# python-easy-ftp-sync

*Actualizado el viernes 10 de enero de 2020.*

*Publicado el martes 12 de julio de 2016.*

## Introducción.

El objetivo de escribir **python-easy-ftp-sync** persigue como fin último la creación de un programa por línea de comandos escrito en lenguaje Python versión 3.4.3 para sincronizar dos carpetas (y sus subcarpetas) por medio de FTP (File Transfer Protocol), así de sencillo, por ello lo de parte del nombre "easy" (fácil -o tranquilo- en idioma inglés). Por supuesto que no vamos a descubrir el agua tibia, antes que nosotros muchos autores han escrito sobre el tema, y lo grandioso del software libre -y la ciencia moderna en sí misma- es que no tenemos que partir desde cero (aunque sería un muy buen ejercicio) sino que podemos estudiar esos trabajos anteriores y sobre ellos construir nuestra herramienta.

## Antecedentes.

Hemos incluido para su estudio los más variopintos proyectos, e incluso algunas preguntas con propuestas incluidas. También hemos descartado muchísima información que consideramos no conducen a nada relevante, así que se preguntarán ¿qué hace eso publicado en Internet? Las razones son dos, al menos:

1. "O inventamos o erramos": evidentemente errar es lo más común, *y aunque parecen inútiles esos datos (inútil no es igual que irrelevante) igual nos aporta información sobre* ***por donde no encaminar nuestros pasos.***
2. Los buscadores o arañas web (insultando la inteligencia salvaje de las arañas) van por ahí recolectando y guardando bytes, y toman todo sin mayor discriminación, por ello podemos nosotros acceder a esos resultados fallidos. Las arañas, por otra parte, tejen su red, trepan por sus hilos y allí cae todo tipo de presa, *pero ellas saben cuál víctima comer y cual bocado soltar y desechar y luego reparar su propia telaraña para alimentar sus crías y tener descendencia evolucionada; ¡****menuda diferencia!***

## Librerías

A esta fecha debemos especificar un poco más que las meras librerías: sí señor, la versión 2.7.x que nos acompañó desde que comenzamos en serios nuestros estudios sobre software libre pues nos abandonó desde el 1° de enero de 2020. Como siempre decimos «lo único constante es el cambio».

https://twitter.com/ks7000/status/1215820807714279425

Para ayudar en la migración a la versión 3 la Fundación Python creó las librerías 2to3 y en este artículo solamente usaremos la versión 3, para ser exactos la 3.6.9; así que ya saben, siempre ejecutar **python3** de ahora en adelante (si usan entornos virtuales no es necesario especificar el número 3).

Ahora sí entrando en materia la librería que usaremos es:

[cc lang="python" width="90%" theme="blackboard"]
from ftplib import FTP
[/cc]

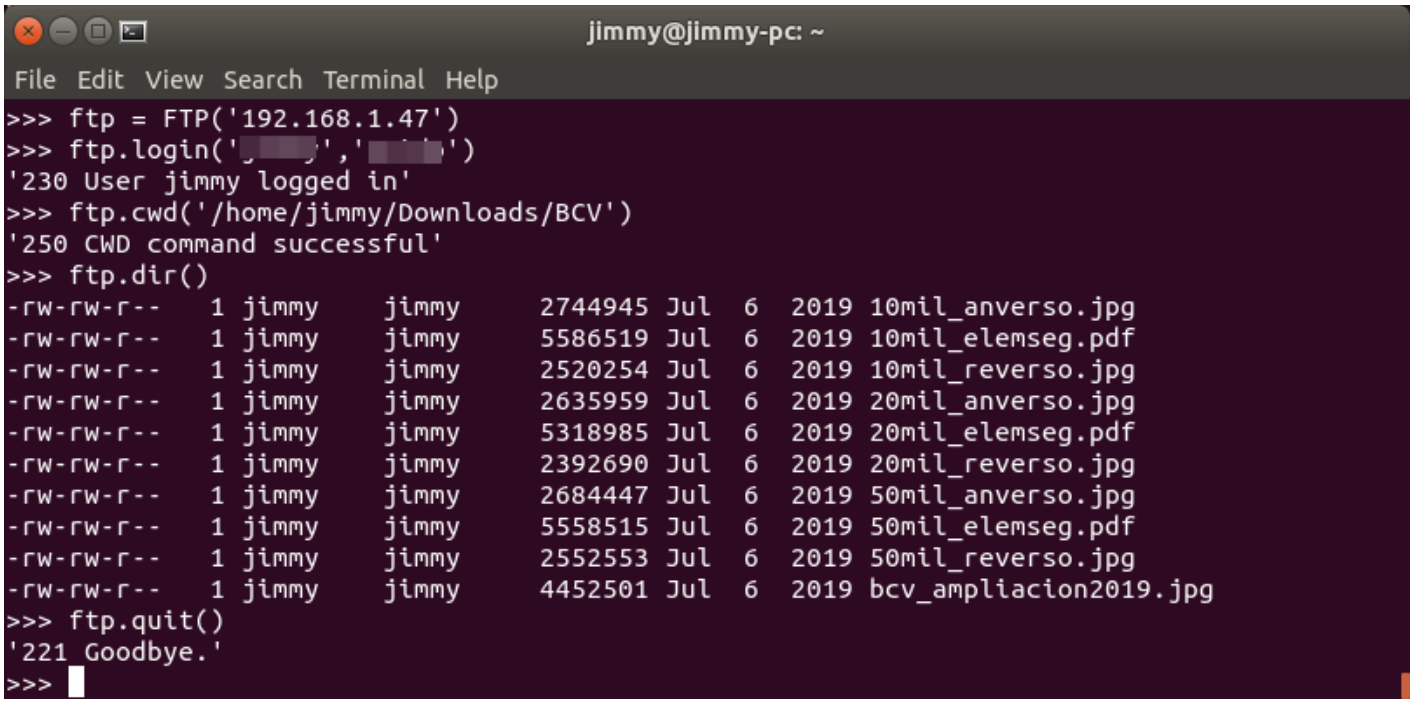
## Conceptos y acciones básicas

Realmente todo el trabajo lo hará la librería **ftplib**, lo que haremos es ubicar y conectar, autenticar a con una cuenta, listar un directorio y salir (lo guardaremos en un archivo llamado **python-easy-ftp-sync.py** ):

[cc lang="python" width="90%" theme="blackboard"]
from ftplib import FTP
ftp = FTP('192.168.1.47')
ftp.login('usuario','contraseña')
ftp.cwd('/home/jimmy/Descargas')
ftp.dir()

ftp.quit()
[/cc]

- La primera línea invocamos el recurso que hará nuestro trabajo.
- Usamos -qué original- un objeto llamado **ftp** indicándole la dirección del servidor FTP, en este caso otra máquina en nuestra red de área local.
- Si el servidor está en línea, oyendo en el puerto 21 (número de puerto habitual) procedemos a pasar el nombre del usuario y su contraseña respectiva.
- Claro, en el otro equipo también tenemos GNU/Linux corriendo proFTPD y entraremos a la carpeta "BCV" del usuario "jimmy". Por ahora meteremos las credenciales así, de plano pero luego haremos un método que las lea desde un archivo, de ser cifrado su contenido, mejor.
- En la penúltima línea listaremos los archivos y carpetas, con ese comando veremos exactamente un listado como si estuviéramos conectado a ese equipo en una terminal de comandos. Esto es ventajoso para nosotros porque nos muestra mucha información pero es engorroso sacar de allí los nombres (ver imagen siguiente).
- Con **ftp.quit()**, de manera cortés, le decimos al servidor que ya no le molestaremos más. Con este comando se espera que el servidor responda y que nosotros recibamos su respuesta. El otro comando es **ftp.close()** que es unilateral, "damos un portazo y nos vamos". Cualquiera de estos dos comandos que usamos nos impide volver a usar **ftp.login()**, osea, deberemos usar el comando de la segunda línea de nuevo.



ftp.dir()

## Descargando un archivo

Como dijimos necesitamos el nombre del fichero (obvio), si estamos nosotros en el teclado pues lo leemos y escribimos y listo, pero necesitamos que esto sea automatizado. Por ello usaremos mejor el comando **ftp.nlst()** el cual devuelve un objeto lista (yo lo llamo matriz) el cual podremos llamar a cada uno de sus miembros por un número.

Otro asunto a resolver es dónde vamos a guardar el fichero recibido (por defecto pues la misma carpeta donde tengamos nuestro programa), para ello necesitamos un objeto que llamaremos **miArchivo** y echaremos mano de otro comando llamado **ftp.retrbinary()** para "unir" los tres componentes.

En este punto es pertinente aclarar que usaremos siempre descargas binarias, que nos ha sucedido que cuando es en formato ASCII se eliminan los finales de línea y retornos de carro y no se pueden "leer" en diferentes plataformas como GNU/Linux y Microsoft Windows: *en pocas palabras, descargaremos los ficheros exactamente iguales como son, con su contenido íntegro siempre y cuando usemos descargas binarias.*

El código para descargar el primero de la lista sería el siguiente, fíjense muy bien en cómo se relacionan los objetos, abrimos, recibimos datos y cerramos -todo esto aparte del proceso FTP en sí-:

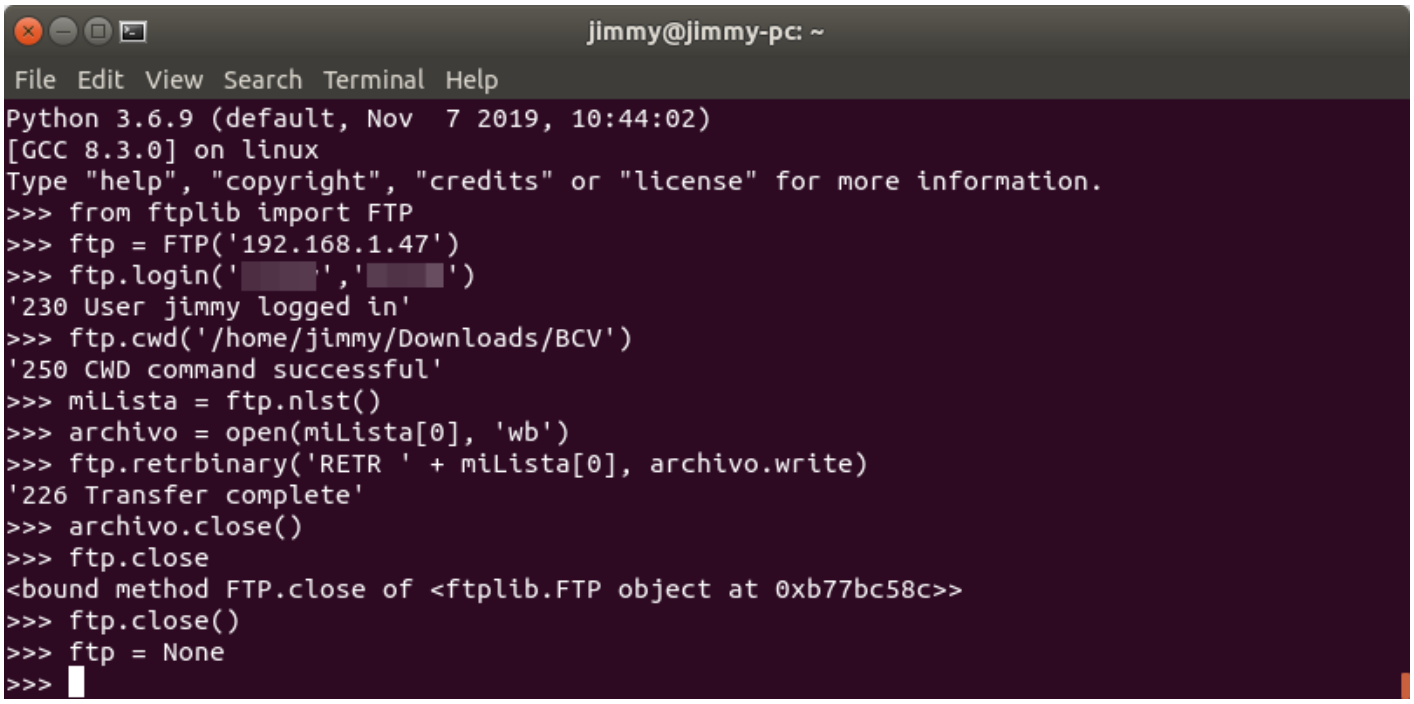
[cc lang="python" width="90%" escaped="true" theme="blackboard"]
from ftplib import FTP
ftp = FTP('192.168.1.47')
ftp.login('usuario','contraseña')
ftp.cwd('/home/jimmy/Descargas/BCV')

miLista = ftp.nlst()
archivo = open(miListata["1"], 'wb')
ftp.retrbinary('RETR ' + miLista["1"], archivo.write)
archivo.close()

ftp.close()
ftp = None
[/cc]

Para variar hemos usado **ftp.close()** -siempre debemos incluir los paréntesis sin contenido, son necesarios- y otra cosa muy importante: borramos por completo de la memoria el objeto **ftp** asignándole un valor **None** (cada lenguaje de programación tiene un método distinto para "recoger la basura", es decir, eliminar y liberar memoria).

Noten también los mensajes que devuelve el servidor, luego veremos que según la respuesta tomaremos una decisión u otra (reintentar, cerrar conexión, descargar el siguiente fichero, etc.):




Uso de ftp.nlst() y ftp.retrbinary()

## Variables de entorno

Una cosa pensada en la automatización es la colocación de variables para que tomen valores de la línea de comandos o bien sean leídas de un archivo. Aquí somos prácticos y comenzaremos, por ahora, en colocar al inicio unas variables que luego las volveremos a visitar para que hagan el trabajo automatizado. Nuestro código quedaría de esta manera (una de las ventajas de Python 3.x es que podemos escribir código con caracteres UTF-8):

[cc lang="python" width="90%" escaped="true" theme="blackboard"]
from ftplib import FTP

```
miServidorFTP = "192.168.1.47"
miUsuario = "usuario"
miContraseña = "contraseña"
miCarpetaRemota = "/home/jimmy/Descargas/BCV"

ftp = FTP(miServidorFtp)
ftp.login(miUsuario, miContraseña')
ftp.cwd(miCarpetaRemota)

miLista = ftp.nlst()
archivo = open(miLista[1], 'wb')
ftp.retrbinary('RETR ' + miLista[1], archivo.write)
archivo.close()

ftp.close()
ftp = None
[/cc]
```

## Preparando las descargas y subidas

La ventaja de las listas en Python, como ya vimos, es que podemos tener los nombres de los archivos en una matriz que podremos recorrerla de arriba hacia abajo o en el orden que necesitemos en un pequeño ciclo **for**. Esto es esencial porque este programa que estamos haciendo es para sincronización de carpetas:

- Debemos "bajar" los ficheros que no tengamos en nuestra máquina local.
- Debemos "subir" los ficheros que no existan en la máquina remota (servidor FTP).

Es así que tenemos entonces que comparar cambures con cambures y mangos con mangos, ahora debemos preparar una lista de los archivos locales con la librería (poderosa) llamada **os**. Aprovecho de agregar otra variable de entorno, que sería la carpeta local que queremos sincronizar ya que no necesariamente debe ser la carpeta donde está nuestro programa, veamos, pongan atención:

```
[cc lang="python" width="90%" escaped="true" theme="blackboard"]
import os
from ftplib import FTP

miServidorFTP = "192.168.1.47"
```

```
miUsuario = "usuario"
miContraseña = "contraseña"
miCarpetaRemota = "/home/jimmy/Descargas/BCV"
miCarpetaLocal = "/home/kevin/Descargas/BCV"

ftp = FTP(miServidorFTP)
ftp.login(miUsuario, miContraseña)
ftp.cwd(miCarpetaRemota)
miListaRemota = ftp.nlst()
print(miListaRemota)

miListaLocal = set(os.listdir(miCarpetaLocal))
print(miListaLocal)

ftp.close()
ftp = None

[/cc]
```

Exacto, antes de almacenar en la variable **miCarpetaLocal** hemos agregado la función **set()**... **¿pero qué hace esto, qué función cumple?** (de aquí en adelante solo trabajaremos con las variables de entorno, es decir, sin especificar explícitamente sus valores, y al final las reintroduciremos por medio de código).

## Uso avanzado de conjuntos

Un **set** lo podemos ver como una lista sin orden alguno *que no tiene miembros repetidos.* Veamos un ejemplo sencillo:

```
[cc lang="python" width="90%" escaped="true" theme="blackboard"]
Python 3.6.9 (default, Nov 7 2019, 10:44:02)
[GCC 8.3.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> frutas = {'cambur', 'naranja', 'piña', 'naranja', 'guanabana', 'aguacate'}
>>> print(frutas) # ningún nombre de fruta se repite
{'guanabana', 'aguacate', 'piña', 'cambur', 'naranja'}
>>> verduras = {'lechuga', 'aguacate' , 'apio' }
>>> print(verduras) # muestra verduras de la lista original sin duplicado alguno
{'aguacate', 'lechuga', 'apio'}
```

```
>>> frutas - verduras # no muestra al aguacate en la lista resultante
{'guanabana', 'piña', 'cambur', 'naranja'}
>>>
```

¿Ven como pudimos "restar" ambos grupos? Esto da para una entrada completa, pero incluimos esta sección para abrir vuestras mentes al trabajo con listas de ficheros , con lo cual podemos obtener "diferencias" de los ficheros.

Solo tenemos que resaltar que las listas se manejan con *corchetes* **[..]** y los conjuntos utilizan *llaves* **{..}**, esto quiere decir que podremos hacer lo siguiente (y son equivalentes):

```
conjunto1 = { "perro", "gato", "ratón", "insecto" }
conjunto2 = set(["perro", "gato", "ratón"])
print(conjunto1-conjunto2) # devuelve solo la palabra "insecto"
```

## Diferencias de conjuntos

Ya estamos preparados para manejar las diferencias entre los archivos que están en la carpeta remota y en la carpeta local, así que lo aplicaremos al código con el uso de dos conjuntos cuyos nombres nemotécnicos serán **paraBajar** y **ParaSubir** *¡vamos a acelerar la marcha agregando también una función, tomen café, atentos y atentas!*

```
# # Copyright (C)

# This program is free software; you can redistribute it and/or modify
# it under the terms of the GNU General Public License as published by
# the Free Software Foundation; either version 2 of the License, or
# (at your option) any later version.

# This program is distributed in the hope that it will be useful,
# but WITHOUT ANY WARRANTY; without even the implied warranty of
# MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
# GNU General Public License for more details.
```

```
# You should have received a copy of the GNU General Public License along
# with this program; if not, write to the Free Software Foundation, Inc.,
# 51 Franklin Street, Fifth Floor, Boston, MA 02110-1301 USA.

def sincronizaFTP(miServidorFTP, miUsuario, miContraseña, miCarpetaRemota, miCarpetaLocal,
subir=False, bajar=True):

  print("\n-- Conectando y comparando carpetas ----\n")

  import os
from ftplib import FTP

  ftp = FTP(miServidorFTP)
ftp.login(miUsuario, miContraseña)
ftp.cwd(miCarpetaRemota)
miListaRemota = set(ftp.nlst())
miListaLocal = set(os.listdir(miCarpetaLocal))

  paraBajar= miListaRemota - miListaLocal
paraSubir = miListaLocal - miListaRemota
vacio = set()

  print("\nPor descargar:\n ")
if ( paraBajar == vacio ):
print("Todos los archivos han sido descargados.\n")
else:
print(paraBajar)

  print("\nPor cargar:\n")
if ( paraSubir == vacio ):
print("Todos los archivos han sido cargados.\n")
else:
print(paraSubir)

  ftp.close()
ftp = None

[/cc]
```

- Recuerden siempre colocar un resumen de la licencia de Software Libre ("Libre Software")

en cada uno de los archivos de vuestros proyectos. Publicarlos en GitHub o GitLab también es una buena idea, en este enlace pueden ver nuestro trabajo en el primer sitio web que mencionamos.

- Declaramos la función de sincronizar, en Python primero van las funciones y luego la rutina principal donde llamaremos a la función o funciones. También podemos definir las funciones en otro u otros ficheros pero en este caso es corta la labor.
- Notar que seguimos incorporando mensajes al usuario pero cuando automaticemos esta tarea más bien necesitaremos lelvar un registro de lo que sucedió. Más adelante le colocaremos unas rutinas para capturar los resultados, errores, etc.

## "Subidas" y "bajadas"

Ahora sí nos concentraremos solamente en la función **sincronizaFTP** y colocamos las instrucciones para descargar o cargar los archivos que no estén en uno u otro sitio.

[cc lang="python" width="90%" escaped="true" theme="blackboard"]
```
def sincronizaFTP(miServidorFTP, miUsuario, miContraseña, miCarpetaRemota, miCarpetaLocal, subir=False, bajar=True):

  print("\n-- Conectando y comparando carpetas ----\n")

  import os
from ftplib import FTP

  if miCarpetaRemota[-1] != "/":
miCarpetaRemota += "/"
if miCarpetaLocal[-1] != "/":
miCarpetaLocal += "/"

  ftp = FTP(miServidorFTP)
ftp.login(miUsuario, miContraseña)
ftp.cwd(miCarpetaRemota)
miListaRemota = set(ftp.nlst())
miListaLocal = set(os.listdir(miCarpetaLocal))

  paraSubir = miListaLocal - miListaRemota
paraBajar = miListaRemota - miListaLocal
vacio = set()
```

```
  if subir:
print("\nPor cargar:\n ")
if ( paraSubir == vacio ):
print("Todos los archivos han sido cargados.\n")
else:
subidos = 0
for fichero in paraSubir:
print(" * Subiendo fichero:" + fichero)
ftp.storbinary('STOR ' + fichero)
subidos += 1

  if bajar:
print("\nPor descargar:\n")
if ( paraBajar == vacio ):
print("Todos los archivos han sido descargados.\n")
else:
descargados = 0
for fichero in paraBajar:
print(" * Descargando fichero:" + fichero)
archivo = open( miCarpetaLocal + fichero, 'wb')
ftp.retrbinary('RETR ' + fichero, archivo.write)
archivo.close()
descargados += 1

  ftp.close()
ftp = None
[/cc]
```

## Listo, *¿y ahora qué?*

- Falta incluir las rutinas para capturar los "errores" típicos, por ejemplo, si el servidor FTP no está en línea ocurrirá una excepción que debemos canalizar y grabar en el registro correspondiente la novedad. Si incluimos este programa en el **crontab** podemos configurar para que nos envíe un mensaje de correo electrónico avisándonos del inconveniente.
- En la función hemos colocado que corrija, si no lo tiene, una *barra* al parámetro carpeta local y carpeta remota, sin embargo esto apenas es el inicio: debemos verificar que ambas carpetas existan en sus sitios respectivos y sino, de tener nosotros permisos de escritura, crearlas y luego proceder con las descargas y/o cargas de ficheros.

- **Más importante aún: debemos poder diferencias directorios (carpetas) de los ficheros (archivos).** Dicho sea de paso, si tenemos una carpeta nueva, en cualquiera de las dos ubicaciones, debemos "abrirla" y ver cuales ficheros tiene... y revisar si tiene subdirectorios para repetir de nuevo el procedimiento. *Esto se logra fácilmente porque podremos siempre llamar de manera recursiva la función (llamarse a sí misma)* ***pero debemos, repito, saber diferenciar carpetas de archivos (directorios de ficheros).***
- Cuando resolvamos el punto anterior también debemos pensar sobre cómo reanudar descargas y/o cargas que hayan sido interrumpidas.
    - Primero: una vez finalizadas las operaciones de carga y/o descarga debemos comprobar el campo **tamaño de archivo** a las dos listas... ¡debemos también considerar si este proceso lo colocamos más bien al principio de la función!
    - Segundo: depende también de si el servidor FTP tiene la capacidad de reanudar y si tenemos derecho de *adicionar datos a un fichero* (**append**). Si el servidor no soporta dicha característica o no tenemos derecho de adicionar, pues debemos interceptar el error y comenzar de nuevo la carga y/o descarga de nuevo...
- La seguridad no debe ser dejada a un lado: debe ser capaz de conectar por medio de certificados con llaves públicas y privadas, los famosos TLS.
- También podemos colocar y dejar listo para hacer traducciones a otros idiomas, recuerden que "en todas partes se cuecen habas" y el Software Libre es su naturaleza el ser difundido a todos y todas.

## Confirmar que existe la carpeta local

Podemos comprobar si existe la carpeta local por medio de la librería **os.makedirs()** la cual crea carpetas y subcarpetas de manera automática y podremos colocar los derechos (por defecto son creadas con 777, ojo con la seguridad):

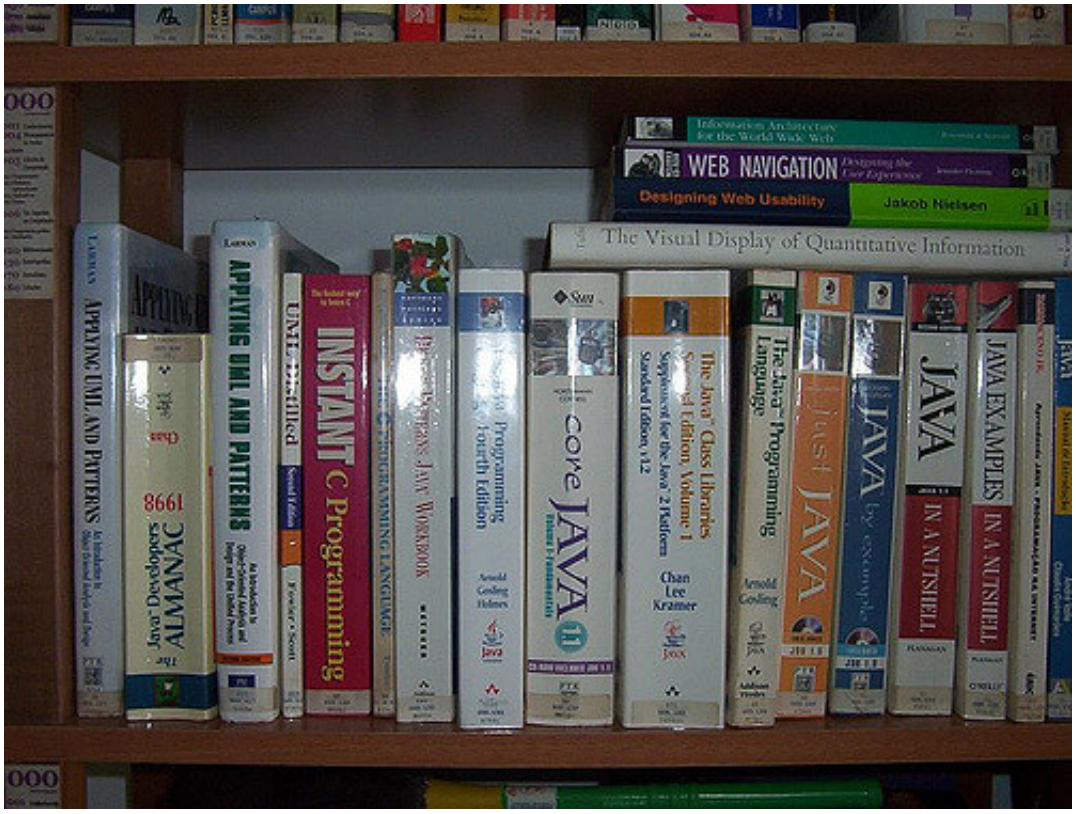
[cc lang="python" width="90%" escaped="true" theme="blackboard"]
def sincronizaFTP(miServidorFTP, miUsuario, miContraseña, miCarpetaRemota, miCarpetaLocal, subir=False, bajar=True):

  print("\n-- Conectando y comparando carpetas ----\n")

  import os
from ftplib import FTP

  if miCarpetaRemota[-1] != "/":
miCarpetaRemota += "/"
if miCarpetaLocal[-1] != "/":

```
miCarpetaLocal += "/"
if not os.path.exists(miCarpetaLocal):
os.makedirs(miCarpetaLocal)

  ftp = FTP(miServidorFTP)
ftp.login(miUsuario, miContraseña)
ftp.cwd(miCarpetaRemota)
miListaRemota = set(ftp.nlst())
miListaLocal = set(os.listdir(miCarpetaLocal))

  paraSubir = miListaLocal - miListaRemota
paraBajar = miListaRemota - miListaLocal
vacio = set()

  if subir:
print("\nPor cargar:\n ")
if ( paraSubir == vacio ):
print("Todos los archivos han sido cargados.\n")
else:
subidos = 0
for fichero in paraSubir:
print(" * Subiendo fichero:" + fichero)
ftp.storbinary('STOR ' + fichero)
subidos += 1

  if bajar:
print("\nPor descargar:\n")
if ( paraBajar == vacio ):
print("Todos los archivos han sido descargados.\n")
else:
descargados = 0
for fichero in paraBajar:
print(" * Descargando fichero:" + fichero)
archivo = open( miCarpetaLocal + fichero, 'wb')
ftp.retrbinary('RETR ' + fichero, archivo.write)
archivo.close()
descargados += 1

  ftp.close()
ftp = None
[/cc]
```




# Fuentes consultadas

## En idioma castellano

- «».
- «».
- «».

## En idioma francés

- «».

- «».
- «».

## En idioma inglés

- "Simple FTP directory synch (Python recipe)" at ActiveState | Code.
- «Retrieving data of generator object for file listing from FTP.mlsd()».
- «Python 'TypeError': 'Generator' object is not subscriptable».
- «Creating and Deleting Directories with Python».
- «ftplib — FTP protocol client».
- «».
- «».
- "pyftpsync" at GitHub.com
- "FTP-Sync" at GitHub.com
- "sftp_s3_sync 2.0.1" at Python.org
- "(Python) Synchronize Remote Directory Tree" at Example-code.com
- "ftpsync2d" at GitHub.com
- "Syncing directories with python's ftplib" at StackOverFlow.com
- 
- "Hashsync: Fast FTP synchronization for static website generators" from THB.it

Crédito de la imagen Gerd Altmann, trabajo, licencia de uso:Pixabay