Lectura 5: El tutorial de Python (Páginas 83-85)

En este material de lectura veremos qué son y cómo crear entornos virtuales y como instalar y actualizar paquetes con pip.

Cantidad de páginas: 3.

Entornos Virtuales y Paquetes

Introducción

Las aplicaciones en Python usualmente hacen uso de paquetes y módulos que no forman parte de la librería estándar. Las aplicaciones a veces necesitan una versión específica de una librería, debido a que dicha aplicación requiere que un bug particular haya sido solucionado o bien la aplicación ha sido escrita usando una versión obsoleta de la interface de la librería.

Esto significa que tal vez no sea posible para una instalación de Python cumplir los requerimientos de todas las aplicaciones. Si la aplicación A necesita la versión 1.0 de un módulo particular y la aplicación B necesita la versión 2.0, entonces los requerimientos entran en conflicto e instalar la versión 1.0 o 2.0 dejará una de las aplicaciones sin funcionar.

La solución a este problema es crear un *entorno virtual*, un directorio que contiene una instalación de Python de una versión en particular, además de unos cuantos paquetes adicionales.

Diferentes aplicaciones pueden entonces usar entornos virtuales diferentes. Para resolver el ejemplo de requerimientos en conflicto citado anteriormente, la aplicación A puede tener su propio entorno virtual con la versión 1.0 instalada mientras que la aplicación B tiene otro entorno virtual con la versión 2.0. Si la aplicación B requiere que actualizar la librería a la versión 3.0, ésto no afectará el entorno virtual de la aplicación A.

Creando Entornos Virtuales

El script usado para crear y manejar entornos virtuales es **pyvenv**. **pyvenv** normalmente instalará la versión mas reciente de Python que tengas disponible; el script también es instalado con un número de versión, con lo que si tienes múltiples versiones de Python en tu sistema puedes seleccionar una versión de Python específica ejecutando `python3` o la versión que desees.

To create a virtual environment, decide upon a directory where you want to place it,

Para crear un virtualenv, decide en que carpeta quieres crearlo y ejecuta el módulo **`venv`** como script con la ruta a la carpeta:

```
python3 -m venv tutorial-env
```

Esto creará la carpeta `tutorial-env` si no existe, y también creará las subcarpetas conteniendo la copia del intérprete Python, la librería estándar y los archivos de soporte.

Una vez creado el entorno virtual, podrás activarlo.

En Windows, ejecuta:

```
tutorial-env\Scripts\activate.bat
```

En Unix o MacOS, ejecuta:

```
source tutorial-env/bin/activate
```

(Este script está escrito para la consola bash. Si usas las consolas **csh** or **fish**, hay scripts alternativos `activate.csh` y `activate.fish` que deberá usar en su lugar).

Activar el entorno virtual cambiará el prompt de tu consola para mostrar que entorno virtual está usando, y modificará el entorno para que al ejecutar `python` sea con esa versión e instalación en particular. Por ejemplo:

```
$ source ~/envs/tutorial-env/bin/activate
(tutorial-env) $ python
Python 3.5.1 (default, May  6 2016, 10:59:36)
  ...
>>> import sys
>>> sys.path
['', '/usr/local/lib/python35.zip', ...,
'~/envs/tutorial-env/lib/python3.5/site-packages']
>>>
```

Manejando paquetes con pip

Es posible instalar, actualizar y quitar paquetes usando un programa llamado **pip**. Por defecto `pip` instalará paquetes desde Python Package Index (Indice de Paquetes Python), <https://pypi.python.org/pypi> . Se puede navegar el Python Package Index ingresando con su navegador de internet, o se puede usar la búsqueda limitada de `pip`'s:

```
(tutorial-env) $ pip search astronomy
skyfield               - Elegant astronomy for Python
gary                    - Galactic astronomy and gravitational dynamics.
novas                   - The United States Naval Observatory NOVAS astronomy library
astroobs                - Provides astronomy ephemeris to plan telescope observations
PyAstronomy             - A collection of astronomy related tools for Python.
...
```

`pip` tiene varios subcomandos: "search", "install", "uninstall", "freeze", etc. (consulta la guía *Instalando módulos de Python* para la documentación completa de `pip`.)

Se puede instalar la última versión de un paquete especificando el nombre del paquete:

```
(tutorial-env) $ pip install novas
Collecting novas
  Downloading novas-3.1.1.3.tar.gz (136kB)
Installing collected packages: novas
  Running setup.py install for novas
Successfully installed novas-3.1.1.3
```

También se puede instalar una verisón específica de un paquete ingresando el nombre del paquete seguido de `==` y el número de versión:

```
(tutorial-env) $ pip install requests==2.6.0
Collecting requests==2.6.0
  Using cached requests-2.6.0-py2.py3-none-any.whl
Installing collected packages: requests
Successfully installed requests-2.6.0
```

Si se re-ejecuta el comando, `pip` detectará que la versión ya está instalada y no hará nada. Se puede ingresar un número de versión diferente para instalarlo, o se puede ejecutar `pip install --upgrade` para actualizar el paquete a la última versión:

```
(tutorial-env) $ pip install --upgrade requests
Collecting requests
Installing collected packages: requests
  Found existing installation: requests 2.6.0
    Uninstalling requests-2.6.0:
      Successfully uninstalled requests-2.6.0
Successfully installed requests-2.7.0
```

`pip uninstall` seguido de uno o varios nombres de paquetes desinstalará los paquetes del entorno virtual.

`pip show` mostrará información de un paquete en particular:

```
(tutorial-env) $ pip show requests
---
Metadata-Version: 2.0
Name: requests
Version: 2.7.0
Summary: Python HTTP for Humans.
Home-page: http://python-requests.org
Author: Kenneth Reitz
Author-email: me@kennethreitz.com
License: Apache 2.0
Location: /Users/akuchling/envs/tutorial-env/lib/python3.4/site-packages
Requires:
```

`pip list` mostrará todos los paquetes instalados en el entorno virtual:

```
(tutorial-env) $ pip list
novas (3.1.1.3)
numpy (1.9.2)
pip (7.0.3)
requests (2.7.0)
setuptools (16.0)
```

`pip freeze` devuelve una lista de paquetes instalados similar, pero el formato de salida es el requerido por`pip install`. Una convención común es poner esta lista en un archivo `requirements.txt`:

```
(tutorial-env) $ pip freeze > requirements.txt
(tutorial-env) $ cat requirements.txt
novas==3.1.1.3
numpy==1.9.2
requests==2.7.0
```

El archivo `requirements.txt` entonces puede ser agregado a nuestro control de versiones y distribuído como parte de la aplicación. Los usuarios pueden entonces instalar todos los paquetes necesarios con`install -r`:

```
(tutorial-env) $ pip install -r requirements.txt
Collecting novas==3.1.1.3 (from -r requirements.txt (line 1))
  ...
Collecting numpy==1.9.2 (from -r requirements.txt (line 2))
  ...
Collecting requests==2.7.0 (from -r requirements.txt (line 3))
  ...
Installing collected packages: novas, numpy, requests
  Running setup.py install for novas
Successfully installed novas-3.1.1.3 numpy-1.9.2 requests-2.7.0
```

`pip` tiene muchas opciones más. Consulta la guía *Instalando módulos de Python* para la documentación de `pip`. Cuando hayas escrito un paquete y desees dejarlo disponible en Python Package Index, consulte la guía *Glosario*.