

PyConES 2013

# Empaquetar es fácil, si sabes cómo

Juan Luis Cano, @Pybonacci  
Madrid, 2013-11-24

**Situación común:**  
Código que se quiere compartir  
(Proyecto PyFoo)

```
$ tree pyfoo/
```

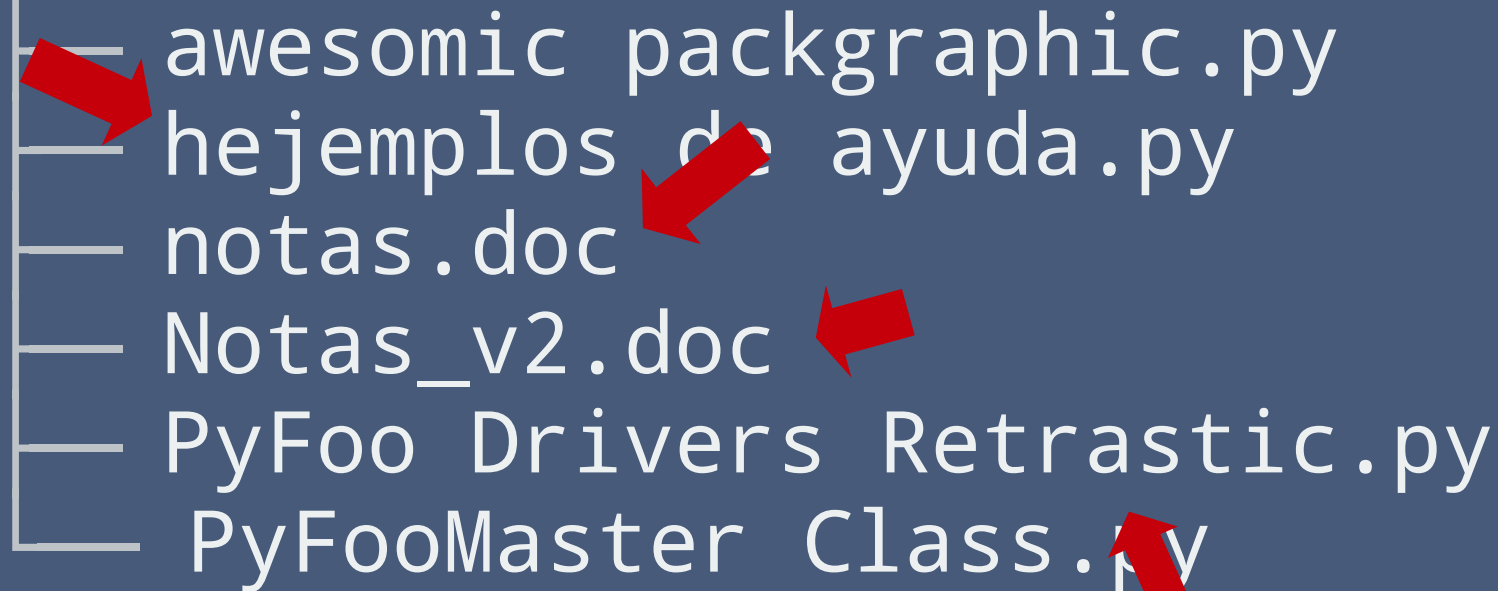
```
pyfoo/
```

```
├── awesome packgraphic.py  
├── ejemplos de ayuda.py  
├── notas.doc  
├── Notas_v2.doc  
├── PyFoo Drivers Retrastic.py  
└── PyFooMaster Class.py
```

```
$ tree pyfoo/
```

```
pyfoo/
```

```
├── awesome packgraphic.py  
├── ejemplos de ayuda.py  
├── notas.doc  
├── Notas_v2.doc  
├── PyFoo Drivers Retrastic.py  
└── PyFooMaster Class.py
```



**MARILIZE**

**LEGAJUANA**

# Problemas comunes

- Ausencia de *notas de instalación*\*
- Archivos sin ningún tipo de *jerarquía*
- Ausencia de *tests* o *documentación*
- ¿?

\*Lo de usar .doc no es tan común, afortunadamente  
†Y lo de las faltas de ortografía tampoco

# ¿De qué va esta charla?

- Cómo *organizar el código* de mi paquete
- Cómo hacerlo fácilmente instalable (por otros)
- Cómo incluir tests y documentación
- Cómo distribuirlo
- Trucos extra

# Jerarquía

- Es importante ser *predecible*

```
$ tree pyfoo
```

```
pyfoo
```

```
├── COPYING
├── doc
├── pyfoo
├── README.rst
├── setup.py
└── test
```



# Jerarquía

- *Siempre* necesitamos un README(.rst)
- *Siempre*\* necesitamos una licencia: COPYING, LICENSE
- Directorios:
  - pyfoo: código del paquete
  - doc: documentación
  - test: tests
- De setup.py hablaremos luego

\*Consulte legislación acerca de derechos de autor correspondiente

**Truco:** ¡Estudia el código de gente  
más lista que tú!\*

\*¡A ver cómo te crees que hemos aprendido todos!

# Paquetes y módulos

- **Módulos:** archivos `.py`
- Contienen todas las definiciones
- *Nombres cortos y en minúsculas* (PEP 8)
- Evitar guiones bajos `_` en lo posible
- Módulo = Unidad lógica

# Paquetes y módulos

- **Paquetes:** grupos de módulos bajo un espacio de nombres
- Cualquier directorio con `__init__.py`
- Fundamentales para categorizar módulos

# Paquetes y módulos

Ejemplo de Celery (<https://github.com/celery/celery>)

```
$ tree celery/  
celery/  
├── celery  
│   └── backends  
│       ├── amqp.py  
│       ├── base.py  
│       ├── __init__.py  
│       ├── mongodb.py  
│       ├── [...]  
│       └── redis.py
```

# Paquetes y módulos

Ejemplo de Celery (<https://github.com/celery/celery>)

```
$ python
```

```
>>> import celery.backends.mongodb
```

```
>>> from celery.backends import mongodb
```

Los paquetes me crean un *espacio de nombres*

# Paquetes FAQ

- ¿Qué es `__init__.py`?

Respuesta: código que se ejecuta al importar el paquete

```
$ cat pyfoo/backends/__init__.py
```

```
print("Hello, " + __package__)
```

```
$ python -q
```

```
>>> from pyfoo.backends import base
```

```
Hello, pyfoo.backends
```

```
>>> from pyfoo.backends import redis
```

```
>>>
```

# Paquetes FAQ

- ¿Qué debo poner en `__init__.py`?

Respuesta: nada (o, en general, poco). Pero...

```
>>> import pyfoo.backends
```

```
Hello, pyfoo.backends
```

```
>>> pyfoo.backends.base
```

```
Traceback (most recent call last):
```

```
  File "<stdin>", line 1, in <module>
```

```
AttributeError: 'module' object has no attribute  
'base'
```



# Paquetes FAQ

A veces conviene:

```
$ cat pyfoo/backends/__init__.py
```

```
from . import base
```

```
from . import mongodb
```

```
$ python -q
```

```
>>> import pyfoo.backends
```

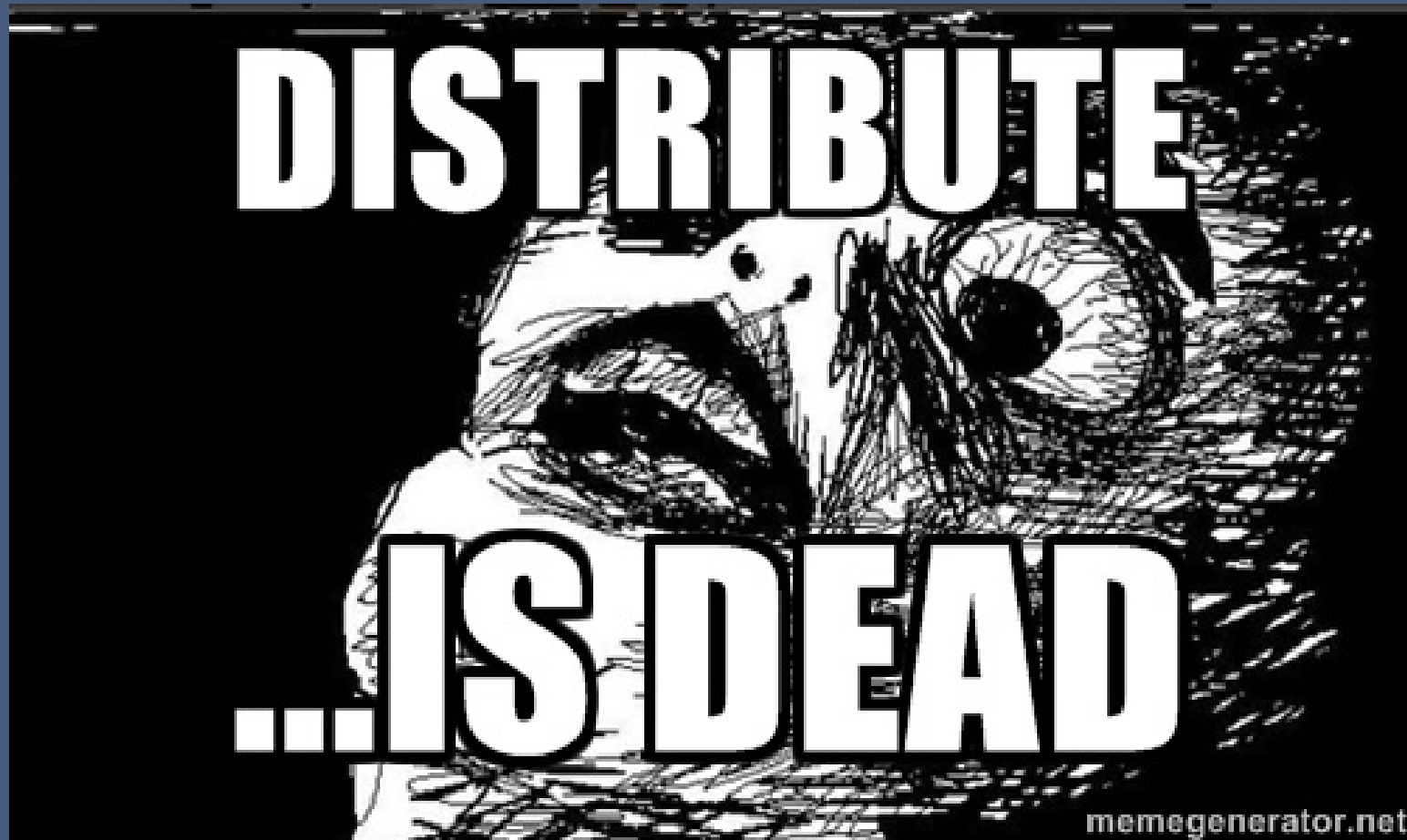
```
>>> pyfoo.backends.base
```

```
<module 'pyfoo.backends.base' from  
'./pyfoo/backends/base.py'>
```

# Distribuir tu paquete

distutils, setuptools, ~~distribute~~, ~~easy\_install~~...  
¡Olvida todo lo que sabes hasta ahora!

Distribuir tu paquete



# Preparando tu paquete

- distutils: biblioteca estándar, básico
- setuptools: extiende y mejora distutils
- pip: instalador de paquetes
- PEP 453: ¡pip en la biblioteca estándar en Python 3.4!
- pip depende de setuptools... de momento

# El archivo `setup.py`

```
from distutils.core import setup

setup(name="pyfoo", version="0.1",
      summary=open("README.rst").read(),
      author="Juan Luis Cano",
      author_email="juanlu001@gmail.com",
      license="BSD",
      url="http://foopy.github.io",
      packages=["foopy"])
```

# Instalando

```
$ python setup.py install
```

# ¡Importante!

Nunca, nunca, nunca, nunca  
usar ~~sudo~~ con pip o setup.py install

Referencias: [1] [2] [3] [4]

# ¡Importante!

En su lugar:

```
$ python setup.py install --user
```

```
$ pip install --user
```



# Dependencias

Primer método: `install_requires`

```
from setuptools import setup

setup(name="pyfoo", version="0.1", # ...
      install_requires=[
          "django", "requests"])
```

# Dependencias

Segundo método: requirements.txt

```
$ cat requirements.txt
```

```
Django==1.5
```

```
requests>=1.0
```

# Dependencias

- ¡Los métodos son diferentes!
- `install_requires`: «dependencias abstractas», bibliotecas
- `requirements.txt`: «dependencias concretas», aplicaciones, por defecto de PyPI

# Últimos preparativos

# Subiendo a PyPI

# El futuro

- Python 3.4 incorporará pip
- Numerosas mejoras y se sigue trabajando
- Formato wheel (¿presente?)

<http://pythonwheels.com/>

¿Preguntas?

Muchas gracias  
y hasta el año que viene :)

@Pybonacci  
<http://pybonacci.wordpress.com/>