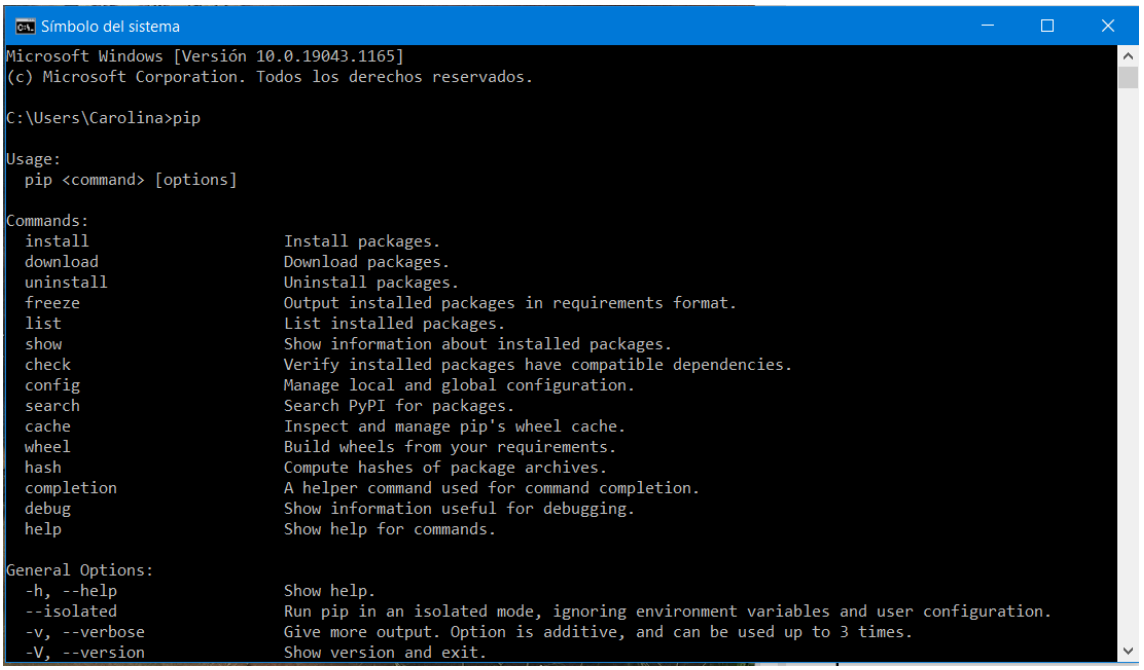
Introducción a Python - Django

Metodología de la Programación
UNJu - Facultad de Ingeniería

PIP (Package Installer for Python)

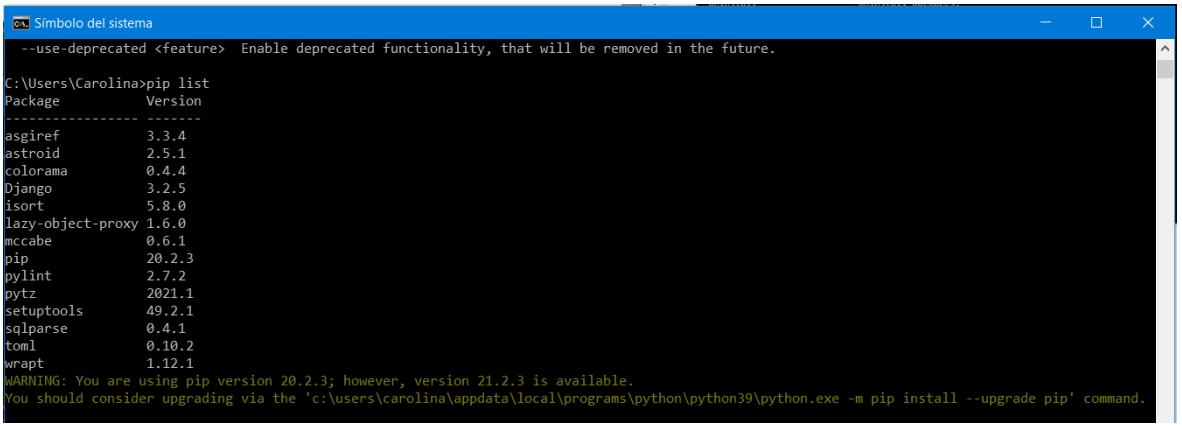
PIP es el Administrador de Paquetes de Python es útil para la creación de aplicaciones utilizando Python. Se puede emplear desde la terminal de VSC o bien desde el símbolo del sistema **cmd** de Windows.
Al ejecutar el **`pip`** se pueden ver una serie de comandos disponibles:



Comandos PIP

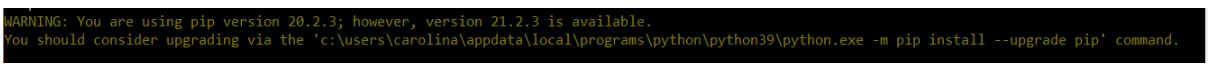
- **`pip list`**

Lista los distintos paquetes que se encuentran instalados y disponibles en el entorno de trabajo de Python.



- **`Actualización versión pip`**

Pip tiene versiones, en la Figura anterior se puede un mensaje en el indica la versión actual instalada y que se encuentra disponible una nueva versión.



La actualización se puede realizar mediante:

```
python.exe -m pip install --upgrade pip
```

Entornos virtuales

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 interfaz 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.
Un **entorno virtual** aísla tu configuración de Python/Django para cada proyecto. Esto quiere decir que cualquier cambio que hagas en un sitio web no afectará a ningún otro que estés desarrollando.

virtualenv

Es la herramienta base para la creación de entornos Python. Fue diseñada para poder desarrollar proyectos en Python trabajando con diferentes versiones de paquetes que, de otra forma, entrarían en conflicto.
Para la instalación se emplea el gestor de paquetes de Python **`pip`** a través de:

```
pip install virtualenv
```

`virtualenv` guarda cada entorno virtual en un directorio con el nombre de ese entorno. Dentro de ese directorio se guardarán todos los archivos necesarios (de ellos nos interesan, en particular, los módulos que instalemos en el entorno y el script que lo inicia).

pipenv

Es la herramienta oficial recomendada para el empaquetado, ya que soluciona defectos que **`virtualenv`** tiene, como por ejemplo no distinguir entre las dependencias del proyecto. **`Pipenv`** simplifica la gestión de dependencias para proyectos Python.

Para instalar **`pipenv`** solo debemos emplear **`pip`** mediante:

```
pip install pipenv
```

Al final se crea un archivo **`pipfile`**. Este archivo contiene información de lo que necesita el proyecto. Tiene una sección package que tienen los paquetes necesarios, por ejemplo, django. También tiene una sección de requerimientos que indica que versión de Python utiliza.

Para ver las opciones disponibles de este gestor de entornos virtuales se ejecuta:

```
pipenv --help
```

- Se puede crear una copia del intérprete de Python de la pc pero aislado, para un proyecto:

  ```
  pipenv --python 3.9
  ```

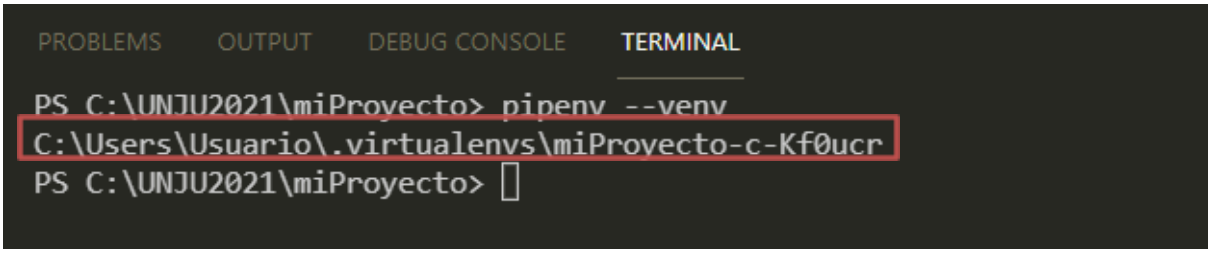
  Este comando crea el entorno virtual con la versión del entorno del sistema.

- Para borrar el entorno virtual instalado (desde el directorio en el que se creó)

  ```
  pipenv --rm
  ```

- Para mostrar la ruta del entorno creado se emplea :

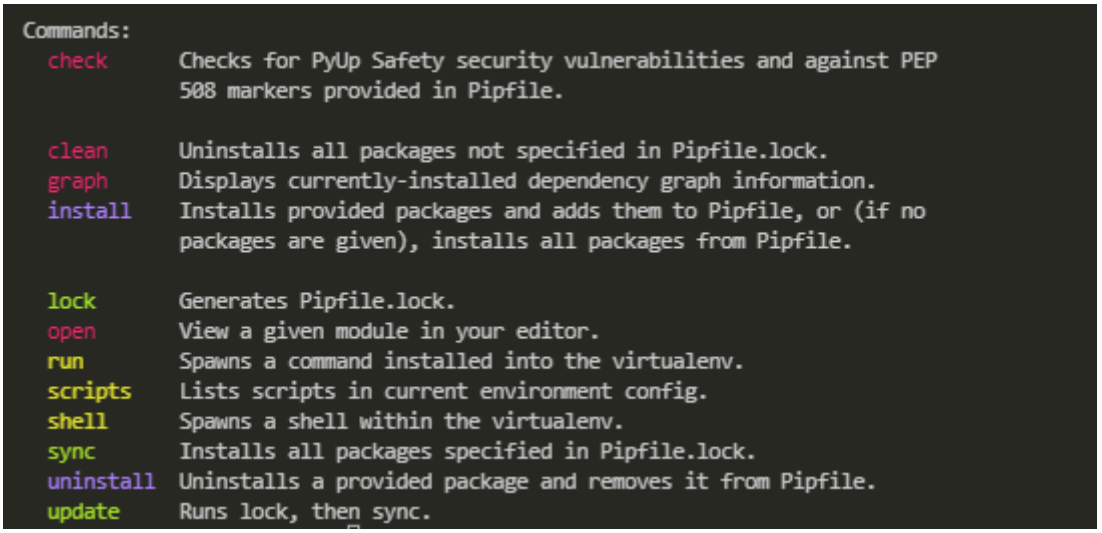
  ```
  pipenv --venv
  ```



En la carpeta
`C:\Users\Usuario\.virtualenvs\miProyecto-c-Kf0ucr\Lib\site-packages`
están los paquetes creados en el entorno virtual.

- Para mostrar la ruta del intérprete de python del entorno virtual se emplea:

`pipenv --py`



- Para acceder al intérprete de Python del entorno virtual

`pipenv run Python`

- Para ver la dependencia entre los módulos se emplea:

`pipenv graph`

- Para instalar otros paquetes debo utilizar el comando

`pipenv install [nombre del paquete]`

Por ejemplo para instalar el paquete numpy en el entorno virtual.

`pipenv install numpy`

- Para desinstalar un paquete utilizar el comando:

```
pipenv uninstall [nombre del paquete]
```

Por ejemplo para desinstalar el paquete instalado anteriormente (numpy)

```
pipenv uninstall numpy
```

- Para mostrar los paquetes del sistema operativo.

```
pip list
```

- Para iniciar/activar un nuevo entorno virtual en el proyecto podemos hacer:

```
pipenv shell
```

- Del entorno virtual se sale mediante:

```
exit
```

Crear un entorno virtual con pipenv

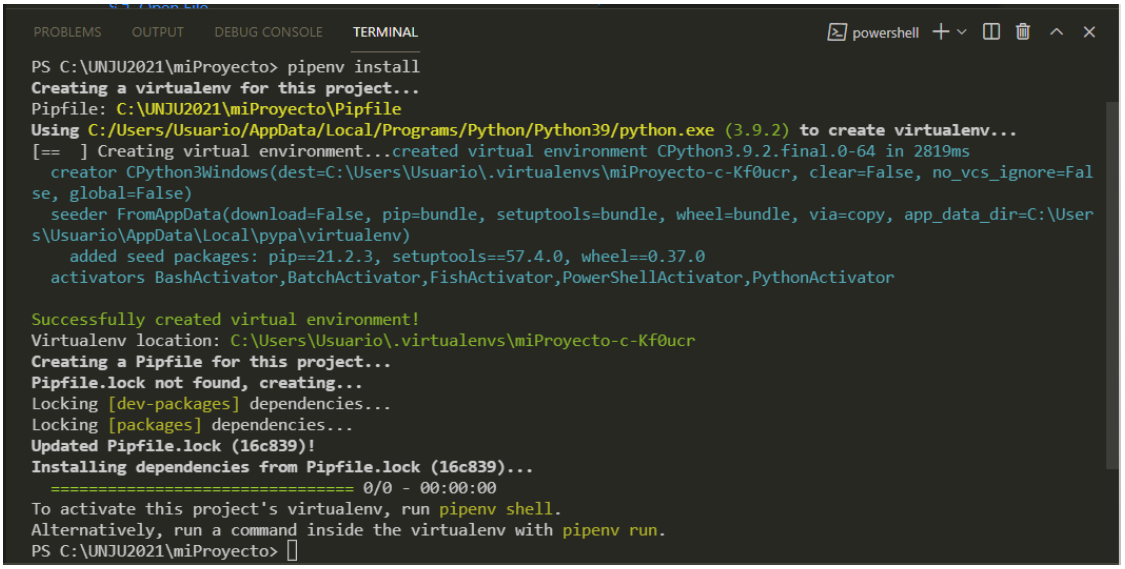
A continuación se muestra la creación de un entorno virtual en un proyecto.

1. Crear una carpeta vacía donde se creará el proyecto. Por ejemplo **miProyecto**, luego la carpeta creada se abrirá con VSC.

2. A través del terminal de VSC (Terminal New Terminal o Ctrl+Shift+ñ), creamos el entorno virtual para dicho proyecto (asegurarse de estar en la carpeta indicada).

   Ejecutamos:

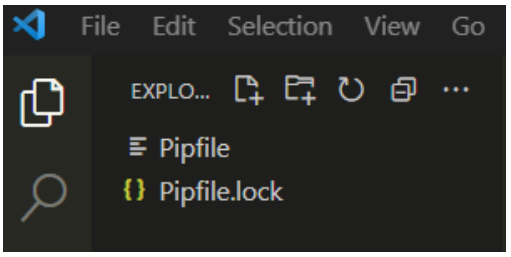
```
pipenv install
```



Cuando se cree el entorno virtual, automáticamente se nombrará con el nombre de la carpeta raíz seguido de un identificador y se guardará en:

- En Linux/Mac: /home/<USUARIO>/.local/share/virtualenvs/
- En Windows: C:\Users\<USUARIO>\.virtualenvs\

Luego de crear el entorno virtual podemos ver que se crearon los archivos **`pipfile`** y **`pipfile.lock`**.



Estos dos archivos son los requisitos de nuestro proyecto con la configuración del entorno virtual exclusivo; que los creó Pipenv al crear el entorno virtual. Cuando se instalen dependencias se completaran las secciones correspondientes (se añadirán líneas con las información del paquete que instalemos, como su nombre y la versión), se podrá modificar para cambiar la configuración a mano (como cambiar la versión de Python), y en última instancia se podrá regenerar desde ellos el entorno virtual (tanto si algo falla del entorno virtual para poder recrearlo, así como para copiar/mover nuestro proyecto a otra máquina para crear el entorno virtual con exactamente la configuración necesaria para que nuestro proyecto funcione).

Importante: no se puede cambiar el nombre de la carpeta del proyecto, los entornos virtuales quedan enlazados a la carpeta del proyecto donde se creó.

3. Para activar el entorno virtual se emplea:

```
pipenv shell
```

```
PS C:\UNJU2021\miProyecto> pipenv shell
Launching subshell in virtual environment...
Windows PowerShell
Copyright (C) 2014 Microsoft Corporation. Todos los derechos reservados.

PS C:\UNJU2021\miProyecto>
```

Para indicar que se activó correctamente aparecerá delante de la ruta (miProyecto), **`pipenv`** ha llamado automáticamente al **`activate`** de **`virtualenv`**.
Otra forma de ver que se está trabajando en el entorno virtual es ver que se cambió el nombre del terminal.

4. Instalar las dependencias (paquetes) a utilizar en el entorno virtual mediante **`pipenv install`**.

Para agilizar la instalación de varios paquetes se puede emplear:

```
pipenv install <Paquete1><Paquete2><Paquete3>
```

En nuestro caso instalaremos los siguientes paquetes:

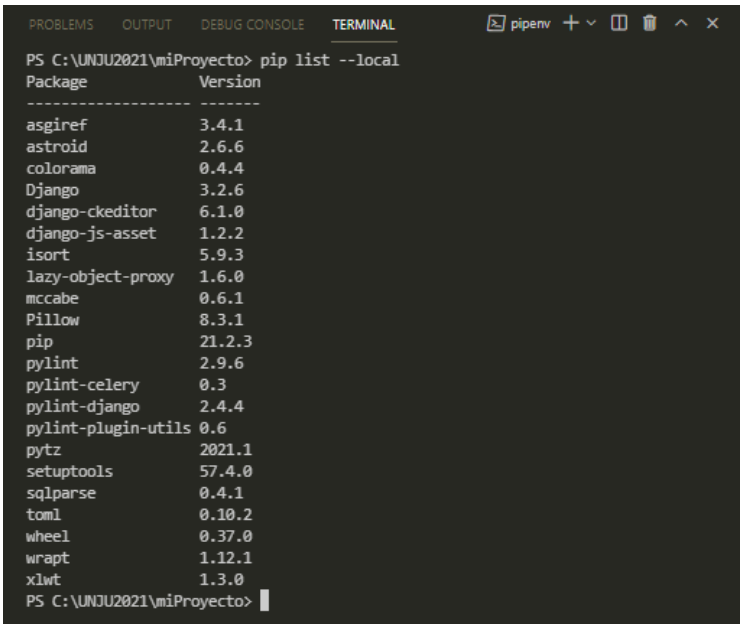
- **`django`**. Es el motor de django.
- **`django-ckeditor`**. Es el paquete que permite colocar un editor de texto enriquecido para django.
- **`Pillow`**. Es un paquete para el manejo de las imágenes en django.
- **`pylint`**. Es la librería que permite validar el código escrito en Python
- **`pylint-django`**:
- **`pylint-celery`**

Para realizar la instalación empleamos:

```
pipenv install django django-ckeditor Pillow pylint pylint-django pylint-celery
```

Para ver todos los paquetes instalados por pip los podemos listar con:

```
pip list –local
```



O también con

`pip graph`

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL
PS C:\UNJU2021\miProyecto> pipenv graph
Django==3.2.6
  - asgiref [required: >=3.3.2,<4, installed: 3.4.1]
  - pytz [required: Any, installed: 2021.1]
  - sqlparse [required: >=0.2.2, installed: 0.4.1]
django-ckeditor==6.1.0
  - django-js-asset [required: >=1.2.2, installed: 1.2.2]
Pillow==8.3.1
pylint-celery==0.3
  - astroid [required: >=1.0, installed: 2.6.6]
    - lazy-object-proxy [required: >=1.4.0, installed: 1.6.0]
    - setuptools [required: >=20.0, installed: 57.4.0]
    - wrapt [required: >=1.11,<1.13, installed: 1.12.1]
  - pylint [required: >=1.0, installed: 2.9.6]
    - astroid [required: >=2.6.5,<2.7, installed: 2.6.6]
      - lazy-object-proxy [required: >=1.4.0, installed: 1.6.0]
      - setuptools [required: >=20.0, installed: 57.4.0]
      - wrapt [required: >=1.11,<1.13, installed: 1.12.1]
    - colorama [required: Any, installed: 0.4.4]
    - isort [required: >=4.2.5,<6, installed: 5.9.3]
    - mccabe [required: >=0.6,<0.7, installed: 0.6.1]
    - toml [required: >=0.7.1, installed: 0.10.2]
  - pylint-plugin-utils [required: >=0.2.1, installed: 0.6]
    - pylint [required: >=1.7, installed: 2.9.6]
      - astroid [required: >=2.6.5,<2.7, installed: 2.6.6]
        - lazy-object-proxy [required: >=1.4.0, installed: 1.6.0]
        - setuptools [required: >=20.0, installed: 57.4.0]
        - wrapt [required: >=1.11,<1.13, installed: 1.12.1]
      - colorama [required: Any, installed: 0.4.4]
      - isort [required: >=4.2.5,<6, installed: 5.9.3]
      - mccabe [required: >=0.6,<0.7, installed: 0.6.1]
      - toml [required: >=0.7.1, installed: 0.10.2]
pylint-django==2.4.4
  - pylint [required: >=2.0, installed: 2.9.6]
    - astroid [required: >=2.6.5,<2.7, installed: 2.6.6]
      - lazy-object-proxy [required: >=1.4.0, installed: 1.6.0]
```

5. Se puede verificar la versión de django instalada en el entorno virtual mediante:

`python -m django - -version`

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL
PS C:\UNJU2021\miProyecto> python -m django --version
3.2.6
PS C:\UNJU2021\miProyecto>
```

6. La prueba anterior funciona. Para ver algo más interesante empleamos **`django-admin`** para crear un proyecto

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL                pipenv
PS C:\UNJU2021\miProyecto> django-admin startproject mytestsite
PS C:\UNJU2021\miProyecto> cd mytestsite
PS C:\UNJU2021\miProyecto\mytestsite>
```

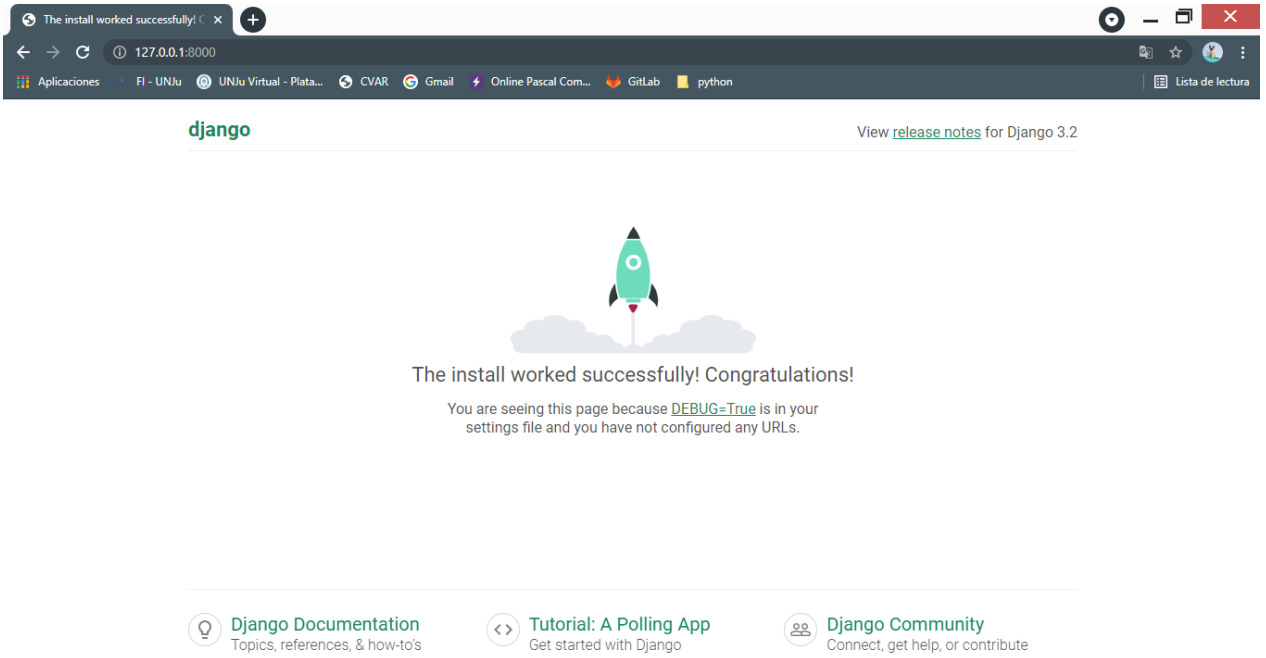
Al realizar **`django-admin startproject mytestsite`** se crea el proyecto con una carpeta a la cual uno se debe cambiar mediante **`cd mytestsite`**

Para ejecutar el proyecto, si ya se activó el entorno virtual se emplea:

```
python manage.py runsever
```

Podemos ver que funciona accediendo al servidor mediante

http://127.0.0.1:8000/



Archivos Pipfile.lock y Pipfile

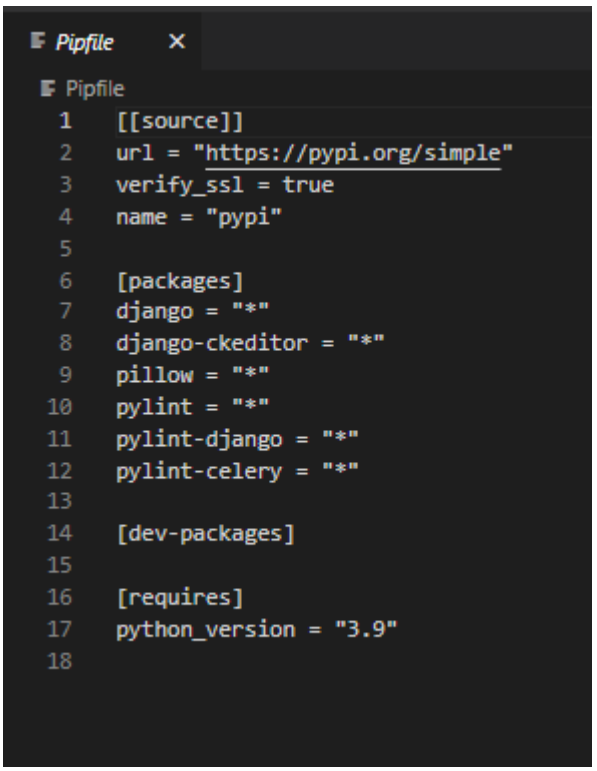
Los archivos "**`Pipfile.lock`**" y "**`Pipfile`**" son los archivos de requisitos donde se guarda la configuración completa del entorno virtual de Python (**`Virtualenv`**) gestionado por **`Pipenv`**.

Estos serán útiles si en un futuro copiamos/distribuimos la carpeta de nuestro proyecto que contenga estos dos ficheros a cualquier otra máquina (otro ordenador u otro servidor), y si volvemos ejecutamos el comando en esa otra máquina:

"Pipfile" por defecto contiene lo siguiente:

- **`[source]`**: desde donde se conecta PIP para descargar los paquetes (se podría configurar para que descargue los paquetes de un repositorio privado de una empresa, por ejemplo).

- **`[dev-packages]`**: los paquetes que sean exclusivos para el desarrollo (como los paquetes para realizar test, para comprobar rendimiento, etc.). Estos paquetes se pueden excluir para cuando tengamos una versión de nuestro proyecto para ser lanzada (release).
- **`[packages]`**: los paquetes que queremos utilizar en nuestro proyecto y que serán incluidos en la versión lanzada (release).
- **`[requires]:`** nos permitirá cambiar la versión de Python si lo deseamos cambiando el número de versión en "python_version" (siempre y cuando tengamos el Python global con la versión deseada).



Podemos modificar a mano cualquier el elemento de este archivo, por ejemplo, la versión de Python, guardaríamos el archivo y es importante eliminar el entorno virtual (sino lo hacemos nos mostrará un mensaje de advertencia) con:

```
pipenv –rm
```

Y lo volvemos a crear con:

```
pipenv install
```

El fichero de requisitos más importante es "Pipfile", con este fichero Pipenv generará el entorno virtual (al ejecutar "pipenv install" sino existe uno previo generará un fichero "Pipfile" por defecto; si existe, lo leerá y aplicará dicha configuración) y creará el fichero "Pipfile.lock" (lo mismo, sino existe se creará y si existe se leerá). El fichero "Pipfile.lock" es también importante para nuestro proyecto ya que definirá las versiones concretas con las que funcionará nuestro proyecto; es decir, nos asegurará una configuración en concreto ("bloqueará una configuración", por eso lo de "lock"), para que no se cambien las versiones (principalmente para que no se actualicen las versiones de paquetes o Python) si copiamos/distribuimos nuestro proyecto y por ello nos aseguramos que funcione correctamente en cualquier máquina.

"Pipfile.lock" no debería ser modificado a mano, mejor regenerarlo desde "Pipfile", contiene información parecida a la siguiente en formato JSON:

En este archivo se ven los paquetes instalados, sus dependencias si las tiene y la versión concreta utilizada, entre otra información como hashes, etc.
Tanto “Pipfile.lock” como “Pipfile” son parte imprescindible del proyecto por lo que deberían ser incluidos en un control de versiones (un repositorio como Git).

Referencias

- Documentación pipenv. Recuperado de: https://docs.pipenv.org/