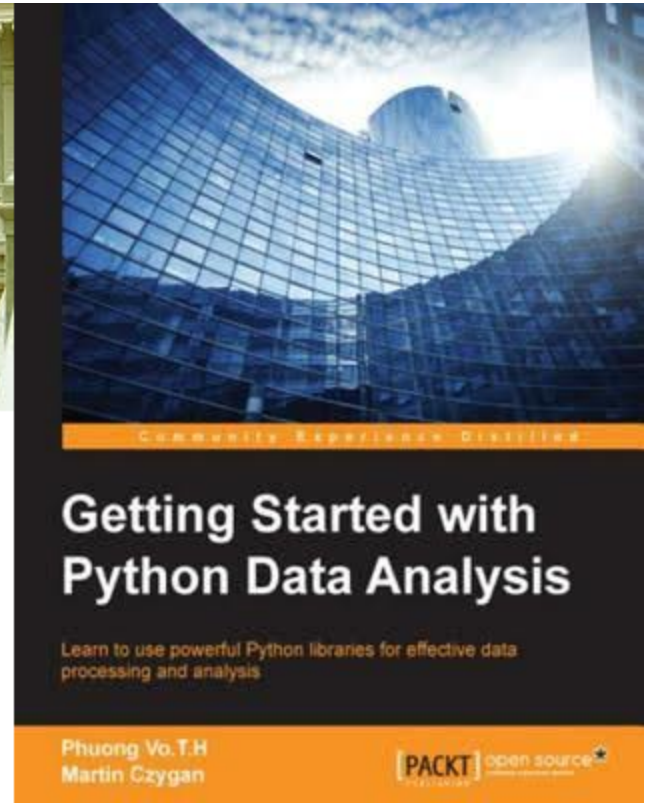
Packaging Python Applications

PyConBalkan, Belgrade, 2018-11-17 16:00-16:30

Martin Czygan / github.com/miku / @cvvfj

About me

- Software developer at Leipzig University Library
- Part-time consultant, co-author *Getting Started with Python Data Analysis* (2015)
- Maintainer of a few open source tools



About me

- interest in (build) automation: code, writing, data
- anecdata: ant, ephemeral VMs, tried to ease adoption of Python at workplace

There is some satisfaction in being able to just run *one command*.
There are also some trade-offs.

Packaging

- loosely defined as *approaches* and *tools* to create usable (installable, shippable) software
- there is this thing called: *The packaging gradient*

Packaging is moderately exiting

- not directly related to features
- many options

> ... plethora of packaging options ...
> (https://packaging.python.org/overview/)

- less formalized, project-dependent

So why care about packaging?

- as individual or team
- as company

Individuals or teams

- to share code
- collaborate or invite contributions

> It might seem strange to think about packaging before writing code, but this process does wonders for avoiding future headaches. (https://packaging.python.org/overview/)

As a company

- Aug 9, 2000: The Joel Test: 12 Steps to Better Code

There are two items related to packaging:

- #2 Can you make a build in one step?
- #3 Do you make daily builds?

On item #2

> By this I mean: how many steps does it take to make a shipping build from the latest source snapshot? On good teams, there's a single script you can run that does a full checkout from scratch, rebuilds every line of code, makes the EXEs, in all their various versions, languages, and #ifdef combinations, creates the installation package, and creates the final media — CDROM layout, download website, whatever.

On item #2

> If the process takes any more than one step, it is prone to errors. And when you get closer to shipping, you want to have a very fast cycle of fixing the "last" bug, making the final EXEs, etc. If it takes 20 steps to compile the code, run the installation builder, etc., you're going to go crazy and you're going to make silly mistakes.

Packaging is only a part of a larger story

- the decade of devops (2009-)
- code--build--test--package--release--configure--monitor

There slides reflect only a few small pieces of the puzzle.

The Packaging Gradient

There is an nice talk called *The Packaging Gradient* by Mahmoud Hashemi at PyBay 2017 (YT:601), BayPiggies2017 (YT:82) - blog.

> One lesson threaded throughout Enterprise Software with Python is that deployment is not the last step of development.

What will we look at?

- "just do nothing"
- modules, packages, distributions, PyPI
- single file deployments (PEP 441)
- reusing linux distribution infrastructure
- images and containers
- frozen software

A single Python file (module)

- with a large standard library, it is possible to write useful things in Python and stdlib only
- deployment cannot get simpler that `scp` or `curl`

```
# scp script.py ...
```

Requirements:

- ssh
- python on target machine (matching version)
- script should have no dependencies

Beautiful, if possible (*simple is better than complex*).

Module, package, distribution

- a module is as single, importable python file
- a package is a directory (containing an `__init__.py`)
- a distribution is a way to bundle zero or more packages (source and built distribution)

A minimal setup.py

Writing a setup.py file can be simple.

> The smallest python project is two files. A setup.py file which describes the metadata about your project, and a file containing Python code to implement the functionality of your project.

However, there are only three required fields: name, version, and packages.

> The name field must be unique if you wish to publish your package on the Python Package Index (PyPI). The version field keeps track of different releases of the project. The packages field describes where you've put the Python source code within your project.

A minimal setup.py

```
$ tree
.
├── hellopkg
│   ├── __init__.py
│   └── hello.py
└── setup.py
```

```
$ cat setup.py
from setuptools import setup

setup(name='hellopkg',
      version='0.1.0',
      packages=['hellopkg'])
```

Additional fields in setup.py

Usually, you project will have dependencies and it might come with command line programs:

```
from setuptools import setup

setup(name='hellopkg',
      version='0.1.0',
      packages=['hellopkg'],
      install_requires=['requests'],
      entry_points={
          'console_scripts': [
                'hellopkg-cli=hellopkg.hello:hello'
          ],
      })
```

Building a source distribution

```
$ python setup.py sdist
...
Writing hellopkg-0.1.0/setup.cfg
creating dist
Creating tar archive
...

$ tree .
.
├── dist
│   └── hellopkg-0.1.0.tar.gz
├── hellopkg
│   ├── __init__.py
│   └── hello.py
└── setup.py
```