Tutoriel distutils

Version 0.1

Cédric Campguilhem

22 May 2011

Table des matières

1 Introduction **1**
- 1.1 Qu'est ce que distutils ? . . . 1
- 1.2 Objectifs du tutoriel . . . 1
- 1.3 Pourquoi utiliser distutils ? . . . 1
- 1.4 Notes à propos de ce tutoriel . . . 1
- 1.5 Notes à propos de l'auteur . . . 2
- 1.6 Quelques définitions avant de commencer . . . 2

2 Packager un script simple **3**
- 2.1 Contexte . . . 3
- 2.2 Configuration basique . . . 4
- 2.3 Création du paquet . . . 4
- 2.4 Configuration avancée . . . 6
- 2.5 Les fichiers `MANIFEST` et `MANIFEST.in` . . . 6
- 2.6 Installation de votre projet . . . 7
- 2.7 Conclusions . . . 8

3 Indices et tables **9**

CHAPITRE 1

Introduction

1.1 Qu'est ce que distutils ?

distutils est un module de la librairie *standard* de Python qui sert à la fois à **packager** des projets Python et à les **installer**.

Pour utiliser **distutils** vous n'avez tout simplement rien à installer en plus de votre distribution Python.

1.2 Objectifs du tutoriel

Ce tutoriel a pour objectif de vous apprendre à utiliser **distutils** pour déployer vos projets Python.

Pour ce faire, nous aborderons 3 thèmes :
– le packaging de scripts simples
– le packaging de modules simples
– le packaging d'applications (IHM [1] incluant des images)
Je vous proposerai également une organisation de répertoire pour votre projet (sources, documentation, exemples...).

A la fin de la lecture de ce tutoriel vous aurez donc les clés en main pour utiliser **distutils** pour déployer vos projets Python.

1.3 Pourquoi utiliser distutils ?

– Il s'agit de la façon standard proposée par Python de diffuser et d'installer des projets Python.
– Les outils permettant des configurations plus poussées du packaging et de l'installation de projets (setuptools et distribute par exemple), sont tous les deux basés sur **distutils**. Préparer son projet Python pour utiliser **distutils** ne sera donc pas une perte de temps.

1.4 Notes à propos de ce tutoriel

Ce tutoriel présente **distutils** des versions 2.x de Python pour plusieurs raisons :

1. Interface Homme Machine

– Malgré l'existence de Python 3.x, les versions 2.x continuent à être très utilisées.
– Je n'ai personnellement pas assez de recul avec Python 3.x.
– Je pense que la très grande majorité des fonctionnalités que je présente sont également disponibles dans la version 3 de Python.

Vous ne trouverez pas dans ce tutoriel :

– La référence complète des fonctionnalités de **distutils** (il y a la documentation officielle pour cela)
– Bien que **distutils** serve à la fois à *packager* et à *installer* des projets Python, je m'étendrais majoritairement sur le *packaging* des projets. Vous pouvez trouver des informations sur la documentation officielle.
– Je n'aborderai pas les aspects relatifs au déploiement de projets Python non purs (c'est à dire incluant du code source écrit dans un autre langage comme C, C++ ou Fortran).

1.5 Notes à propos de l'auteur

Je développe des applications Python depuis 6 ans dans un domaine industriel avec des utilisateurs un peu partout dans les monde dans le cadre de mon travail. Plusieurs environnements de production existent, aussi bien sur Windows, Linux qu'Unix. Le passage à **distutils** pour l'ensemble de ces applications nous a permis de rationnaliser et de faciliter le déploiement et l'installation de ces applications.

1.6 Quelques définitions avant de commencer

Je tiens à préciser certains termes que vous me verez utiliser régulièrement au cours de ce tutoriel :

projet Ensemble des fichiers sources, exemples, ressources (images, fichiers de configuration,...), documentation et tests.

module Un module est un fichier source Python dont la finalité est d'être importé.

script Un script est un module Python dont la finalité est d'être exécuté.

librairie Une librairie est un ensemble de modules Python, éventuellement organisés en packages dont la finalité est d'être importée.

package Un package Python est un répertoire contenant un fichier __init__.py, éventuellement vide. Un package sert à regrouper des modules Python.

application Une application peut être composé de plusieurs librairies et plusieurs scripts mais également de fichiers ressources (images, fichiers de configuration) dont la finalité principale est d'être exécuté.

packager Réaliser un paquet de votre projet en vue de le rendre disponible à d'autres utilisateurs et de faciliter son installation.

CHAPITRE 2

Packager un script simple

2.1 Contexte

Vous avez écrit un script simple, par exemple :

```
#!/usr/bin/env python

import os
import sys


def tree(top, depth=0, char='|'):
    """Directory tree generator (deep-first search algorithm)"""
    if depth == 0:
        print top
    elif depth == 1:
        print '|-- ' + os.path.basename(top)
    else:
        print '|   ' + '    ' * (depth-2) + '%s-- ' % char + \
            os.path.basename(top)
    names = os.listdir(top)
    names.sort()
    n = len(names)
    for i, name in enumerate(names):
        char = '`' if (i == n-1) else '|'
        fullname = os.path.join(top, name)
        if os.path.isdir(fullname):
            tree(fullname, depth=depth+1, char=char)
        else:
            if depth == 0:
                print "%s-- " % char + name
            else:
                print '|   ' + '    ' * (depth-1) + '%s-- ' % char + name


if __name__ == '__main__':
    try:
        path = sys.argv[1]
    except IndexError:
```

```
        path = os.curdir
    tree(path)
```

qui permet, à la manière de la commande tree sur Linux d'afficher l'arborescence de fichiers en mode texte dans un terminal et vous voulez partager ce script `tree.py`.

2.2 Configuration basique

Le coeur de l'utilisation de **distutils** réside dans la création d'un script Python de configuration. Conventionnellement, ce fichier est nommé `setup.py`.

Notre répertoire sera organisé comme ceci :

```
.
|-- setup.py
`-- tree.py
```

c'est à dire que les scripts `tree.py` et `setup.py` sont dans le même répertoire.

Le fichier `setup.py` quant à lui sera :

```
#!/usr/bin/env python

from distutils.core import setup

setup(name='tree',
      version='1.0',
      scripts=['tree.py'],
     )
```

Ce script comporte deux instructions :
– l'import de la fonction `setup` depuis le module `distutils.core`
– l'appel de cette fonction `setup` avec un jeu de paramètres.
Les deux premiers paramètres de la fonction `setup` parlent d'eux même :
– *name* permet de définir le nom de votre projet Python
– *version* précise le numéro de version
Le dernier paramètre *scripts* est une liste dans lequel chaque élément est un script que vous voulez packager.

Par exemple, si vous aviez eu un second script `autre.py` ce paramètre aurait été :

```
scripts = ['tree.py', 'autre.py']
```

Et c'est tout ! Vous venez de rendre possible le déploiement de votre script grâce à **distutils**.

2.3 Création du paquet

Une fois que le fichier `setup.py` est opérationnel, il n'y a plus qu'à exécuter le script `setup.py` afin de créer un paquet :

```
python setup.py sdist
```

Dans le jargon **distutils**, `sdist` est une **commande**. `sdist` est une abréviation pour *source distribution*.

Après exécution de cette commande, votre répertoire projet va changer et ressembler à ceci :

```
.
|-- MANIFEST
|-- dist
|   `-- tree-1.0.tar.gz
|-- setup.py
`-- tree.py
```

Et si vous êtes vigilants, vous allez vous rendre compte que vous avez quelques avertissements lors de l'exécution de `setup.py` :

```
warning: sdist: missing required meta-data: url
warning: sdist: missing meta-data: either (author and author_email) or (maintainer and maintainer_ema
warning: sdist: manifest template 'MANIFEST.in' does not exist (using default file list)
warning: sdist: standard file not found: should have one of README, README.txt
```

Rien de grave rassurez vous ! Avant de voir ce qui n'a pas bien fonctionné, regardons ce qui a bien fonctionné.

Vous disposez à présent d'un répertoire `dist` contenant le paquet que vous venez de créer : `tree-1.0.tar.gz`.

Quelques remarques :
– le nom du package correspond aux paramètres *nom* et *version* que vous avez passés à la fonction `setup` dans le script `setup.py`.
– ce package possède l'extension tar.gz (c'est ce que l'on appelle une tarball). Sur Linux c'est le format par défaut, sur Windows on aurait obtenu un fichier `tree-1.0.zip`. Mais c'est exactement le même principe, c'est une archive compressée contenant les scripts de votre projet.

Parce qu'on n'est jamais trop prudents, vérifions le contenu de cette archive. Je vous rappelle que l'on peut la désarchiver grace à la commande tar sur Linux :

```
cd dist
tar -zxf tree-1.0.tar.gz
```

Le répertoire tree-1.0 est comme ceci :

```
tree-1.0
|-- PKG-INFO
|-- setup.py
`-- tree.py
```

On retrouve donc bien notre script `tree.py` ainsi que le script `setup.py` qui nous a servi à configurer **distutils**, mais également un fichier nommé `PKG-INFO` qui contient les informations suivantes :

```
Metadata-Version: 1.0
Name: tree
Version: 1.0
Summary: UNKNOWN
Home-page: UNKNOWN
Author: UNKNOWN
Author-email: UNKNOWN
License: UNKNOWN
Description: UNKNOWN
Platform: UNKNOWN
```

C'est un fichier qui collecte les méta-données relatives au projets que nous avons (ou non) renseignées dans le script `setup.py`. On retrouve notamment les paramètres *name* et *version*.

2.4 Configuration avancée

Revenons maintenant aux avertissements que nous avons pu lire sur notre terminal au moment de l'exécution du script `setup.py` :
– Il nous manque un paramètre *url* pour préciser le page internet du projet.
– Il nous manque un paramètre *author* et *author_email* pour préciser l'auteur du projet.
– Il nous manque un fichier README pour donner des informations sur notre projet.
Je reviendrai sur l'avertissement au sujet du fichier `MANIFEST.in` un peu plus tard.

Voici comment nous pouvons modifier le fichier `setup.py` de notre projet pour qu'il définisse ces nouvelles informations :

```
#!/usr/bin/env python

from distutils.core import setup

setup(name='tree',
      version='1.0',
      author='Cedric Campguilhem',
      author_email='campguilhem@mail.com',
      url='http://tree.project.fr',
      scripts=['tree.py'],
     )
```

Nous pouvons également définir un fichier README comme suit. Le but de ce fichier est de donner des informations essentielles à propos de votre projet :
– comment l'installer
– comment l'utiliser
Voici le fichier README que je vous propose :

```
Tree

Procédure d'installation
========================

python setup.py install

Utilisation
===========

tree.py [path]
```

2.5 Les fichiers `MANIFEST` et `MANIFEST.in`

Comme nous avons pu le voir lorsque nous avons réalisé le paquet auparavant, un fichier `MANIFEST` a été crée dans le répertoire du projet. De plus, nous avons eu un avertissement de la part de **distutils** à propos de l'absence du fichier `MANIFEST.in`.

Le fichier `MANIFEST` est produit par **distutils**, c'est en fait un listing de tous les fichiers qui ont été mis dans votre paquet.

Si vous ouvrez ce fichier vous allez voir :

```
setup.py
tree.py
```

Ce qui correspond bien à ce que l'on voulez.

Le fichier `MANIFEST.in` quant à lui permet de lister des fichiers supplémentaires à ajouter. Et c'est votre rôle de développeur de le remplir.

Si vous ne fournissez pas ce fichier, **distutils** utilise le fichier MANIFEST qu'il a précédemment généré. C'est ce que nous indique cet avertissement :

```
warning: sdist: manifest template 'MANIFEST.in' does not exist (using default file list)
```

Je vous conseille très vivement de **toujours** définir par vous même un fichier `MANIFEST.in`, même vide. En effet, les fichiers ajoutés au paquet par distutils peuvent parfois ne pas correspondre à ce à quoi on s'attend.

Dans notre cas, on peut ajouter le fichier `README` au paquet en écrivant le fichier `MANIFES.in` suivant :

```
include README
```

En pratique, ce n'est pas nécessaire, **distutils** ajoute de lui même les fichiers suivants :
– le script `setup.py`,
– les scripts que vous avez défini dans le `setup.py` (ici `tree.py`),
– le fichier README ou README.txt s'il existe.
Nous pouvons à présent re-générer le paquet comme précedemment :

```
python setup.py sdist
```

Ce coup-ci, plus d'avertissements de la part de **distutils** !

Vous pouvez également constater que l'ancienne archive a été écrasée.

2.6 Installation de votre projet

Mettons nous à la place d'un utilisateur qui a récupéré le paquet que vous avez généré : `tree-1.0.tar.gz`.

Nous allons désarchiver ce paquet :

```
tar -zxf tree-1.0.tar.gz
```

En lisant le fichier *README* présent, nous voyons qu'il suffit de lancer la commande suivante pour installer le paquet :

```
python setup.py install
```

`install` est une commande **distutils**, comme `sdist` que nous avons vu avant dans ce tutoriel.

C'est là que la magie de **distutils** va opérer, c'est à dire qu'en fonction de la plateforme (Linux, Windows, Mac,...) il va déterminer tout seul où installer le script `tree.py`.

Sur Windows, ce sera par exemple dans `C:\Python26\Scripts\tree.py`. Sur Linux, ce sera par défaut dans `/usr/bin`.

La commande `install` vous permet toutefois de préciser un autre répertoire d'installation (par exemple si vous n'avez pas les autorisations nécessaires pour écrire dans les emplacements susnommés).

Voici la marche à suivre :

```
python setup.py install --prefix=/home/cedric/python
```

Et cela installera `tree.py` dans le répertoire `/home/cedric/python/bin`. Afin de pouvoir l'utiliser de n'importe où, j'ai besoin de redéfinir la variable d'environnement `PATH` :

```
export PATH=$PATH:/home/cedric/python/bin
```

Et voilà, j'ai installé le paquet **tree** sur mon ordinateur !

2.7 Conclusions

Dans cette partie, nous avons appris à :
– écrire le script setup.py de **distutils**,
– écrire les fichiers `README` et `MANIFEST.in` qui sont attendus par **distutils**,
– à générer le paquet en utilisant la commande `sdist`,
– à installer le paquet en utilisant la commande `install`.
Comme on a pu le voir, la mise en place de ces fichiers pour **distutils** est très rapide et nous à déjà permis de faciliter le déploiement et l'installation de notre projet.

CHAPITRE 3

Indices et tables

– *Index*
– *Module Index*
– *Search Page*