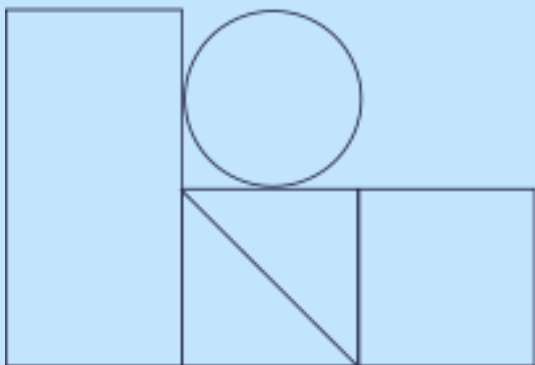


Django

Configurando el entorno virtual



Índice

Introducción	3
Virtualenv	4
Diferencia entre virtualenv y venv	4
Instalando Virtualenv usando pip3	5
Creación de un entorno virtual con una versión personalizada de Python	5
Desactivando virtualenv	7
Eliminar el entorno virtual	7
Posibles problemas	8

Introducción

Los entornos virtuales en Python son una herramienta que se usa en la práctica totalidad de los proyectos de cierta envergadura. Son tan importantes que forman parte de la librería estándar de Python, pero, realmente, ¿qué hacen?

Imaginemos que estamos desarrollando dos proyectos, cada uno para una empresa diferente. En el primero de los proyectos estamos desarrollando una nueva funcionalidad para un sitio web corporativo que utiliza Django 2.2. Un par de las librerías usadas en el anterior proyecto se actualizan con poca frecuencia, por lo que, para evitar problemas de compatibilidad, decidimos conservar esa versión de Django. Vamos a nombrar a este proyecto “Pro 2.2”.

En el segundo proyecto, nuestros clientes nos piden que desarrollemos una aplicación web desde cero. Para aprovechar las nuevas características del framework utilizamos la versión más nueva de Django en este proyecto. Nombramos a este segundo proyecto “Pro-newest”.

Empiezan los problemas.

Esa misma tarde empezamos a trabajar en el primer proyecto, Pro 2.2. Más tarde, empezamos a trabajar en el segundo proyecto. De repente, cuando necesitamos volver a trabajar nuevamente en Pro 2.2, el problema aparece claro. Cada vez que trabajemos en Pro 2.2 será necesario desinstalar la versión más nueva de Django y, cuando escribamos código para el segundo proyecto, tendremos que instalar la versión más reciente. Y peor aún, esta situación se repite para cada dependencia del proyecto.

Pero no solo eso. Cuando entregamos los archivos a nuestro cliente nos responde que el código no se ejecuta.

Nuestro proyecto está desarrollado con la versión más reciente de Django y no funciona en el equipo del cliente debido a incompatibilidades en las versiones.

¿Y si usamos máquinas virtuales?

Podríamos solucionar el problema anterior instalando una máquina virtual, como virtualbox. Dentro de cada máquina virtual seríamos capaces de instalar las dependencias de nuestro proyecto a medida. Además, tendríamos tantas como proyectos. Y funcionaría, ¿no? Bueno, sí, pero con un gran inconveniente: hay que cargar todo un sistema operativo completo para tener unas cuentas dependencias. Es demasiada carga a nuestro sistema para resultar práctico.

Las máquinas virtuales consumen demasiado espacio en disco duro y el tiempo de arranque de cada máquina virtual es desalentador. La interacción entre nuestro sistema y una máquina virtual puede llegar a ser complicada. Después de todo, no necesitamos cargar todo un sistema operativo, sino solo código Python.

La solución, un entorno virtual en Python.

Un entorno virtual, simplificando al máximo su explicación, es un espacio aislado del resto de nuestro sistema operativo, donde tendremos una serie de dependencias instaladas de manera local. Es como si especificaras un lugar desde donde Python tomará sus librerías, en lugar del predeterminado que usa tu sistema operativo. Estas dependencias son independientes de las que tengamos previamente instaladas en nuestro sistema operativo. Y, lo mejor, podemos tener tantos de estos espacios aislados como deseemos.

Imaginemos una carpeta donde tengamos instalado Django 2.2 y otra para la versión más nueva de Django. Al ser entornos aislados, no importa si nuestro sistema operativo ni siquiera tiene instalado Django.

Podremos cambiar entre un entorno virtual y otro, sin tiempos de carga excesivos, y el comportamiento será el mismo que si los tuviéramos instalados en nuestro sistema operativo.

Sobra decir que los entornos virtuales solucionan bastantes problemas. Y es una práctica altamente recomendada, por no decir casi obligatoria, cuando se trabaja con código Python.

Virtualenv es una herramienta utilizada para crear un entorno Python aislado. Este entorno tiene sus propios directorios de instalación que no comparten bibliotecas con otros entornos virtualenv (y, opcionalmente, tampoco accede a las bibliotecas instaladas globalmente).

Virtualenv es la forma más fácil y recomendada de configurar un entorno de Python personalizado.

Virtualenv

Diferencia entre virtualenv y venv

venv es un paquete que viene con Python 3. Python 2 no contiene **venv**.

virtualenv es una biblioteca que ofrece más funcionalidades que **venv**.

En el siguiente enlace se muestra una lista de las características que **venv** no ofrece en comparación con **virtualenv**.

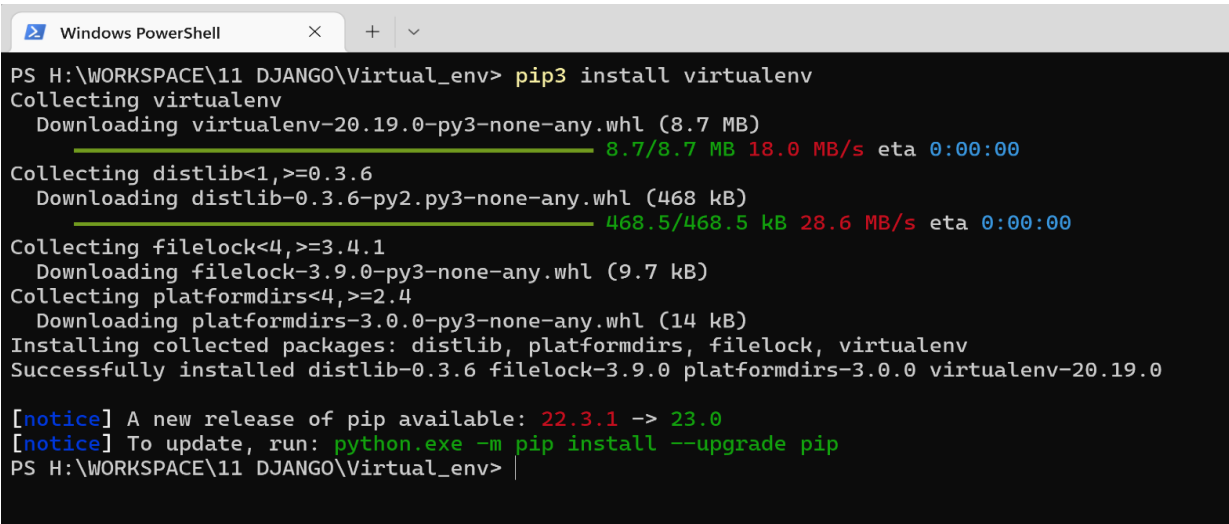
<https://virtualenv.pypa.io/es/estable/>

Aunque podemos crear un entorno virtual usando **venv** con Python3, se recomienda que se instale y use **virtualenv** en su lugar.

Instalando Virtualenv usando pip3

Virtualenv solo se instala en servidores DreamHost para Python 2. Si está trabajando con Python 3, debe instalar **virtualenv** usando **pip3**. Para ello, en la consola de comandos:

pip3 install virtualenv



```
PS H:\WORKSPACE\11 DJANGO\Virtual_env> pip3 install virtualenv
Collecting virtualenv
  Downloading virtualenv-20.19.0-py3-none-any.whl (8.7 MB)
    8.7/8.7 MB 18.0 MB/s eta 0:00:00
Collecting distlib<1,>=0.3.6
  Downloading distlib-0.3.6-py2.py3-none-any.whl (468 kB)
    468.5/468.5 kB 28.6 MB/s eta 0:00:00
Collecting filelock<4,>=3.4.1
  Downloading filelock-3.9.0-py3-none-any.whl (9.7 kB)
Collecting platformdirs<4,>=2.4
  Downloading platformdirs-3.0.0-py3-none-any.whl (14 kB)
Installing collected packages: distlib, platformdirs, filelock, virtualenv
Successfully installed distlib-0.3.6 filelock-3.9.0 platformdirs-3.0.0 virtualenv-20.19.0

[notice] A new release of pip available: 22.3.1 -> 23.0
[notice] To update, run: python.exe -m pip install --upgrade pip
PS H:\WORKSPACE\11 DJANGO\Virtual_env> |
```

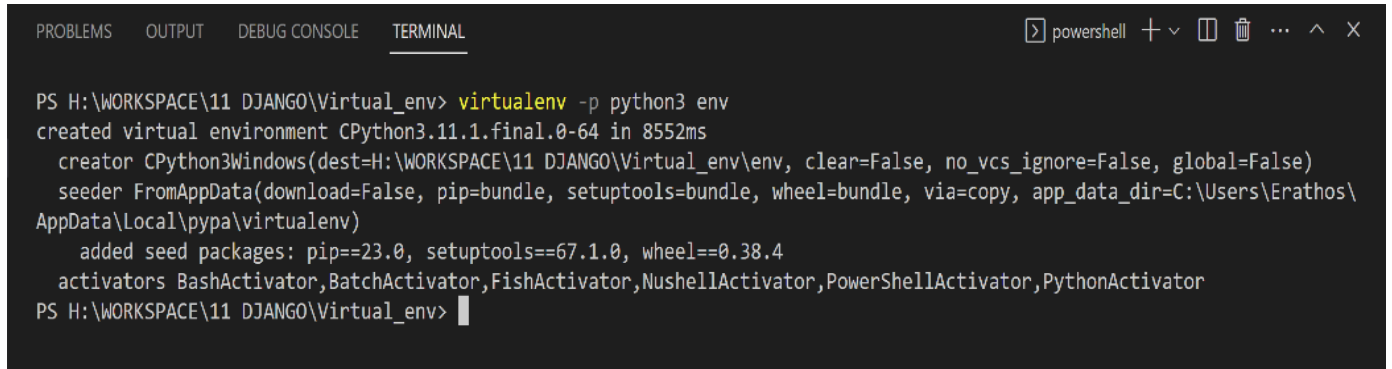
Creación de un entorno virtual con una versión personalizada de Python

Hay solo otros pocos comandos útiles que deberías conocer (hay más en la documentación de la herramienta, pero estos son los que usarás de forma habitual:

- **deactivate** — Salir del entorno virtual Python actual
- **workon** — Listar los entornos virtuales disponibles
- **workon name_of_environment** — Activar el entorno virtual Python especificado
- **rmvirtualenv name_of_environment** — Borrar el entorno especificado.

Tenga en cuenta que es posible que deba reinstalar Python después de una actualización del sistema operativo del servidor.

Cuando se trabaja con entornos virtuales en Python, es común usar una versión personalizada de Python en lugar de la versión del servidor. Para crear un nuevo entorno virtual utilizando su versión personalizada instalada de Python, deberemos ejecutar en la consola de comandos:



```
PS H:\WORKSPACE\11 DJANGO\Virtual_env> virtualenv -p python3 env
created virtual environment CPython3.11.1.final.0-64 in 8552ms
  creator CPython3Windows(dest=H:\WORKSPACE\11 DJANGO\Virtual_env\env, clear=False, no_vcs_ignore=False, global=False)
  seeder FromAppData(download=False, pip=bundle, setuptools=bundle, wheel=bundle, via=copy, app_data_dir=C:\Users\Erathos\
AppData\Local\pypa\virtualenv)
    added seed packages: pip==23.0, setuptools==67.1.0, wheel==0.38.4
  activators BashActivator,BatchActivator,FishActivator,NushellActivator,PowerShellActivator,PythonActivator
PS H:\WORKSPACE\11 DJANGO\Virtual_env>
```

Por ejemplo, el siguiente comando crea un **virtualenv** llamado '**venv**' y usa el indicador **-p** para especificar la ruta completa a la versión de Python3 que acaba de instalar:

Podemos nombrar el **virtualenv** como queramos.

```
virtualenv -p
/home/username/opt/python-
3.10.1/bin/python3 venv

Running virtualenv with
interpreter
/home/username/opt/python-
3.10.1/bin/python3

Using base prefix
'/home/username/opt/python-
3.10.1'

New python executable in
/home/username/example.com/env/bi
n/python3

Also creating executable in
/home/username/example.com/env/bi
n/python
```

```
Installing setuptools, pip,  
wheel...done.
```

Este comando crea una copia local de su entorno específico para este sitio web. Mientras trabajemos en este sitio web, deberemos activar el entorno local para asegurarnos de que estamos trabajando con las versiones correctas de nuestras herramientas y paquetes.

Nota: Es posible que nos aparezca el siguiente error al instalar.

```
setuptools pip failed with error  
code 1` error
```

En este caso, deberemos ejecutar lo siguiente:

```
pip3 install --upgrade setuptools
```

Volvemos a intentarlo y debería poder instalarse sin errores.

Ya tenemos el entorno virtual instalado, ahora debemos activarlo. Para activar el nuevo entorno virtual, ejecutaremos lo siguiente:

```
source venv/bin/activate
```

El nombre del entorno virtual actual aparece a la izquierda del aviso. Por ejemplo:

```
(venv) [servidor]$
```

Para verificar la versión correcta de Python, ejecute lo siguiente:

```
python -V
```

```
Python 3.10.1
```

Cualquier paquete que instalemos usando **pip** ahora se coloca en la carpeta del proyecto de entornos virtuales, aislado de la instalación global de Python.

Desactivando virtualenv

Cuando terminemos de trabajar en el entorno virtual, podemos desactivarlo ejecutando lo siguiente:

```
deactivate
```

Esto nos devuelve a la configuración predeterminada de su usuario de Shell.

Eliminar el entorno virtual

Para eliminar un entorno virtual, simplemente eliminaremos la carpeta del proyecto. Usando el ejemplo anterior, ejecute el siguiente comando:

```
rm -rf venv
```

Posibles problemas

Errores al crear un virtualenv

Es posible que nos aparezcan los siguientes errores al intentar crear un entorno virtual con Python 3.7:

AttributeError: module 'importlib._bootstrap' has no attribute 'SourceFileLoader'

OSError: Command

/home/username/venv/bin/python3 -c "import sys, pip; sys...d\" + sys.argv[1:])" setuptools pip failed with error code 1

Agregar la siguiente línea al instalar una versión personalizada de OpenSSL en su **.bash_profile** resuelve esto.

```
export LC_ALL="en_US.UTF-8"
```

Usa la ruta completa a su virtualenv personalizado

También es posible que cuando ejecutamos el comando **virtualenv**, estemos usando una versión fuera de la instalación personalizada. Intenta ejecutar esto en su lugar para confirmar la ruta completa a tu Python3 **virtualenv**.

```
which virtualenv
/home/username/opt/python-
3.8.0/bin/virtualenv
```

Esto debería responder con la versión en el directorio Python3 personalizado. Luego puedes crear un **virtualenv** usando la ruta completa:

```
/home/username/opt/python-
3.8.0/bin/virtualenv -p
/home/username/opt/python-
3.8.0/bin/python3 venv
```