jueves, 30 de julio de 2015

Empaquetado y distribución de proyectos Python (y II)

Introducción

Si hemos desarrollado un módulo/paquete/aplicación Python que creemos puede resultar de utilidad a otras personas, una alternativa para conseguir que llegue a los potenciales interesados consiste en empaquetar el proyecto y en subirlo con posterioridad al repositorio **PyPI** de la comunidad Python.

Para dar a conocer los nuevos proyectos, **PyPI** en su página principal, va presentando a sus visitantes los últimos desarrollos que se han publicado con una breve reseña. Además, incorpora un buscador y un navegador de paquetes (*browse packages*) para facilitar el acceso a los miles de desarrollos que hay registrados.

Si tenemos el propósito de compartir un proyecto y la fase de codificación, en principio, ha concluido antes de comenzar a empaquetar tendremos que recopilar alguna información importante entre la que se encuentra el nombre del proyecto, su versión (seguir recomendaciones PEP-400), tipo de software, tipo de licencia (GPL V3, MIT, etc), público al que va dirigido, estado del desarrollo (Alfa, Beta, Producción/Estable, etc.), versiones de Python que soporta (2.x/3.x) y las dependencias con otros paquetes, si existen.

Evidentemente, las personas que mantienen **PyPI** han pensado en todo y ofrecen un índice de clasificación con información normalizada que tendremos que utilizar para registrar con éxito nuestro proyecto. Esto evita que los desarrolladores "inventemos" más de lo debido y ordena con cierto criterio todos los proyectos del repositorio.

Después, antes de proceder al empaquetado tendremos que organizar en una carpeta, siguiendo un orden preestablecido, todos los archivos del proyecto, que básicamente son los que contienen el código fuente y otros que se podrán crear en un editor de textos como el instalador, archivos de configuración, la información que debe aparecer en **PyPI**, archivos con ejemplos, etcétera.

Para finalizar, tan sólo será necesario registrarse como usuario de **PyPI** antes de subir el proyecto y ponerlo a disposición de toda la comunidad.

Requerimientos

Una vez que **pip** y **setuptools** estén disponibles en nuestra instalación Python, para poder empaquetar y distribuir un proyecto es necesario contar con los módulos **wheel** y **twine**.

Wheel se utiliza para empaquetar y **twine** se utiliza para subir los proyectos a **PyPI**.

Para instalar **wheel** y **twine** (si no estuvieran disponibles):

```
$ pip3 install wheel
$ pip3 install twine
```

Caso práctico de empaquetado y distribución de un proyecto

Buscar

Buscar

Python para impacientes

Python
IPython
EasyGUI
Tkinter
JupyterLab
Numpy

Anexos

Guía urgente de MySQL
Guía rápida de SQLite3

Entradas + populares

Dar color a las salidas en la consola
En Python para dar color a las salidas en la consola (o en la terminal de texto) existen varias posibilidades. Hay un método basado ...

Instalación de Python, paso a paso
Instalación de Python 3.6 A finales de 2016 se produjo el lanzamiento de Python 3.6 . El propósito de esta entrada es mostrar, pas...

Añadir, consultar, modificar y suprimir elementos en Numpy
Acceder a los elementos de un array. [], [,], ... Acceder a un elemento de un array. Para acceder a un elemento se utiliz...

Variables de control en Tkinter
Variables de control Las variables de control son objetos especiales que se asocian a los widgets para almacenar sus valore...

Cálculo con arrays Numpy
Numpy ofrece todo lo necesario para obtener un buen rendimiento cuando se trata de hacer cálculos con arrays. Por como está concebid...

Tkinter: interfaces gráficas en Python
Introducción Con Python hay muchas posibilidades para programar una interfaz gráfica de usuario (GUI) pero Tkinter es fácil d...

Operaciones con fechas y horas. Calendarios
Los módulos datetime y calendar amplían las posibilidades del módulo time que provee funciones para manipular expresiones de ti...

Convertir, copiar, ordenar, unir y dividir arrays Numpy
Esta entrada trata sobre algunos métodos que se utilizan en Numpy para convertir listas en arrays y viceversa; para copiar arrays d...

Tkinter: Tipos de ventanas
Ventanas de aplicación y de diálogos En la entrada anterior tratamos los distintos gestores de geometría que se utilizan para di...

Threading: programación con hilos (I)
En programación, la técnica que permite que una aplicación ejecute

Para explicar, paso a paso, todo el proceso nos vamos a basar en un módulo Python llamado **Pysaurio**, que hemos desarrollado, que permite buscar y extraer información dispersa en múltiples archivos de texto con la posibilidad de guardar el resultado en un único archivo, por ejemplo, del tipo CSV.

El código fuente se encuentra en un único archivo .py y contiene una clase llamada **Pysaurio** con varios métodos que podremos utilizar en cualquier programa una vez que importemos el módulo:

from pysaurio import Raptor

Para organizar todos los archivos del proyecto hemos creado una carpeta llamada '**pysaurio**' con el siguiente contenido:

Pysaurio/

- **setup.py**: script de instalación.
- **setup.cfg**: archivo de configuración del script de instalación.
- **README.rst**: archivo de texto (ReStructuredText) con una descripción del proyecto.
- **README.md**: archivo de texto (Markdown) con una descripción del proyecto.
- **MANIFEST.in**: archivo con lista de archivos que se incluyen/excluyen del proyecto.
- **LICENSE.txt**: archivo con la licencia elegida para el proyecto.
- **pysaurio (carpeta del paquete)**: fuentes del módulo.
- **examples (carpeta con ejemplos)**: ejemplos (.py) y archivos de texto para pruebas.

Otros archivos y directorios que podrían haberse incluido:

- **REQUIREMENTS.txt**: Listado de dependencias que deben satisfacerse.
- **CHANGELOG.txt**: Cambios incorporados al proyecto ordenados cronológicamente por lanzamientos.
- **ROADMAP.txt**: Mejoras previstas en versiones futuras del proyecto.
- Carpetas para TEST, DATOS complementarios, imágenes, scripts, etc.

Descripción de archivos

setup.py

Este archivo que depende de **setuptools** se encuentra en el directorio raíz del proyecto. Es el más importante y tiene dos funciones principales:

- Sirve para describir y configurar diversos aspectos del proyecto. El script contiene una función global llamada setup() cuyos argumentos definen los detalles específicos de un proyecto.: autor, versión, descripción, etc.
- Se puede ejecutar desde la línea de comandos para realizar tareas relacionadas con la propia gestión del paquete. Para consultar las opciones disponibles: **python3 setup.py --help-commands**

Contenido del archivo setup.py:

```
from setuptools import setup
from codecs import open
from os import path

here = path.abspath(path.dirname(__file__))

with open(path.join(here, 'README.rst'), encoding='utf-8') as f:
    long_description = f.read()

setup(
    name='pysaurio',
    version='0.2.0',
    description='A tool for searching & extracting information...',
    long_description=long_description,
    url='https://pypi.python.org/pypi/pysaurio',
    author='Antonio',
    author_email='dir@correo.com',
    license='GNU GPLv3',
    classifiers=[
        'Development Status :: 4 - Beta',
        'Intended Audience :: Developers',
        'Intended Audience :: System Administrators',
        'Topic :: Utilities',
        'License :: OSI Approved :: GNU General Public License v3 (GPLv3)',
        'Programming Language :: Python :: 3',
    ],
    keywords='pysaurio search extract text csv',
    packages=['pysaurio'],
```

simultáneamente varias operaciones en el mismo espacio de proceso se...

Archivo

julio 2015 (2)

python.org

pypi.org

Sitios

- ActivePython
- Anaconda
- Bpython
- Django
- Flask
- Ipython
- IronPython
- Matplotlib
- MicroPython
- Numpy
- Pandas
- Pillow
- PortablePython
- PyBrain
- PyCharm
- PyDev
- PyGame
- Pypi
- PyPy
- Pyramid
- Python.org
- PyTorch
- SciPy.org
- Spyder
- Tensorflow
- TurboGears

```
    package_dir = {'pysaurio':'pysaurio'},
)
```

Parámetros de la función setup():

- **name**: Nombre del paquete. Es recomendable buscar en Pypi que el nombre no está en uso.
- **version**: Versión del paquete.
- **description**: Descripción breve del proyecto.
- **long_description**: Descripción larga del proyecto que se leerá del archivo "README.rst".
- **url**: Dirección web del proyecto.
- **author**: Nombre o nick del autor.
- **author_email**: Dirección de correo electrónico del autor.
- **license**: Licencia del proyecto.
- **classifiers**: Lista de categorías para clasificar el paquete
- **keywords**: Lista de palabras que describen el proyecto. Se pueden usar para encontrarlo.
- **packages**: Lista de paquetes Python que se incluirán en el instalable del proyecto.
- **package_dir**: Diccionario con el nombre de los paquetes incluidos y sus directorios.

Otros usos y otros parámetros que no se incluyen en el caso práctico pero que podrían utilizarse en otros proyectos.

- **download_url**: Página de descargas externa a Pypi.
- **packages**: Con la función find_packages() es posible encontrar los paquetes a incluir en el instalable. El argumento 'exclude' puede utilizarse para incluir todos excepto los especificados en la lista: packages=find_packages(exclude=['paquete1']). Requiere importar el módulo find_packages de setuptools.
- **install_requires**: Lista de dependencias. Esta lista se puede exportar previamente a un archivo REQUIREMENTS.txt que puede ser leído como el parámetro long_description.
- **package_data**: Diccionario con archivos que acompañan al proyecto para instalar dentro del paquete. package_data = {'archivo': ['archivo.dat']}
- **data_files**: Lista de archivos que acompañan al proyecto para instalar en otras ubicaciones diferentes a la ubicación del paquete: data_files=[('datos', ['dat/archivo.dat'])]
- **entry_points**: Funciones (de scripts) que pueden ser llamadas directamente por un usuario. entry_points = {'console_scripts': ['Script1 = script1: main',],}.
- **zip_safe**: Con el valor 'false' se impide que se instalen los paquetes en formato comprimido.

Más información sobre **setuptools**.

Ver otro ejemplo de lo que puede contener un archivo setup.py.

setup.cfg

El archivo setup.cfg contiene los valores predeterminados de diferentes opciones que se utilizan con los argumentos de setup.py.
Contenido del archivo setup.cfg en Pysaurio:

[bdist_wheel]
universal=0

La opción universal con el valor '0' establece que el paquete no esta soportado por las dos líneas de Python: 2.x y 3.x. El paquete sólo podrá instalarse en un sistema con Python 3.x.

README.rst

Todos los proyectos deben contener un archivo README.rst en el que se haga una descripción detallada del proyecto.

El formato que se utiliza con más frecuencia para escribir su contenido es reStructuredText aunque no es un requisito indispensable.

Contenido del archivo README.rst:

```
Pysaurio
----------

**Raptor** for extracting and displaying information from a set of files of the same type; and creating a single CSV file with all the selected information.
```

The information in the files may be in multiple rows::

```
PC01.txt:
User=ms123
Name=Mayra Sanz
OS=GNU/Linux
IP=10.226.140.1
```

But, also, the information may be in several columns. It is possible to read data from multiple fields in a single line::

...
...
...

MANIFEST.in

El archivo MANIFEST.in se utiliza en casos en que se necesiten empaquetar archivos adicionales que setup.py no incluya automáticamente.

Ver archivos que están incluidos por defecto en una distribución.

Contenido del archivo MANIFEST.in:

include README.md
include LICENSE.txt
include examples/*
include examples/txt/*

En MANIFEST.in se incluyen los archivos:

- **README.md**: tiene el mismo contenido que **README.rst** pero el texto está escrito utilizando el lenguaje de marcado MarkDown.
- **LICENSE.txt:** con el texto de la licencia GNU GPL Version 3.
- También se incluye la carpeta **'examples'** que contiene varios programas Python de ejemplos y algunos archivos de texto en el directorio **'txt'** necesarios para probar los programas.

Otros instrucciones permitidas: *exclude, recursive-include, recursive-exclude, global-include, global-exclude, graft* y *prune*

Ver detalles sobre cómo escribir un archivo MANIFEST.in.

Carpeta 'pysaurio'

Es la carpeta de mayor importancia porque es la que contiene el paquete a distribuir.

La práctica más común consiste en incluir los módulos y paquetes Python bajo un único paquete de nivel superior con el mismo nombre del proyecto (o similar).

En nuestro caso contiene un archivo __init__.py con el código fuente de **Pysaurio**.

Es imprescindible documentar los fuentes de los paquetes para que cualquier usuario que lo instale pueda consultarla con **pydoc3**.

Carpeta 'examples'

La carpeta **'examples'** contiene varios programas Python con ejemplos y una carpeta llamada **'txt'** con algunos archivos de texto necesarios para probar los programas.

También se incluye un archivo con la extensión .rap que es similar a un archivo **.INI** que es utilizado por unos de los programas de ejemplo.

El módulo Pysaurio utiliza los archivos .rap para salvar información relacionada con una operación de búsqueda y extracción de datos de una serie de archivos de texto. Entre otros **Pysaurio** tiene un método para guardar un archivo .rap y otro para abrirlo.

Empaquetar el proyecto

Para que un proyecto se pueda instalar desde **PyPI** es necesario crear al menos un paquete de distribución. En el capítulo anterior se comentaron los dos tipos más usados. A continuación, trataremos sobre la forma de crearlos:

Crear un paquete source o sdist

Para crear un paquete source o sdist (distribución de código fuente):

$ python3 setup.py sdist

Una distribución **source** requiere el paso de ser generada o construida cuando se instala con **pip**. Incluso si la distribución es python puro (no contiene extensiones) incluye una fase de construcción para los metadatos de instalación a partir del archivo **setup.py**.

Durante la generación de la distribución se crean dos carpetas nuevas en la carpeta del proyecto: una llamada "NombrePaquete.egg-info" que contiene archivos de texto con información del paquete o "huevo" (egg) y otra llamada '**dist**' con el "huevo" propiamente dicho o el paquete generado: "paquete-version.tar.gz".

Crear un paquete wheel

También es posible crear un paquete **wheel** que se podrá instalar sin necesidad de pasar por el proceso de construcción. La instalación de paquetes **wheel** es sustancialmente más rápida para el usuario final que la instalación de una distribución **source**.

Si el proyecto está basado en Python puro (no contiene extensiones compiladas) y soporta Python 2.x y 3.x, es posible crear un paquete **wheel universal**.

Para construir un paquete **wheel universal** (Esta opción **NO** sería la adecuada para Pysaurio porque el proyecto solo está soportado por Python 3.x):

$ python3 setup.py bdist_wheel --universal

También se puede establecer la variable '**universal**' con el valor **1** en el archivo **setup.cfg** con:

[bdist_wheel]
universal=1

Si el proyecto está basado en Python puro, pero no soporta de forma nativa Python 2.x y 3.x, entonces hay que crear el paquete con (Está opción **SÍ** sería la idónea para Pysaurio):

$ python3 setup.py bdist_wheel

bdist_wheel detectará que el código es Python puro y construirá un paquete con un nombre apropiado (según sea la instalación de Python) con la versión adecuada de Python: 2.x o 3.x. (Para más detalles sobre nombres de archivos de paquetes consultar PEP-425).

En la carpeta 'dist' se incluirá el paquete **wheel** "nombrepaquete-version-py3-none-any.whl" y en el carpeta del proyecto aparecerá una nueva carpeta llamada '**build**' con archivos generados durante el proceso de construcción.

Si se quiere dar soporte tanto a Python 2.x como a 3.x pero con un código fuente distinto se puede ejecutar **setup.py bdist_wheel** dos veces, una vez con Python 2.x y otra con Python 3.x.

Y **si el proyecto contiene extensiones compiladas**, entonces será necesario crear lo que se denomina paquete **wheel de plataforma**:

$ python3 setup.py bdist_wheel

bdist_wheel detectará que el código no es Python puro y construirá un paquete **wheel de plataforma** con un nombre apropiado.

PyPI actualmente sólo permite paquetes wheel de plataforma para Windows y OS X.

Subir el proyecto a PyPI

Para realizar pruebas es recomendable utilizar el sitio que ha dispuesto **PyPI** para ellas: **PyPI test site**

Es recomendable consultar las instrucciones de uso del sitio de pruebas. No hay problema en hacer todas la pruebas que se necesiten porque el sitio se "autolimpiará" cada cierto tiempo.

Crear una cuenta de usuario

Para subir un proyecto a **PyPI** es necesario tener una cuenta de usuario. Dicha cuenta puede crearse de forma manual utilizando el siguiente formulario o bien en el mismo procedimiento de registro del proyecto.

Registrar el proyecto

Hay dos caminos que se pueden seguir para registrar un proyecto:

1) (**Recomendado**) Utilizar el formulario de registro de **PyPI**

2) Registrar el proyecto con:

$ python3 setup.py register

Si no disponemos de una cuenta de usuario el asistente nos guiará para crear una.

Si la cuenta se generó de forma manual tendremos que crear el archivo **~/.pypirc** con el siguiente contenido:

```
[distutils]
index-servers=pypi

[pypi]
repository = https://upload.pypi.org/legacy/
username = usuario_PyPI
password = contraseña
```

Subir el paquete del proyecto a Pypi

Finalmente, para subir la distribución a **PyPI**, hay dos opciones:

1) (**Recomendado**) Ejecutar:

```
$ twine upload dist/*
```

La principal razón para utilizar **twine** es que **python3 setup.py upload** basa la subida en el uso de archivos de texto plano, con el riesgo de seguridad que supone exponer la cuenta del usuario de **PyPI** y su contraseña.

2) Utilizar **setuptools**:

```
$ python3 setup.py sdist bdist_wheel upload
```

Casi de inmediato, una vez se han subido los archivos a PyPI, la web anunciará en su página inicial la disponibilidad del paquete ofreciendo la fecha de publicación, el nombre del paquete, su versión y la descripción breve que se incluyó en el archivo setup.py.

Y si seleccionamos el enlace del nombre del paquete accederemos a la página del proyecto.

Y lo más importante cualquier usuario, si necesidad de registrarse, podrá instalar Pysaurio en su sistema Python con:

```
$ pip3 install pysaurio
```

Herramientas

- **Pyroma** es una aplicación que analiza la configuración de setup.py para comprobar si cumple las buenas prácticas recomendadas.

El modo desarrollo

Aunque no es imprescindible, un proyecto se puede instalar en modo "desarrollo" mientras se está trabajando en él:

```
$ python3 setup.py develop
```

ó

```
$ pip3 install -e
```

Ambos comandos instalarán las dependencias declaradas con "install_requires" y también los scripts declarados en "console_scripts".

Ir al índice del tutorial de Python

Publicado por Pherkad en 9:09

Etiquetas: Python3

Entrada más reciente Inicio Entrada antigua