Home
All Linux (& Debian)	`/usr/`
OSX	`/Library/Frameworks/Python.framework/Versions/3.6/`

Debian-based Linux Distros: More Differences

Installed By	Location
OS App Store	`/usr/lib/python3.6/dist-packages/`
Root	`/usr/local/lib/python3.6/dist-packages/`
User	`/home/User-Name/.local/lib/python3.6/site-packages/`

Used 3.6 as example, good for all versions

Environment Variables (all OS's)

Internally, Python tries to figure out locations of modules (installed & Standard Library).

To help it out:

Variable	What is it?
`PYTHONPATH`	*Augments* path for resolving module references, with separators = **`os.pathsep`** ("`:`" in Linux, "`;`" in Windows)
`PYTHONHOME`	*Changes* location of the Python Standard Library

But *Everything* Was Done ***By Hand***

Construct `PYTHONPATH`

Switch `PYTHONPATH` when switching projects (no virtual environments)

Determine dependencies among modules

Find (& track changes to) 3rd party modules

Manually track version numbers

Packaging Can Do All That, Plus…

Auto Download & Install Missing Libraries

Check version number automatically

Metadata (version #, author, URL, etc.)

We have central, maintained package store (PyPI)

First Shot at Packaging

`distutils` & **`setup.py`**, low-level libraries (1.6)

Python Package Index (PyPI) **`https://pypi.python.org`**

On top of distutils, add (2.4?)

- **`setuptools`**: dependencies, metadata
- "egg" binary format
- **`easy_install`**: main utility for packaging

Problems & missing features, so…

Fixes by Individuals & Ad-Hoc Groups

Low Level Libraries:

- **`Setuptools`**, **`Distribute`**, **`Distutils2`**

Virtual Environments:

- **`venv`**, **`pyenv`**, **`virtualenv`**, **`virtualenvwrapper`**

`Buildout`

Confusion! Design problems! Clashing egos!

Packaging Today (1/3) The Big Slide!

Packaging managed by

- Python Packaging Authority (PyPA), www.pypa.io
- Python Packaging User Guide

  https://packaging.python.org

PyCon 2017 "Python packaging without complication":

www.youtube.com/watch?v=qOH-h-EKKac

Warning: obsolete info about packaging everywhere!

Packaging Today (2/3)

`pip`, not **`easy_install`**

Wheel file (**`*.whl`**), not egg

Using **`setuptools`** & **`setup.py`**

`Distutils` being absorbed into **`setuptools`**

Packaging Today (3/3)

3 Types of wheels:

- "Universal": python only, for both 2.x ***AND*** 3.x
- "Pure Python": python only, for 2.x ***XOR*** 3.x
- "Platform": extension written in C
  (one wheel file per supported platform,
  for one python version)

Use **`sys.path`**, Not **`PYTHONPATH`** (1.6+)

`site` module constructs **`sys.path`**

More info in **`site`** module docs

Using pip (1/5)

pip reference: https://pip.pypa.io/en/stable/

Verify that python & pip work:

```
python --version
pip --version
```

If pip is not installed:

```
python -m ensurepip --default-pip
```

Using pip (2/5)

Install latest version of **`projectX`**

```
pip install projectX
```

Install specific version of **`projectX`**

```
pip install 'projectX==5.6'
```

Upgrade to current version

```
pip install --upgrade projectX
```

Using pip (3/5)

Can install from URL that's not PyPI

Install from local source

```
pip install /path/to/source
```

Download without installing

```
pip download projectX
```

Uninstall

```
pip uninstall projectX
```

Using pip (4/5)

See list of installed packages (not Std Lib)

```
pip list
```

Show info about a particular package

```
pip show projectX
```

Search PyPI for packages whose name or summary contains ‘kumquat’

```
pip search ‘kumquat’
```

Using pip (5/5)

Install a list of packages:

```
pip install -r requirements.txt
```

Generate a wheel file

Virtual Environments (1/5)

You have a working project
with globally defined module X (version A)

Add project & incompatible version B of X

First project now references Version B

You just broke the first project

Virtual Environments (2/5)

Virtual Environments completely isolate projects

Each project has it’s own:

- root directory
- python version
- Standard & installed libraries

Virtual Environments (3/5)

`virtualenv` is first package to do virtual environs

Not in Standard Library, owned by PyPA
Works with Python 2.x & 3.x

https://virtualenv.pypa.io/en/stable/

`virtualenvwrapper` is a set of extensions to `virtualenv`

Virtual Environments (4/5)

`venv` also creates virtual environments

Modeled after **`virtualenv`**

Added to Standard Library in 3.3, won't work in 2.x

`https://docs.python.org`

`pyvenv` is a wrapper around **`venv`**

It's in the Standard Library, deprecated in Python 3.6

Virtual Environments (5/5)

Use **`pyenv`** to handle multiple versions of Python

`pipenv` merges **`pip`** & **`virtualenv`**

Author uses PyDev, not virtual environments

Rest of talk, assume we're working with 1 project

Package Source Layout

```
data/   ———— data_file
LICENSE.txt
MANIFEST.in
package/ ——┬— __init__.py
           ├— core-code.py
           └— helper-code.py
README.rst
setup.cfg
setup.py
test/  ——————┬— __init__.py
             └— test_package.py
```

The core code

Give package name to code directory

```
data/   ———————— data_file
LICENSE.txt
MANIFEST.in
package/ ———┬— __init__.py
            ├— core-code.py
            └— helper-code.py

README.rst
setup.cfg
setup.py
test/  ————————┬— __init__.py
               └— test_package.py
```

The core code

Alternative:

Combine all source code into 1 file

```
data/   ———— data_file
LICENSE.txt
MANIFEST.in
package.py
README.rst
setup.cfg
setup.py
test/  –———┬— __init__.py
           └— test_package.py
```

Test code

Unit tests of your package

Tests correctness of customizations

```
data/   ———— data_file
LICENSE.txt
MANIFEST.in
package/ ——┬— __init__.py
           ├— core-code.py
           └— helper-code.py
README.rst
setup.cfg
setup.py
test/  ———┬— __init__.py
          └— test_package.py
```

Data file(s)

Any data needed by your package

Perhaps test or sample data

```
data/   ———————— data_file
LICENSE.txt
MANIFEST.in
package/ ———┬— __init__.py
            ├— core-code.py
            └— helper-code.py
README.rst
setup.cfg
setup.py
test/  ————————┬— __init__.py
               └— test_package.py
```

LICENSE.txt

What can others do with your code? MIT, GPL, etc.

https://opensource.org/licenses
https://choosealicense.com

```
data/   ———————— data_file
LICENSE.txt
MANIFEST.in
package/ ———┬— __init__.py
            ├— core-code.py
            └— helper-code.py
README.rst
setup.cfg
setup.py
test/  ————————┬— __init__.py
               └— test_package.py
```

MANIFEST.in

To get non-coding files into the package

```
data/   ———— data_file
LICENSE
MANIFEST.in
package/ ——┬— __init__.py
           ├— core-code.py
           └— helper-code.py
README.rst
setup.cfg
setup.py
test/  ————┬— __init__.py
           └— test_package.py
```

Example of MANIFEST.in

```
include AUTHORS.rst
include CONTRIBUTING.rst
include HISTORY.rst
include LICENSE
include README.rst

recursive-include tests *
recursive-exclude * __pycache__
recursive-exclude * *.py[co]

recursive-include docs *.rst conf.py Makefile make.bat *.jpg *.png *.gif
```

<u>README.rst</u>

The project's goal

How to run it

How to unit test

Key dependencies

Future enhancements

```
data/   ———————— data_file
LICENSE
MANIFEST.in
package/ ———┬— __init__.py
            ├— core-code.py
            └— helper-code.py

README.rst
setup.cfg
setup.py
test/  ————————┬— __init__.py
               └— test_package.py
```

setup.py

Command line program to do packaging

Configuration of your project

Uses function setup(),
with keyword args, sample:

```
name='my_package',
version='1.2.0',
url='https://github.com/…',
author='Bullwinkle',
author_email='yada@ya.da',
```

```
data/   ———— data_file
LICENSE
MANIFEST.in
package/ ——┬— __init__.py
           ├— core-code.py
           └— helper-code.py
README.rst
setup.cfg
setup.py
test/  ————┬— __init__.py
           └— test_package.py
```

setup.cfg

Ini file with option defaults for setup.py commands

```
data/   ———— data_file
LICENSE
MANIFEST.in
package/ ——┬— __init__.py
           ├— core-code.py
           └— helper-code.py
README.rst
setup.cfg
setup.py
test/  ———┬— __init__.py
          └— test_package.py
```

Can Have Other Files In Project Root

CHANGELOG

CONTRIBUTING

AUTHORS

requirements.txt

.gitignore

Creating a Package (1/6)

Create directory tree from scratch

Or can use **`cookiecutter`** (PyPI)
(On Windows only runs in bash shell)

Add source code & tests

Customize the non-code files

Creating a Package (2/6)

Run your test

Create account for source code host

Make client repository for source code host (github, bitbucket, etc.): **`git init`**

`git push`

Creating a Package (3/6)

Create PyPI account (`https://pypi.python.org`)

Practice on `https://testpypi.python.org`

Save PyPI settings to `.pypirc`

(don't put in package, contains your login)

Creating a Package (4/6)

Check if **`wheel`** installed

Install the package for uploading to PyPI
`pip install twine`

Make desired package type (& put in **`dist/`**)

- Make source package
  `python setup.py sdist`

Creating a Package (5/6)

- Make wheel package:
  `python setup.py bdist_wheel`

- Make binary package:

`python setup.py bdist`

- On Windows, makes an executable
- On Linux, makes linux package (rpm, deb, etc.)
- Self-contained? Need Python externally?

Creating a Package (6/6)

Upload to PyPI:
`twine upload dist/*`

Make sure it all works:
`pip install package-name`

Option to post documentation to readthedocs.io

Make Executables with Freezing

Freezing utils: turns package source into executable
Varying support for 2.x or 3.x on Linux, Windows, or OSX

- PyInstaller, py2exe, Freeze, cx_freeze, …

All executables contain a Python interpreter,
no external one needed

https://wiki.python.org/moin/DistributionUtilities

Python Distributions Besides CPython

Starting in IronPython 2.7.5, **`pip`** can be used.

In Anaconda & Jython: you can install from PyPI with **`pip`**

But Jython also has **`jip`**, and Anaconda also has **`conda`**

See distribution docs to see when to use **`pip`**
& when to use **`jip`** or **`conda`**.

Commercial Interruption

I'm looking for entry-level
Python programming job.

Long experience in software support,
especially with databases.

2 Quarters Python class
4 Quarters Java class (3 Intro to C.S.)

www.LinkedIn.com/in/DavidJLambert

Questions?

Talk available at

https://github.com/David-J-Lambert/PyPkg

Thank you, Les Faby, for your feedback.

Feel free to look at

www.LinkedIn.com/in/DavidJLambert