[Distribuir aplicaciones Python](http://mundogeek.net/archivos/2008/09/23/distribuir-aplicaciones-python/" \o "Distribuir aplicaciones Python)

Una vez terminemos con el desarrollo de nuestra nueva aplicación es conveniente empaquetarla de forma que sea sencillo para los usuarios instalarla, y para nosotros distribuirla.

En Python existen dos módulos principales para este cometido: distutils, que es parte de la librería estándar y era el método más utilizado hasta hace poco, y setuptools, que extiende la funcionalidad de distutils y es cada vez más popular.

En este capítulo veremos el funcionamiento de ambas herramientas, y terminaremos explicando cómo crear ejecutables .exe para Windows a partir de nuestro programa en Python.

DISTUTILS

Todo programa distribuido con distutils contiene un script llamado por convención setup.py, que se encarga de instalar la aplicación llamando a la función setup de distutils.core. Esta función tiene montones de argumentos, que controlan, entre otras cosas, cómo instalar la aplicación.

Destinados a describir la aplicación tenemos los siguientes argumentos:

* name: El nombre del paquete.
* version: El número de versión.
* description: Una línea describiendo el paquete.
* long\_description: Descripción completa del paquete.
* author: Nombre del autor de la aplicación.
* author\_email: Correo electrónico del autor.
* maintainer: Nombre de la persona encargada de mantener el paquete, si difiere del autor.
* maintainer\_email: Correo de la persona encargada de mantener el paquete, si difiere del autor.
* url: Web de la aplicación.
* download\_url: Url de la que descargar la aplicación.
* license: Licencia de la aplicación

También tenemos argumentos que controlan los archivos y directorios que deben instalarse, como son packages, py\_modules, scripts y ext\_modules.

El parámetro scripts, que es una lista de cadenas, indica el nombre del módulo o módulos principales, es decir, los que ejecuta el usuario. Si nuestra aplicación consistiera, por ejemplo, en un solo script ejemplo.py, el código de setup.py podría tener un aspecto similar al siguiente:

[view plaincopy to clipboardprint?](http://mundogeek.net/archivos/2008/09/23/distribuir-aplicaciones-python/)

1. **from** distutils.core **import** setup
3. setup(name="Aplicacion de ejemplo",
4. version="0.1",
5. description="Ejemplo del funcionamiento de distutils",
6. author="Raul Gonzalez",
7. author\_email="zootropo en gmail",
8. url="http://mundogeek.net/tutorial-python/",
9. license="GPL",
10. scripts=["ejemplo.py"]
11. )

Si hemos escrito otros módulos para ser utilizados por el script principal, estos se indican mediante el parámetro py\_modules. Por ejemplo, supongamos que la aplicación consiste en un script principal ejemplo.py, y un módulo de apoyo apoyo.py:

[view plaincopy to clipboardprint?](http://mundogeek.net/archivos/2008/09/23/distribuir-aplicaciones-python/)

1. **from** distutils.core **import** setup
3. setup(name="Aplicacion de ejemplo",
4. version="0.1",
5. description="Ejemplo del funcionamiento de distutils",
6. author="Raul Gonzalez",
7. author\_email="zootropo en gmail",
8. url="http://mundogeek.net/tutorial-python/",
9. license="GPL",
10. scripts=["ejemplo.py"],
11. py\_modules=["apoyo"]
12. )

Para instalar paquetes Python (directorios que contienen varios módulos y un archivo \_\_init\_\_.py) usaríamos el parámetro packages. Si además del módulo ejemplo.py quisiéramos instalar los paquetes gui y bbdd, por ejemplo, haríamos algo así:

[view plaincopy to clipboardprint?](http://mundogeek.net/archivos/2008/09/23/distribuir-aplicaciones-python/)

1. **from** distutils.core **import** setup
3. setup(name="Aplicacion de ejemplo",
4. version="0.1",
5. description="Ejemplo del funcionamiento de distutils",
6. author="Raul Gonzalez",
7. author\_email="zootropo en gmail",
8. url="http://mundogeek.net/tutorial-python/",
9. license="GPL",
10. scripts=["ejemplo.py"],
11. packages=["gui", "bbdd"]
12. )

ext\_modules, por último, sirve para incluir extensiones que utilice el programa, en C, C++, Fortran, …

Veamos ahora cómo se utilizaría el archivo setup.py una vez creado.

Al ejecutar el comando

python setup.py install

los módulos y paquetes especificados por py\_modules y packages se instalan en el directorio Lib de Python. Los programas indicados en scripts, se copian al directorio Scripts de Python.

Una vez hemos comprobado que la aplicación se instala correctamente, procedemos a crear archivos mediante los que distribuir la aplicación a los usuarios. Para crear archivos con el código fuente se utiliza la opción sdist de setup.py, que crea por defecto un archivo tar.gz en Unix y un zip en Windows.

python setup.py sdist

Sin embargo se puede utilizar --formats para especificar el formato o formatos que queramos generar

|  |  |
| --- | --- |
| bztar | .tar.bz2 |
| gztar | .tar.gz |
| tar | .tar |
| zip | .zip |
| ztar | .tar.Z |

Para crear un archivo tar.bz2, un tar.gz y un zip, por ejemplo, se utilizaría la siguiente orden:

python setup.py sdist –formats=bztar,gztar,zip

Para generar un archivo de distribución binaria, se usa la opción bdist:

python setup.py bdist

Los formatos que soporta bdist son los siguientes:

|  |  |
| --- | --- |
| rpm | RPM |
| gztar | .tar.gz |
| bztar | .tar.bz2 |
| ztar | .tar.Z |
| tar | .tar |
| wininst | Instalador Windows |
| zip | .zip |

Para crear un archivo rpm y un instalador de Windows, por ejemplo, escribiríamos:

python setup.py bdist –formats=wininst,rpm

También es posible crear otros tipos de archivos de distribución utilizando scripts que extienden distutils, como es el caso de los paquetes deb mediante el script stdeb (http://stdeb.python-hosting.com/)

SETUPTOOLS

setuptools extediende distutils añadiendo una serie de funcionalidades muy interesantes: introduce un nuevo formato de archivo para distribución de aplicaciones Python llamado egg, se encarga de buscar todos los paquetes que deben instalarse y añadir las posibles dependencias, permite instalar paquetes de PyPI con un solo comando, etc.

Además, como setuptools se basa en distutils, un script de instalación básico utilizando setuptools es prácticamente igual a su equivalente con distutils. Tan sólo cambiaría la sentencia de importación.

[view plaincopy to clipboardprint?](http://mundogeek.net/archivos/2008/09/23/distribuir-aplicaciones-python/)

from setuptools import setup

2. setup(name="Aplicacion de ejemplo",
3. version="0.1",
4. description="Ejemplo del funcionamiento de distutils",
5. author="Raul Gonzalez",
6. author\_email="zootropo en gmail",
7. url="http://mundogeek.net/tutorial-python/",
8. license="GPL",
9. scripts=["ejemplo.py"],
10. )

El único inconveniente que podríamos encontrar al uso de setuptools es que no está incluido por defecto en Python 2.5, aunque es probable que esto cambie en próximas versiones debido a su gran uso. Pero los desarrolladores de setuptools han pensado en todo, e incluso esto no debería suponer ningún problema, ya que con un mínimo esfuerzo por nuestra parte podemos hacer que setuptools se descargue e instale automáticamente en la máquina del usuario si este no se encuentra ya en el sistema. Basta distribuir con nuestro paquete un pequeño módulo extra ez\_setup.py que viene incluido por defecto con setuptools (http://peak.telecommunity.com/dist/ez\_setup.py) y llamar a la función use\_setuptools del módulo al inicio de setup.py:

[view plaincopy to clipboardprint?](http://mundogeek.net/archivos/2008/09/23/distribuir-aplicaciones-python/)

1. **from** ez\_setup **import** use\_setuptools
2. use\_setuptools()
4. **from** setuptools **import** setup
6. setup(name="Aplicacion de ejemplo",
7. version="0.1",
8. description="Ejemplo del funcionamiento de distutils",
9. author="Raul Gonzalez",
10. author\_email="zootropo en gmail",
11. url="http://mundogeek.net/tutorial-python/",
12. license="GPL",
13. scripts=["ejemplo.py"],
14. )

Veamos ahora con más detenimiento algunos de los cambios y novedades que introduce setuptools.

Integración con PyPI

Al estilo de CPAN en Perl setuptools permite instalar de forma fácil y sencilla los paquetes pertenecientes a PyPI, el Índice de Paquetes Python (http://pypi.python.org/pypi), así como subir nuestros propios paquetes.

PyPI cuenta en el momento de escribir estas líneas con 4782 paquetes, por lo que poder instalar los paquetes de este repositorio con un simple comando supone una ayuda muy a tener en cuenta.

Instalar un paquete de PyPI es tan sencillo como pasar al comando easy\_install el nombre del paquete a instalar

easy\_install docutils  
Searching for docutils  
Reading http://pypi.python.org/simple/docutils/  
Reading http://docutils.sourceforge.net/  
Best match: docutils 0.5  
Downloading http://prdownloads.sourceforge.net/docutils/docutils-0.5.tar.gz?download  
Processing docutils-0.5.tar.gz  
Running docutils-0.5/setup.py -q bdist\_egg –dist-dir /tmp/easy\_install-wUAyUZ/docutils-0.5/egg-dist-tmp-kWkkkv  
“optparse” module already present; ignoring extras/optparse.py.  
“textwrap” module already present; ignoring extras/textwrap.py.  
zip\_safe flag not set; analyzing archive contents…  
docutils.writers.newlatex2e.\_\_init\_\_: module references \_\_file\_\_  
docutils.writers.pep\_html.\_\_init\_\_: module references \_\_file\_\_  
docutils.writers.html4css1.\_\_init\_\_: module references \_\_file\_\_  
docutils.writers.s5\_html.\_\_init\_\_: module references \_\_file\_\_  
docutils.parsers.rst.directives.misc: module references \_\_file\_\_  
Adding docutils 0.5 to easy-install.pth file  
Installing rst2pseudoxml.py script to /usr/bin  
Installing rst2html.py script to /usr/bin  
Installing rst2latex.py script to /usr/bin  
Installing rst2s5.py script to /usr/bin  
Installing rst2newlatex.py script to /usr/bin  
Installing rstpep2html.py script to /usr/bin  
Installing rst2xml.py script to /usr/bin  
Installed /usr/lib/python2.5/site-packages/docutils-0.5-py2.5.egg  
Processing dependencies for docutils  
Finished processing dependencies for docutils

Poder subir nuestros paquetes a PyPI requiere de un proceso un poco más laborioso. Primero registramos los detalles de nuestra aplicación en PyPI mediante la opción register del script setup.py, el cuál nos preguntará por nuestro nombre de usuario, contraseña y correo electrónico si no tenemos cuenta en PyPI, o nombre de usuario y contraseña si nos registramos anteriormente:

python setup.py register  
running register  
running egg\_info  
creating Aplicacion\_de\_ejemplo.egg-info  
writing Aplicacion\_de\_ejemplo.egg-info/PKG-INFO  
writing top-level names to Aplicacion\_de\_ejemplo.egg-info/top\_level.txt  
writing dependency\_links to Aplicacion\_de\_ejemplo.egg-info/dependency\_links.txt  
writing manifest file ‘Aplicacion\_de\_ejemplo.egg-info/SOURCES.txt’  
reading manifest file ‘Aplicacion\_de\_ejemplo.egg-info/SOURCES.txt’  
writing manifest file ‘Aplicacion\_de\_ejemplo.egg-info/SOURCES.txt’  
We need to know who you are, so please choose either:  
1. use your existing login,  
2. register as a new user,  
3. have the server generate a new password for you (and email it to you), or  
4. quit  
Your selection [default 1]: 1  
Username: zootropo  
Password:  
Server response (200): OK  
I can store your PyPI login so future submissions will be faster.  
(the login will be stored in /home/zootropo/.pypirc)  
Save your login (y/N)?y

Para crear y subir una distribución con el código fuente de nuestra aplicación se utiliza la opción sdist upload:

python setup.py sdist upload

También podríamos crear y subir un egg (un formato de archivo para distribuir aplicaciones Python que veremos en la próxima sección) utilizando la opción bdist\_egg upload:

python setup.py bdist\_egg upload

O combinar los tres pasos en un solo comando:

python setup.py register sdist bdist\_egg upload

Una vez subido el paquete cualquier persona podría instalarlo en su sistema utilizando easy\_install, de la misma forma que cualquier otro paquete de PyPI:

easy\_install mi-paquete

Eggs

Los eggs (huevo en inglés) son archivos de extensión .egg mediante los que distribuir aplicaciones en Python. Serían algo así como el equivalente a los archivos .jar del mundo Java. Son multiplataforma, permiten manejar dependencias, y permiten instalar distintas versiones del mismo paquete.

La forma más sencilla de instalar aplicaciones distribuidas como archivos egg es mediante el comando easy\_install, el cuál comentamos brevemente en el punto anterior al hablar sobre su uso para instalar paquetes de PyPI. Para instalar un archivo egg no tenemos más que pasarle el nombre del archivo al comando easy\_install:

easy\_install mi-aplicacion.egg

o bien podemos pasarle la URL de la que descargar el egg:

easy\_install http://mundogeek.net/mi-aplicacion.egg

Para construir nuestros propios eggs podemos utilizar el comando bdist\_egg de setup.py, de forma similar a la manera en que construíamos paquetes RPM o instaladores para Windows con distutils:

python setup.py bdist\_egg

**Otros cambios destacables**

Uno de los cambios más interesantes es la incorporación de un nuevo argumento para la función setup llamado install\_requires, que consiste en una cadena o lista de cadenas que indica los paquetes de los que depende la aplicación. Si nuestra aplicación necesitara tener instalado el paquete apoyo para poder ejecutarse, por ejemplo, escribiríamos lo siguiente:

[view plaincopy to clipboardprint?](http://mundogeek.net/archivos/2008/09/23/distribuir-aplicaciones-python/)

1. install\_requires = ["apoyo"]

Y de esta forma, easy\_install se encargaría de buscar e instalar el paquete si fuera necesario, bien en PyPI, o en cualquier otro repositorio indicado por el parámetro dependency\_links.

Además podemos especificar que se necesita una versión concreta del paquete requerido, que sea mayor o menor que una cierta versión, o que no se trate de una versión determinada utilizando operadores relacionales (==, !=, <, <=, >, >=):

[view plaincopy to clipboardprint?](http://mundogeek.net/archivos/2008/09/23/distribuir-aplicaciones-python/)

1. install\_requires = ["apoyo >= 1.0 < 2.0"]

También existen argumentos similares para declarar paquetes que deben instalarse para poder ejecutar el script de instalación (setup\_requires), para poder ejecutar las posibles pruebas incluídas con el paquete (tests\_require) y para conseguir funcionalidades adicionales (extras\_require, que consiste en este caso en un diccionario).

setuptools incluye también atajos útiles, como la función find\_packages() que nos evita tener que listar todos y cada uno de los paquetes que utiliza nuestro script en el parámetro packages, como era el caso de distutils:

[view plaincopy to clipboardprint?](http://mundogeek.net/archivos/2008/09/23/distribuir-aplicaciones-python/)

1. **from** ez\_setup **import** use\_setuptools
2. use\_setuptools()
4. **from** setuptools **import** setup, find\_packages
6. setup(name="Aplicacion de ejemplo",
7. version="0.1",
8. description="Ejemplo del funcionamiento de distutils",
9. author="Raul Gonzalez",
10. author\_email="zootropo en gmail",
11. url="http://mundogeek.net/tutorial-python/",
12. license="GPL",
13. scripts=["ejemplo.py"],
14. packages = find\_packages()
15. )

**CREAR EJECUTABLES .EXE**

Tanto en Mac OS como en la mayor parte de las distribuciones Linux el intérprete de Python está instalado por defecto, por lo que los usuarios de estos sistemas no tienen mayor complicación a la hora de instalar y ejecutar aplicaciones escritas en Python.

En el caso de Windows, esto no es así, por lo que sería interesante que los usuarios de este sistema operativo no tuvieran que instalar el intérprete de Python. También sería interesante que nuestro programa consistiera en un archivo .exe en lugar de uno o varios archivos .py, para simplificar las cosas.

Todo esto lo podemos lograr gracias a py2exe, una extensión para distutils que, como su nombre indica, permite crear ejecutables para Windows a partir de código Python, y que permite ejecutar estas aplicaciones sin necesidad de tener instalado el intérprete de Python en el sistema.

Py2exe funciona examinando nuestro código fuente en busca de los módulos y paquetes que utilizamos, compilándolos y construyendo un nuevo archivo que incluye estos archivos y un pequeño intérprete de Python integrado.

Para probar el funcionamiento de py2exe creemos un pequeño programa ejemplo.py

[view plaincopy to clipboardprint?](http://mundogeek.net/archivos/2008/09/23/distribuir-aplicaciones-python/)

1. **print** "Soy un .exe"

y el archivo setup.py correspondiente. Los cambios que tenemos que realizar a setup.py son sencillos: importar py2exe, y utilizar los argumentos console y windows para indicar el nombre del script o scripts que queramos convertir en ejecutables de consola o ejecutables de interfaz gráfica, respectivamente.

[view plaincopy to clipboardprint?](http://mundogeek.net/archivos/2008/09/23/distribuir-aplicaciones-python/)

1. **from** distutils.core **import** setup
2. **import** py2exe
4. setup(name="Aplicacion de ejemplo",
5. version="0.1",
6. description="Ejemplo del funcionamiento de distutils",
7. author="Raul Gonzalez",
8. author\_email="zootropo en gmail",
9. url="http://mundogeek.net/tutorial-python/",
10. license="GPL",
11. scripts=["ejemplo.py"],
12. console=["ejemplo.py"]
13. )

Para crear el ejecutable, utilizamos una nueva opción de línea de comandos para setup.py disponible tras importar el módulo y llamada, cómo no, py2exe:

python setup.py py2exe

Con esto py2exe generará un directorio build, con las librerías compiladas, y un directorio dist, con los archivos que conforman nuestra aplicación.

Entre los archivos que podemos encontrar en dist tendremos uno o varios ejecutables con el mismo nombre que los scripts indicados en console y windows, un archivo python\*.dll, que es el intérprete de Python, y un archivo library.zip, que contiene varios archivos pyc que son los módulos que utiliza la aplicación compilados.

Si queremos reducir el número de archivos a distribuir, podemos utilizar la opción --bundle de py2exe para añadir a library.zip las dll y los pyd (--bundle 2) o las dll, los pyd y el intérprete (--bundle 1).

python setup.py py2exe --bundle 1

o bien podemos añadir un nuevo argumento options a la función setup que indique el valor a utilizar (opción bundle\_files), de forma que no tengamos que añadir el flag --bundle cada vez que usemos el comando py2exe:

[view plaincopy to clipboardprint?](http://mundogeek.net/archivos/2008/09/23/distribuir-aplicaciones-python/)

1. **from** distutils.core **import** setup
2. **import** py2exe
4. setup(name="Aplicacion de ejemplo",
5. version="0.1",
6. description="Ejemplo del funcionamiento de distutils",
7. author="Raul Gonzalez",
8. author\_email="zootropo en gmail",
9. url="http://mundogeek.net/tutorial-python/",
10. license="GPL",
11. scripts=["ejemplo.py"],
12. console=["ejemplo.py"],
13. options={"py2exe": {"bundle\_files": 1}}
14. )

Por último podemos incluso prescindir de library.zip e incrustarlo en el ejecutable utilizando el argumento zipfile=None

[view plaincopy to clipboardprint?](http://mundogeek.net/archivos/2008/09/23/distribuir-aplicaciones-python/)

1. **from** distutils.core **import** setup
2. **import** py2exe
4. setup(name="Aplicacion de ejemplo",
5. version="0.1",
6. description="Ejemplo del funcionamiento de distutils",
7. author="Raul Gonzalez",
8. author\_email="zootropo en gmail",
9. url="http://mundogeek.net/tutorial-python/",
10. license="GPL",
11. scripts=["ejemplo.py"],
12. console=["ejemplo.py"],
13. options={"py2exe": {"bundle\_files": 1}},
14. zipfile=None
15. )

[distutils](http://mundogeek.net/etiqueta/distutils/), [exe](http://mundogeek.net/etiqueta/exe/), [py2exe](http://mundogeek.net/etiqueta/py2exe/), [python](http://mundogeek.net/etiqueta/python/), [setuptools](http://mundogeek.net/etiqueta/setuptools/), [tutorial](http://mundogeek.net/etiqueta/tutorial/)

**COMENTARIOS**

1. [fidojones](http://fidojones.com/)

Yo recomendaria también pyinstaller que me ha dado mejores resultados con PyQT que el py2exe. Eso si recomiendo usar la version del SVN que es la que está mas actualizada.

<http://pyinstaller.python-hosting.com/>

Y para empaquetar para Mac OS, py2app. Aunque con las QT4 da problemas con las rutas.

<http://svn.pythonmac.org/py2app/py2app/trunk/doc/index.html>

El que no quiera volverse loco para empaquetar PyQT para Mac OS que mire este post, a mi me ha funcionado y he podido finalmente empaquetarlo:

<http://markmail.org/message/6evkmysjce6lgjry>

[Responder](http://mundogeek.net/archivos/2008/09/23/distribuir-aplicaciones-python/?replytocom=279948#respond)

## CREAR EJECUTABLE (“.EXE”) DE UN PROGRAMA PYTHON

Programar en Python en nuestro GNU/Linux es una belleza. Pero cuando también nos toca soportar dicho Software en plataformas MS Windows lo vemos más complicado. O tal vez pueda darse el caso de que el profesor solicite entregar la tarea programada en un “.exe” y nos de la libertad de escoger el lenguaje de programación que nosotros queramos (cualquier relación con la realidad es pura coincidencia).

Pero por dicha podemos hacer un freeze de nuestros scripts en Python, es decir, empaquetar en un “.exe” nuestro programa, las bibliotecas Python que utiliza y el mismísimo intérprete Python con lo que podremos distribuir el Software y ejecutarlo sin la necesidad de tener que descargar Python para Windows primero.

Para ello necesitamos tener un MS Windows en el cual realizaremos lo siguiente:

1. Instalar y configurar Python.
2. Instalar como administrador Py2Exe.
3. Crear script de distribución **setup.py**
4. Llamar al script de distribución con el parámetro py2exe

### Instalar y configurar Python

Es necesario descargar Python para Windows desde [la página oficial](http://www.python.org/download/releases/).

Es importante descargarse la **última versión de la rama 2.X** ya que py2exe no está disponible para Python 3 todavía (y no creo que pronto). En mi caso para la máquina virtual (mi GNU/Linux es de 64bits, la máquina virtual de 32) descargué la versión Python 2.7.3 para 32bits, es decir [Windows x86 MSI Installer](http://www.python.org/ftp/python/2.7.3/python-2.7.3.msi).

Una vez descargado se ejecuta e instala para todos los usuarios. Recomiendo cambiar la ruta de C:\Python2.7 a C:\Program Files\Python2.7 por una cuestión de orden.

Una vez instalado se procede a configurar el PATH de Windows para que encuentre el ejecutable de Python, para así no tener que escribir toda la ruta del ejecutable cuando queramos correr un script. Para ello vamos al editor de variables de entorno de Windows. En Windows 7 es:

Inicio -> Click derecho en My Computer -> Propiedades -> Advanced System Settings -> Pestaña Advanced -> Botón Environment Variables

Editamos la que dice Path en el cuadro de variables de entorno y adjuntamos al final del string la ruta de la carpeta de instalación de Python, separándola con un punto y coma **;** , es decir:

[lo que habia escrito];[ruta de instalacion python]

En mi caso:

%SystemRoot%\system32;%SystemRoot%;%SystemRoot%\System32\Wbem;

%SYSTEMROOT%\System32\WindowsPowerShell\v1.0\;C:\Program Files\Python2.7

Ok, ok, ok. Ahora abrimos la PowerShell (que de Power no tiene nada si la comparas con Bash) y escribimos el comando python y debería iniciar el interprete. Para abrir la PowerShell ingresa al menú de inicio y busca PowerShell. Listo.

### Instalar como administrador Py2Exe

Py2Exe es el paquete que hará la magia de crearnos nuestro “.exe”. Descargamos la última versión desde la [página oficial](http://www.py2exe.org/) para nuestra versión de Python y arquitectura: [descargar](http://sourceforge.net/projects/py2exe/files/py2exe/).

En mi caso, es [py2exe-0.6.9.win32-py2.7.exe](http://sourceforge.net/projects/py2exe/files/py2exe/0.6.9/py2exe-0.6.9.win32-py2.7.exe/download).

Un vez descargado instalamos el paquete como administrador: click derecho - ejecutar como administrador. Si no lo ejecutamos como administrador el instalador se quedará congelado y no avanzará. Listo.

### Crear script de distribución setup.py

Creamos un archivo Python llamado setup.py encargado de la distribución del Software. El estricto mínimo funcional sería:

from distutils.core import setup  
import py2exe

setup(console=['nombrearchivo.py'])

Ahora bien un ejemplo más completo sería:

# -\*- coding: utf-8 -\*-

import sys

from distutils.core import setup

kwargs = {}

if 'py2exe' in sys.argv:

import py2exe

kwargs = {

'console' : [{

'script' : 'nombrescript.py',

'description' : 'Descripcion del programa.',

'icon\_resources' : [(0, 'icon.ico')]

}],

'zipfile' : None,

'options' : { 'py2exe' : {

'dll\_excludes' : ['w9xpopen.exe'],

'bundle\_files' : 1,

'compressed' : True,

'optimize' : 2

}},

}

setup(

name='nombreproyecto',

author='Nombre del autor',

author\_email='autor@correo.com',

\*\*kwargs)

Lo bueno de éste último es que creará realmente un único archivo .exe. Y la verdad se puede volver [tan complejo como ustedes quieran](http://svn.code.sf.net/p/nestededitor/code/trunk/setup.py). Toda la documentación de cada uno de éstos elementos pueden encontrarlo en la wiki de Py2Exe.

### Llamar al script de distribución con el parámetro py2exe

Desde el PowerShell:

cd C:\Carpeta De Proyecto\

python setup.py py2exe

Y luego encontrarán el .exe en la carpeta dist/.

Saludos