Python
Organisation d'un projet python
pip, pyenv, pipenv, git

Sébastien Bonnegent, Nicolas Delestre

Installation de *frameworks*, *packages*, etc.

Pourquoi ne pas utiliser `dpkg`

- Votre programme ne sera pas portable
- La mise à jour de l'OS peut rendre votre programme inutilisable
- L'OS ne propose peut-être pas la dernière version d'une bibliothèque

`pip`

- Python possède son propre gestionnaire de paquets : `pip`
- Il permet d'installer, de mettre à jour de désinstaller des paquets
- Commandes principales :

```
  install      Install packages.
  download     Download packages.
  uninstall    Uninstall packages.
  freeze       Output installed packages in requirements format.
  list         List installed packages.
  show         Show information about installed packages.
...
```

Avantages, inconvénients

Avantages

- Multi plateformes
- Intégré de base à python
- Avoir n'importe quelles versions des bibliothèques

Inconvénients

- Impossible de faire cohabiter des versions différentes de paquets. Cela peut poser des problèmes lorsque des applications python utilisent des paquets avec des versions non compatibles
- Impossible d'utiliser une version spécifique de python

Solution

- Il faudrait pouvoir isoler les applications python du système

Installation

Caractéristiques

- Permet d'utiliser n'importe quelle version de python

Installation

```
$ git clone https://github.com/pyenv/pyenv.git ~/.pyenv
$ echo 'export PYENV_ROOT="$HOME/.pyenv"' >> ~/.bashrc
$ echo 'export PATH="$PYENV_ROOT/bin:$PATH"' >> ~/.bashrc
$ echo -e 'if command -v pyenv 1>/dev/null 2>\&1; then\n \
           eval "$(pyenv init -)"\nfi' >> ~/.bashrc
$ exec "$SHELL"
```

Utilisation

Exemple d'utilisation

```
$ python3 -V
Python 3.6.7
$ pyenv versions
* system (set by /home/bonnegent/.pyenv/version)
$ pyenv install 3.7.3
# ... installation dans ~/.pyenv/versions/3.7.3/

$ pyenv versions
* system (set by /home/bonnegent/.pyenv/version)
  3.7.3
$ pyenv global 3.7.3
$ python3 -V
Python 3.7.3
$ pyenv global system
$ python3 -V
Python 3.6.7
```

venv 1 / 3

Caractéristiques

- Historiquement `virtualenv`, intégré à python à partir de la version 3.4
- Isole les applications python du système d'exploitation
- Utilisation de `pip` pour l'installation des paquets
- Gestion des environnements à l'aide du module `venv`

venv 2 / 3

Création de l'environnement (bonne pratique)

1. Création du répertoire de l'application : `mkdir NomDuProjet && cd NomDuProjet`
2. Création de l'environnement virtuel : `python3 -m venv venv`
   - Création d'un répertoire `venv` dans le projet (avec une arborescence « système » (répertoires bin, include, lib, etc.)
   - Par défaut n'utilise pas les paquets systèmes
3. Activation de l'environnement : `source venv/bin/activate`
   - les variables systèmes sont modifiées
   - le prompt est modifié
4. Mise à jour de pip et installation des paquets
5. Enregistrement de la configuration : `pip freeze > requirements.txt`

venv 3 / 3

Utilisation de l'environnement

- La version de base de `python` est `python3`
- `ipython` n'est pas installé de base (attention son installation avec `pip` donne accès à `ipython` et `ipython3`)
- Configuration si nécessaire de `PYTHONPATH`

Sortir de l'environnement

- Commande : `deactivate`

Reprise d'une configuration

1. Création et activation de l'environnement virtuel (étapes 2 et 3)
2. Installation des paquets :
   `pip install -r requirements.txt`
3. Configuration si nécessaire de `PYTHONPATH`

L'arme ultime

Description

- `https://docs.pipenv.org/en/latest/`
- Surcouche à venv et à pip
- Outil recommandé par python et les mainteneurs de pip
- Gestion automatique du répertoire du virtualenv
- Plusieurs environnements possibles (prod et dev)
- Mise à jour facile
- Contrainte sur la version de python
- Différence entre les dépendances nécessaires et les dépendances des dépendances

Installation

```
$ sudo pip3 install pipenv
$ sudo apt install pipenv
```

Mise en place

Pour un nouveau projet

```
$ pipenv --python 3.7
Virtualenv location: /home/bonnegent/.local/share/virtualenvs/test-hxkKlP5o
Creating a Pipfile for this project...
$ pipenv install ipython
$ pipenv install black==19.3b0 --dev
$ pipenv install pytest --dev
```

Mise à jour

```
$ # affichage des mises à jours disponibles
$ pipenv update --outdated
$ # réalisation des mises à jours (lock + sync)
$ pipenv update
```

Comment ça marche ?

Le fichier Pipfile

```
$ ls
Pipfile  Pipfile.lock
$ cat Pipfile
[[source]]
url = "https://pypi.org/simple"
verify_ssl = true
name = "pypi"

[packages]
ipython = "*"

[dev-packages]
black = "==19.3b0"
pytest = "*"

[requires]
python_version = "3.7"
```

Utilisation

Utilisation

```
$ pipenv shell
$ pipenv run ipython
```

Commandes utiles

```
$ # vérification des mises à jour de sécurité
$ pipenv check
$ # arbre des dépendances
$ pipenv graph
$ # suppression du virtual env
$ pipenv --rm
$ # chemin du virtual env
$ pipenv --venv
$ # installation / mise à jour de l'environnement de production
$ pipenv install
$ # installation / mise à jour de l'environnement de production+dev
$ pipenv install --dev
```

Configuration pour git

Qu'est-ce qui ne doit pas être *pushé* ?

- le virtual env (`.venv`)
- les résultats des compilations (`.pyc`, `__pycache__`, etc)

Comment empêcher automatiquement ces fichiers d'être *pushés*

- en ajoutant un fichier `.gitignore` à la racine du projet
- le dépôt git de git propose des fichiers types pour chaque langage (`https://github.com/github/gitignore`), celui de python prend en compte ces exigences

Organisation d'un projet python

Proposition d'une organisation de fichiers, répertoires

```
NomDuProjet
  + package_racine
    + __init__.py
    + sous_package_1
      + __init__.py
      + ...
    + sous_package_2
      + __init__.py
      + ...
    + module_1
    + ...
  + Pipfile
  + Pipfile.lock
  + tests
    + reprendre la hiérarchie des packages et mettre autant de fichier
test_XXX.py qu'il y a de modules (attention tous avec des noms différents)
  + programme_principal.py (si plusieurs, utilisation d'un répertoire bin)
  + README
  + .gitignore
```

Exemple : le projet CAO 1 / 4

Organisation du projet

```
ASI_CAO
  + cao
    + __init__.py
    + core
      + __init__.py
      + point.py
      + polyligne.py
    + lib
      + __init__.py
      + point.py
      + polyligne.py
  + tests
    + core
      + test_core_point.py
      + test_core_polyligne.py
    + lib
      + test_lib_point.py
      + test_lib_polyligne.py
  + main.py
  + Pipfile
  + Pipfile.lock
```

Exemple : le projet CAO 2 / 4

Création de l'environnement

```
ASI_CAO$ pipenv --python 3.7
```

Installation de pytest

```
ASI_CAO$ pipenv install --dev pytest
Creating a virtualenv for this project...
Pipfile: /tmp/test2/Pipfile
Using /home/bonnegent/.pyenv/versions/3.7.3/bin/python3.7m (3.7.3) to create virtua
Running virtualenv with interpreter /home/bonnegent/.pyenv/versions/3.7.3/bin/pytho
Using base prefix '/home/bonnegent/.pyenv/versions/3.7.3'
/usr/lib/python3/dist-packages/virtualenv.py:1086: DeprecationWarning: the imp modu
  import imp
New python executable in /home/bonnegent/.local/share/virtualenvs/test2-h6PFdzq5/bi
Also creating executable in /home/bonnegent/.local/share/virtualenvs/test2-h6PFdzq5
Installing setuptools, pkg_resources, pip, wheel...done.

Virtualenv location: /home/bonnegent/.local/share/virtualenvs/test2-h6PFdzq5
Creating a Pipfile for this project...
```

Exemple : le projet CAO 3 / 4

C'est python3 qui est le seul python

```
ASI_CAO$ pipenv run python --version
Python 3.7.3
ASI_CAO$
ASI_CAO$ pipenv shell
(venv)ASI_CAO$ python --version
Python 3.7.3
(venv)ASI_CAO$ desactivate
```

Lancement des tests unitaires

```
ASI_CAO$ pipenv run black .
ASI_CAO$ pipenv run pytest .
============================= test session starts ==============================
platform linux -- Python 3.7.3, pytest-3.0.7, py-1.4.33, pluggy-0.4.0
rootdir: .../ASI_CAO, inifile:
collected 18 items

tests/core/test_core_point.py ..
tests/core/test_core_polyligne.py ..........
tests/lib/test_lib_point.py ..
tests/lib/test_lib_polyligne.py ...
```

Exemple : le projet CAO 4 / 4

Utilisation en production

```
# activation dans notre shell
$ export PIPENV_VENV_IN_PROJECT="yes"
# enregistrement
$ echo 'export PIPENV_VENV_IN_PROJECT="yes"' >> /root/.bash_profile
$ pipenv install --ignore-pipfile
$ pipenv --venv
/opt/asi_cao/.venv
```