

CÓMO CONVERTIR TUS SCRIPTS PYTHON .PY A EJECUTABLES DE WINDOWS .EXE

By Wa4Ya HaCK
Aka. FerSuper2



ÍNDICE

0.- INTRODUCCIÓN

1.- NUESTRA ALTERNATIVA

2.- EMPEZANDO...

3.- NOS METEMOS EN EL MEOLLO!

4.- POR FIN! AMPLIANDO HORIZONTES

5.- Y PARA TERMINAR...

*Este paper se distribuye bajo la licencia
Creative Commons 3.0 :*

*Usted puede copiar, reproducir o modificar
este documento **mientras** reconozca al
autor original:*

WaAYa HaCK / FerSuper2

0.- INTRODUCCIÓN

Bienvenidos/as! Este es un documento cuyo objetivo principal es **compilar scripts de Python (.py) a ejecutables de Windows (.exe)** .

Sin embargo, no iremos por la vía casual...

El material de este curso es:

- Intérprete de Python, versión 2.7.2
- SO Windows XP SP3 de 32 bits
- ~~Una mierda de notebook~~
- Mi script de Python
- El “compilador” ...

Bueno, antes que nada, os voy a presentar a nuestro nuevo amigo...

1.- NUESTRA ALTERNATIVA

¿Cuántas veces habéis buscado en Google “compilar .py” , “python a ejecutable”, “.py a .exe”, y demás?

¿Y de éstas, cuántas veces os ha salido **py2exe** ? Sí, py2exe está muy bien, pero... empieza a fallar. Últimamente mucha gente tiene problemas con py2exe. A veces sólo funciona con la versión 2.6 de Python; a veces no va el **--bundle**; a veces no le da la gana funcionar...

Bueno, para los que no nos hace falta tanto **py2exe** ese para compilar nuestros scripts (entre los que me incluyo), hay *algo más sencillo y efectivo*...

Y aquí está nuestro amigo **PyInstaller** !!!!!!!!!!!!!!!

No sé si alguno de vosotros lo conocía ya; no sé si diréis: “¿*Para qué tanto misterio si yo ya lo conocía?*” . Lo que pasa es que el monopolio de la compilación pythoniana (:P) está en py2exe. Por lo tal, voy a innovar y elaborar un manual, pero con **PyInstaller**.

PyInstaller es un programa que convierte scripts de Python (.py) a ejecutables (.exe). Sus ventajas principales son :

- **Multiplataforma:** Windows XP, Vista y 7, 32 y 64 bits; Mac OS X 32 bits; Linux 32 y 64 bits.
- **Librerías:** Acepta las librerías por defecto, las de PyQt, las de Django y las de Matplotlib.
- **Fluidez y facilidad:** PyInstaller es rápido y no requiere escribir código.

Su versión actual es la 1.5, y funciona de Python 2.2 a Python 2.7 .

Necesita un complemento llamado **pywin32** para Windows.

Página oficial (Los de Windows, descargad el archivo .tar igualmente: <http://www.pyinstaller.org/>

Descarga del **pywin32**: <http://sourceforge.net/projects/pywin32/files/pywin32/>

Y... ¡empezamos!

2.- EMPEZANDO...

Ahora que ya tenemos todo lo necesario (~~¡falta el cubata, quizás?~~) vamos a iniciar la batalla.

Hemos descargado el **pywin32** y procederemos a instalarlo. Yo he seleccionado el **pywin32-win32-py2.7.exe**, puesto que tengo un Windows de 32 bits y mi Python es 2.7 .

PAQUETE PYTHON	WINDOWS 32b	WINDOWS 64b
2.7 32b	pywin32-win32-py2.7	pywin32-win32-py2.7
2.7 64b	-	pywin32-win64-py2.7
2.6 32b	pywin32-win32-py2.6	pywin32-win32-py2.6
2.6 64b	-	pywin32-win64-py2.6
etc.

Creo que veis el patrón en la tabla, ¿no? Si tienes Windows de 32bits, sólo puedes tener instalado Python de 32bits; por lo que debes escoger **pywin32-win32-py2.?** (acorde con la versión de Python que tengas).

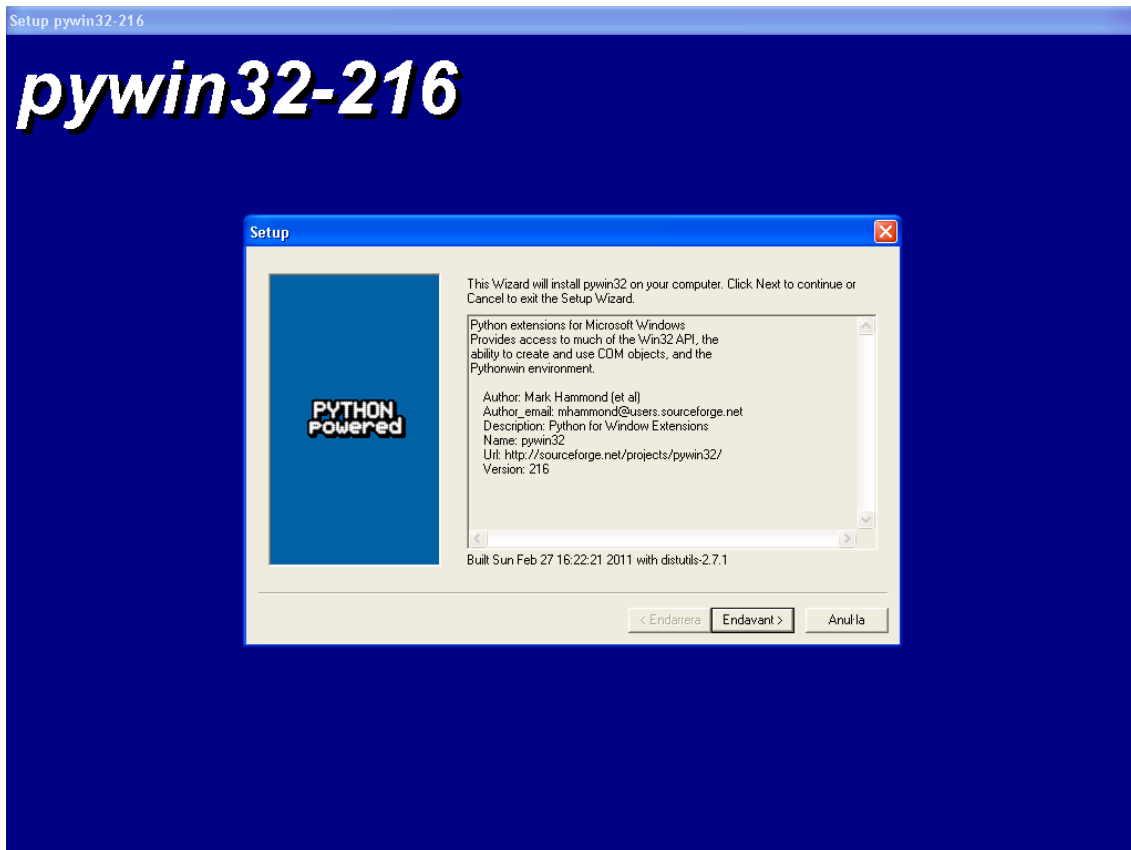
Sin embargo, si tienes Windows de 64bits, tu Python puede ser tanto de 32bits como de 64bits; debes escoger el **pywin32** según el **intérprete de Python**: si es de 32bits, deberás escoger **pywin32-win32** aunque tu SO sea de 64bits.

Una vez hecha la selección, nos descargamos el **pywin32** y obtendremos...



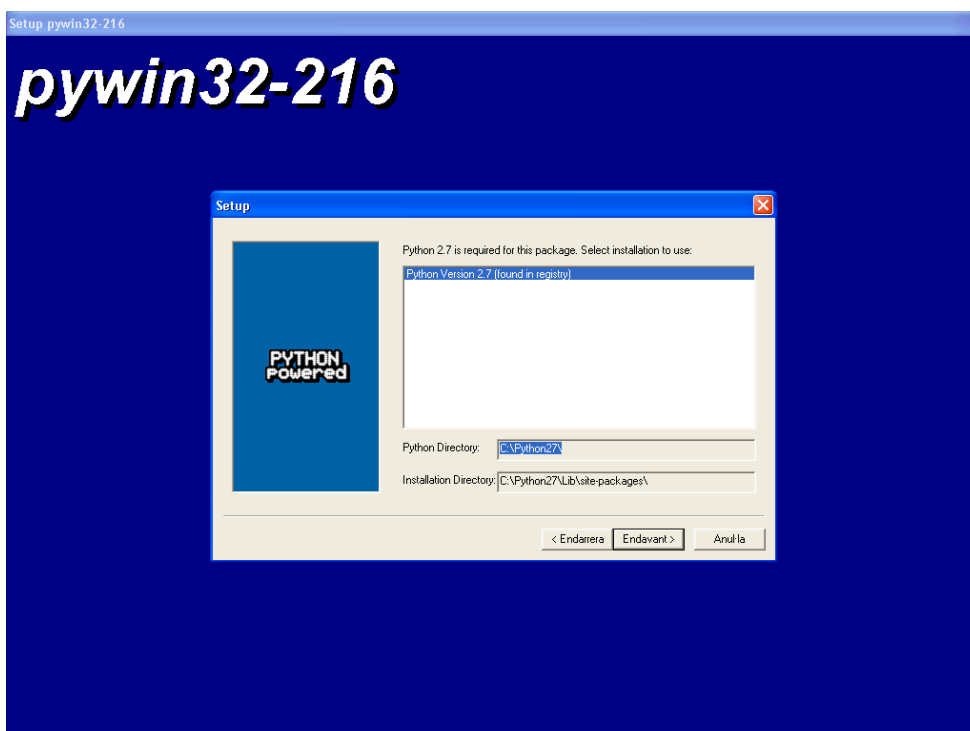
*El instalador de **pywin32**. Como podéis observar, tengo Windows de 32bits (por tanto, el intérprete es de 32bits) y la versión de Python 2.7 .*

Ahora procederemos a ejecutarlo:

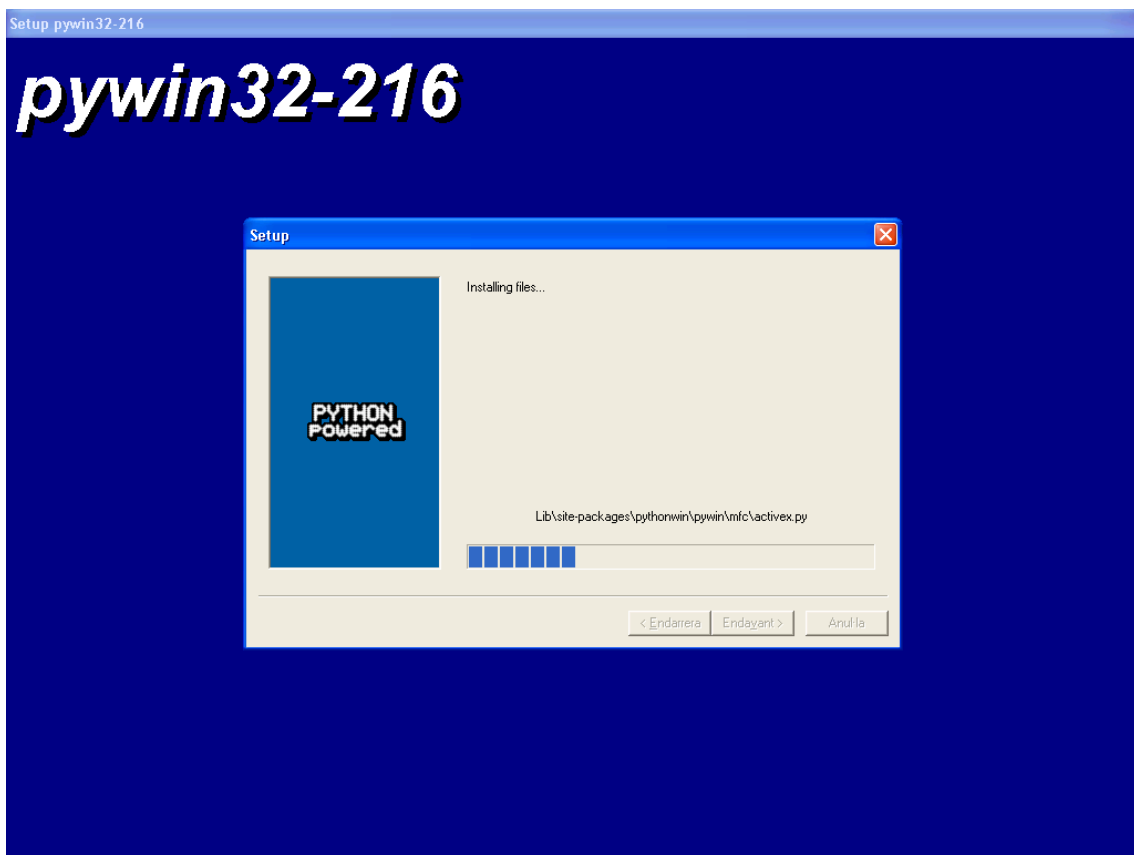


Instalación típica de Windows, como el “Sí mamá, que sí; sí, esto también; ¡sí, claro!; sí, lo haré... adiós. ¡Sí!”

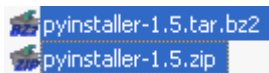
Ahora, si habéis elegido bien, detectará el intérprete de Python y podremos proseguir:



Y... ¡lo encontré! Por el momento todo correcto, ahora a instalar.



Cuando haya terminado, cerramos y vamos a otro lugar: al **PyInstaller** (al fin y al cabo es el protagonista, ¿no?)



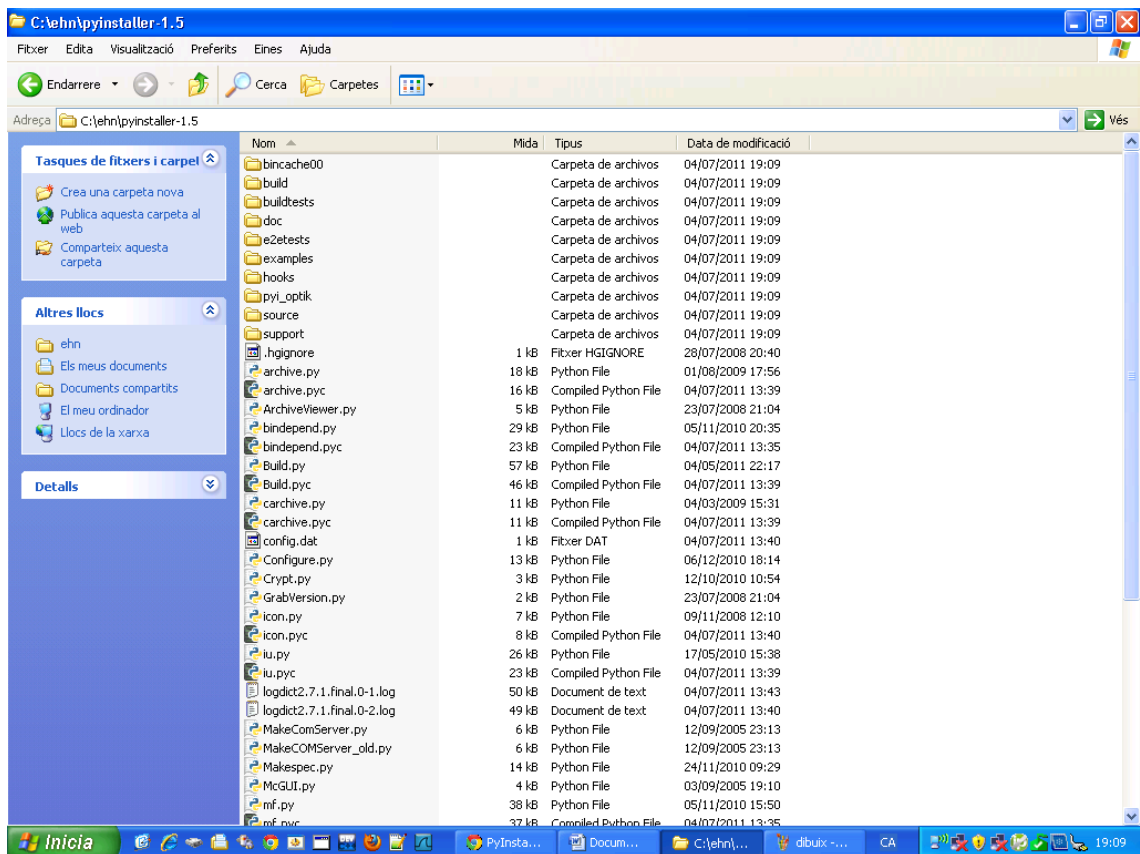
*Sí, tengo el .zip y el .tar.bz2 ... eso es porque me descargué el .zip y estaba incompleto :P , me salieron un montón de errores... así que **descargaros el .tar!***

Descomprimos el **.tar.bz2** y nos encontramos con la carpeta

PyInstaller-1.5

...

dentro de la cual encontramos un montón de archivos:



Diría que ya va siendo hora de ir a lo fuerte...

3.- NOS METEMOS EN EL MEOLLO!

“Bueno, bueno... qué poca acción... ese sopa nos va a tener instalando tonterías y descomprimiendo cosas... ¡eso ya lo puedo hacer yo solito!”

-¡Esperaaaad! Ahora *nos metemos en el meollo...*

Lo primero será crear (o elegir) un script.py . En este caso, le voy a llamar **compilar.py** .

El código de este script es simple:

```
# compilar.py
# Apartaos, que voy!

print "Hola!"
print "Este es un script ya compilado... o no?"
nombre = raw_input("Escribe un nombre... ")
print "Has dicho " + nombre
print ""
print "Pues muy bien!"
print ""
print ""
print "-----"
print ""
print "Dame dos numeros menores de 50 y te hago una lista
de uno al otro!"
x = int(raw_input("Dame un numero: "))
y = int(raw_input("Dame otro numero: "))
print "Has dicho: "
print x
print y
```

```

print "Pues vamos!"
rango = range(x, y, 1)
print ""
print "Y tu lista es... "
print "-----"
print ""
print rango
print "Adios!"
# Fin del script

```

Ahora que tenemos nuestro **compilar.py**, vamos a comprobar que no posee ningún fallo: nos movemos a C:\ehn\ , donde tengo yo el script:

```
python compilar.py
```

Después de comprobar cómo funciona correctamente, vamos a probar de compilarlo: nos vamos a C:\ehn\pyinstaller-1.5\ , donde está PyInstaller:

```
python Configure.py
```

Y él solito se va a encargar de reparar todo lo que haga falta. Conviene hacer esto siempre antes de compilar, para evitar posibles errores:

```

C:\Documents and Settings\Administrador>cd..
C:\Documents and Settings>cd..
C:\>cd ehn
C:\ehn>cd pyinstaller-1.5
C:\ehn\pyinstaller-1.5>python Configure.py
I: read old config from config.dat
I: computing EXE dependencies
I: Finding TCL/TK...
I: Analyzing C:\Python27\DLLs\_tkinter.pyd
I: Adding tcl85.dll dependency of _tkinter.pyd
I: Adding tk85.dll dependency of _tkinter.pyd
I: Skipping KERNEL32.dll dependency of _tkinter.pyd
I: Adding python27.dll dependency of _tkinter.pyd
I: Skipping MSVCRT90.dll dependency of _tkinter.pyd
I: Analyzing C:\Python27\DLLs\_hashlib.pyd

```

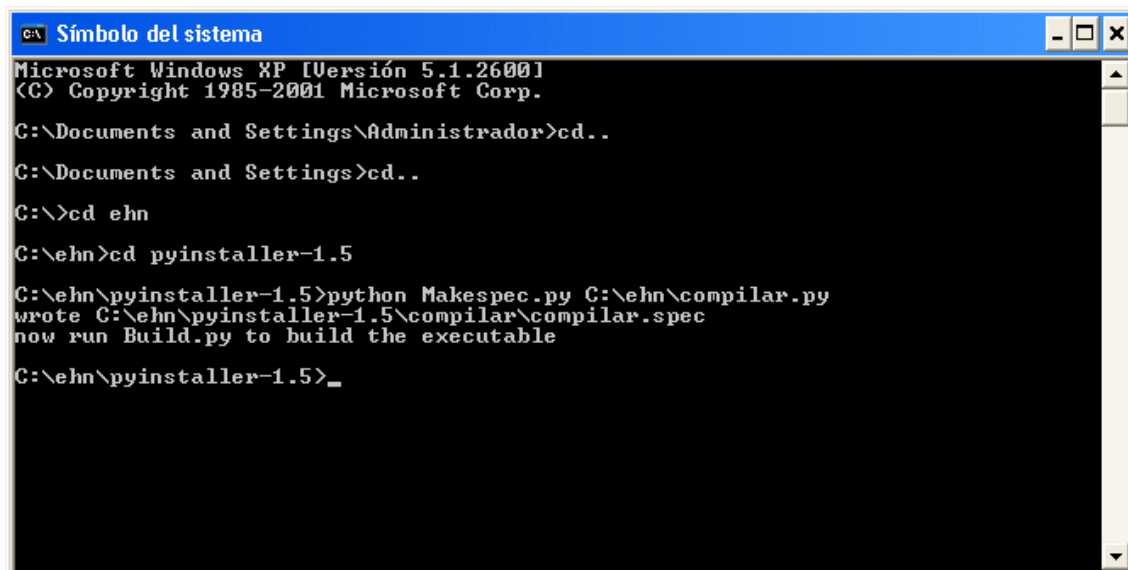
Configure.py va añadiendo o reparando los archivos que están perdidos o dañados, automáticamente. También actualiza los directorios.

Después de eso, vamos a crear un archivo **.spec** . Aquí es el momento: el sustitutivo del “setup.py” de cuando usábamos el **py2exe**, de cuando nos peleábamos con el **--bundle** , con el **zipfile=None**... pero lo explicaremos más tarde.

```
python Makespec.py C:\ehn\compilar.py
```

Recordad que **compilar.py** es el script que he creado antes.

Entonces...



```
C:\> Símbolo del sistema
Microsoft Windows XP [Versión 5.1.2600]
(C) Copyright 1985-2001 Microsoft Corp.

C:\Documents and Settings\Administrador>cd..
C:\Documents and Settings>cd..
C:\>cd ehn
C:\ehn>cd pyinstaller-1.5
C:\ehn\pyinstaller-1.5>python Makespec.py C:\ehn\compilar.py
wrote C:\ehn\pyinstaller-1.5\compilar\compilar.spec
now run Build.py to build the executable
C:\ehn\pyinstaller-1.5>_
```

Podemos observar que la salida del comando nos dice **que se ha creado una carpeta con el nombre de nuestro script**, y que dentro **hay un archivo llamado compilar.spec** .

Lo único que tenemos que hacer es escribir:

```
python Build.py C:\ehn\pyinstaller-1.5\compilar\compilar.spec
```

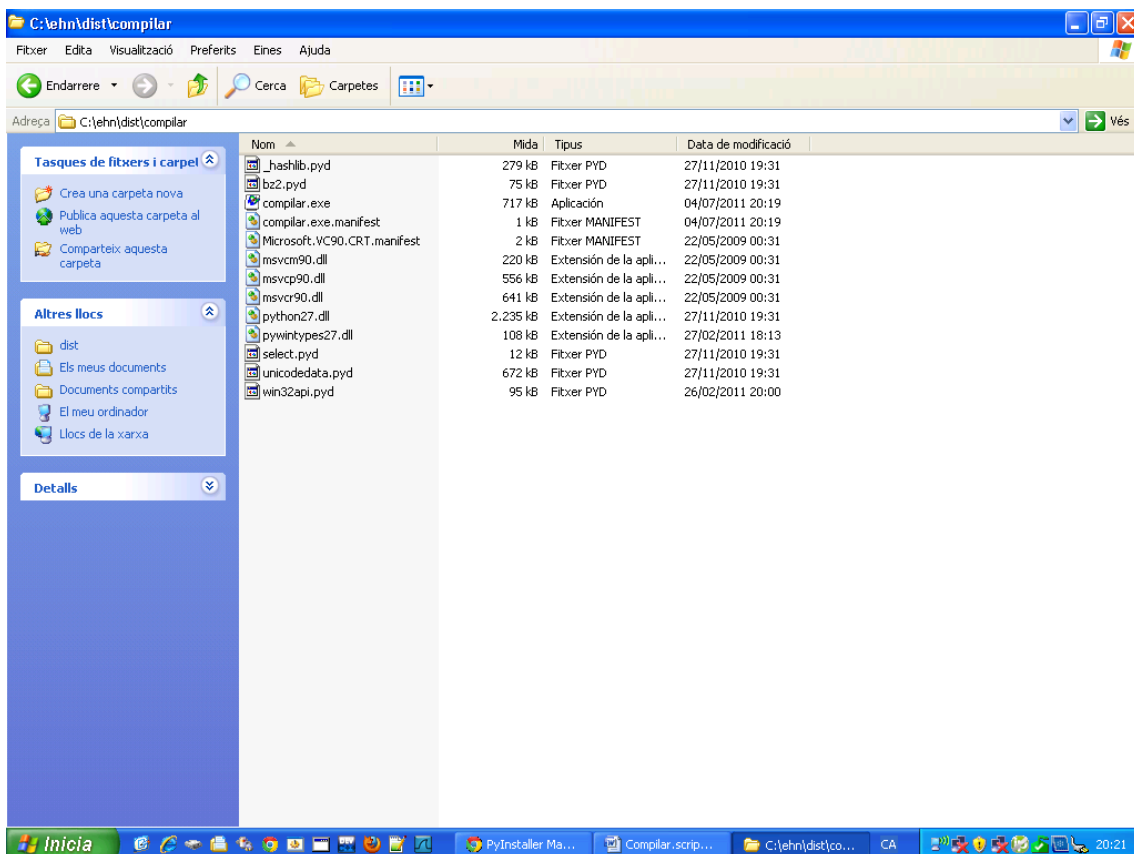
Y veremos la salida del comando...

Y un regalito...

```
Símbolo del sistema - python Build.py C:\ehn\compilar.spec
I: x86_Microsoft.VC90.CRT_1fc8b3b9a1e18e3b_9.0.21022.8_x-ww
Adding Microsoft.VC90.CRT to dependent assemblies of final executable
I: Searching for assembly x86_Microsoft.VC90.CRT_1fc8b3b9a1e18e3b_9.0.21022.8_x-ww...
I: Found manifest C:\WINDOWS\WinSxS\Manifests\x86_Microsoft.VC90.CRT_1fc8b3b9a1e18e3b_9.0.21022.8_x-ww_d08d0375.manifest
I: Searching for file msucr90.dll
I: Found file C:\WINDOWS\WinSxS\x86_Microsoft.VC90.CRT_1fc8b3b9a1e18e3b_9.0.21022.8_x-ww_d08d0375\msucr90.dll
I: Searching for file msucp90.dll
I: Found file C:\WINDOWS\WinSxS\x86_Microsoft.VC90.CRT_1fc8b3b9a1e18e3b_9.0.21022.8_x-ww_d08d0375\msucp90.dll
I: Searching for file msucm90.dll
I: Found file C:\WINDOWS\WinSxS\x86_Microsoft.VC90.CRT_1fc8b3b9a1e18e3b_9.0.21022.8_x-ww_d08d0375\msucm90.dll
I: Adding Microsoft.VC90.CRT.manifest
I: Adding msucr90.dll
I: Adding msucp90.dll
I: Adding msucm90.dll
I: Adding python27.dll dependency of python.exe
I: Skipping KERNEL32.dll dependency of python.exe
I: Analyzing C:\WINDOWS\system32\python27.dll
I: Dependent assemblies of C:\WINDOWS\system32\python27.dll:
I: x86_Microsoft.VC90.CRT_1fc8b3b9a1e18e3b_9.0.21022.8_x-ww
```

Y... ¡voilà! En la carpeta **C:\ehn** tengo un directorio **dist** con todos los archivos!

Vamos a examinar este directorio:

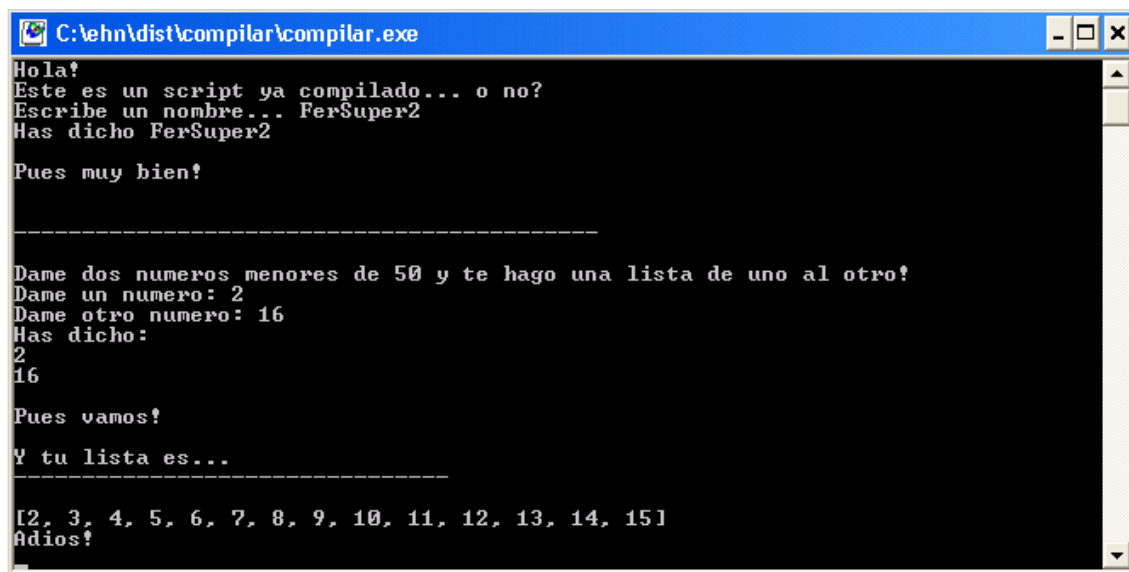


Tenemos un montón de cosas.

Vamos a verlas una por una:

- Hay diversos archivos **.pyd**, gracias a Root que están ahí...
- Observamos un archivo **compilar.exe** !!!!!!!!!!!!!
- Hay dos archivos **.manifest**, necesarios para el correcto funcionamiento después del compilado.
- También hay tres librerías **.dll**; si en el código del script **compilar.py** hubiera importado algo (como **import os, sys, socket**) generalmente habría más.
- Observamos la librería **python27.dll**: evidentemente, es el intérprete de Python 2.7 (el mío, en este caso, es el 2.7.2 de 32bits).
- También vemos la librería **pywintypes27.dll**: esta va ligada a nuestro **pywin32**, que sin él no sería posible.

Ahora voy a comprobar si el archivo compilado **compilar.exe** realmente funciona:



```
C:\ehn\dist\compilar\compilar.exe
Hola!
Este es un script ya compilado... o no?
Escribe un nombre... FerSuper2
Has dicho FerSuper2
Pues muy bien!

-----

Dame dos numeros menores de 50 y te hago una lista de uno al otro?
Dame un numero: 2
Dame otro numero: 16
Has dicho:
2
16
Pues vamos!
Y tu lista es...
-----
[2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15]
Adios!
```

¡FUNCIONA! Y además sin problema alguno.

Lo hemos conseguido: nuestro script **compilar.py** ya es **compilar.exe** .

Ahora vamos a ver más opciones de sacarle el jugo al PyInstaller...

4.- POR FIN! AMPLIANDO HORIZONTES

El momento *mágico* de la compilación es en el **Makespec**: este script posee unas opciones muy interesantes. La sintaxis del comando es:

```
python Makespec.py [opciones] ruta\al\script.py
```

Donde **[opciones]** pueden ser:

-OPCIÓN	--OPCIÓN	DESCRIPCIÓN
-F	--onefile	Creamos un solo archivo
-D	--onedir	Crea sólo un directorio(por defecto)
-K	--tk	Por si queréis incluir TCL/TK en el proyecto
-a	--ascii	No incluir Unicode
-d	--debug	Usa versiones en Debug (verbose mode) de los ejecutables
-w	--windowed --noconsole	Usa el subsistema de Windows para el programa
-c	--nowindowed --console	Usa el subsistema de la consola(por defecto)
-s	--strip	El programa correrá a través de un strip
-X	--upx	Si tienes UPX, se usará para comprimir el ejecutable
-o DIR	--out=DIR	Crea el archivo .spec en DIR
-p DIR	--paths=DIR	Define las PATH para la importación
	--icon=<icono.ico>	Define un icono para el ejecutable
-v ARCHIVO	--version=ARCHIVO	Añade el ARCHIVO como versión del ejecutable
-n NOMBRE	--name=NOMBRE	Otro nombre para el proyecto (por defecto, el nombre del script)

Como podéis ver, hay algunas opciones muy jugosas. El lío del **--bundle 1** o del **zipfile=None**, por ejemplo, se soluciona con la opción **-F** o **--onefile**.

5.- Y PARA TERMINAR...

Bueno, vamos a aclarar algunas cosas y después recordamos los pasos que hemos seguido:

Al extraer el archivo **.tar**, la carpeta resultante se define como **INSPATH**. Cuando creamos un **.spec**, se crea en **INSPATH\nombredelscript\nombredelscript.spec**.

Cuando hacemos el **Build**, se crean dos directorios, **build** y **dist**, en la carpeta del **.spec**, o bien (a veces) **en el directorio original del script**. Si al hacer el **.spec** sale algún error de “archivo perdido”, o “file not found”, puede ser a causa de que has movido el **.spec**, la **INSPATH** no está actualizada, etc. Todo esto se soluciona con el **Configure.py**.

También se crea un archivo llamado **warnnombredelscript.txt**, donde se graban todas las alertas ocurridas durante el proceso de compilación.

0.- PREVENTIVO: ejecutar **Configure.py**

1.- SPEC: crear el archivo **.spec** con el **Makespec.py**, con nuestras opciones personalizadas.

2.- CONSTRUIR: usar el archivo **Build.py** con el archivo **.spec**.

Distribuir tu aplicación compilada perfectamente es una realidad con **PyInstaller**.

Espero que os haya gustado y que os haya servido de ayuda.

Saludos!

WaAYa HaCK

5 de julio del 2011

foro.elhacker.net