Getting Started With `setuptools` and `setup.py`

`setuptools` is a rich and complex program. This tutorial will focus on the bare minimum basics you need to get setuptools running so you can:

- Register your package on pypi.
- Build egg, source, and window installer ‘distributables’.
- Upload these ‘distributables’ to pypi.

Installing setuptools and easy install

To install setuptools visit http://pypi.python.org/pypi/setuptools and follow the instructions for your operating system. Also, check out http://peak.telecommunity.com/DevCenter/EasyInstall for more instructions on how to install setup tools.

Currently (as of November, 2009), setuptools is pretty easy to install for Python version 2.3 through 2.6.

Bootstrapping setuptools

If you are having trouble setting up setuptools for your platform, you may want to check out the ‘bootstrap’ setuptools script at http://peak.telecommunity.com/dist/ez_setup.py.

You can run this like this:

```
$ python ez_setup.py
```

and it will install setuptools for whichever version of Python `python` refers to. For example on windows:

```
$ C:\Python24\python.exe ez_setup.py
```

will install a setuptools for your python24 distribution.

Setting up `setup.py`

All the features of what can go into a `setup.py` file is beyond the scope of this simple tutorial. I'll just focus on a very basic and common format needed to get this project onto pypi.

The contents of `setup.py` is just pure python:

```
import os
from setuptools import setup

# Utility function to read the README file.
# Used for the long_description.  It's nice, because now 1) we have a top level
# README file and 2) it's easier to type in the README file than to put a raw
# string in below ...
def read(fname):
    return open(os.path.join(os.path.dirname(__file__), fname)).read()

setup(
    name = "an_example_pypi_project",
    version = "0.0.4",
    author = "Andrew Carter",
    author_email = "andrewjcarter@gmail.com",
    description = ("An demonstration of how to create, document, and publish "
                                   "to the cheese shop a5 pypi.org."),
    license = "BSD",
    keywords = "example documentation tutorial",
    url = "http://packages.python.org/an_example_pypi_project",
    packages=['an_example_pypi_project', 'tests'],
    long_description=read('README'),
    classifiers=[
        "Development Status :: 3 - Alpha",
        "Topic :: Utilities",
        "License :: OSI Approved :: BSD License",
    ],
)
```

Directory Structure

The directory structure, so far, should look like this:

```
some_root_dir/
|-- README
|-- setup.py
|-- an_example_pypi_project
|   |-- __init__.py
|   |-- useful_1.py
|   |-- useful_2.py
|-- tests
|-- |-- __init__.py
|-- |-- runall.py
|-- |-- test0.py
```

README

A nice idea stolen from http://pypi.python.org/pypi/Sphinx-PyPI-upload is to include a README text file which your code. This would be visible when someone, say, cloned your repo.

Using the simple `read` function, it is easy to include this in the `long_description` keyword arg for the **`setuptools.setup()`** function.

Classifiers

A really nice website is http://pypi.python.org/pypi?%3Aaction=list_classifiers which lists all the classifiers you can use in the `setup` call.

A sample of this website is:

```
Development Status :: 1 - Planning
Development Status :: 2 - Pre-Alpha
Development Status :: 3 - Alpha
Development Status :: 4 - Beta
Development Status :: 5 - Production/Stable
Development Status :: 6 - Mature
Development Status :: 7 - Inactive
Environment :: Console
```

```
Environment :: Console :: Curses
Environment :: Console :: Framebuffer
Environment :: Console :: Newt
Environment :: Console :: svgalib
```

Using `setup.py`

The basic usage of `setup.py` is:

```
$ python setup.py <some_command> <options>
```

To see all commands type:

```
$ python setup.py --help-commands
```

And you will get:

```
Standard commands:
  build             build everything needed to install
  build_py          "build" pure Python modules (copy to build directory)
  build_ext         build C/C++ extensions (compile/link to build directory)
  build_clib        build C/C++ libraries used by Python extensions
  build_scripts     "build" scripts (copy and fixup #! line)
  clean             clean up temporary files from 'build' command
  install           install everything from build directory
  install_lib       install all Python modules (extensions and pure Python)
  install_headers   install C/C++ header files
  install_scripts   install scripts (Python or otherwise)
  install_data      install data files
  sdist             create a source distribution (tarball, zip file, etc.)
  register          register the distribution with the Python package index
  bdist             create a built (binary) distribution
  bdist_dumb        create a "dumb" built distribution
  bdist_rpm         create an RPM distribution
  bdist_wininst     create an executable installer for MS Windows
  upload            upload binary package to PyPI

Extra commands:
  rotate            delete older distributions, keeping N newest files
```

```
  develop           install package in 'development mode'
  setopt            set an option in setup.cfg or another config file
  saveopts          save supplied options to setup.cfg or other config file
  egg_info          create a distribution's .egg-info directory
  upload_sphinx     Upload Sphinx documentation to PyPI
  install_egg_info  Install an .egg-info directory for the package
  alias             define a shortcut to invoke one or more commands
  easy_install      Find/get/install Python packages
  bdist_egg         create an "egg" distribution
  test              run unit tests after in-place build
  build_sphinx      Build Sphinx documentation

usage: setup.py [global_opts] cmd1 [cmd1_opts] [cmd2 [cmd2_opts] ...]
   or: setup.py --help [cmd1 cmd2 ...]
   or: setup.py --help-commands
   or: setup.py cmd --help
```

Intermezzo: .pypirc file and gpg

In order to interact with pypi, you first need to setup an account. Go to http://pypi.python.org/pypi and click on `Register`.

Now, once registered, when you run `setup.py` commands that interact with pypi you'll have to enter your username and password each time.

To get around this, place a `.pypirc` file in your `$HOME` directory on linux. On windows, an you'll need to set a `HOME` environ var to point to the directory where this file lives.

The structure of a `.pypirc` file is pretty simple:

```
[pypirc]
servers = pypi
[server-login]
username:your_awesome_username
password:your_awesome_password
```

Note: There's probably away around having your plain text password in this file, but I don't know of the solution and haven't looked into it.

Also, you often want to `sign` the files using gpg encryption. Visit http://www.gnupg.org/ on linux or http://www.gpg4win.org/ on windows to install this software.

Registering Your Project

With your `setup.py` and `.pypirc` in place, registering your project is pretty simple. Just type:

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

I would say more, but it is just that simple.

Uploading Your Project

There are three major `setup.py` commands we will use:

- `bdist_egg`: This creates an egg file. This is what is necessary so someone can use `easy_install your_project`.
- `bdist_wininst`: This will create an `.exe` that will install your project on a windows machine.
- `sdist`: This create a raw source distribution which someone can download and run `python setup.py` directly.

> **Note:** A key point here is you need to run these commands with the version of python you want to support. We'll cover this in the Putting It All Together With The Full Windows Script below.

You can run these commands by themselves and simply create the files but not upload them. However, for this project, we always marry these commands with the `upload` directive which will both *build* and *upload* the necessary files.

Putting It All Together With The Full Windows Script

This project was build on a windows machine. To best understand how it all works and the other options used when using `setup.py` let's just look at the `.bat` file I use to build the package and upload it to pypi:

```
set HOME=C:\Users\Owner\
cd C:\eclipse\workspace\HG_AN_EXAMPLE_PYPI_PROJECT
C:\Python24\python.exe setup.py bdist_egg upload --identity="Andrew Carter" --sign --quiet
C:\Python25\python.exe setup.py bdist_egg upload --identity="Andrew Carter" --sign --quiet
C:\Python26\python.exe setup.py bdist_egg upload --identity="Andrew Carter" --sign --quiet
C:\Python24\python.exe setup.py bdist_wininst --target-version=2.4 register upload --identity="Andr
C:\Python25\python.exe setup.py bdist_wininst --target-version=2.5 register upload --identity="Andr
C:\Python26\python.exe setup.py bdist_wininst --target-version=2.6 register upload --identity="Andr
C:\Python26\python.exe setup.py sdist upload --identity="Andrew Carter" --sign
pause
```

For linux, it would be pretty much the same commands, just changing around the directories to point to the correct python versions.

Note: I use the `set HOME=C:\Users\Owner\` instead of setting an environ variable on windows