¿Cómo Trabajar con el Código de Ejemplo?

El historial de confirmación en este repositorio se creó cuidadosamente para que coincida con el orden en que se presentan los conceptos. La forma recomendada de trabajar con el código es revisar las confirmaciones comenzando desde la más antigua, luego avanzar a través de la lista de confirmaciones a medida que avanzamos. Github permite descargar cada confirmación como un archivo ZIP. El siguiente comando descarga el código de ejemplo usando GIT:

```
$ git clone https://github.com/miguelgrinberg/flasky.git
```

El comando git clone instala el código fuente de GitHub en una carpeta flasky2 que se crea en el directorio actual. Esta carpeta contiene una copia del repositorio de Git con el historial completo de los cambios realizados en la aplicación.

En el primer capítulo se le pedirá que revise la versión inicial de la aplicación, y luego, en los lugares adecuados, se le indicará que avance en la historia. El comando Git que le permite moverse por el historial de cambios es git checkout. Aquí hay un ejemplo:

```
$ git checkout 1a
```

El **1a** al que se hace referencia en el comando es una etiqueta: un punto con nombre en el historial de confirmación del proyecto. Este repositorio está etiquetado de acuerdo con los capítulos del libro, por lo que la etiqueta **1a** utilizada en este ejemplo establece los archivos de la aplicación a la versión inicial utilizada en el Capítulo 1. La mayoría de los capítulos tienen más de una etiqueta asociada con ellos, por ejemplos, las etiquetas 5a, 5b, etc. son versiones incrementales presentadas en el Capítulo 5. Cuando se ejecuta un comando de git como se muestra, GIT mostrará un mensaje de advertencia que le informa que está en un estado de “CABEZA desconectada”. Esto significa que no se encuentra en ninguna rama de código especifica que pueda aceptar nuevas confirmaciones, sino que está viendo una confirmación particular en medio del historial de cambios del proyecto.

No hay ninguna razón para alarmarse con este mensaje, pero debe tenerse en cuenta que si se realiza modificaciones en cualquier archivo mientras se encuentra en este estado, la emisión de otro git checkout fallará, porque Git no sabrá qué hacer con los cambios que usted ha hecho. Por lo tanto, para poder seguir trabajando con el proyecto, deberá revertir los archivos que cambió a su original. La forma más fácil de hacer esto es con el comando git reset:

```
$ git reset --hard
```

Este comando destruirá los cambios locales que se haya realizado, por lo que debe guardar todo lo que no quiera perder antes de usar este comando.

Además de verificar los archivos de origen para una versión de la aplicación, en ciertos puntos deberá realizar tareas de configuración adicionales.

De vez en cuando, es posible que desee actualizar su repositorio local desde el de GitHub, donde se han aplicado correcciones de errores y mejoras. Los comandos que logran esto son:

```
$ git fetch --all
$ git fetch --tags
$ git reset --hard origin/master
```

Los comandos git fetch se usan para actualizar el historial de confirmaciones y las etiquetas en su repositorio local desde el remoto en GitHub, pero nada de esto afecta a los archivos de origen reales, que se actualizan con el comando git reset. Una vez más, tenga en cuenta que cada vez que usa git reset, perderá los cambios locales que haya realizado.

Otra operación útil es ver todas las diferencias entre dos versiones de la aplicación; esto puede ser útil para comprender un cambio en los detalles. Desde la línea de comandos, el comando git diff puede hacer esto. Por ejemplo, para ver la diferencia entre las revisiones 2a y 2b, use:

```
$ git diff 2a 2b
```

Las diferencias se muestran como un parche, que no es un formato muy intuitivo para revisar los cambios si no está acostumbrado a trabajar con archivos de parche. Es posible que las comparaciones gráficas mostradas por GitHub sean mucho más fáciles de leer. Por ejemplo, las diferencias entre las revisiones 2a y 2b se pueden ver en GitHub en https://github.com/miguelgrinberg/flasky/compare/2a...2b.

CAPÍTULO 1

INSTALACIÓN

Flask es un marco pequeño para la mayoría de los estándares, razón por la cual como para llamarlo microframework y esto hace que se pueda entender de manera sencilla su código fuente. Flask fue diseñado como un marco extensible desde cero, proporcionando un núcleo sólido con los servicios básicos, mientras que las extensiones hacen el resto.

Flask tiene 3 dependencias principales:

- Sistemas de enrutamiento, depuración e interfaz de puerta de enlace de servidor web (WSGI) que provienen de **Werkzeug**.
- El soporte de Plantilla que lo proporciona **JINJA2**
- Integración de la línea de comandos que proviene de **CLICK**

Flask no tiene soporte para acceder a bases de datos, validar formularios web, autenticar usuarios u otras tareas de alto nivel. Estos y muchos servicios están disponibles a través de extensiones que se integran con los paquetes principales.

En contraste con los framework (marcos) más grandes donde la mayoría de las elecciones ya están hechas y son difíciles de cambiar; como desarrollador tienes el poder de elegir las extensiones que mejor se adapten para el proyecto.

Aprenderemos a instalar flask; único requisito es tener Python instalado.

Entornos virtuales.

La forma más conveniente de instalar Flask es utilizar un entorno virtual.

- Un entorno virtual es una copia del intérprete de Python en el que se puede instalar paquetes de forma privada, sin afectar al intérprete global de Python.
- Los entornos virtuales evitan el desorden de paquetes y conflicto de versiones en el intérprete de Python del sistema.
- La creación de entorno virtual para cada proyecto garantiza que las aplicaciones tengan acceso solo a los paquetes que utilizan, mientras que el intérprete global permanece ordenado y limpio y sirve solo como una fuente a partir del cual se pueden crear más entornos virtuales.
- Cada entorno virtual es independiente y se puede crear y administrar.

Creación de un entorno virtual con Python 3

Para la creación de entornos virtuales con python 3 es de manera nativa con el paquete **venv** que forma parte de la biblioteca estándar de Python. Para un sistema Ubuntu Linux el paquete venv hay que instalarlo: $ sudo apt-get install python3-venv

El comando para crear un entorno virtual tiene la siguiente estructura:

$ python3 -m venv virtual-environment-name

Nota: como estamos guiando de un código fuente, vamos a crear el entorno virtual dentro de la carpeta “flasky”.

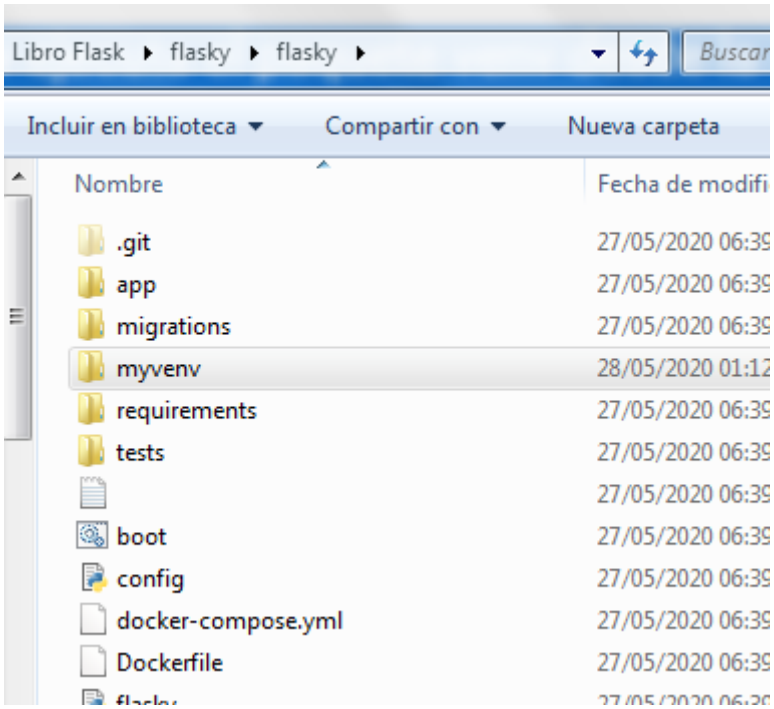
nota 2: Es común llamar “venv” a los entornos virtuales.

Vamos a crear un entorno virtual dentro de la carpeta (directorio) flasky.

```
C:\Users\PEDRITO\Libro Flask\flasky\flasky>python -m venv myvenv

C:\Users\PEDRITO\Libro Flask\flasky\flasky>
```

ahora tenemos un directorio de nombre “myvenv” donde se ha creado el entorno virtual



Creación de un entorno virtual con Python 2

Python 2 no tiene un paquete venv, los entornos virtuales se crean con utilidades de terceros virtualvenv.

Para Linux o macOs, el comando es: $ sudo pip install virtualenv

Para Windows, el comando es: $ pip install virtualvenv

Para crear un entorno virtual: $ virtualenv venv

se habrá creado un entorno virtual con el nombre “venv”

Trabajando con Entornos Virtuales

Cuando se desee comenzar a usar un entorno virtual, debe “activarlo”.

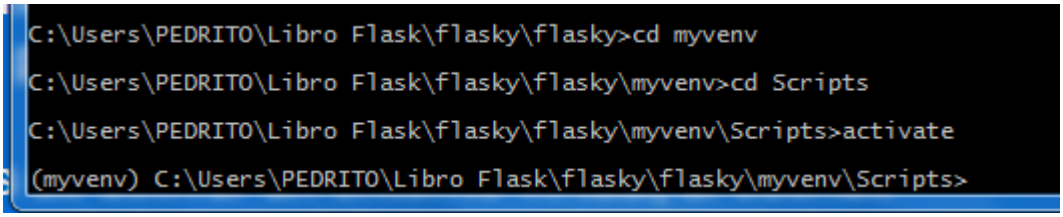
- Para Linux o macOs, puede activar el entorno virtual con el comando:
  $ source venv/bin/activate
- Para Microsoft Windows:
  $ venv\Scritps\activate

Cuando se activa un entorno virtual, la ubicación de su intérprete de Python se agrega a la variable de entorno de PATH en su sesión de comando actual, que determina dónde buscar ejecutables. Para saber que se ha activado un entorno virtual el comando de activación modifica su símbolo del sistema para incluir el nombre del entorno:

(myvenv) $

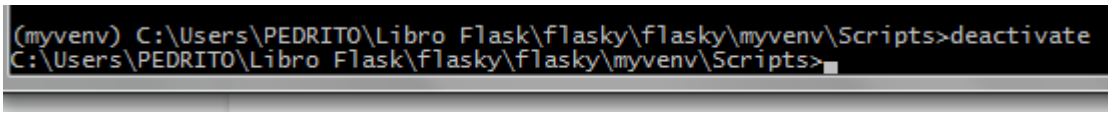
Después de activar el entorno virtual, al escribir python en el símbolo del sistema invocará al intérprete desde el entorno virtual en lugar del intérprete de todo el sistema. Si se usa más de un entorno virtual, se debe activar cada una de ellas.

En nuestro caso:



Para desactivar el entorno virtual escribir "**deactivate**" para restaurar la variable de entorno PATH para su sesión terminal y el símbolo del sistema a sus estados originales.

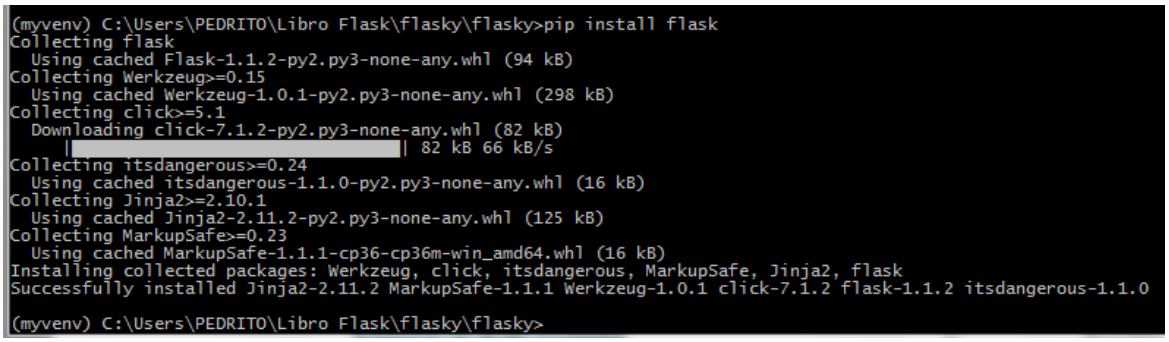
En nuestro caso:



Instalando Paquetes de Python con PIP

Los paquetes de Python se instalan con el administrador de paquetes pip que se incluyen en todos los entornos virtuales. Al igual que el comando python, escribir pip en una sesión de símbolo de sistema invocará la versión de esta herramienta que pertenece al entorno virtual activado. Para instalar FLASK en el entorno virtual, asegurar que el entorno virtual esté activado y luego ejecutar el siguiente comando:

- **(venv) $ pip install flask**



Cuando se ejecuta el comando, pip no solo instala Flask, sino también todas sus dependencias, se puede verificar los paquetes instalados con el comando:

- **(venv) $ pip freeze**

```
(myvenv) C:\Users\PEDRITO\Libro Flask\flasky\flasky>pip freeze
click==7.1.2
Flask==1.1.2
itsdangerous==1.1.0
Jinja2==2.11.2
MarkupSafe==1.1.1
Werkzeug==1.0.1

(myvenv) C:\Users\PEDRITO\Libro Flask\flasky\flasky>
```

La salida de **pip freeze** incluye número de versiones detallados para cada paquete instalado.

Nota: Por lo general la versión de pip una vez instalado el entorno virtual está desactualizado, nos sugieren actualizar: