Distribuer un projet Python

Présentation des outils les plus utilisées

Auteur: Jérémie DECOCK
Contact: jd.jdhp@gmail.com

Version: 0.1
Date: 01/05/2016

Licence: Creative Commons 4.0 (CC BY-SA 4.0)

Sommaire

> ***Avis***
>
> Veuillez noter que ce document est en cours de rédaction. Dans son état actuel, il n'est pas destiné à être lu par d'autres personnes que ses auteurs.

1 Introduction

Documentation "officielle":

- PyPA: https://www.pypa.io/
- PyPUG: https://python-packaging-user-guide.readthedocs.org/

Documentation intéressante:

- https://python-packaging-user-guide.readthedocs.org/
- http://sametmax.com/creer-un-setup-py-et-mettre-sa-bibliotheque-python-en-ligne-sur-pypi/ (nov 2012)
- **http://sametmax.com/travailler-sur-une-lib-externe-a-votre-projet-proprement-en-python/ (oct 2014)**

  astuce très propre et très utile pour ne pas avoir a résinstaller à chaque modification un programme ou une bibliothèque qu'on est en train de développer!

- http://stackoverflow.com/questions/6344076/differences-between-distribute-distutils-setuptools-and-distutils2 (sept 2014)

1.1 Voc

- Distribution package: https://python-packaging-user-guide.readthedocs.org/en/latest/glossary.html#term-distribution-package
- Wheel: https://python-packaging-user-guide.readthedocs.org/en/latest/glossary.html#term-wheel
- Egg: https://python-packaging-user-guide.readthedocs.org/en/latest/glossary.html#term-egg
- Source Distribution (or “sdist”): https://python-packaging-user-guide.readthedocs.org/en/latest/glossary.html#term-source-distribution-or-sdist

1.2 Installer un paquet en local

Petit tour d'horizon des principaux outils disponibles:

- distutils: basique mais intégré à la lib standard
- setuptools: standard de fait, plus riche mais pas dans la bibliothèque standard
  - https://setuptools.pypa.io/en/latest/setuptools.html
  - fourni aussi easy_install
- distutils2: abandonné
- dsitribute: mergé dans setuptools
- bento: non standard, non mature

1.3 Installer un paquet depuis internet

PyPI (http://pypi.python.org) : le site web où sont entreposés les paquets.

Petit tour d'horizon des principaux outils disponibles:

- easy_install
- pip
    - cf. http://sametmax.com/votre-python-aime-les-pip/
    - "pip est un easy_install en plus court à taper, plus puissant, plus souple, plus mieux".
    - pip va plus loin qu'easy_install, et permet carrément de gérer toutes les dépendances de son projet.
    - A partir des versions 2.7.9 et et 3.4, pip est fournit automatiquement avec Python.

2 Distutils

- https://wiki.python.org/moin/Distutils/Tutorial
- https://docs.python.org/3/library/distutils.html
- https://wiki.python.org/moin/Distutils/

Fichiers à la racine du projet:

- MANIFEST.in
- setup.py

Voir https://wiki.python.org/moin/Distutils/Tutorial

```
top
|-- package
|   |-- __init__.py
|   |-- module.py
|   `-- things
|       |-- cross.png
|       |-- fplogo.png
|       `-- tick.png
|-- runner
|-- MANIFEST.in
|-- README
`-- setup.py
```

2.1 MANIFEST.in

Ce fichier contient la liste des fichiers qui ne font pas partie du "paquet python" mais qu'on souhaite ajouter au "paquet" (typiquement fichiers de données tels que la documentation, les images, les fichiers json, les fichiers de configuration, etc.)...

2.2 setup.py

...

2.3 Usage

Help (seulement les commandes "communes"):

```
python3 setup.py -h
```

Help (toutes les commandes):

```
python3 setup.py --help-commands
```

Commands:

```
Standard commands:
  build            build everything needed to install
  build_py         "build" pure Python modules (copy to build directory)
  build_ext        build C/C++ extensions (compile/link to build directory)
  build_clib       build C/C++ libraries used by Python extensions
  build_scripts    "build" scripts (copy and fixup #! line)
  clean            clean up temporary files from 'build' command
  install          install everything from build directory
  install_lib      install all Python modules (extensions and pure Python)
  install_headers  install C/C++ header files
  install_scripts  install scripts (Python or otherwise)
  install_data     install data files
  sdist            create a source distribution (tarball, zip file, etc.)
  register         register the distribution with the Python package index
  bdist            create a built (binary) distribution
  bdist_dumb       create a "dumb" built distribution
  bdist_rpm        create an RPM distribution
  bdist_wininst    create an executable installer for MS Windows
  check            perform some checks on the package
  upload           upload binary package to PyPI
```

Install:

```
python3 setup.py install
```

Make a source distribution file (sdist != bdist):

```
python3 setup.py sdist
```

Uninstall:

"Notably, distutils does not have an uninstall command at this time." (http://stackoverflow.com/questions/402359/how-do-you-uninstall-a-python-package-that-was-installed-using-distutils)

Register to PyPI:

```
python3 setup.py register
```

Submitting to PyPI:

```
python3 setup.py sdist upload
```

Clean up temporary files from 'build' command:

```
python3 setup.py clean
```

Create a built (binary) distribution:

```
bdist
```

Create a "dumb" built distribution:

```
bdist_dumb
```

Create an RPM distribution:

```
bdist_rpm
```

Create an executable installer for MS Windows:

```
bdist_wininst
```

3 Setuptools

Installation sur Debian:

```
apt-get install python-setuptools
```

3.1 Distutils VS setuptools en pratique

Le fichier setup.py peut utiliser distutils ou setuptools...

distutils:

```
from distutils.core import setup
```

setuptools:

```
from setuptools import setup
```

3.2 Usage

Help (seulement les commandes "communes"):

```
python3 setup.py -h
```

Help (toutes les commandes):

```
python3 setup.py --help-commands
```

Commands:

```
Standard commands:
  build             build everything needed to install
  build_py          "build" pure Python modules (copy to build directory)
  build_ext         build C/C++ extensions (compile/link to build directory)
  build_clib        build C/C++ libraries used by Python extensions
```

```
  build_scripts     "build" scripts (copy and fixup #! line)
  clean             clean up temporary files from 'build' command
  install           install everything from build directory
  install_lib       install all Python modules (extensions and pure Python)
  install_headers   install C/C++ header files
  install_scripts   install scripts (Python or otherwise)
  install_data      install data files
  sdist             create a source distribution (tarball, zip file, etc.)
  register          register the distribution with the Python package index
  bdist             create a built (binary) distribution
  bdist_dumb        create a "dumb" built distribution
  bdist_rpm         create an RPM distribution
  bdist_wininst     create an executable installer for MS Windows
  check             perform some checks on the package
  upload            upload binary package to PyPI

Extra commands:
  bdist_egg         create an "egg" distribution
  develop           install package in 'development mode'
  egg_info          create a distribution's .egg-info directory
  test              run unit tests after in-place build
  setopt            set an option in setup.cfg or another config file
  upload_docs       Upload documentation to PyPI
  alias             define a shortcut to invoke one or more commands
  rotate            delete older distributions, keeping N newest files
  install_egg_info  Install an .egg-info directory for the package
  easy_install      Find/get/install Python packages
  saveopts          save supplied options to setup.cfg or other config file
```

Publier sur PyPI:

```
python3 setup.py register
python3 setup.py sdist upload

./setup.py register
[...]
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]:
2
Username: [...]
Password: [...]
 Confirm: [...]
   EMail: [...]
Registering [...] to https://pypi.python.org/pypi
You will receive an email shortly.
Follow the instructions in it to complete registration.
```

Mot de pass enregistré dans ~/.pypirc

4 PyPI

TODO...

5 Easy install

TODO...

6 Pip

http://sametmax.com/votre-python-aime-les-pip/ (sept 2012)

Linux:

```
pip install pyax12
pip install --upgrade pyax12
pip uninstall pyax12
```

Pour installer des versions de pre-release ou versions de développement:

```
pip install --pre  pyax12
```

(`--pre` include pre-release and development versions, by default, pip only finds stable versions)

Windows:

```
py -m pip install pyax12
py -m pip install --upgrade pyax12
py -m pip uninstall pyax12
```

6.1 Twine

Jusqu'en decembre 2013, les communications avec PyPI n'étaient pas cryptées, le mot de passe etait transmi en clair ! https://anthony-monthe.me/weblog/2015/08/23/never-use-python-setuppy-upload/

Twine était une alternative sécurisée https://github.com/pypa/twine.

Depuis le problème est corrigé http://bugs.python.org/issue12226

7 Environnements virtuels Python

UPDATE: à lire: - http://masnun.com/2016/04/10/python-pyenv-pyvenv-virtualenv-whats-the-difference.html - https://realpython.com/blog/python/python-virtual-environments-a-primer/ - http://pewetheb.blogspot.fr/2015/11/virtual-environment-in-python.html (ok) - venv https://www.python.org/dev/peps/pep-0405/ - venv https://docs.python.org/3/library/venv.html - virtualenv https://virtualenv.pypa.io/en/stable/ - virtualenv et virtualenvwrapper http://sametmax.com/les-environnement-virtuels-python-virtualenv-et-virtualenvwrapper/ (oct 2012) - http://homework.nwsnet.de/releases/8aff/

> "la notion simple d'avoir des installations de Python isolées de l'OS, et séparées les unes des autres pour chaque projet" (sametmax.com)

Pour résumer:

Ancien outil tiers (toujours très utilisé ?) pour virtualliser les paquets python: - virtualenv : outil externe populaire qui était là bien avant venv. à l'avantage de fonctionner avec de vielles versions de Python

(<3.3) - virtualenvwrapper: basically, this is a script to wrap virtualenv so that it is easier for us to work on multiple virtual environments.

La nouvelle alternative officielle à virtualenv pour virtualliser les paquets python: - venv : une librairie intégrée à Python depuis Python 3.3 pour faire la même chose que virtualenv mais intégré de base dans python (c'est le seul outil "officiel" de la liste). It looks pretty much similar to virtualenv but "venv has less features compared to virtualenv". - pyvenv_ (à ne pas confondre avec pyenv): un script de venv (déprécié depuis Python 3.6 au profit de *python3 -m venv*)

Outil tiers (scripts shell) pour installer plusieurs interprètes Python: - pyenv (à ne pas confondre avec pyvenv) : outil tiers (scripts shell) pour installer plusieurs *interprètes* Python sur un même système (par ex. CPython 2.6, CPython 3.4, Pypy, IronPyhon, ...). Rien à voir avec les outils précédents mais son nom entretient une certaine confusion avec venv et pyvenv - Pythonbrew: ancien nom de pyenv (?)

Misc: - buildout : puissant mais compliqué

7.1 Virtualenv

Virtualenv est un outil **tiers** développé par Ian Bicking qui permet d'isoler des site packages

- https://www.python.org/dev/peps/pep-0405/#id4
- https://virtualenv.pypa.io/en/stable/

Tout est très bien expliqué dans le tutoriel suivant:

- http://sametmax.com/les-environnement-virtuels-python-virtualenv-et-virtualenvwrapper/ (oct 2012)

Pour résumer, principales commandes:

Créer un environnement virtuel:

```
virtualenv <PATH>
```

où `<PATH>` est le chemin du répertoire où créer l'environnement virtuel.

Créer un environnement virtuel pour une version spécifique de python (par ex python 3.4):

```
virtualenv -p /usr/bin/python3.4 <PATH>
```

Créer un environnement virtuel vierge (ie qui n'hérite pas des libs du système et a seulement accès aux libs standards de Python):

```
virtualenv --no-site-packages <PATH>
```

Pour desactiver setuptools et pip:

```
--no-setuptools --no-pip
```

Entrer dans l'environnement virtuel (Unix):

```
source <PATH>/bin/activate
```

Entrer dans l'environnement virtuel (Windows):

```
<PATH>\Scripts\activate.bat
```

Sortir de l'environnement virtuel:

```
deactivate
```

7.2 Virtualenvwrapper

Pour passer facilement d'un environnement virtuel à l'autre.

Cf:

- http://sametmax.com/les-environnement-virtuels-python-virtualenv-et-virtualenvwrapper/ (oct 2012)

7.3 venv

- https://docs.python.org/3/library/venv.html#venv-def
- La PEP qui défini venv: https://www.python.org/dev/peps/pep-0405/

Definition (https://www.python.org/dev/peps/pep-0405/#abstract):

> This PEP proposes to add to Python a mechanism for lightweight "virtual environments" with their own site directories, optionally isolated from system site directories. Each virtual environment has its own Python binary (allowing creation of environments with various Python versions) and can have its own independent set of installed Python packages in its site directories, **but shares the standard library with the base installed Python**.

Motivations: https://www.python.org/dev/peps/pep-0405/#motivation

> "To use venv, the workflow is quite similar to virtualenv. First, you create the virtual environment directory. Next, activate the virtual environment."
>
> http://pewetheb.blogspot.fr/2015/11/virtual-environment-in-python.html

Intégré à Python v3.3 et plus.

Installation sur Debian/Ubuntu/... (il est officiellement "intégré" à Python3 mais Debian a éclaté Python3 en plusieurs paquets):

```
# aptitude install python3-venv
```

- Documentation officielle: https://docs.python.org/3/library/venv.html
- http://askubuntu.com/questions/279959/how-to-create-a-virtualenv-with-python3-3-in-ubuntu (cf. la 2e réponse)

Sur Debian/Ubuntu, il y a un "bug" (en réalité c'est un choix des dev Debian TODO...): par défaut, les sitepackages ne sont pas accessibles depuis l'environnement virtuel Python (par exemple `import numpy` échoue même si numpy est installé sur le système...). Pour résoudre ce problème, il suffit de déclarer la variable d'environnement `PYTHONPATH` comme suit à la fin du fichier `~/.bashrc`:

```
export PYTHONPATH=${PYTHONPATH}:/usr/lib/python3/dist-packages
```

et de recharger le .bashrc en rouvrant le terminal ou en tapant:

```
source ~/.bashrc
```

Pour plus de détails, cf. http://stackoverflow.com/questions/19210964/how-to-get-virtualenv-to-use-dist-packages-on-ubuntu.

Créer un environnement virtuel:

```
pyvenv <PATH>
```

où <PATH> est le chemin du répertoire où créer l'environnement virtuel.

Entrer dans l'environnement virtuel (Unix):

```
source <PATH>/bin/activate
```

Sortir de l'environnement virtuel:

```
deactivate
```

7.4 Pyenv

https://github.com/yyuu/pyenv

pyenv does...

- Let you change the global Python version on a per-user basis.
- Provide support for per-project Python versions.
- Allow you to override the Python version with an environment variable.
- Search commands from multiple versions of Python at a time. This may be helpful to test across Python versions with tox.

In contrast with pythonbrew and pythonz, pyenv does not...

- Depend on Python itself. pyenv was made from pure shell scripts. There is no bootstrap problem of Python.
- Need to be loaded into your shell. Instead, pyenv's shim approach works by adding a directory to your $PATH.
- Manage virtualenv. Of course, you can create virtualenv yourself, or pyenv-virtualenv to automate the process.

Pyenv n'a rien à voir avec *virtualenv* et *pyvenv*.

> "*Pyenv* allows you to easily install multiple copies and multiple flavors of the Python interpreter. So you can not only install different versions of CPython, you can also install PyPy, Jython, Stackless Python and their different versions. [...]
>
> *pyvenv* and *virtualenv* allow you to create virtual environments so we can isolate our project dependencies. Why are they helpful? Say for example, you have one project which uses Django 1.6 still while your newer projects start with 1.9. When you install one version of Django, it replaces the other one, right? Virtual environments can rescue us from such situation."
>
> http://masnun.com/2016/04/10/python-pyenv-pyvenv-virtualenv-whats-the-difference.html
>
> "Pyenv and virtualenv are very different tools that work in different ways to do different things:
>
> Pyenv is a bash extension - will not work on Windows - that intercepts your calls to python, pip, etc., to direct them to one of several of the system python tool-chains. So you always have all the libraries that you have installed in the selected python version available - as such it is good for users who have to switch between different versions of python.
>
> VirtualEnv, is pure python so works everywhere, it makes a copy of, optionally as specific version of, python and pip local to the activate environment which may or may not include links to the current system tool-chain, if it does not you can install just a known subset of libraries into that environment. As such it is almost certainly much better for testing and deployment as you know exactly which libraries, at which versions, are used and a global change will not impact your module."

http://stackoverflow.com/questions/29950300/what-is-the-relationship-between-virtualenv-and-pyenv

Un tutoriel pour pyenv: https://amaral.northwestern.edu/resources/guides/pyenv-tutorial

8 Licence

Ce document est distribué selon les termes de la licence Creative Commons 4.0 (CC BY-SA 4.0).