Python http.server

La verdad es que revisando los temas publicados por nuestros colegas de GNULinuxBlog (por ahora quien publica el tema, el sr. Elías Rodríguez Martín y el sr. José Miguel, creador del blog) encontramos uno en particular muy útil y que nosotros habíamos enfocado de una manera un tanto complicada en comparación con la facilidad de lo que allí proponen. Cuando conseguimos que alguien es más listo que nosotros *inmediatamente lo reconocemos y aprendemos de dichas personas, no tenemos rubor en admitirlo,* por eso os pedimos que nos acompañéis en nuestro artículo de hoy: compartir ficheros de una manera rápida y sencilla con Python.

Introducción.

Pasar ficheros de una máquina a otra en algún momento nos tocó o tocará hacerlo y en nuestro caso fue de una máquina con Ubuntu hacia otra con software privativo ubicada fuera de nuestra red de área local. Para ello habilitamos un servidor FTP, en este caso ProFTPd, que conecta con las credenciales de otro usuario creado de manera exprofeso y con derechos de lectura sobre los archivos en cuestión. *Eso a nosotros nos pareció fácil, pero ¿qué tal hacer algo que se pueda ejecutar en varios sistemas operativos?*

Para ello nada mejor que Python: es legendario y ya en 1996 corría en sistemas tan diverso como Linux, IRIX, Compaq Tru64, OS X, Solaris, and Windows (ver caso de éxito "Industrial Light & Magic", "Lucas Films LTD"). Al momento de escribir estas líneas Python (que ya va por la versión 3.6.1) está disponible para ser descargado y usado en Linux/Unix, Windows, AS/400 (OS/400), Beos, MorphOS, MS-DOS (sí, así como lo leen, no ha muerto, aún), OS/2, OS/390 and z/OS, RIC-OS, Symbian OS series 60, VMS, ¡¡¡ Windows CE or Pocket PC !!!, HP-UX y hasta podemos comprar una versión privativa a la cual no podremos acceder a su código fuente, y cuya distribución es llamada ActivePython (sabrá Dios en cuales plataformas de hardware ejecutará este último "sabor") ***pero como véis, de que tenemos opciones y libertad de escoger es indudable, y si nos proponemos hasta podemos nosotros mismos hacer nuestra propia distribución de Python por nosotros elaborada (palabras mayores).*** Esto último es conocido como "implementaciones" y las podéis conocer más en este enlace -en idioma inglés-; ya por último debemos recalcar que Python es aceptado como lenguaje dentro del manejador de base de datos PostgreSQL ¡¿maravilloso, cierto?! 8-) .

Versión Python 2.x

En dicha versión 2.x se utiliza "**SimpleHTTPServer**" y lo probamos y comprobamos desde nuestra red de área local y desde una máquina externa, ya en la entrada que nos inspiró bien lo explican: solo debemos lanzar el siguiente comando (las librerías vienen integradas en Python):

```
python -m SimpleHTTPServer 8000
```

¿Fácil cierto? Nos vamos en una ventana terminal hasta la carpeta que contiene los archivos "a compartir", lanzamos el guión en Python indicandole que abrimos el puerto 8000 y en el otro ordenador lanzamos nuestro navegador web con la dirección IP destino (si tenemos un enrutador pues configurad ALGO parecido a lo nuestro):

Reconocimiento a la labor del sr. Elías.

En su repositorio de aplicaciones -gentilmente alojadas por GitHub.com- el sr. Elías nos enseña como agregar valor al código existente por medio de una bonita interfaz gráfica para escoger la carpeta a compartir y cambiar el puerto de escucha -vamos que antes igual teníamos esa opción por la línea de comandos- **pero con otro botón adicional: el botón de detener el servicio ya que por línea de comandos solo con CTRL+C se detenía el proceso.** En la siguiente imagen comparto una carpeta de respaldo de un antiguo aparatito reproductos de MP3 y disfruto de música que tenía tiempo que no escuchaba:

Lo bonito del asunto del asunto (el guion, **no la utilería gráfica PyShare**) es que podemos navegar en nuestras carpetas, iniciar el servicio, detenerlo, ir a otra carpeta y repetir el proceso; *y por si queda alguna duda sobre su funcionamiento se abre de una vez una pestaña en nuestro navegador web predeterminado para visualizar la carpeta compartida, todo muy práctico:*

Lo que véis en la última imagen es el guión desarrollado por el Sr. Elías y el último error es simplemente un mal manejo de los caracteres de nuestro bello idioma castellano (compartí la carpeta llamada "Vídeos" que tiene un acento). Alegremente lo primero que hisimo fue lo que hacen los niñatos y niñatas de los parvularios, COPIAR Y PEGAR en un archivo llamado "python_easy_server.py" ***pero obviando la regla de oro: UTILIZAR GIT PARA CONTROLAR (y colaborar) EN EL CONTROL DE VERSIONES*** (mirad nuestro humilde tutorial al respecto).

No os vamos a "aburrir" con el tema de GitHub pero si os indicamos que propusimos unos cambios ("fork" y "pull request") que podréis analizar con detalle en nuestros repositorios alojados -de nuevo, la publicidad- en GitHub.

Versión Python 3.X

Como bien especifica el sr. Elías en Python versión 3.x el **SimpleHTTPServer** ha sido sustituido por **http.server** y en esencia tiene la misma línea de comandos pero si analizamos el código *pues*

bueno, todo de acuerdo a la nueva versión (3.6 en este caso). En el repositorio correspondiente en el módulo **server.py** leemos 1211 líneas y está muy bien explicado, un trabajo muy compacto con las deficiones básicas del protocolo HTTP y sobre cómo funciona por defecto esta librería escrita en Python.

Nos agrada mucho que vayan directo y al grano explicando cómo funciona un servidor web, *eso es poesía informática, para muestra un botón: el inicio de la clase* ***BaseHTTPRequestHandler:***

> The following explanation of HTTP serves to guide you through the code as well as to expose any misunderstandings I may have about HTTP (so you don't need to read the code to figure out I'm wrong :-).

Que traducido lo exponemos de la siguiente manera:

> La siguiente explicación de HTTP sirve para guiarlo a través del código, así como para exponer cualquier malentendido que yo pueda tener sobre HTTP (por lo que no necesita leer el código para averiguar que estoy equivocado).

Allí se fajan a explicar cómo un servidor web recibe una petición y como se contesta, ya sea con el protocolo "HTTP/0.9" (por defecto en este módulo, línea N° 261), "HTTP/1.0" o "HTTP/1.1". Esencialmente funciona de la siguiente manera (*vamos a ser redundantes con propósitos didácticos*):

1. Un servidor web está funcionando con un "socket" abierto que se interpreta (y nos abstraemos) como un puerto abierto en un número conocido (por defecto es el puerto 80 pero como estamos usando algo "privado" usamos el puerto 8000).
2. Un programa cliente, ya sea en nuestra propia red de área local o desde el internet, envía una petición a ese puerto conociendo de antemano la dirección IP del servidor web (no ahondaremos en el tema de los DNS, para los lectores más avezados).
3. Esa petición tiene *un protocolo específico:* ejemplo "**GET / HTTP/1.1"** ¿Pero que significa esto? Veamos el siguiente punto no sin antes mencionar que una petición puede contener

dos líneas adicionales **que son opcionales (por ahora no complicaremos la labor)**.

4. Ante una petición tipo **GET** (en este caso nos exigen usar un protocolo específico **HTTP/1.1**) el servidor web procede a revisar el directorio por defecto (estamos hablando del **http.server** de Python, un servidor Apache es mucho más avanzado) y si encuentra un archivo **index.html** lo enviará, *de lo contrario, si no lo halla entonces procederá a mostrar un listado de los archivos presentes en la carpeta desde donde ejecutamos el comando "**python -m http.server 8000**"* (de hecho así también se comporta un servidor web Apache). Este último comportamiento lo podemos detallar en la función **send_head()** de la clase **SimpleHTTPRequestHandler**. Incluso podríamos aquí empoderarnos de nuestro idioma e incluir comandos para buscar también un archivo llamado **"índice.html"**, es decir, que 'busque' en inglés y castellano ¿Por qué no? (amén de traducir al castellano todos los demás mensajes). Así mismo, con acentos y todo ***el hecho de nosotros hablar dos o más idiomas ejercita nuestro cerebro, ampliamos nuestra memoria y nos ayuda a crear sinapsis nuevas en nuestras neuronas.***
5. La manera como envía esta respuesta debe estar ajustada al protocolo en el cual nos lo solicitaron. Para este ejemplo debemos contestar según el protocolo que nos pidieron [más abajo veremos que esto no es necesariamente cierto;-)] y el cual consta de tres partes.
6. La primera parte es una línea que está compuesta a su vez en tres segmentos:
    - Versión en la que estamos respondiendo (este ejemplo "**HTTP/1.0**") más un espacio como separador al final.
    - Un código numérico de tres dígitos que está bien normalizado (¡A que alguna vez habrán visto una respuesta "**404**"!). Para este ejercicio contestaremos con un "**200**" (más el consabido espacio en blanco como separador).
    - Un tercer segmento que explica ¡cómo no, en inglés! un mensaje descriptivo para nosotros los seres humanos (el código anterior, numérico, es muy agradable a nuestros amigos los ordenadores). El código **200** tiene la descripción simple y llana: **OK**.
7. Para separar la primera parte de la segunda se recomienda utilizar la combinación de caracteres especiales de retorno de carro y avance de línea, pero debemos estar prestos a utilizar solo el avance de línea para una compatibilidad más amplia (de nuevo, ver **BaseHTTPRequestHandler**).
8. La segunda parte son varias líneas llamadas encabezados y ajustados a la norma RFC-822 (la norma data del año 1982 pero pensamos está plenamente vigente).
9. Un punto muy especial es que para separar la segunda parte de la tercera, además de utilizar el CRLF -ver punto 7- *debemos insertar una línea en blanco,* ¿cómo se hace esto? Pro favor ver la función **end_headers()** {línea 516} y os sorprenderéis.
10. La tercera parte son los preciados datos en sí mismos cuyas características están especificadas de antemano en la segunda parte (recordad RFC-822).

Todo esto **QUE PARECE COMPLICADO EN TEORÍA os lo podemos demostrar en la práctica:** abrimos una ventana terminal y ejecutamos el servidor web Python, abrimos una segunda ventana terminal y ejecutamos la siguiente orden con el parámetro **-i** (o la manera larga **--include**) el cual

muestra los encabezados HTTP:

```
curl -i http://192.168.1.47:8000/
```

Tras lo cual recibiremos la siguiente respuesta:

```
HTTP/1.0 200 OK  Server: SimpleHTTP/0.6 Python/3.5.2  Date: Sun, 16 Apr 2017  01:24:24  GMT   Content-type:  text/html;  charset=utf-8   Content-Length: 1370      (...
```

Acá la "magia" la realiza nuestro navegador web: según el **Content-type** se prestará a interpretar y mostrar de manera correcta los datos recibidos. Si es "**text/html**" pues vemos que es el lenguaje de marcado muy famoso y que además debemos mostrarlo con codificación **utf-8**. ¿Estáis confundido o perdido? Consultad nuestro tutorial sobre HTML.

Notad la respuesta del servidor identificandose a sí mismo "**SimpleHTTP/0.6 Python/3.5.2**", maravilloso para nosotros. Mirad en la siguiente figura a lo que nos referimos:

Solicitando descargar un archivo.

Sigamos entonces con nuestra práctica: ya sabemos que no vamos a mostrar un archivo **index.html** -ni **índice.html**- sino que mostramos un listado de archivos, ahora nos iremos a la carpeta donde clonamos el "fork" del proyecto **py-qt-share** y lanzamos allí nuestro servidor web Python.

Allí tenéis de una vez la tres líneas completas, un abreboca, pero veamos la primera y segunda líneas -olvidemos la tercera, por el momento- y abramos de nuevo otra ventana terminal donde lanzamos el comando **curl** para retribuir el listado de archivos disponibles:

Una vez recibimos e interpretamos el HTML solicitamos de nuevo con otro comando **GET** el fichero **README.md**, observemos lo que responde:

Diseccionemos la respuesta:

```
Server: SimpleHTTP/0.6 Python/3.5.2  Date: Sun, 16 Apr 2017 02:55:23 GMT
 Content-type: application/octet-stream  Content-Length: 575  Last-
Modified: Sat, 15 Apr 2017 23:33:16 GMT
```

¿Cómo es posible que en el **Content-type** nos indique que es una aplicación? ???

Sabemos plenamente que un fichero con extensión **.md** *generalmente* contiene un lenguaje de marcado tipo **MarkDown** (de nuevo, mirad nuestro otro tutorial relacionado), si acaso se asemeja más a un **text/html** que a una aplicación ***¿Qué piensan ustedes?***

Los lectores más experimentados nos responderán "**¿de dónde saca que .md es un MIME type?**" Pues veamos los tipos principales definidos en RFC-2046:

> The "text" media type is intended for sending material which is principally textual in form. A "charset" parameter may be used to indicate the character set of the body text for "text" subtypes, notably including the subtype "text/plain", which is a generic subtype for plain text.

De manera independiente a la opinión que tengamos si **MarkDown** es un fichero de texto o una aplicación debemos primero analizar en qué se basan para programar nuestro servidor web en Python para clasificar los archivos que listamos para compartir con otras computadoras. Analizemos la función dedicada a esto:

```
  def guess_type(self, path):       """Guess the type of a file.       Ar
gument is a PATH (a filename).       Return value is a string of the form
 type/subtype,       usable for a MIME Content-type header.       The def
ault implementation looks the file's extension       up in the table self
.extensions_map, using application/octet-stream       as a default; howev
er it would be permissible (if       slow) to look inside the data to mak
e a better guess.       """
```

Como bien podemos leer PRIMERO se trata de identificar el archivo por la extensión del archivo,

lo cual es una solución rápida -y sucia- de identificar con qué tipo de datos estamos lidiando, **y por defecto asumimos que es una aplicación "application/octet-stream" ya que ponerse a la tarea de abrir cada archivo y ver su contenido es una tarea muy tediosa (y que puede bloquear nuestro ordenador *si tenemos demasiados ficheros almacenados*).**

"import mimetypes"

Es por ello necesario que pasemos a revisar la librería que importamos llamada **mimetypes**. Si revisamos la documentación de Python al respecto pues... *la metodología es la misma, ¡"adivinar" por la extensión del fichero!*

No obstante se apoyan en un método más "científico": revisar las aplicaciones que tenemos en nuestro sistema operativo para poder ofrecer así una mayor variedad de opciones:

```
knownfiles = [   "/etc/mime.types",   "/etc/httpd/mime.types", # Mac OS X
   "/etc/httpd/conf/mime.types", # Apache   "/etc/apache/mime.types", # A
pache 1   "/etc/apache2/mime.types", # Apache 2   "/usr/local/etc/httpd/c
onf/mime.types",   "/usr/local/lib/netscape/mime.types",   "/usr/local/et
c/httpd/conf/mime.types", # Apache 1.2   "/usr/local/etc/mime.types", # A
pache 1.3  ]
```

Actualizado el jueves 20 de julio de 2017.

En este enlace de la "Web del Programador" podreis descargar un fichero en formato PDF con la mayoría de las extensiones de fichero conocidas, muy completa lista, os lo recomendamos.

De acá notamos que rapidamente podemos agregar nuestras propias extensiones, empezemos por la primera línea **/etc/mime.types**: en una ventana terminal metemos el comando **"sudo nano /etc/mime.types"** y podremos ver un encabezado parecido al siguiente:

```
###############################################################################
######  #  #  MIME media types and the extensions that represent them.  #
  #  The format of this file is a media type on the left and zero or more
  #  filename extensions on the right.  Programs using this file will map
  #  files ending with those extensions to the associated type.  #  #  Th
is file is part of the "mime-support" package.  Please report a bug using
  #  the "reportbug" command of the "reportbug" package if you would like
```

```
 new  #  types or extensions to be added.  #  #  The reason that all type
s are managed by the mime-support package instead  #  allowing individual
 packages to install types in much the same way as they  #  add entries i
n to the mailcap file is so these types can be referenced by  #  other pr
ograms (such as a web server) even if the specific support package  #  fo
r that type is not installed.  #  #  Users can add their own types if the
y wish by creating a ".mime.types"  #  file in their home directory.  Def
initions included there will take  #  precedence over those listed here.
 #  ##################################################################
##########
```

En el último párrafo especifican que podremos crear nuestros propios tipos de ficheros por medio de un archivo en nuestro directorio personal de datos ("home") y que tendrá preferencia aquella lista sobre esta. Pero si modificamos la lista principal los efectos tomarán para todos los usuarios en el ordenador donde estemos compartiendo archivos.

Es así que agregamos la siguiente línea, ya sea al final o buscáis el último grupo de líneas:

```
text/markdown                                   md
```

Previamente, por medio de nuestro editor de texto favoritos, **nano**, buscamos que la palabra "markdown" no esté registrado ni tampoco la extensión de archivo "**md**". Luego le damos guardar y salir y relanzamos de nuevo nuestro servidor web escrito en **Python** para compartir archivos y solicitamos descargar de nuevo el archivo markdown del guion que le hicimos "fork" antes de resubirlo a GitHub. Nuestro navegador ahora nos mostrará el diálogo de descarga de una manera diferente y hasta nos propondrá editarlo con nuestro editor de texto con entrono gráfico favorito **gedit**, mirad:

Y claro, solicita abrirlo con **gedit** porque le estamos especificando que es un archivo de texto pero con una nueva clasificación "por nosotros propuesta", tipo **markdown**.

Presentando ficheros markdown como HTML.

Para ensanchar nuestros conocimientos haremos un ejercicio interesante en cuanto a agregar funcionalidades a nuestro servidor web escrito en Python. La idea consiste en agregar una librería a nuestro Python y el cual permite convertir código markdown en HTML. Para ello debemos

primero descargar el software necesario, tanto para Python 2.X como Python 3.X, mirad la imagen:

Una vez tengamos instalada la librería, corremos una terminal Python y hacemos unas cuantas pruebas y confirmar como convierte en código HTML lo que le pasemos en código markdown:

```
Python 3.5.2 (default, Nov 17 2016, 17:05:23)  [GCC 5.4.0 20160609] on l
inux  Type "help", "copyright", "credits" or "license" for more informati
on.  >>> import markdown  >>> print(markdown.markdown("* Lista "));
```

- Lista

```
>>> print(markdown.markdown("# Título 1 "));
```

Título 1

```
>>>
```

Ya tenemos nuestro conversor; la idea que tenemos en mente es mostrar todo archivo con extensión "**.md**" en la carpeta que compartimos en ambos formatos pero con una diferencia: el enlace mostrado en el navegador en realidad NO es un fichero real en disco sino una "traducción" del fichero tipo markdown que sí existe en disco. *Por supuesto nuestro navegador web al hacer click en uno de estos enlaces y recibir la cabecera de datos que recibe del servidor web Python se comportará de diferentes maneras: ante un fichero ".html" no ofrecerá guardar en disco* ***sino directamente mostrarlo por pantalla.*** Esto que queremos hacer pareciera no tener sentido alguno pero si alguna vez llegáis a estudiar el lenguaje PHP entenderéis como un guión escrito en un lenguaje es convertido a un lenguaje de marcación como HTML. Cuesta un poco acostumbrarse a esta manera de trabajar en PHP pero una vez que uno le toma el gusto todo va sobre ruedas.

Ahora presentamos un código de prueba para exhibir un fichero markdown por pantalla en formato HTML, para ello también echaremos mano de la librería **codecs** que viene por defecto en las instalaciones de Python, veamos (recordad que tenemos abierta una ventana terminal en la carpeta que estamos compartiendo):

```
>>> import markdown  >>> import codecs  >>> archivo_md = codecs.open("REA
DME.md", mode="r", encoding="utf-8")  >>> texto = archivo_md.read()  >>>
print(markdown.markdown(text);
```

Utilidad escrita por el sr. Elías Rodríguez Martín

Entrada hecha en Linux GNU Blog para compartir archivos con Python

Adición hecha por Jimmy Olano para soportar caracteres no ascii (utf-8) en os.chdir()

Entrada hecha en KS7000 + WP para ampliar el alcance de dicho guion

Propongo tenga una licencia Apache que es adecuada para pequeños proyectos 8-)

Por supuesto que hay algunos detalles que corregir: primero se debe colocar el encabezado del archivo HTML, luego el "cuerpo" que es el código anterior y al final las etiquetas de cierre del archivo HTML. Segundo se debe comprobar que el código sea "limpiado" y presentado correctamente ya que proviene de una fuente "no confiable". Para ello recomiendan una herramienta llamada **Bleach** (Blanqueador) el cual se debe instalar via **pip** y no explicaremos más para no poner pesada esta entrada (más pesada, queremos decir).

Ya tenemos la idea de lo que vamos a hacer, ahora analizamos el código que vamos a modificar el cual podemos inspeccionar y descargar desde este enlace. En realidad son dos funciones que modificaremos: la función que muestra los ficheros en un archivo virtual en formato HTMl con un listado de los archivos a compartir y luego la otra función que entrega el ficero en sí mismo.

La primera función es llamada **list_directory** y la segunda es llamada **do_GET** y ambas pertenecen a la clase **SimpleHTTPRequestHandler.**

Fuentes consultadas:

En idioma castellano:

- «Compartir archivos fácil y rápidamente gracias a Python» por sr. Elías Rodríquez Martín.
- «PDF de programación - Extensiones de archivos» en "La Web del Programador".

En idioma inglés:

- "Tech Tip: Really Simple HTTP Server with Python" by Mohammed Hisamuddin.
- "Industrial Light & Magic Runs on Python" at Python.org
- "Download Python for Other Platforms at Python.org
- "How to fix: “UnicodeDecodeError: 'ascii' codec can't decode byte”.
- "http.server — Base Classes for Implementing Web Servers".
- "Difference between Carriage Return, Line Feed and End of Line Characters."
- "RFC-822".
- "RFC-2045".
- "RFC-2046".
- "mimetypes" at Python.org
- "mimetypes" at Python.org's Github account.
- "Using Markdown as a Python Library".
- "Bleach" by Mozilla.