An Introduction to Distutils

Using the Distutils, one can easily distribute Python modules, concentrating on the role of developer/distributor.

Concepts & Terminology

Distutils contains some responsibilities:

- write a setup script (`setup.py` by convention)
- (optional) write a setup configuration file
- create a source distribution
- (optional) create one or more built (binary) distributions

A Simple Example

The setup script is usually quite simple., The setup script may be run multiple times in the course of building and installing your module distribution.

If all you want to do is distribute a module called `foo`, contained in a file `foo.py`, then your setup script can be as simple as this:

setup.py:

```
from distutils.core import setup
setup(name='foo',
      version='1.0',
      author='Greg Ward',
      author_email='gward@python.net',
      url='https://www.python.org/sigs/distutils-sig/',
      py_modules=['foo'],
      )
```

Foo.py:

```
Print “Hello World, Welcome to Distutils”
```

Some observations:

- most information that you supply to the Distutils is supplied as keyword arguments to the `setup()` function
- those keyword arguments fall into two categories: package metadata (name, version number) and information about what’s in the package (a list of pure Python modules, in this case)
- **modules are specified by module name, not filename** (the same will hold true for packages and extensions)

- it’s recommended that you supply a little more metadata, in particular your name, email address and a URL for the project.

To create a source distribution for this module, you would create a setup script, `setup.py`, containing the above code, and run this command from a terminal:

```
python setup.py sdist
```

Output:

```
running sdist
running check
warning: sdist: manifest template 'MANIFEST.in' does not exist (using default file list)

warning: sdist: standard file not found: should have one of README, README.txt

writing manifest file 'MANIFEST'
creating foo-1.0
making hard links in foo-1.0...
hard linking foo.py -> foo-1.0
hard linking setup.py -> foo-1.0
creating dist
Creating tar archive
removing 'foo-1.0' (and everything under it)
```

Directory: Some new files are created after running sdist command

```
├── dist
│   └── foo-1.0.tar.gz
├── foo.py
├── MANIFEST
└── setup.py
```

MAINFEST:

```
# file GENERATED by distutils, do NOT edit
foo.py
setup.py
```

First, this command create a MAINFEST file and a directory called dist.

Sdist will create an archive file (e.g., tarball on Unix, ZIP file on Windows) containing your setup script `setup.py`, and your module `foo.py`. The archive file will be named `foo-1.0.tar.gz in dist directory`, and will unpack into a directory `foo-1.0`.

If an end-user wishes to install your `foo` module, all she has to do is download `foo-1.0.tar.gz` (or `.zip`), unpack it, and—from the `foo-1.0` directory

extract : `foo-1.0.tar.gz`

```
tar -xvf foo-1.0.tar.gz
```

After extract contains:

```
foo-1.0/
foo-1.0/foo.py
foo-1.0/setup.py
foo-1.0/PKG-INFO
```

In PKG-INFO:

```
Metadata-Version: 1.0
Name: foo
Version: 1.0
Summary: UNKNOWN
Home-page: https://www.python.org/sigs/distutils-sig/
Author: Greg Ward
Author-email: gward@python.net
License: UNKNOWN
Description: UNKNOWN
Platform: UNKNOWN
```

— then run

```
python setup.py install
```

which will ultimately copy `foo.py` to the appropriate directory for third-party modules in their Python installation.

Output:

```
running install
running build
running build_py
creating build
creating build/lib
copying foo.py -> build/lib
running install_lib
copying build/lib/foo.py -> /usr/lib/python2.7/site-packages
byte-compiling /usr/lib/python2.7/site-packages/foo.py to foo.pyc
running install_egg_info
```

```
Removing /usr/lib/python2.7/site-packages/foo-1.0-py2.7.egg-info
Writing /usr/lib/python2.7/site-packages/foo-1.0-py2.7.egg-info
```

After running install command:

```
foo-1.0
├── build
│   └── lib
│       └── foo.py
├── foo.py
├── PKG-INFO
└── setup.py
```

install: installs everything from build directory.

build: build command is responsible for putting the files to install into a build directory.

build_py: build_py command is responsible for copying the **package data files** to the build directory build/lib.

install_lib: install_lib installs all python modules (extensions and pure python)

install_egg_info: installs an egg infor directory for the packages.

Test this as running python:

```
[root@localhost foo-1.0]# python
Python 2.7.5 (default, Jun 24 2015, 00:41:19)
[GCC 4.8.3 20140911 (Red Hat 4.8.3-9)] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> import foo
Hello World, Welcome to Distutils
>>>
```

*This simple example demonstrates some fundamental concepts of the Distutils. First, both developers and installers have the same basic user interface, i.e. the setup script. The difference is which Distutils commands they use: the **sdist** command is almost exclusively for module developers, while **install** is more often for installers (although most developers will want to install their own code occasionally).*