

# Concepts Python avancés

## Les Virtualenvs

Par Alexandre GALODE  

Date de publication : 22 avril 2015

Dernière mise à jour : 22 avril 2015

CONFIRMÉ

Aujourd'hui, je vous propose d'aborder une fonctionnalité Python qui n'est pas la plus connue : les virtualenvs.

Comme le laisse deviner leur nom, elles créent un environnement virtuel permettant ainsi de construire des espaces de travail Python indépendants. Avec les virtualenvs, on peut ainsi avoir différentes configurations d'environnements et de modules en parallèle pour travailler sur de multiples projets à la fois.

**Commentez**

I - Introduction.....	3
II - Principe de fonctionnement.....	3
III - Prérequis.....	3
IV - Installation.....	3
IV-A - Options.....	4
V - Création d'une virtualenv.....	4
VI - Manipulation d'une virtualenv.....	5
VI-A - Activation.....	5
VI-B - Désactivation.....	6
VI-C - Lancement d'un programme avec une virtualenv.....	7
VI-D - Copie et déploiement de virtualenv.....	8
VII - Conclusion.....	14
VIII - Remerciements.....	14

## I - Introduction

Le terme « Virtualenv » décrit à la fois un outil et ses créations. L'outil va permettre de créer un ensemble de dossiers et de fichiers qui constitueront une virtualenv.

Les virtualenvs sont couramment utilisées en entreprise, où il est souvent nécessaire d'avoir plusieurs configurations en parallèle de Python afin de tester les applications, leur déploiement ou encore de travailler sur plusieurs projets à la fois et cela sans conflit.

Imaginez que vous disposez de deux scripts Python, le premier utilisant la version 1 d'un module, et le second la version 2 du même module. Vous ne pouvez pas faire cohabiter ces deux versions d'un même module. C'est ici que les virtualenvs révèlent leur utilité.

Elles permettent de compartimenter une installation Python. Chaque logiciel possède alors sa virtualenv attitrée, ce qui permettra de faire tourner plusieurs logiciels Python en parallèle avec différentes versions et modules de Python potentiellement incompatibles entre eux.



*Virtualenv est l'outil pour Python 2.x. Python 3.x est livré avec son propre outil dont la documentation officielle se trouve **ICI**. En cas d'absence de `pyvirtualenv`, utilisez « `python -m virtualenv` » où `python` sera votre installation 3.x.*

## II - Principe de fonctionnement

Au lieu de lancer directement votre programme, vous commencez par activer votre environnement virtuel dans une console, puis vous appelez votre programme.

De fait, votre programme n'utilisera pas l'installation Python du système, mais exclusivement celle de votre virtualenv. Cela signifie que votre programme est alors totalement compartimenté, et qu'il ne va pas interférer avec d'autres programmes Python.

Tout module installé dans une virtualenv, n'est accessible, et visible, qu'à l'intérieur de cette virtualenv.

Parmi les contraintes que cela impose, nous retiendrons principalement le fait qu'une fois créée, déplacer une virtualenv est fortement déconseillé, au risque de la voir ne plus fonctionner. Dans l'idéal, vous devez la créer à chaque fois dans son lieu d'exécution.

Néanmoins, une option d'installation, que nous verrons plus loin, tente de pallier ce souci. Il s'agit cependant d'une contrainte minimale par rapport aux avantages fournis.

## III - Prérequis

Tout d'abord, vous devez bien évidemment disposer d'une installation fonctionnelle de Python.

Une fois n'est pas coutume, le moyen le plus simple d'installer l'outil est de passer par pypi. Pour cela, utiliser l'outil **pip** que vous pouvez installer grâce à la commande suivante :

```
1. python get-pip.py
```

Quel que soit votre système, je vous recommande de passer root/administrateur pour l'installation.

## IV - Installation

Une fois cela fait, utilisez simplement la commande suivante :

```
1. pip install virtualenv
```

## IV-A - Options

La commande virtualenv dispose d'un certain nombre d'options utiles dont voici les principales :

Option	Description
-v	Mode verbeux
-q	Mode silencieux
-p	Indique la version de python à utiliser. Par défaut la version par défaut du système est choisie, mais vous pouvez indiquer une autre version.
--system-site-packages	Autoriser la commande à avoir accès à tous les paquets système. Par défaut, sans cette option, seuls les paquets Python seront accessibles.
--always-copy	Empêche la création de liens symboliques et impose la copie de fichiers. Plus volumineux au final, mais évite parfois des soucis.
--relocatable	Passer les chemins de la virtualenv du mode absolu au mode relatif. Est censé permettre de déplacer la virtualenv. ATTENTION : n'est utilisable que sur une virtualenv existante.
--no-pip	Empêche l'installation de pip sur la virtualenv.

## V - Création d'une virtualenv

```
virtualenv -p python2.7 --system-site-packages /home/alex/test/my_virtualenv
virtualenv --relocatable /home/alex/test/my_virtualenv
```

```

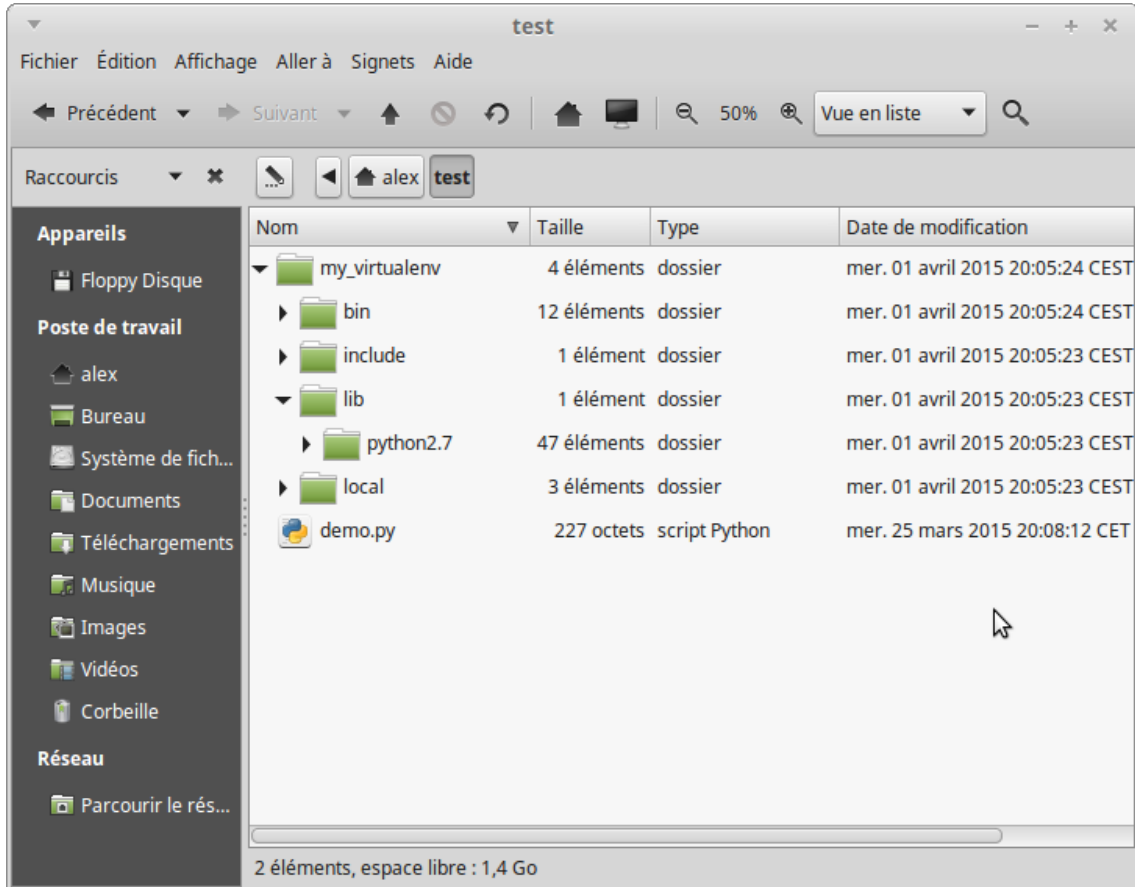
alex@alex-virtual-machine ~/test $ virtualenv -p python2.7 --system-site-packages /home/alex/test/my_virtualenv
Running virtualenv with interpreter /usr/bin/python2.7
New python executable in /home/alex/test/my_virtualenv/bin/python2.7
Also creating executable in /home/alex/test/my_virtualenv/bin/python
Installing setuptools, pip...done.
alex@alex-virtual-machine ~/test $ virtualenv --relocatable /home/alex/test/my_virtualenv
Making script /home/alex/test/my_virtualenv/bin/pip2 relative
Making script /home/alex/test/my_virtualenv/bin/easy_install relative
Making script /home/alex/test/my_virtualenv/bin/easy_install-2.7 relative
Making script /home/alex/test/my_virtualenv/bin/pip relative
Making script /home/alex/test/my_virtualenv/bin/pip2.7 relative
alex@alex-virtual-machine ~/test $

```

Voici la ligne de commande classique : on stipule la version de python à utiliser, ici la 2.7, et on autorise l'accès à tous les paquets.

Puis, une fois la virtualenv créée, on utilise l'option relocate afin de pouvoir déplacer notre virtualenv.

Voici notre virtualenv créée à l'endroit demandé.



Nous pouvons constater que la version de Python qui est embarquée est bien celle demandée, à savoir la 2.7.

## VI - Manipulation d'une virtualenv

### VI-A - Activation

Pour lancer votre environnement, rien de difficile : rendez-vous dans un des sous-dossiers à la recherche du script « active », et lancez-le de la façon suivante :

```
1. . activate
```

```

Terminal
Fichier Édition Affichage Rechercher Terminal Aide
alex@alex-virtual-machine ~/test $ virtualenv -p python2.7 --system-site-package
s /home/alex/test/my_virtualenv
Running virtualenv with interpreter /usr/bin/python2.7
New python executable in /home/alex/test/my_virtualenv/bin/python2.7
Also creating executable in /home/alex/test/my_virtualenv/bin/python
Installing setuptools, pip...done.
alex@alex-virtual-machine ~/test $ virtualenv --relocatable /home/alex/test/my_v
irtualenv
Making script /home/alex/test/my_virtualenv/bin/pip2 relative
Making script /home/alex/test/my_virtualenv/bin/easy_install relative
Making script /home/alex/test/my_virtualenv/bin/easy_install-2.7 relative
Making script /home/alex/test/my_virtualenv/bin/pip relative
Making script /home/alex/test/my_virtualenv/bin/pip2.7 relative
alex@alex-virtual-machine ~/test $ cd ./my_virtualenv/bin/
alex@alex-virtual-machine ~/test/my_virtualenv/bin $ . activate
(my_virtualenv)alex@alex-virtual-machine ~/test/my_virtualenv/bin $
  
```

Comme on peut le voir, une fois rentré dans une virtualenv, le prompt de la console est précédé du nom de la virtualenv, de sorte que l'on sait toujours dans quel environnement on se trouve. Pour cette raison, il est important de toujours choisir un nom explicite pour nos virtualenvs.

## VI-B - Désactivation

Pour éteindre une virtualenv en cours d'utilisation, rien de compliqué, il suffit d'appeler le script deactivate.

```
1. deactivate
```



```

alex@alex-virtual-machine ~/test $ virtualenv -p python2.7 --system-site-package
s /home/alex/test/my_virtualenv
Running virtualenv with interpreter /usr/bin/python2.7
New python executable in /home/alex/test/my_virtualenv/bin/python2.7
Also creating executable in /home/alex/test/my_virtualenv/bin/python
Installing setuptools, pip...done.
alex@alex-virtual-machine ~/test $ virtualenv --relocatable /home/alex/test/my_v
irtualenv
Making script /home/alex/test/my_virtualenv/bin/pip2 relative
Making script /home/alex/test/my_virtualenv/bin/easy_install relative
Making script /home/alex/test/my_virtualenv/bin/easy_install-2.7 relative
Making script /home/alex/test/my_virtualenv/bin/pip relative
Making script /home/alex/test/my_virtualenv/bin/pip2.7 relative
alex@alex-virtual-machine ~/test $ cd ./my_virtualenv/bin/
alex@alex-virtual-machine ~/test/my_virtualenv/bin $ . activate
(my_virtualenv)alex@alex-virtual-machine ~/test/my_virtualenv/bin $ deactivate
alex@alex-virtual-machine ~/test/my_virtualenv/bin $
  
```

## VI-C - Lancement d'un programme avec une virtualenv

Pour bien illustrer le sujet, le programme suivant indiquera le type d'OS ainsi que la version et le chemin de l'interpréteur Python utilisé.

```

1. #!/usr/bin/env PYTHON
2. # -*- coding: utf-8 -*-
3.
4. import os
5. import sys
6. import platform
7.
8.
9. version = platform.python_version_tuple()
10.
11. print("OS de type: %s" % (os.name))
12. print("Python Version %s.%s.%s" % (version[0], version[1], version[2]))
13. print("chemin python: %s" % (sys.executable))
  
```

Nous allons d'abord l'exécuter sur l'OS directement, puis nous activerons la virtualenv, et nous le lancerons à nouveau.

```

alex@alex-virtual-machine ~/test $ python demo.py
OS de type: posix
Python Version 2.7.6
chemin python: /usr/bin/python
alex@alex-virtual-machine ~/test $ . ./my_virtualenv/bin/activate
(my_virtualenv)alex@alex-virtual-machine ~/test $ python demo.py
OS de type: posix
Python Version 2.7.6
chemin python: /home/alex/test/my_virtualenv/bin/python
(my_virtualenv)alex@alex-virtual-machine ~/test $ deactivate
alex@alex-virtual-machine ~/test $ . ./my_virtualenv/bin/activate && python demo
.py && deactivate
OS de type: posix
Python Version 2.7.6
chemin python: /home/alex/test/my_virtualenv/bin/python
alex@alex-virtual-machine ~/test $
  
```

Comme nous pouvons le constater, exécutée directement sur notre système, c'est la version par défaut de Python du système, Python 2.7, qui est utilisée, depuis /usr/bin.

Une fois dans notre virtualenv, le même script nous indique le Python par défaut de notre virtualenv, à savoir Python 2.7, non pas depuis /usr/bin, mais depuis les dossiers de notre virtualenv.

Nous sommes ainsi entièrement isolés par rapport à notre installation locale.

La dernière commande lancée vous montre comment faire pour lancer votre logiciel à l'intérieur d'une virtualenv.

## VI-D - Copie et déploiement de virtualenv

Rendu à ce niveau, vous vous demandez peut-être, si au-delà du test/développement, cela peut également servir au déploiement. Eh bien la réponse est : oui.

Tout commence lors de votre développement. Vous devez vous créer une virtualenv fonctionnelle pour votre logiciel.

Une fois votre développement terminé, il faudra évidemment que les utilisateurs finals disposent du même environnement (branche de Python, paquets...) afin que votre code fonctionne.

Partons du principe que vous voulez livrer une virtualenv. Vous disposez déjà de la procédure pour en créer une. Il vous manque cependant la façon de faciliter le déploiement des paquets tiers.

Nous allons utiliser ici l'outil pip. Cela impose qu'il soit installé dans la virtualenv.

Nous allons commencer par lister les paquets installés au sein de la virtualenv. La procédure suivante va vous créer un fichier contenant tous les paquets installés, ainsi que leurs versions respectives, dans votre virtualenv.



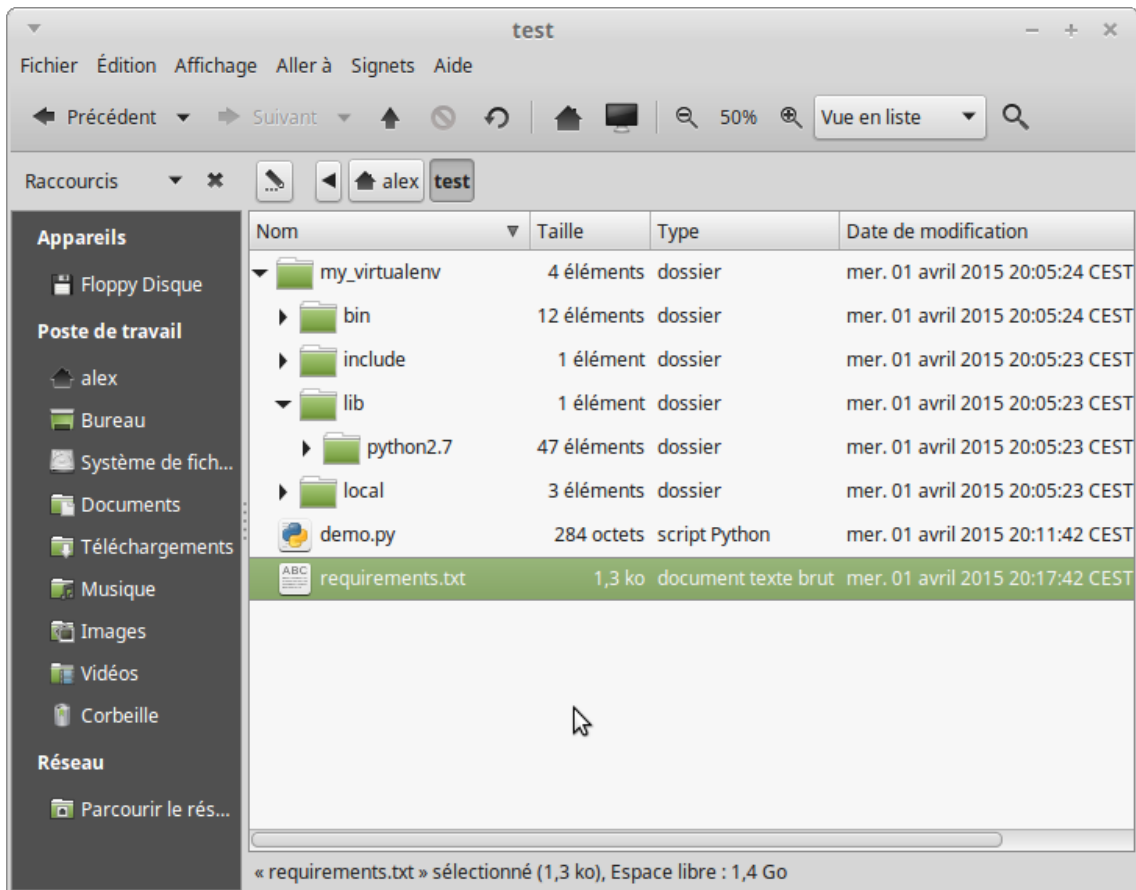
**i** Si nous utilisons cette procédure, ici, dans le cadre des virtualenvs, cela fonctionne également pour une installation locale standard.

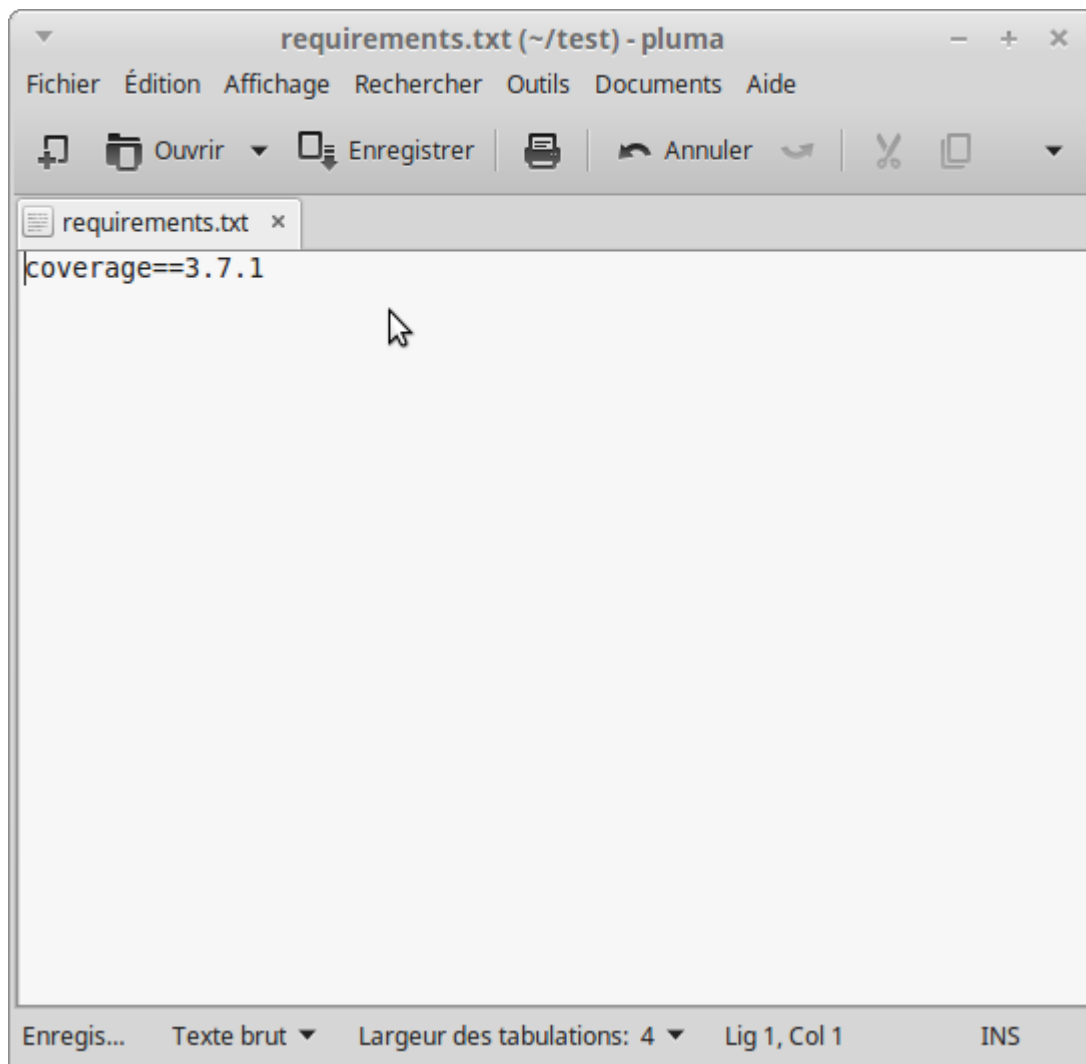
```
1. pip freeze > /home/alex/test/requirements.txt
```

The screenshot shows a terminal window titled "Terminal" with a menu bar containing "Fichier", "Édition", "Affichage", "Rechercher", "Terminal", and "Aide". The terminal output is as follows:

```
alex@alex-virtual-machine ~/test $ ./my_virtualenv/bin/activate
(my_virtualenv)alex@alex-virtual-machine ~/test $ pip freeze > requirements.txt
(my_virtualenv)alex@alex-virtual-machine ~/test $
```

The background of the terminal window features a dark, chalkboard-like texture with faint, handwritten mathematical equations and diagrams, including trigonometric functions like  $y = A + B \cos 2x$  and various geometric notations.





Comme on peut le voir, mon installation ne contient qu'un paquet tiers : coverage, qui permet de mesurer la couverture de code ([voir cet article](#)).

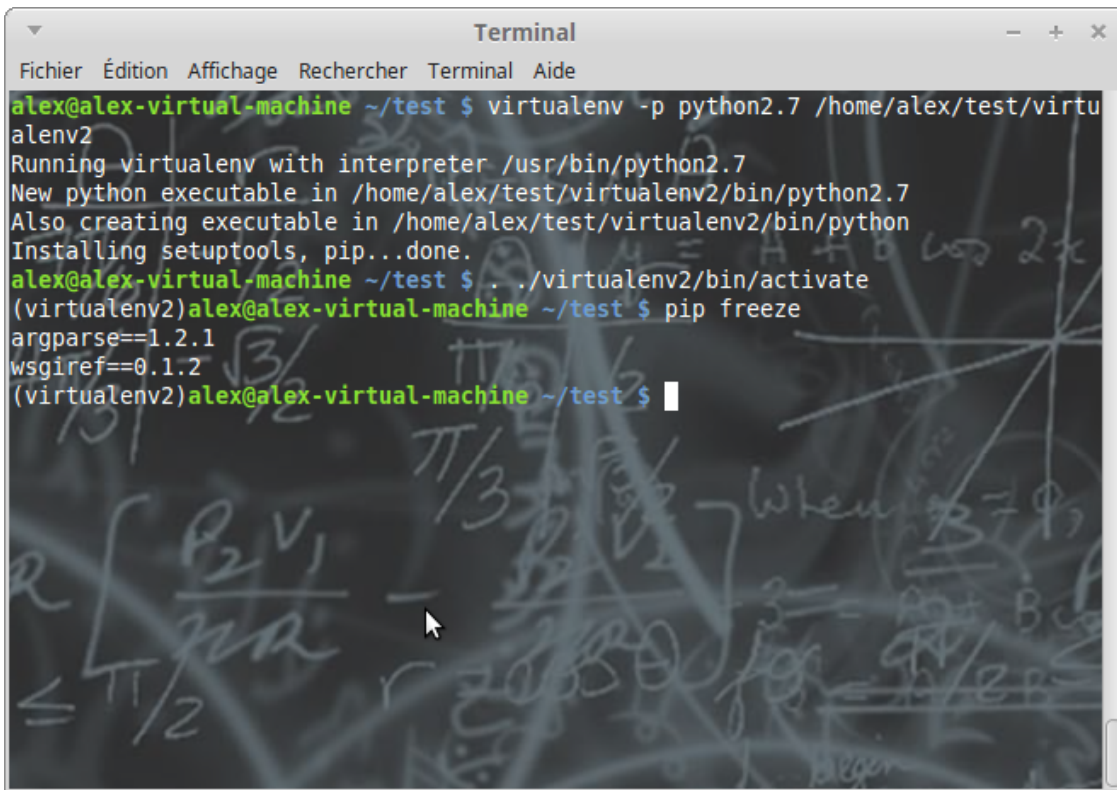
*Vous n'êtes pas obligé de stipuler un numéro de version. Vous auriez ainsi pu lire juste « coverage » au lieu de « coverage==3.7.1 ». La dernière version disponible serait alors systématiquement téléchargée.*



*Cependant, pour des raisons évidentes de stabilité, il est préférable de toujours stipuler un numéro de version.*

Nous allons maintenant réutiliser ce fichier.

Tout d'abord, créons une nouvelle virtualenv, en ne copiant que les paquets Python.



```

alex@alex-virtual-machine ~/test $ virtualenv -p python2.7 /home/alex/test/virtualenv2
Running virtualenv with interpreter /usr/bin/python2.7
New python executable in /home/alex/test/virtualenv2/bin/python2.7
Also creating executable in /home/alex/test/virtualenv2/bin/python
Installing setuptools, pip...done.
alex@alex-virtual-machine ~/test $ . ./virtualenv2/bin/activate
(virtualenv2)alex@alex-virtual-machine ~/test $ pip freeze
argparse==1.2.1
wsgiref==0.1.2
(virtualenv2)alex@alex-virtual-machine ~/test $
  
```

Nous n'avons pas utilisé l'option `--system-site-packages` et pouvons constater que la commande `pip freeze` ne renvoie que deux paquets.

Bien, maintenant, utilisons notre fichier de listing.

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

```

Terminal
Fichier Édition Affichage Rechercher Terminal Aide
(virtualenv2)alex@alex-virtual-machine ~/test $ pip install -r requirements.txt
Downloading/unpacking coverage==3.7.1 (from -r requirements.txt (line 1))
Downloading coverage-3.7.1.tar.gz (284kB): 284kB downloaded
Running setup.py (path:/home/alex/test/virtualenv2/build/coverage/setup.py) eg
g info for package coverage
warning: no previously-included files matching '*.pyc' found anywhere in dis
tribution
Installing collected packages: coverage
Running setup.py install for coverage
building 'coverage.tracer' extension
i686-linux-gnu-gcc -pthread -fno-strict-aliasing -DNDEBUG -g -fwrapv -O2 -Wa
ll -Wstrict-prototypes -fPIC -I/usr/include/python2.7 -c coverage/tracer.c -o bu
ild/temp.linux-i686-2.7/coverage/tracer.o
i686-linux-gnu-gcc -pthread -shared -Wl,-O1 -Wl,-Bsymbolic-functions -Wl,-Bs
ymbolic-functions -Wl,-z,relro -fno-strict-aliasing -DNDEBUG -g -fwrapv -O2 -Wal
l -Wstrict-prototypes -D_FORTIFY_SOURCE=2 -g -fstack-protector --param=ssp-buffe
r-size=4 -Wformat -Werror=format-security build/temp.linux-i686-2.7/coverage/tra
cer.o -o build/lib.linux-i686-2.7/coverage/tracer.so
warning: no previously-included files matching '*.pyc' found anywhere in dis
tribution
Installing coverage2 script to /home/alex/test/virtualenv2/bin
Installing coverage-2.7 script to /home/alex/test/virtualenv2/bin
Installing coverage script to /home/alex/test/virtualenv2/bin
Successfully installed coverage
Cleaning up...
(virtualenv2)alex@alex-virtual-machine ~/test $

```

```

Terminal
Fichier Édition Affichage Rechercher Terminal Aide
(virtualenv2)alex@alex-virtual-machine ~/test $ pip freeze
argparse==1.2.1
coverage==3.7.1
wsgiref==0.1.2
(virtualenv2)alex@alex-virtual-machine ~/test $

```

On peut constater que malgré quelques erreurs, le paquet est installé avec succès et apparaît bien si on effectue la commande pip freeze.

De même, si vous lancez une console Python, au sein de votre virtualenv, vous pourrez importer le module coverage.

*Cette méthode utilise Pypi. Pour que cela fonctionne, il faut donc disposer d'un accès Internet.*  
**!** *De plus, si vous stipulez dans votre fichier de requirements un paquet non disponible pour la branche Python dont vous disposez dans la virtualenv, ou un numéro de version erroné, alors l'installation échouera.*

## VII - Conclusion

Comme nous venons de voir ensemble, les virtualenvs, bien que peu répandues, peuvent se révéler très utiles. Par exemple, tester un nouveau module sans compromettre notre installation locale, travailler sur plusieurs projets utilisant des modules incompatibles entre eux... Les exemples de cas d'application ne manquent pas.

Très utilisé en milieu professionnel pour des raisons évidentes de compartimentage, c'est un outil qui se révélera à vous comme des plus pratiques, aussi bien pour de simples tests que pour vos futurs développements.

## VIII - Remerciements

Merci aux personnes suivantes pour leur aide :

- **LittleWhite**
- **Wiztricks**
- **chrtophe**
- **ClaudeLELOUP**