Keeping Scientists Happy: Creating Easy to Install Scientific Python Packages.

Jonathan Helmus and Scott Collis
Argonne National Laboratory, Argonne, IL

Introduction

Modern science owes much of its success to the widespread availability of specialized instrumentation. Scientific software is another type of instrument which scientists and engineers use daily in their research and work. Just as the lack of a specific instrument can hinder the productivity of a scientist, software that is difficult to install can also limit scientific capability. Scientists are most productive when they have access to high-quality, low-cost, easy to install software.

Unfortunately, many open source Python modules are not easy to install. This leads to frustrated users who are often driven to use inferior software products. Developers and scientists creating software benefit if the end product has a simple installation process as this leads to a larger, more active user base. Increased recognition and citations of the software, additional collaborations, and an increased probability of securing future grants for continued development of the software are other possible benefits.

All-in-one Python distributions such as Enthought Canopy or Continuum Analytics' Anaconda product have made it simple for non-technical scientists to install the core Scientific Python stack (NumPy, SciPy, matplotlib, and IPython), but installing smaller, domain-specific packages is still challenging. In this poster, we will look at tools which can be used to make Python packages easier to install and manage, leading to happy productive scientists.

pip

pip is a tool for installing and managing Python packages from the Python Package Index, PyPI. PyPI is the official third-party software repository for Python which contains over 45,000 packages.

Pros

- Easy to install or included with many Python distributions.
- Included in Python 3.4 and future Python versions.
- Promoted and supported by the core Python community, including the Python Packing Authority (PyPA).
- Many packages available in PyPI, over 45,000 packages.
- Open Source (MIT).

Cons

- Limited to Python packages (no C libraries, etc.).
- Compiles all C extensions from source*. End users need a compiler.
- Requires* a build step for all packages including pure Python packages.
- Not all packages properly use pip's dependency handling.

* - Unless there are binary wheels available, see "Wheels" section

Example

Building a source package.

```
$ python setup.py sdist
running sdist
running egg_info
running build_src
...
creating dist
Creating tar archive
removing 'scikit-image-0.10.0' (and everything under it)
```

Installing a package using pip.

```
$ pip install scikit-image
Downloading/unpacking scikit-image
  Downloading scikit-image-0.10.0.tar.gz (9.1MB): 9.1MB downloaded
...
Successfully installed scikit-image
Cleaning up...
```

This poster has been created by UChicago Argonne, LLC, Operator of Argonne National Laboratory ("Argonne"). Argonne, a U.S. Department of Energy Office of Science laboratory, is operated under Contract No. DE-AC02-06CH11357. This research was supported by the Office of Biological and Environmental Research of the U.S. Department of Energy as part of the Atmospheric Radiation Measurement Climate Research Facility.

Wheels

A wheel file is a new Python distribution format which can be used to provide binary installation of Python packages. Wheel files can be "unpacked" into the appropriate directory with no compiling or build step required.

Pros

- Integrated into PyPI and pip.
- No compiling is required when installing C extensions.
- Fast installation, no build step is required.
- Reference implementation is Open Source (MIT)

Cons

- Limited to Python packages.
- PyPI only hosts Windows and Mac wheel files.
- Still a relatively new format; few projects provide wheels and many that do only provide wheels for limited number of platforms and Python versions.
- Linked third-party libraries need to be distributed in the wheel or installed in the same manner as the system which created the wheel.

Example

Building a wheel for OS X.

```
$ python setup.py bdist_wheel
running bdist_wheel
running build
...
Copying scikit_image.egg-info to build/.../wheel/scikit_image-0.10.0-py2.7.egg-info
running install_scripts
creating build/bdist.macosx-10.9-x86_64/wheel/scikit_image-0.10.0.dist-info/WHEEL
```

Installing a wheel using pip

```
$ pip install --use-wheel scikit-image
Downloading/unpacking scikit-image
  Downloading scikit_image-0.10.0-cp27-none-....whl (19.1MB): 19.1MB downloaded
Installing collected packages: scikit-image
Successfully installed scikit-image
Cleaning up...
```

Platform Specific Hints

Creating software which works on multiple operating systems is difficult but not impossible. pip, wheels, and conda have all been designed for use on Windows, OS X, and Linux. With a bit of work, it is possible to create Python packages for all three operating systems. For pure Python modules, a single universal wheel or conda package can be created which will work on all three systems. Modules with C extensions require additional packages to be created for each platform which typically must be created on that platform.

Windows

- When compiling C extensions on Windows using Visual C++, the version of Visual C++ must match the Python version being targeted..
  - Python 2.6 and 2.7 extensions use Visual Studio 2008 (VC++ 9.0).
  - Python 3.3 and 3.4 extensions require Visual Studio 2010 (VC++ 10.0).
- The MinGW toolchain can be used to compile extensions, both mingw32 and mingw-w64.
- Continuum provides a version of mingw32 in their 32-bit Windows Anaconda distribution.
- DLL dependencies must be included, the *Dependency Walker* tool can help.
- AppVeyor, a CI service, and Amazon EC2 can provide cloud hosted Windows build platforms.

OS X

- Use the C/C++ compilers provided by XCode for building extensions.
- Use *otool* and *install_name_tool* to examine and change the path of shared libraries.
- Matthew Brett's *delocate* utility can find and copy dynamic libraries into Python wheels.
  - https://github.com/matthew-brett/delocate.
- Wheel's must be renamed to be installed automatically on various Python installs.
- Travis CI's OS X build environment can be used to build packages.

Linux

- PyPI does not host wheel packages for Linux, but they can be created and installed using pip.
- Shared library requirements and paths can be examined and changed using *ldd* and *patchelf.*
- Wheels and conda packages will link extensions to the version of glibc of the build system.
  - The *nm* and *objectdump* tools can examine the glibc functions and versions.
  - Using older toolchains and glibc libraries will produce packages which work on a wider variety of hosts.

Conda

Conda is a binary package manager created by Continuum Analytics. The cross-platform tool is at the heart of the Anaconda distribution but may be used with other Python systems.

Pros

- Can install and manage any type of package including C libraries, R packages, and more.
- No compiling is required when installing packages.
- Fast installs, no build step.
- Package dependencies and conflicts are resolved.
- Possible to create independent environments (like virtualenv).
- SciPy Stack and many additional packages are provided for multiple platforms by Continuum.
- Binstar.org can be used to host user created conda packages.
- Open Source (BSD).

Cons

- Available packages are limited; some only support select platforms.
- Closely coupled with Anaconda, although installation in other Python systems is possible.
- Required creating a "conda recipe" to build a package.
- Conda packages need to be available or created for all linked dependencies or the libraries must be in the same location as in the system that built the package.

Example

Building a conda package.

```
$ conda build scikit-image/
BUILD START: scikit-image-0.10.0-np18py27_0
Fetching package metadata: ...
Solving package specifications: .
The following packages will be linked:

    package                    |            build
    ---------------------------|-----------------
    cython-0.20.2              |           py27_0   hard-link
    numpy-1.8.1                |           py27_0   hard-link
    openssl-1.0.1h             |                0   hard-link
    python-2.7.7               |                2   hard-link
    readline-6.2               |                2   hard-link
    setuptools-3.6             |           py27_0   hard-link
    six-1.7.2                  |           py27_0   hard-link
    sqlite-3.8.4.1             |                0   hard-link
    tk-8.5.15                  |                0   hard-link
    zlib-1.2.7                 |                1   hard-link

Linking packages ...
[      COMPLETE      ] |##################################################| 100%
checkout: u'v0.10.0'
Cloning into '/Users/jhelmus/anaconda/conda-bld/work'...
...
===== testing package: scikit-image-0.10.0-np18py27_0 =====
import: u'skimage'
===== scikit-image-0.10.0-np18py27_0 OK =====
TEST END: scikit-image-0.10.0-np18py27_0
# If you want to upload this package to binstar.org later, type:
#
# $ binstar upload /Users/.../scikit-image-0.10.0-np18py27_0.tar.bz2
#
# To have conda build upload to binstar automatically, use
# $ conda config --set binstar_upload yes
```

conda recipe for building scikit-image.

```
$ cat scikit-image/meta.yaml
package:
  name: scikit-image
  version: 0.10.0

source:
  git_url: git://github.com/scikit-image/scikit-image.git
  git_tag: v0.10.0

requirements:
  build:
    - python
    - numpy
    - cython
    - six
    - setuptools
  run:
    - python
    - numpy
    - scipy
    - six

test:
  imports:
    - skimage

about:
  home: http://scikit-image.org/
  license: BSD

$ cat scikit-image/build.sh
#!/bin/bash
$PYTHON setup.py install
```

Installing a package using conda

```
$ conda install scikit-image
Fetching package metadata: ..
Solving package specifications: .
Package plan for installation in environment /Users/jhelmus/anaconda/envs/test_poster:

The following packages will be linked:

    package                    |            build
    ---------------------------|-----------------
    scikit-image-0.10.0        |       np18py27_0   hard-link

Proceed ([y]/n)? y

Linking packages ...
[      COMPLETE      ] |##################################################| 100%
```

Recommendations

- Encourage users without an existing Python installation to use an all-in-one Scientific Python distribution.
- For pure Python modules, a source-only package which can be installed with pip should be supplied at a minimum. Creating universal wheel and conda packages for these modules is relatively straightforward and will make installation simpler for users.
- Python modules with C extensions but few dependencies outside of the SciPy stack are well served by wheel files.
- Projects with multiple dependencies should create conda packages for the software and required libraries.
- The best option is to provide all three: source packages, wheels for Windows and OS X, and conda packages for all three operating systems.
- Continuous integration platforms like Travis CI and AppVeyor as well as cloud based virtual servers can be used to build packages reproducibly.