Managing Python Package Dependencies

Constant problems are not a requirement

Jeremy Bowman
jbowman@edx.org

About Me

Jeremy Bowman

Principal Software Engineer at edX

Tools Team

It looks so simple!

setup.py:

```
...
install_requires = ['django', 'pytz']
...
```

But...

setup.cfg:

```
install_requires =
  django
  pytz
```

requirements.txt:

```
django>=1.11<2.0
pytz==2017.2
```

...It can get rather complicated

Subset of ipython's setup.py:

```
extras_require = dict(
    parallel = ['ipyparallel'],
    qtconsole = ['qtconsole'],
    doc = ['Sphinx>=1.3'],
    test = ['nose>=0.10.1', 'requests', 'testpath', 'pygments', 'nbformat', 'ipykernel'
    terminal = [],
    kernel = ['ipykernel'],
    nbformat = ['nbformat'],
    notebook = ['notebook', 'ipywidgets'],
    nbconvert = ['nbconvert'],
)

install_requires = [
    'setuptools>=18.5',
    'jedi>=0.10',
    'decorator',
    'pickleshare',
    'simplegeneric>0.8',
    'traitlets>=4.2',
    'prompt_toolkit>=2.0.0,<2.1.0',
    'pygments',
```

No problem, my needs are basic

```
install_requires = ['django', 'social-auth-app-django']
```

```
Running Django-2.0.6/setup.py -q bdist_egg --dist-dir
/var/folders/td/v6rq12dx5dq4fy1fzphk2bs40000gn/T/easy_install-HAclRZ/Django-2.0.6/egg-d

==========================
Unsupported Python version
==========================

This version of Django requires Python 3.4, but you're trying to
install it on Python 2.7.
```

Right, just until I finish porting my service to Python 3

```
install_requires = ['django<2.0', 'social-auth-app-django']
```

“Hey, I’d like to use your app, but I’m running Django 2.0.6…”

“I tried using your app on Django 1.2, but it didn’t work…”

...Ok, this should do it

```
# In setup.py
install_requires = ['django>=1.8', 'social-auth-app-django']
```

```
# In requirements.txt
django>=1.8<2.0
social-auth-app-django
```

(A few weeks later, tests start failing when a new release of social-auth-app-django is released...)

This will DEFINITELY work

```
# In setup.py
install_requires = ['django>=1.8', 'social-auth-app-django<2.0.0']
```

```
# In requirements.txt
django==1.11.13
social-auth-app-django==1.2.0
```

(A month later, tests start failing when a new release of social-auth-core is released...)

ARGH! Fine, pip freeze it is.

```
# In setup.py
install_requires = ['django>=1.8', 'social-auth-app-django<2.0.0', 'social-auth-core<1.
```

```
# In requirements.txt
django==1.11.13
pytz==2016.3
social-auth-app-django==1.2.0
social-auth-core==1.6.0
...
```

(A year later, the app hasn't been tested with current releases of any of its dependencies.)

It can be easier than this

Pieces of the dependencies puzzle

- distutils
- setuptools
- Environment markers
- pip
- pip-tools or pipenv
- Your preferred task runner

distutils

- Legacy utilities for building Python packages
- Part of the standard library
- Says outright: “Use setuptools instead”
- But setuptools uses parts of it

setuptools

- Toolkit for defining and building packages
- It’s a package on PyPI
- Works with all still-supported Python versions
- Defines syntax for setup.py and setup.cfg

setup.py

```
from setuptools import setup, find_packages

setup(
    name='django-example-app',
    version='1.2',
    author='Jeremy Bowman',
    author_email='jbowman@edx.org',
    packages=find_packages(exclude=['tests']),
    include_package_data=True,
    url='https://github.com/jmbowman/django-example-app',
    description='Example Django application with typical setup.py',
    long_description='Lots more words, paragraphs even...',
    install_requires=[
        'Django',
    ],
    classifiers=[
```

Notable setup.py Characteristics

- It’s Python code, not markup
- Need to run it to parse it
- Contains some information you need to change often
- Tempts you into trying to do clever things

setup.cfg

```
[metadata]
name = django-example-app
version = file: src/django-example-app/VERSION.txt
description = Example Django application with typical setup.py
long_description = file: README.rst, CHANGELOG.rst
classifiers =
    Framework :: Django
    Programming Language :: Python :: 3
    Programming Language :: Python :: 3.5

[options]
packages = find:
install_requires =
  Django
```

Notable setup.cfg characteristics

- Recent addition to setuptools (inspired by pbr, etc.)
- Standard .ini file format
- Can be parsed without execution
- Still need setup.py, but trivially short
- Helpers for common cases (load text from file, etc.)
- Not code, so can’t handle some corner cases

setup_requires, tests_require, extras_require

- `setup_requires` - requirements for package to build
- `install_requires` - requirements for package to work
- `tests_require` - requirements for `python setup.py test`
- `extras_require` - additional requirements for optional features
- `python_requires` - Completely different; supported Python versions specifier

Environment markers

- Constrain when a dependency is required
- Can depend on Python version, operating system, Python implementation, etc.
- Follow a colon in setup.py requirements

```
"futures : python_version == '2.7'"
"pywin32>1.0 : sys.platform == 'win32'"
"unittest2>=2.0,<3.0 : python_version == '2.4' or python_version == '2.5'"
```

pip

- Utility for installing and uninstalling packages
- It's a package on PyPI
- `pip install Django`
- `pip install -r requirements.txt`

Requirements files

- Plain text
- But defined format
- Order unimportant except for the reader
- Comments allowed
- Can be generated!

pip-tools

```
# requirements/travis.in
codecov      # Code coverage reporting
tox          # Virtualenv management for tests
tox-battery  # Makes tox aware of requirements file changes
```

```
# requirements/travis.txt
#
# This file is autogenerated by pip-compile
# To update, run:
#
#    pip-compile --upgrade -o requirements/travis.txt requirements/travis.in
#
certifi==2018.4.16        # via requests
chardet==3.0.4            # via requests
codecov==2.0.15
coverage==4.5.1           # via codecov
idna==2.6                 # via requests
pluggy==0.6.0             # via tox
py==1.5.3                 # via tox
requests==2.18.4          # via codecov
six==1.11.0               # via tox
tox-battery==0.5.1
tox==3.0.0
urllib3==1.22             # via requests
virtualenv==16.0.0        # via tox
```

pip-compile and pip-sync

- Both included in the pip-tools package
- pip-compile: generate a comprehensive requirements file from a high-level requirements file
- pip-sync: install everything in the given requirements file(s), and uninstall anything not in them

pipenv

- Kind of like pip-compile + virtualenv
- Pipfile and Pipfile.lock instead of requirements files
- Very active project
- Only supports 2 sets of dependencies: regular and dev

make, invoke, paver, etc.

```
requirements: ## install development environment requirements
  pip install -qr requirements/dev.txt
  pip install -e .

upgrade: export CUSTOM_COMPILE_COMMAND=make upgrade
upgrade: ## update the pip requirements files to use the latest releases satisfying our
  pip install -qr requirements/pip-tools.txt
  # Make sure to compile files after any other files they include!
  pip-compile --upgrade -o requirements/pip-tools.txt requirements/pip-tools.in
  pip-compile --upgrade -o requirements/base.txt requirements/base.in
  pip-compile --upgrade -o requirements/django.txt requirements/django.in
  pip-compile --upgrade -o requirements/test.txt requirements/test.in
  pip-compile --upgrade -o requirements/doc.txt requirements/doc.in
  pip-compile --upgrade -o requirements/travis.txt requirements/travis.in
  pip-compile --upgrade -o requirements/dev.txt requirements/dev.in
  # Let tox control the Django version for tests
  sed '/^[dD]jango==/d' requirements/test.txt > requirements/test.tmp
  mv requirements/test.tmp requirements/test.txt
```

Identify contexts with different dependencies

- Core, test, docs, dev, CI, etc.
- Each one gets a `*.in` requirements file or a Pipfile category
- Identify your top-level dependencies for each context
- Only use version constraints when necessary
- Don’t list indirect dependencies unless there are constraints on them
- Try to use only a single requirements file per context

Context inheritance

- Don’t repeat dependencies in `*.in` files if avoidable
- Include generated `*.txt` file, not original `*.in` file
- Generate the requirements files in the correct order

```
# In test.in
-r base.txt   # Core dependencies of the service being tested
```

Auto-generate install_requires if feasible

```
def load_requirements(*requirements_paths):
    requirements = set()
    for path in requirements_paths:
        requirements.update(
            line.split('#')[0].strip() for line in open(path).readlines()
            if is_requirement(line.strip())
        )
    return list(requirements)

def is_requirement(line):
    return not (
        line == '' or
        line.startswith('-r') or
        line.startswith('#') or
        line.startswith('-e') or
        line.startswith('git+')
    )
```

```
install_requires=load_requirements('requirements/base.in'),
```

Make upgrading easy

- Have a task that handles pip-compile or pipenv for you
- Do not manually edit the generated output
- Run this task often
- Run the task on a schedule (cron, Jenkins, etc.) to generate pull requests
- Don’t let pins to old versions fester too long

For more advice - OEP-18

http://open-edx-proposals.readthedocs.io/en/latest/oep-0018-bp-python-dependencies.html

- OEP = “Open edX Proposal”
- Recently established guidelines for managing Python dependencies in Open edX projects
- Repositories are open source, feel free to use as examples or provide feedback

Thank you!

Questions?

Jeremy Bowman
jbowman@edx.org