Python, pip, setuptools, PyPI, oh my!

BV

[2014-09-03 Wed 13:11]

I started to use PyPI to distribute releases of Worch and related Python packages. This is new territory for me so this topic captures my understanding

1 Package layout

1.1 The `setup.py` file

The `setup.py` file describes how the package source area is organized. I put Python modules either at top-level or under `python/`. In the latter case one needs to add this keyword argument to the `setup()` function.

```
package_dir = {"": "python"},
```

The rest of this section gives other parts of `setup.py` which are associated with an additional file.

1.2 The `requirements.txt` file

Any dependencies are asserted in `requirements.txt`:

```
worch == 1.2.1
ups-utils == 0.1
```

and brought into `setup.py` with:

```
install_requires = [l for l in open("requirements.txt").readlines() if l.strip()],
```

Maybe there is a nicer way?....

1.3 The `MANIFEST.in` file

Files that are not picked up by "`python setup.py sdist` can be added to `MANIFEST.in`

```
include requirements.txt
include examples/*.cfg
recursive-include examples *.cfg
```

Files that should be installed need to be declared to `setup()` like:

```
data_files = [('target/path', glob('source/path/*'),]
```

Here is how to handle a recursive set of `data_files` so that the directory structure is preserved:

```
import os
from glob import glob
setup(...
      data_files = [('blah', glob('blah/*'))] + \
   [('prefix/path/'+x[0],
     map(lambda y: x[0]+'/'+y, x[2])) for x in os.walk.('thedir')]
      ...)
```

2 Package production

To make a local package:

1. Check and update the version string in `setup.py`
2. Maybe `git commit` and `git tag`
3. `python setup.py sdist`

The package is under `dist/` and can be installed with `pip install dist/pkg-X.Y.tar.gz`.

3 Dependencies

Package dependencies are encoded in to the `requirements.txt` file as above and with explicit equality relations asserted.

4 Release

1. hack and commit
2. bump version number in `setup.py`
3. commit and tag
4. `git push && git push --tags`
5. `python setup.py sdist upload`

5 Distribution

Use PyPI. Register for an account and for each new package register it:

```
$ python setup.py register
```

To upload:

```
(check and update version in setup.py)
$ git commit -a -m "...."
$ git tag X.Y.Z
$ git push --tags
$ python setup.py sdist upload
```

6 Installation

Once in PyPI, installation is trivial.

```
$ virtualenv venv
$ source venv/bin/activate
$ pip install the-package
```

7 Development

To set up for development, follow normal installation as above and then:

```
$ pip uninstall -y the-package
$ git clone git@github.com:brettviren/the-package.git
$ cd the-package
```

```
$ python setup.py sdist
$ pip install dist/the-package-X.Y.Z.tar.gz
(hack)
$ pip uninstall -y the-package
$ pip install dist/the-package-X.Y.Z.tar.gz
```

8 Managing all this

A system to manage releases would be helpful.

8.1 Problems

While incredibly nice from the point of view of the installer/user, this organization comes at a price on the developer. Some issues:

8.1.1 Tight coupling of dependency versions.

The Internet tells me that I should put equality assertions in the `requirements.txt` file. When a bug is found and fixed in a low-level package its version number is bumped and a new release is made. For an installation of a high-level package to gain this fix the low-level version number in the `requirements.txt` file needs to be updated and a new high-level package release made.

8.1.2 Developing on the stack

Setting up and maintaining a development environment requires getting several repositories in place. Hacking on their code and then installing them is error prone if done manually. A global "make install" type thing is needed to do all the sdist/pip dancing.

8.1.3 Release management

Making a release requires some actions to be coordinated:

- change version in `setup.py`
- test locally
- git commit and git tag with the version
- git push and upload to PyPI

- test from PyPi

Ways to tweak the version string in `setup.py` after a release to make it be different and indicate one is in a development phase.

8.1.4 Status

Some way to know the status of all the repositories:

- are they dirty with uncommitted/modified files?
- are they "off tag" (setup.py version says one thing, but not git tag

9 Best practices:

`http://pytest.org/latest/goodpractises.html` use virtualenv and pip, look into tox

`http://dcreager.net/2010/02/10/setuptools-git-version-numbers/` set version from git. Use `setup.cfg` file in dev branch to tack on `dev-YYYYMMDD` to version string, but do better. Use `get describe --tags --match "[0-9]*\.[0-9]*` to generate version number based on the last tag

10 UI

I want some groovy tool where I can automate some of this stuff. It might look a bit like Google `repo` in parts. I want to use it like:

```
<cli> status --cloned
 --> list cloned repositories
<cli> status --available
 --> list available repositories

<cli> repo add <name> <url> [dep1 ...]
 --> make a repository available

<cli> project add <name> [repo1 ...]
 --> add a project with zero or more repositories
<cli> project list
 --> list repositories in project
```

```
  <cli> project clone
   --> clone repositories in project
  <cli> project versions
   --> show version of each repository as per setup.py and git-describe

...
```

It needs a name....