

# Entornos de producción

Llevo todo el curso repitiendo entornos de producción por aquí, entornos de producción por allá, así que ha llegado la hora de aprender un poco más sobre ellos.

Cuando hablamos de entornos normalmente hacemos referencia a un “lugar” donde se ejecuta el código, aunque quizá sería más acertado entenderlo como una configuración que envuelve la ejecución del propio software.

Los entornos más comunes son el de desarrollo y el de producción, aunque también existen otros como los de pruebas.

A grandes rasgos, el entorno de desarrollo es donde los programadores crean las aplicaciones, y suele contener configuraciones que les ayudan en el proceso. En Django eso se refleja muy bien, por ejemplo, cuando tenemos el DEBUG en True, pues cualquier error que surja se nos mostrará en el frontend para facilitarnos la depuración.

En cambio el entorno de producción es el que está dirigido al usuario final, el lugar donde correrá el código una vez está funcionando públicamente. Aquí no tendría mucho sentido activar el DEBUG y de hecho sería una completa falla de seguridad, pues cualquiera podría provocar un fallo y analizar el registro para recopilar información privada.

Otro cambio importante podría ser a la hora de servir ficheros estáticos, pues si bien durante el desarrollo se puede configurar Django para servirlos y facilitarnos la vida, esta funcionalidad no está pensada para un entorno de producción con multitud de usuarios. De esa tarea se encargarían servidores preparados para ello como Nginx o Apache.

Incluso podríamos hablar de bases de datos, pues JAMÁS deberíamos trabajar con la misma en ambos entornos, y de hecho sería uno de los errores más desafortunadas que podéis cometer como desarrolladores de software. Sólo imaginaros que haciendo pruebas borrais un montón de registros... pues como no tengáis una copia de seguridad os puede caer una buena demanda por parte del cliente eh.

Por tanto queda claro que no se puede utilizar la misma configuración mientras desarrollamos nuestra aplicación que cuando la tenemos colgada en Internet, y se requieren realizar una serie de cambios para que todo funcione.

Bueno, pues este proceso de adaptar el código de un entorno a otro se conoce como la fase de despliegue, y contiene la instalación, configuración y unas pruebas básicas de funcionamiento.

En definitiva “Desplegar Django” es el nombre que recibe el proceso de copiar y adaptar el proyecto desde un entorno de desarrollo a un entorno de producción. Sin embargo como copiar los datos literalmente de un sitio a otro sería muy

tedioso y poco seguro, existen los repositorios, hablaremos de ellos en la siguiente lección.

Por ahora, os dejo un enlace con la lista de cosas que desde la documentación oficial Django nos recomiendan modificar de cara a un buen despliegue: <https://docs.djangoproject.com/en/dev/howto/deployment/checklist/>

## Repositorios y controles de versiones

Nos habíamos quedado en que gracias a los repositorios hoy en día ya no debemos ir copiando datos directamente de un entorno de desarrollo a uno de producción, algo bastante tedioso e inseguro. Así que lo primero es dejar claro qué es exactamente un repositorio, pues no hay que confundirlo con un sistema control de versiones.

- Un repositorio es un sitio centralizado donde se almacenan y organizan archivos informáticos, cuya función principal es distribuirlos, ya sea a través de una red (pública o privada) o un medio físico, como un disco compacto.
- Por otro lado, un sistema control de versiones es un programa pensado para la gestión de los cambios que se realizan en un producto informático, ya sea en forma de revisiones, versiones o ediciones del mismo. Existen muchas alternativas entre las que caben destacar: Apache Subversion (abreviado SVN), Mercurial o la más utilizada hoy en día: Git.

La clave es la unión de ambos conceptos, pues de ella surge la idea de un repositorio compatible con un control de versiones, y precisamente hay varias empresas que se dedican a dar este servicio. Las más utilizadas las encontramos para el sistema GIT y son: Github, Gitlab y Bitbucket.

Github es el servicio por excelencia, la gestión es excelente y nos permite almacenar tantos repositorios públicos como queramos, el negocio sin embargo, lo hacen con los repositorios privados, pues hay que pagar una cuota para poder almacenarlos.

Por su parte Gitlab y Bitbucket no son tan famosos, pero nos permiten crear tanto repositorios públicos como privados de forma gratuita, y sus servicios de pago están más centrados en la gestión de usuarios, equipos y seguridad extra.

En cualquier caso crear un repositorio es bastante intuitivo y el despliegue se hace exactamente igual, por eso creo que lo mejor es empezar con Github y luego probáis por vuestra cuenta otras alternativas si queréis almacenar proyectos de forma privada. Además Github lo podéis utilizar como portafolio, os aconsejo publicar vuestros proyectos personales ahí y compartir el enlace en el currículum para hacer gala de vuestros logros. Por cierto, mi usuario en Github es hcosta (mostrar enlace), podéis seguirme si queréis ver en que ando metido.

Mi repositorio personal en Github: <https://github.com/hcosta>

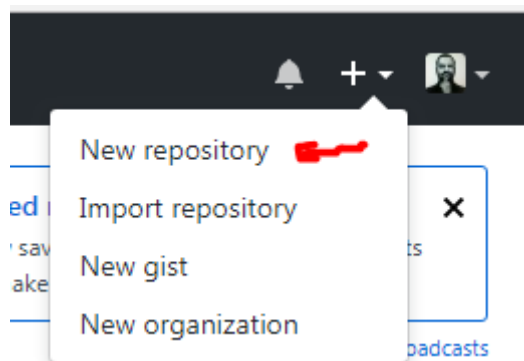
Repositorio de mi academia: <https://github.com/hektorprofe>

## Creando un repositorio en Github

En esta lección vamos a crear un repositorio para almacenar uno de nuestros proyectos y utilizarlo en el despliegue. No me voy a centrar en enseñaros Git a fondo, sólo los cuatro comandos básicos para crear el repo y enviar los datos.

Así que vamos a suponer que todos tenemos una cuenta en [github.com](https://github.com) y tenemos Git instalado en nuestra computadora, al fin y al cabo era un requisito para utilizar Visual Studio Code. Si no lo tenéis podéis descargarlo en [git-scm.com](https://git-scm.com).

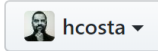
A continuación voy a crear mi repositorio en Github para almacenar mi proyecto webempresa, ya que este es el que os enseñaré a desplegar:



## Create a new repository

A repository contains all the files for your project, including the revision history.

Owner



Repository name

/ web-personal-curso-django-2 ✓

Great repository names are short and memorable. Need inspiration? How about [fluffy-spork](#).

Description (optional)

Repositorio para almacenar la web que hemos creado en el curso Django 2



Public

Anyone can see this repository. You choose who can commit.



Private

You choose who can see and commit to this repository.



Initialize this repository with a README

This will let you immediately clone the repository to your computer. Skip this step if you're importing an existing repository.



Add .gitignore: Python ▾

Add a license: None ▾



Create repository

hcosta Initial commit

Latest commit d9ea34a just now

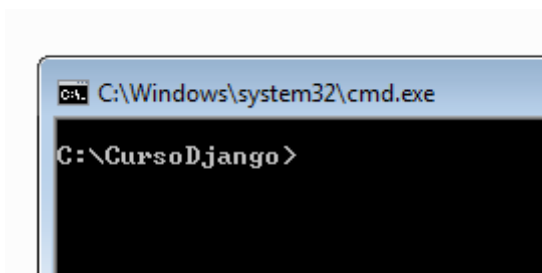
.gitignore	Initial commit	just now
README.md	Initial commit	just now

README.md

# web-personal-curso-django-2

Repositorio para almacenar la web que hemos creado en el curso Django 2

Ahora vamos a clonar el repositorio en nuestro equipo. Abrimos una terminal en el directorio donde queremos clonarlo, en mi caso CursoDjango:



Copiamos la URL para clonarlo:

## Clone with HTTPS ?

Us

Use Git or checkout with SVN using the web URL.

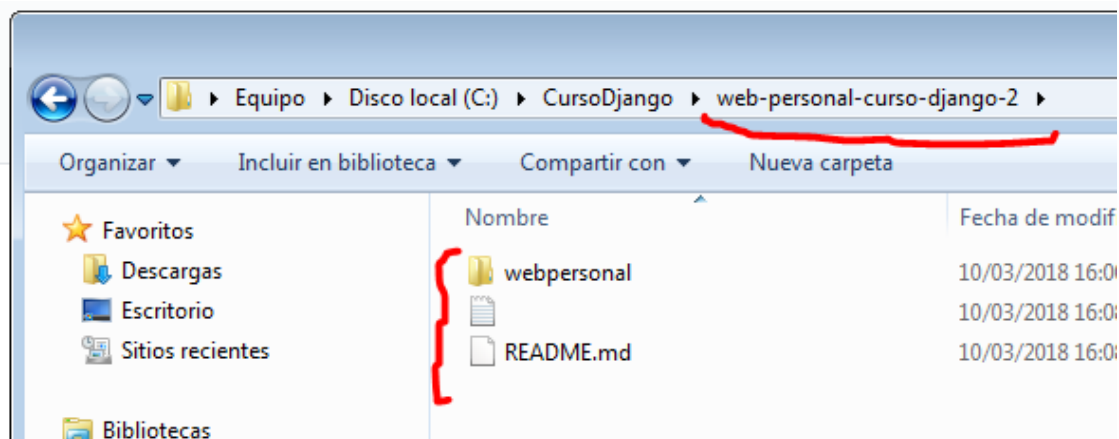
<https://github.com/hcosta/web-personal-cur>

Y utilizamos el comando git clone \<url>:

```
C:\Windows\system32\cmd.exe

C:\CursoDjango>git clone https://github.com/hcosta/web-personal-curso-django-2
Cloning into 'web-personal-curso-django-2'...
remote: Counting objects: 4, done.
remote: Compressing objects: 100% (4/4), done.
remote: Total 4 (delta 0), reused 0 (delta 0), pack-reused 4
Unpacking objects: 100% (4/4), done.
C:\CursoDjango>_
```

Se creará una carpeta con nuestro repositorio, trasladaremos el proyecto a ella, pues a partir de ahora trabajaremos ahí:



Accederemos a la carpeta a través de la terminal y ejecutaremos el comando: git add . (para añadir nuestro proyecto al repositorio).

```
C:\CursoDjango>cd web-personal-curso-django-2
C:\CursoDjango\web-personal-curso-django-2>git add .
warning: LF will be replaced by CRLF in webpersonal/c
og.js.
The file will have its original line endings in your
```

A continuación crearemos un commit para configurar una actualización con todos los cambios que hemos hecho:

```
C:\CursoDjango\web-personal-curso-django-2>git commit -m "Añadido el proyecto"
*** Please tell me who you are.

Run

  git config --global user.email "you@example.com"
  git config --global user.name "Your Name"

to set your account's default identity.
Omit --global to set the identity only in this repository.

fatal: unable to auto-detect email address (got 'Hektor@Eureka.<none>')
C:\CursoDjango\web-personal-curso-django-2>
```

Al hacerlo por primera vez nos pedirá antes configurar un perfil con un email y nombre. Utilizaremos el comando git config dos veces. Si vamos a utilizar siempre este email y nombre en todos los proyectos podemos usar la cláusula global tal como nos indica:

```
C:\CursoDjango\web-personal-curso-django-2>git config --global user.email "hcosta
aguzman@gmail.com"

C:\CursoDjango\web-personal-curso-django-2>git config --global user.name "Héctor
Costa"
```

Volveremos a hacer el commit:





```
C:\CursoDjango\web-personal-curso-django-2>git commit -m "Añadido el proyecto"
[master fecca881] Añadido el proyecto
 98 files changed, 47242 insertions(+)
```

Veremos un resumen con los cambios, ya sólo falta enviar los datos al repositorio, que haremos con un git push origin master:

```
C:\CursoDjango\web-personal-curso-django-2>git push origin master
Username for 'https://github.com': hcosta
Password for 'https://hcosta@github.com':
Counting objects: 122, done.
Delta compression using up to 4 threads.
Compressing objects: 100% (113/113), done.
Writing objects: 100% (122/122), 5.53 MiB | 2.71 MiB/s, done.
Total 122 (delta 11), reused 0 (delta 0)
remote: Resolving deltas: 100% (11/11), done.
To https://github.com/hcosta/web-personal-curso-django-2.git
 d9ea34a..fecca88 master -> master

C:\CursoDjango\web-personal-curso-django-2>
```

Después de poner nuestro usuario y contraseña, veremos que se han publicado los cambios en la web del repositorio y podemos navegar en él:

 hcosta Añadido el proyecto		Latest commit fecca88 3 minutes ago
 webpersonal	Añadido el proyecto	3 minutes ago
 .gitignore	Initial commit	18 minutes ago
 README.md	Initial commit	18 minutes ago

El paso de introducir el usuario y la contraseña nos lo podemos ahorrar generando una clave SSH, os dejaré un enlace os explican cómo se hace: <https://help.github.com/articles/generating-a-new-ssh-key-and-adding-it-to-the-ssh-agent/>

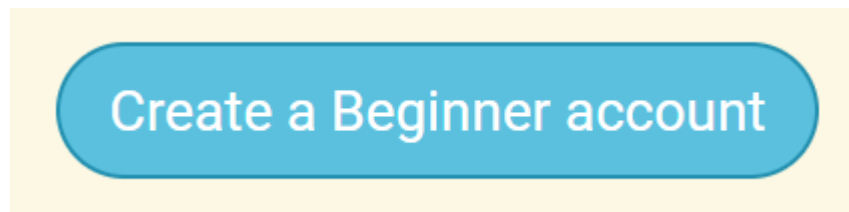
Sea como sea ya tenemos el proyecto en un repositorio, así que en la próxima lección veremos como adaptarlo para desplegarlo en un servicio online gratuito.

## Desplegando un proyecto de prueba

Os aviso de antemano de que esta lección será muy larga, realizar un despliegue no es cualquier cosa y requiere configurar un montón de opciones, pero el resultado es muy grato y estoy seguro de que será una buena experiencia.

Existen muchas empresas que nos permiten desplegar nuestros proyectos en la nube, como Heroku o AWS (Amazon Web Services), incluso se puede desplegar en un servidor privado, pero en cualquier caso se requieren conocimientos de administración de sistemas. Os compartiré tutoriales y manuales para algunos de estos servicios en la siguiente lección. Y por mi parte me centraré en enseñaros cómo hacerlo en PythonAnywhere, un Hosting especializado en proyectos Python, que nos da la posibilidad de desplegar un proyecto gratuito de una forma bastante fácil, así por lo menos os sirve de experiencia práctica.

Lo primero será ir a la web <http://www.pythonanywhere.com> y crear una cuenta de principiante. El nombre de la cuenta será la dirección donde se publicará nuestro proyecto, así que elegid un buen nombre.



## Create your account

**Username:**

Your username can only contain letters and numbers.

**Email:**

**Password:**

**Password (again):**

☒ I agree to the [Terms and Conditions](#)

**Register**

We promise not to spam or pass your details on to anyone else.

Al registrarnos contaremos con nuestro propio subdominio, en mi caso: <http://webpersonal.pythonanywhere.com/> , con hasta 512 MB de espacio.

Bien, ahora deberíamos iniciar una terminal y clonar nuestro repositorio, configurarlo para producción y poner el servidor en marcha. Suena fácil, pero en realidad no lo es, básicamente porque cada hosting tiene sus normas y pautas a la hora de desplegar Django o cualquier otro servicio, obviamente en el caso de PythonAnywhere no iba a ser diferente.

En el enlace de la documentación `DeployExistingDjangoProject` (<https://help.pythonanywhere.com/pages/FollowingTheDjangoTutorial/> + <https://help.pythonanywhere.com/pages/DjangoStaticFiles>) se detalla punto a punto como desplegar correctamente Django, os lo dejaré en los recursos porque es básicamente lo que vamos a hacer.

Vamos a ir al apartado consoles y vamos a iniciar Bash para trabajar como si estuviéramos en la terminal del servidor.

El siguiente paso es crear un entorno virtual para ejecutar nuestro proyecto, pero en lugar de utilizar Anaconda aquí tenemos que crearlo con la herramienta VirtualEnv, es la forma más clásica:

```
16:06 ~ $ mkvirtualenv --python=python3.6 webpersonal
```

De forma similar a Anaconda podemos desactivar el entorno con:

```
deactivate
```



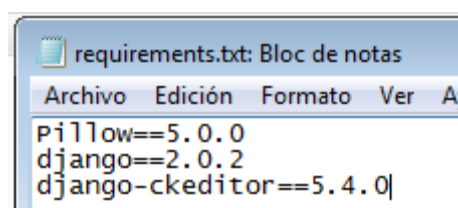
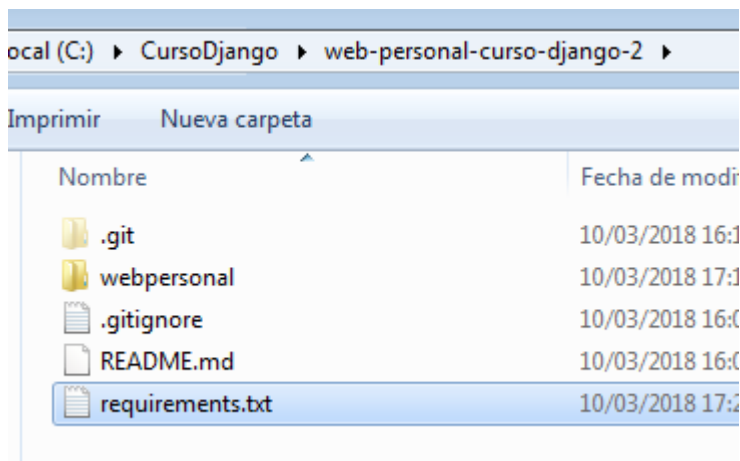
Sin embargo para activarlo aquí deberemos ejecutar el script activate desde su propio directorio con un source:

```
16:11 ~ $ source ~/.virtualenvs/webpersonal/bin/activate
(webpersonal) 16:12 ~ $
```

Podemos hacer un pip list para ver qué paquetes nos vienen instalados:

```
(webpersonal) 16:12 ~ $ pip list
DEPRECATION: The default format will switch to columns in your pip.conf under the [
pip (9.0.1)
setuptools (38.5.2)
wheel (0.30.0)
(webpersonal) 16:14 ~ $
```

Obviamente si vamos a utilizar Django y otros paquetes como Pillow y django-ckeditor deberíamos instalarlos aquí mismo, pero es una mejor práctica crear un fichero requirements.txt en nuestro repositorio para luego instalarlo de forma mucho más cómoda. Iremos a nuestro directorio del repositorio, crearemos el fichero y añadiremos lo siguiente, haciendo referencia al nombre del paquete y a la versión que sabemos es estable en nuestro proyecto:



Ahora vamos a publicar los cambios:

```
C:\CursoDjango\web-personal-curso-django-2>git add .
C:\CursoDjango\web-personal-curso-django-2>git commit -m "Añadido requirements.txt"
[master a97029a] Añadido requirements.txt
1 file changed, 3 insertions(+)
create mode 100644 requirements.txt
C:\CursoDjango\web-personal-curso-django-2>git push origin master
```

Ahora vamos a clonar el repositorio desde la terminal Bash. Haremos un simple git clone (podemos pegar con Control+V):

```
(webpersonal) 16:14 ~ $ git clone https://github.com/hcosta/web-personal-curso-django-2.git
Cloning into 'web-personal-curso-django-2'...
remote: Counting objects: 129, done.
remote: Compressing objects: 100% (109/109), done.
remote: Total 129 (delta 13), reused 124 (delta 11), pack-reused 0
Receiving objects: 100% (129/129), 5.53 MiB | 0 bytes/s, done.
Resolving deltas: 100% (13/13), done.
Checking connectivity... done.
```

Con esto tenemos el repositorio ya en el servicio de PythonAnywhere, vamos a instalar las dependencias en el entorno virtual:

```
(webpersonal) 16:24 ~ $ cd web-personal-curso-django-2/
(webpersonal) 16:25 ~/web-personal-curso-django-2 (master)$ ls
README.md requirements.txt webpersonal
(webpersonal) 16:25 ~/web-personal-curso-django-2 (master)$ pip install -r requirements.txt
```

Esperamos un rato que instale todo...

```
Installing collected packages: Pillow, pytz, django, django-js-asset, django-ckeditor
Successfully installed Pillow-5.0.0 django-2.0.2 django-ckeditor-5.4.0 django-js-asset-1.0.0 pytz-2018.3
(webpersonal) 16:29 ~/web-personal-curso-django-2 (master)$
```

En este punto todos los paquetes habrán quedado instalados en nuestro entorno virtual y deberíamos ser capaces de ejecutar el scripts manage.py de forma normal.

Así que vamos a ejecutar el comando: python manage.py check --deploy

```
(webpersonal) 16:34 ~/web-personal-curso-django-2/webpersonal (master)$ python manage.py check --deploy
System check identified some issues:

WARNINGS:
?: (security.W004) You have not set a value for the SECURE_HSTS_SECONDS setting. If your entire site is served only over SSL, you may want to consider setting a value and enabling HTTP Strict Transport Security. Be sure to read the documentation first; enabling HSTS carelessly can cause serious, irreversible problems.
?: (security.W006) Your SECURE_CONTENT_TYPE_NOSNIFF setting is not set to True, so your pages will not be served with an 'x-content-type-options: nosniff' header. You should consider enabling this header to prevent the browser from identifying content types incorrectly.
?: (security.W007) Your SECURE_BROWSER_XSS_FILTER setting is not set to True, so your pages will not be served with an 'x-xss-protection: 1; mode=block' header. You should consider enabling this header to activate the browser's XSS filtering and help prevent XSS attacks.
?: (security.W008) Your SECURE_SSL_REDIRECT setting is not set to True. Unless your site should be available over both SSL and non-SSL connections, you may want to either set this setting True or configure a load balancer or reverse-proxy server to redirect all connections to HTTPS.
?: (security.W012) SESSION_COOKIE_SECURE is not set to True. Using a secure-only session cookie makes it more difficult for network traffic sniffers to hijack user sessions.
?: (security.W016) You have 'django.middleware.csrf.CsrfViewMiddleware' in your MIDDLEWARE, but you have not set CSRF_COOKIE_SECURE to True. Using a secure-only CSRF cookie makes it more difficult for network traffic sniffers to steal the CSRF token.
?: (security.W018) You should not have DEBUG set to True in deployment.
?: (security.W019) You have 'django.middleware.clickjacking.XFrameOptionsMiddleware' in your MIDDLEWARE, but X_FRAME_OPTIONS is not set to 'DENY'. The default is 'SAMEORIGIN', but unless there is a good reason for your site to serve other parts of itself in a frame, you should change it to 'DENY'.
?: (security.W020) ALLOWED_HOSTS must not be empty in deployment.

System check identified 9 issues (0 silenced).
(webpersonal) 16:35 ~/web-personal-curso-django-2/webpersonal (master)$
```

Es interesante ver cómo el propio Django analiza la configuración para avisarnos de todo lo que deberíamos solucionar antes de hacer poner el servidor en marcha en este entorno de producción.

No voy a entrar a comentar cada uno de ellos, la mayoría son poner a True algunas opciones en el settings.py, otros son recomendaciones que se entienden leyendo la descripción. Sin embargo hay dos que son sumamente importantes:

**You should not have DEBUG set to True in deployment.**

**ALLOWED\_HOSTS must not be empty in deployment.**

Así que tenemos que arreglarlos antes de continuar.

Simplemente editaremos el fichero settings.py con algún editor a través de la terminal:

```
nano webpersonal/settings.py
```

```
# SECURITY WARNING: don't run with debug turned on in production!  
DEBUG = False  
  
ALLOWED_HOSTS = ['localhost', '127.0.0.1', 'webpersonal.pythonanywhere.com']
```

Así desactivaremos el Debug y daremos permiso a los HOST locales, incluyendo el propio dominio de la página.

La base de datos no tenemos que hacer nada con ella, ya que la hemos publicado en el repositorio y se ha clonado tal cuál la teníamos. Ya os dije que esto no es una buena práctica, pero casualmente las bases de datos SQLite se gestionan en ficheros, por lo que esta es una copia de la otra, por lo que no tenemos el problema de borrar datos accidentalmente. En todo caso si utilizáis otro SGBD no uséis la misma. Deberíais migrar manualmente y crear el superadministrador desde la terminal bash.

Sea como sea ahora tenemos la configuración mínima recomendada para realizar el despliegue, y para ello debemos crear una webapp que se encargue de mantener el servicio de Django en marcha.

Abriremos otra pestaña en la sección

Apps: <https://www.pythonanywhere.com/user/webpersonal/webapps/> y crearemos una nueva app:

[+ Add a new web app](#)

You have no web apps

To create a PythonAnywhere-hosted web app, click the "Add a new web app" button to the left.

Aunque en la lista nos salga la opción Django, no seleccionaremos esa opción, primero porque se crearía un nuevo proyecto y nosotros ya tenemos uno existente, y segundo porque tampoco tiene soporte para Django 2 automático. Así que vamos a seleccionar la configuración Manual para Python 3.6:

» **Manual configuration** (including virtualenvs)

» **Python 3.6**

+

Ahora nos indicará que debemos configurar nuestro propio fichero WSGI:

## Manual Configuration

Manual configuration involves editing your own **WSGI** configuration file in `/var/www/`. Usually this imports a WSGI-compatible application which you've stored elsewhere

When you click "Next", we will create a WSGI file for you, including a simple "Hello World" app which you can use to get started, as well as some comments on how to use other frameworks.

You will also be able to specify a *virtualenv* to use for your app.

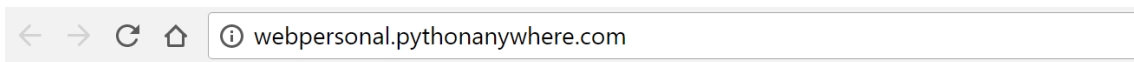
¿Recordáis al principio del curso cuando os comenté que dentro del proyecto Django, en el directorio donde está `settings.py` también hay un script llamado `wsgi.py`? Pues es hora de que este fichero entre en escena, aunque de una forma distinta, ya que pythonanywhere utiliza su propia versión. Así que presionamos NEXT para crear la web app y acto seguido, una vez se cree, llegaremos a la página de gestión de nuestra web app:

# Configuration for [webpersonal.pythonanywhere.com](https://webpersonal.pythonanywhere.com)

Reload:

 Reload [webpersonal.pythonanywhere.com](https://webpersonal.pythonanywhere.com)

De hecho ya podemos acceder a la página, aunque sólo nos aparecerá información de prueba:



## Hello, World!


This is the default welcome page for a [PythonAnywhere](#) hosted web application.

Find out more about how to configure your own web application by visiting the [web app setup](#) page

¿De dónde sale esta información de prueba? Pues ni nada menos que del fichero wsgi que se está utilizando ahora mismo:

Code:

What your site is running.

Source code:	<a href="#">Enter the path to your web app source code</a>	
Working directory:	<a href="#">/home/webpersonal/</a>	<a href="#">Go to directory</a>
WSGI configuration file:	<a href="#">/var/www/webpersonal_pythonanywhere_com_wsgi.py</a>	
Python version:	3.6 	

Virtualenv:

Use a virtualenv to get different versions of flask, django etc from our default system ones. [More info here](#). You need to **Reload your web app** to activate it; NB - will do nothing if the virtualenv does not exist.


[Enter path to a virtualenv, if desired](#)

Luego volveremos a él, pero antes vamos a realizar la configuración inferior. Debemos introducir el directorio con el código fuente del proyecto y también el

## del entorno virtual:

Code:

What your site is running.

Source code:	<a href="#">Enter the path to your web app source code</a>	
Working directory:	<a href="#">/home/webpersonal/</a>	<a href="#">Go to directory</a>
WSGI configuration file:	<a href="#">/var/www/webpersonal_pythonanywhere_com_wsgi.py</a>	
Python version:	3.6	


Virtualenv:

Use a virtualenv to get different versions of flask, django etc from our default system ones. [More info here](#). You need to **Reload your web app** to activate it; NB - will do nothing if the virtualenv does not exist.

[Enter path to a virtualenv, if desired](#)

Code:

What your site is running.

Source code:	<a href="#">Enter the path to your web app source code</a>	
Working directory:	<a href="#">/home/webpersonal/</a>	<a href="#">Go to directory</a>
WSGI configuration file:	<a href="#">/var/www/webpersonal_pythonanywhere_com_wsgi.py</a>	
Python version:	3.6	

Virtualenv:

Use a virtualenv to get different versions of flask, django etc from our default system ones. [More info here](#). You need to **Reload your web app** to activate it; NB - will do nothing if the virtualenv does not exist.

[Enter path to a virtualenv, if desired](#)

El directorio del código fuente es obviamente donde encuentra el proyecto. Podemos consultarlo haciendo un pwd en la terminal, seleccionarlo y copiarlo con Control+C:

```
(webpersonal) 17:10 ~/web-personal-curso-django-2/webpersonal (master)$ pwd
/home/webpersonal/web-personal-curso-django-2/webpersonal
(webpersonal) 17:10 ~/web-personal-curso-django-2/webpersonal (master)$
```

\

Source code: [/home/webpersonal/web-personal-curso-django-2/webpersonal](#)

El del entorno virtual podemos consultarlo haciendo un which python, y copiando la parte hasta el nombre del entorno:

```
(webpersonal) 17:10 ~/web-personal-curso-django-2/webpersonal (master)$ which python
/home/webpersonal/.virtualenvs/webpersonal/bin/python
```

\

[/home/webpersonal/.virtualenvs/webpersonal](#)

Ahora vamos de vuelta al fichero WSGI que utiliza nuestro proyecto, si accedemos a él veremos que contiene una configuración de prueba en lugar de poner en marcha nuestro servidor Django:



/var/www/webpersonal-pythonanywhere\_com\_wsgi.py

```
1 # This file contains the WSGI configuration required to serve up your
2 # web application at http://webpersonal.pythonanywhere.com/
3 # It works by setting the variable 'application' to a WSGI handler of some
4 # description.
5 #
6
7 # ++++++ GENERAL DEBUGGING TIPS ++++++
8 # getting imports and sys.path right can be fiddly!
```

Así que evidentemente tenemos que configurar una serie de cosillas para que todo funcione. Vamos a borrar todo el contenido, sí, todo. En su lugar vamos a copiar la configuración recomendada por la propia empresa. La encontramos en el enlace de configuración que os compartí al principio de la lección y lo editaremos cambiando el directorio de nuestro proyecto y el de la variable de entorno DJANGO\_SETTINGS\_MODULE para que haga referencia al settings del proyecto:

<https://help.pythonanywhere.com/pages/FollowingTheDjangoTutorial/>

MUCHO OJO CON LOS ESPACIOS (COMO EN LA CAPTURA EN EL PATH):



/var/www/webpersonal-pythonanywhere\_com\_wsgi.py

```
1 import os
2 import sys
3
4 path = os.path.expanduser('~/web-personal-curso-django-2/webpersonal ')
5 if path not in sys.path:
6     sys.path.append(path)
7 os.environ['DJANGO_SETTINGS_MODULE'] = 'webpersonal.settings'
8 from django.core.wsgi import get_wsgi_application
9 from django.contrib.staticfiles.handlers import StaticFilesHandler
10 application = StaticFilesHandler(get_wsgi_application())
11
```

Luego lo guardaremos y recargaremos el proyecto:



En teoría una vez se recargue, si visitamos nuestra página deberíamos funcionar perfectamente:

<https://webpersonal.pythonanywhere.com/>





Copyright © 2018 · Juan Pérez (Ingeniero Industrial)

Podemos también visitar el panel de administrador:

<https://webpersonal.pythonanywhere.com/admin/> (hector -> hola1234)

## Administración de Django

Inicio › Portafolio › Proyectos › Envase innovador con cierre fácil

### Modificar proyecto

**Título:**

Envase innovador con cierre fácil

**Descripción:**

Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

Todos nos funcionará, excepto por una cosa: los ficheros media. Y es que estos ficheros no sé si lo recordaréis pero os dije muchas veces que se suelen servir con servidores como Nginx o Apache y por defecto Django no los servirá en producción. De hecho si lo pensáis detenidamente, ¿cómo es que los ficheros media no los sirve pero los estáticos sí? Bueno eso es gracias a que hemos utilizado la buena lógica de crear directorios static dentro de las apps, así que él se encarga de servirlos. Pero tampoco es buena idea, deberíamos hacer unos



ajustes finales para que nuestra webapp sirva los ficheros estáticos de forma correcta.



Para ello vamos a hacer lo siguiente, iremos al apartado FILES de la webapp:

Static files:

Files that aren't dynamically generated by your code, like CSS, JavaScript or uploaded files, can be served much faster straight off the disk if you specify them here. You need to **Reload your web app** to activate any changes you make to the mappings below.

URL	Directory	Delete
<i>Enter URL</i>	<i>Enter path</i>	

Aquí añadiremos dos urls y dos directorios:

URL	Directory	Delete
<a href="#">/static/</a>	<a href="#">/home/webpersonal/web-personal-curso-django-2/webpersonal/static/</a>	
<a href="#">/media/</a>	<a href="#">/home/webpersonal/web-personal-curso-django-2/webpersonal/media/</a>	

Sé lo que estaréis pensando... El directorio media se entiende, pero ¿porque hemos puesto un directorio static si no existe en nuestra aplicación? Pues muy sencillo, porque vamos a crearlo manualmente y copiaremos todos los ficheros estáticos ahí! Pero tranquilos, que no tenemos que hacerlo nosotros, Django lo hará por nosotros.

Simplemente debemos ir al fichero settings y justo abajo en la configuración de ficheros estáticos añadir la siguiente variable:

```
STATIC_URL = '/static/'  
STATIC_ROOT = os.path.join(BASE_DIR, "static")
```

Guardamos el fichero, y ahora en nuestra terminal y con el entorno virtual en marcha vamos a ejecutar la magia potagia:

```
python manage.py collectstatic  
185 static files copied to '/home/webpersonal/web-personal-curso-django-2/webpersonal/static'.
```

Habiendo hecho esto, reiniciamos la webapp:

↻ Reload [webpersonal.pythonanywhere.com](http://webpersonal.pythonanywhere.com)

Accedemos de nuevo a nuestra página y...



### Envase innovador con cierre fácil

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Donec eget mi aliquam, fringilla lorem eget, tincidunt odio. Pellentesque sed nisl et nunc molestie maximus ac in diam. Ut et urna ornare, sollicitudin justo id, ultrices est..

[Más información](#)



### Estudio sobre la optimización de los envases plásticos

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Donec eget mi aliquam, fringilla lorem eget, tincidunt odio. Pellentesque sed nisl et nunc molestie maximus ac in diam. Ut et urna ornare, sollicitudin justo id, ultrices est..

Valorar si añadir un mailbox nuevo y probar a enviar un correo o no hace falta.

Ya lo tenemos todo funcionando. Nuestra web app está lista, sirviendo ficheros estáticos y media con el servidor de ficheros estáticos de Python Anywhere. ¿Qué os parece?

Sólo una última cosa, debemos volver al panel de la webapp una vez cada tres meses si no queremos que la desactiven por inactividad:

This site will be disabled on **Sunday 10 June 2018**

Run until 3 months from today

Y bueno, todo lo que hemos visto es muy parecido en los demás servicios, dejando de banda las opciones de configuración específicas de cada uno, pero estoy seguro de que os habrá servido mucho, porque por lo menos habréis experimentado lo que es el proceso de despliegue en vuestras propios manos, que ya es algo.