Python : Environnement virtuel

Environnement Python

Mise en place d'un environment virtuel Python

Jérôme Buisine
jerome.buisine@univ-littoral.fr

5 septembre 2023

L'objectif de ce document est la mise en place de l'environnement de travail pour Python à l'aide de l'outil `pyenv`.

1 Ressources

Ce support vous invite à configurer votre environnement afin de ne pas avoir de problèmes de dépendances et de versions de Python.

Voici un ensemble de ressources qui peuvent vous être utiles :

— 1. pyenv : permet la gestion d'environnement Python.

— 2. pyenv-virtualenv : plugin de gestion des environnements virtuels Python.

2 Installation de `pyenv`

Ce projet requiert Python 3.8. Si vous ne le posséder par, il vous faut configurer l'environnement pour ce projet. D'ailleurs, même si vous l'avez, **ne passez pas forcément cette étape** puisqu'elle peut vous être utile pour gérer des environnements virtuels Python pour tout projet.

Pour cela, nous allons installer `pyenv`. Pour ceux qui sont sur **Windows**, vous pouvez installer l'outil pyenv-win.

Note : `pyenv-win` ne permet pas de gérer des environnements virtuels, mais vous permet au moins d'installer des versions différentes de Python et de pointer sur l'une d'entre elle.

Pour ceux sur **Linux**, vérifiez d'abord que vous avez bien toutes les dépendances requises installées. Puis procéder à l'installer en suivant la documentation suivante : pyenv-installer. N'oubliez pas de vérifier que les lignes de chargements de l'outil soient présentes dans votre fichier `~/.bashrc`.

```
1 export PATH="$HOME/.pyenv/bin:$PATH"
2 eval "$(pyenv init --path)"
3 eval "$(pyenv virtualenv-init -)"
```

Redémarrez ensuite votre terminal avec la commande : `exec $SHELL`.

3 Les principales commandes de `pyenv`

Depuis `pyenv`, il vous est possible d'installer une version spécifique de Python. Installons par exemple, la version 3.8.0 de Python avec la commande suivante :

```
# installation de version demandée
pyenv install 3.8.0
# vérification de la disponibilité de la version
pyenv versions
# spécification de l'utilisation de la version à utiliser localement
pyenv local 3.8.0
```

Sous **Linux** uniquement, il est possible de gérer des environnements virtuels, environnements séparés pour une même version de Python :

```
# création d'un environnement à partir d'une version demandée de Python
pyenv virtualenv 3.8.0 mon-venv
# vérification des environnements disponibles
pyenv virtualenvs
# spécification de l'utilisation de l'environnement à utiliser localement
pyenv local mon-venv
```

Pour plus d'informations, vous pouvez vous référer à la documentation du plugin pyenv-virtualenv.

4 Installation/gestion des dépendances

`pip` est le gestionnaire de packages Python. Il vous permet d'installer, mettre à jour et supprimer des packages. En réalité, `pip` est lui même un package, pour pointer sur les dernières versions de packages, il faut vous le mettre à jour :

```
# mettre à jour la version de pip
pip install --upgrade pip
```

Voici quelques commandes importantes de `pip` pour la gestion de `package` qui peuvent vous être utiles :

```
# chercher un paquet, ou vérifier s'il est à jour
pip search <paquet>
# installer un paquet (une version précise peut être spécifiée)
pip install <paquet>
# mettre à jour un paquet
pip install <paquet> --upgrade
# lister les paquets installés et leurs versions
pip list
```

L'installation des packages peut donc se faire de la manière suivante :

```
pip install pandas numpy
```

Généralement, au sein d'un projet, on souhaite fixer les versions des dépendances pour permettre à quiconque de posséder le même environment de développement. En Python, cela se fait par le biais d'un fichier nommé « requirements.txt », dont le contenu pourrait être le suivant :

```
numpy==1.23.2
pandas==1.4.4
```

De cette manière, il est possible d'installer l'ensemble des dépendances à l'aide de la commande :

```
pip install -r requirements.txt
```