Trabajando con documentos PDF

El formato PDF es hoy en día ubicuo, sin embargo no nos detenemos a pensar mucho sobre cómo funcionan y qué podemos hacer con ellos. Anteriormente publicamos un problemilla que tuvimos para extraer los datos allí contenidos y los cuales ofuscaron de una manera muy intelegente, admirable en verdad, muy bien pensado. Pero eso no nos detiene y allí propusimos una solución que luego lo vamos a complementar con otro de nuestros artículos (todos nuestros tutoriales están interconectados o relacionados para configurar un todo). Empezemos, pues, ¡VAMOS!

Introducción.

Esta entrada no va a ir sobre la historia y normas de los documentos PDF ("**P**ortable **D**ocument **F** ormat" por sus siglas en idioma inglés) ya que sobre eso hay bastante material en la red ***y, además, que nosotros somos terriblementes pragmáticos.*** Ojo, no estamos diciendo que estudiar la teoría es mala, sino que en este caso la teoría es compleja, e incluso hay quienes piensan que las especificaciones establecidas son "maléficas" ya que permiten muchos fallos en la seguridad de nuestros ordenadores, pero ¿por qué a devenido esta situación? Veamos.

Brevísima historia del PDF.

Para todos los efectos, en vez de escribir a cada rato "Formato de Documento Portátil" os mostraremos el acrónimo de tres letras **PDF** por el cual es ampliamente conocido a nivel mundial por ser la norma *de facto* digital.

Aunque vosotros no lo creaís, el PDF comenzó siendo un formato privativo por parte del **Doctor John Warnock,** allá en 1991 (cuando nosotros en esa época montábamos redes de área local con Netware Novell con IPX/SPX y cable de red coaxial RG58 de 10 mbps -y estudiábamos ingeniería en la Universidad de Carabobo y aún Linus Torvalds estaba por comenzar a estudiar en la universidad-). Originalmente el proyecto se llamaba "Camelot" y para 1992 obtuvo su madurez técnica.

La idea era "sencilla": poder abrir un fichero bajo diferentes sistemas operativos (y, por ende, en distintos hardwares) de la misma forma y manera, tanto por pantalla como por impresora. Debido a las limitaciones de los ordenadores y el costo de los mismos (y vaya que nosotros lo podemos atestiguar, lo sufrimos en carne propia) la ardua tarea era lo que hoy damos por sentado -recordad que el internet no estaba masificado aún-: que un documento se viera igual a como lo habían diseñado, "artísticamente", los autores respectivos. En aquel tiempo la idea era revolucionaria, apenas comenzábamos a usar sistemas operativos con entorno gráfico y dispositivos apuntadores -léase ratones-.

Para ello el Dr. John Warnock y su socio Charles Geschke basaron las especificaciones en un

archivo autocontenido que contuviera las fuentes (tipos de letras) e imágenes -tan simple como eso- y en base a esas especificaciones programar diferentes ejecutables que pudieran abrir y mostrar dichos documentos en diferentes ambientes. *Pero ellos no estaban solos en su trabajo, ambos (Warnock y Geschke) trabajaban en "Xerox's Palo Alto Research Center (Xerox PARC)" y estaban tratando de convencer a dicha empresa acerca de un nuevo lenguaje gráfico que ellos denominaban* ***Interpress*** *pero no le prestaron atención tras lo cual decidieron renunciar y fundar la empresa* ***Adobe (¿la conoceís acaso?)*** y rebautizaron dicho lenguaje como PostScript. ***Como véis, era software privativo e hicieron dinero a raudales por aquello de que "en tierra de ciegos, el tuerto es el rey".***

Fallas en la seguridad del PDF.

Como podremos imaginar, lo único constante es el cambio, y para **PDF** ésto no es la excepción. Con el devenir de los años y el aumento de la potencia de los equipos aunado a la masificación de la internet, se hizo necesario -por ejemplo- que en los archivos PDF pudiera insertarse enlaces web para ampliar la información que uno leía. Pero las aplicaciones que leen y muestran documentos PDF no fueron hechas para navegar por internet, *eso ya es un mundo completamente aparte y diferente, sumamente difícil de incrustar (ya veremos después que al final sucedió lo contrario: los modernos navegadores web pueden mostrar documentos PDF sin ningún tipo de problema debido a las bien conocidas especificaciones de formato de archivo).*

La solución para la época fue que las aplicaciones lectoras de documentos PDF (lo abreviaremos de ahora en adelante como visor PDF) se les dió la "facultad" de lanzar aplicaciones no solo para los enlaces web contenidos en el documento sino también para todo el "multimedia" -audio y vídeo- e incluso la ejecución de ciertos lenguaje scomo JavaScript . Acá un ejemplo, en vídeo, de la situación en el año 2010:

https://www.youtube.com/watch?v=jTlwxfRqODs

El señor Neil J. Rubenking en el año 2010 se hizo la pregunta ¿acaso **PDF** significa **Pretty Dangerous Format** (Bonito y Peligroso Formato)? y a la conclusión que él llega es que en ese año, ahora en 2017 y allá por 1990 *la solución a los problemas de seguridad en los archivos PDF es volver a sus raíces: mostrar correctamente un documento tal como fue concebido y eliminarle todos los agregados "multimedia" que pueden hacer daño a nuestros ordenadores.*

PDF como "formato abierto".

Irónicamente el artículo que data del año 2010, y al cual hacemos referencia en la sección anterior, no nombra para nada el hecho que en el año 2005 la Asociación Internacional para la Normalización ("International Organization for Standardization" o **ISO** por su abreviatura) emitió la norma ISO 19005-1 mejor conocida como **PDF/A** la cual está basada en la consabida norma **PDF** *dictada por Adobe* la cual recoge los sueños y anhelos de los más puristas defensores del

documento de formato abierto.

Las características que hacen al **PDF/A** una joya a ser conservada son las siguientes:

1. Debe ser independiente del dispositivo donde se visualize.
2. Debe ser autocontenido, no necesita descargar ningún recurso fuera de los que contiene el archivo en sí.
3. Debe ser autodocumentado, lo cual garantiza a futuro su interpretación y correcta lectura.

Debido a estas tres características -cada fichero debe contener las instrucciones para abrirlo junto con el texto e imágenes también- resulta en que dichos archivos son ligeramente voluminosos. **Pero es un costo que muchos estamos dispuestos "a pagar"** e incluso existe una asociación formal que defienden a capa y espada dicho formato y desarrollan aplicaciones que garantizan mantener sus especificaciones.

El **PDF/A** garantiza su "pureza" al prohibir expresamente las siguientes características:

- No deben contener ni audio ni vídeo.
- No deben contener imágenes en formato JPG (deben ser imágenes de formato abierto, el JPG es privativo).
- El lenguaje JavaScript y el llamado a ejecutar aplicaciones está prohibida.
- Todas las fuentes o tipos de letras deben estar contenidas y se debe garantizar que no tienen restricción alguna para ser reproducidas y mostradas (entiéndase fuentes de dominio público).
- La manera de definir y describir los colores debe ser independiente del dispositivo utilizado (entiendase especificaciones abiertas o normalizadas de color).
- La encriptación está deshabilitada (lo cual es totalmente diferente a comprimir y/o encriptar un fichero PDF/A para enviarlo a otras personas o instituciones).
- El uso de metadatos (datos que se describen a sí mismos) basados en normas internacionales reconocidas ***es obligatoria.***

Subsiguientes normas PDF/A.

La norma **ISO PDF/A** en realidad es la norma **PDF/A-1** y luego le sucedieron dos normas adicionales:

- **PDF/A-2** emitida en el año 2011 la cual extiende las capacidades del **PDF/A-1.**
- **PDF-/A-3** emitida en el año 2012 el cual permite la incrustación de diverson formatos de archivos tales como XML, CSV, CAD, hojas de cálculo, etc.

Desde luego, las especificaciones son más complejas, pero en aras de la simplicidad, así lo hemos explicado. Tal vez, y a futuro, publicaremos algo sobre formularios en **PDF** lo cual

consideramos algo verdaderamente útil: recibes un fichero donde puedes llenar ciertos campos (tu nombres, apellidos, etc.) y mandarlo a la impresora. *Lo que no consideramos para nada útil es devolver ese mismo fichero por correo electrónico ya que para eso mucho mejor es crear una página web segura (**https**) para recolectar dicha información en línea y guardarla en una base de datos.*

Ah, y otra cosa, no creaís que **Adobe** está tranquila y jactándose de ser "padres" del **PDF/A-1**, incluso tienen tutoriales acerca de cómo eliminar dicha especificación, eliminar todo rastro -más bien restricción- que establece dicha norma en lo archivos que osen usarla.

Extrayendo elementos de un fichero PDF.

Si habéis aguantado hasta acá nuestra resumida descripción del mundo **PDF** os felicito y ahora pasemos a la práctica. Como ya os dijimos el **PDF** fue creado para *transmitir información* y su contenido jamás ni nunca fue diseñado para ser editado ni transformado.

Es por ello que a veces suceden cosas como la acontecida a nuestro colega de "**Linux GNU Blog**":

`https://twitter.com/LinuxGnuBlog/status/847531417017962496`

https://twitter.com/ks7000/status/843462504420458497

https://twitter.com/ks7000/status/843465094621290498

Lo que sucede es lo siguiente: es un fichero **PDF** que contiene las instrucciones para un electrodoméstico y con cualquier visor **PDF** abre perfectamente y de maravilla. Pero he aquí lo extraño: pesa 9 megabytes (exactamente 9.212.740 bytes) lo cual es MUY grande para la información que contiene, considerando que los gráficos son en escala de grises y no tienen mayor detalle que el estrictamente necesario para describir al aparato electrónico. **Lo más extraño viene luego, si extraemos un rango de sus páginas, el fichero resultante tiene mayor tamaño incluso que el original, lo cual contraviene cualquier tipo de lógica.**

https://twitter.com/ks7000/status/843476633302552576

De hecho arriba vemos que si lo abrimos con **Libre Office Draw** tardará muchísimo tiempo en ser abierto, e incluso tal vez no lo hará. Eso es debido a que tiene en capas transparentes que en modo normal no se visualizan pero que a la hora de "editar" el fichero aparece como polígonos aparentemente aleatorios que llenan y "engordan" todo el documento. **De hecho Libre Office Draw NO ESTÁ EDITANDO EL ARCHIVO PDF, está simplemente extrayendo su información**

para que pueda ser editado en formato de dibujo ".odg", y el hecho que pueda ser exportado de nuevo a formato **PDF** pues es un extra añadido, ya que si queremos seguir editando dicho documento cada vez que queramos, debemos guardarlo en formato "**.odg**".

https://twitter.com/ks7000/status/843532136258949120

Nosotros hicimos un análisis exportando el contenido **PDF** a formato **XML** y eso ocupó más de 50 megabytes debido a la técnica de ofuscamiento basado en dibujar incontables polígono que, en teoría, evitarían que sea "editado" o más bien transformado a otro documento **PDF**: ¡tanto misterio para un manual de usuario de un artefacto electrónico! ***Pues más bien nos causa risa tal comportamiento.***

Usando aplicaciones para unir y cortar documentos PDF.

Cada quien tiene un estilo diferente de publicar sus conocimientos, *en nuestro caso lo orientamos más bien hacia la programación (Python, PHP, lenguaje C, etc.)* pero hay excelentes aplicaciones gráficas que suplen rápidamente nuestras necesidadas **e incluso algunas como Ghoscript corren en una ventana terminal.** Os recomendamos un excelente artículo escrito en castellano donde podréis ampliar vuestro conocimientos sobre el tema PDF.

Usando Python para trabajar con ficheros PDF.

Ya os hemos explicado sobre el cómo programar -y por supuesto instalar- el lenguaje Python, incluso viene incluído en las mayorías de las distribuciones basadas en GNU/Linux. La utilería elegida para sacar información es **PyPDF2** y podemos instalarlo por medio de una línea de comando en una ventana terminal(recordad anteceder el comando **sudo**):

```
pip install PyPDF2
```

Luego que haya descargado pues estamos prestos y listos a lanzar nuestras líneas de código. Acá os colocamos el ejemplo dado el caso vosotros utilicéis **Python3**, ya que debéis utilizar **pip3**(se sugiere utilizar el comando **sudo** con la opción "-H" para referirnos a nuestro propio directorio "**home**"):

Primero podéis descargar el manual de marras cuyo enlace está en uno de nuestros mensajes tipo "tuit":

https://twitter.com/ks7000/status/843499601894260736

Sobre ese manual es que vamos a practicar los comandos -programa- en Python con **PyPDF2** y para todos los efectos prácticos nosotros descargamos y renombramos dicho archivo a "**manual.pdf**" para escribir menos código y mayor legilbilidad didáctica. Comencemos , pues, a desgranar el fichero en una cónsola terminal con Python en modo interactivo:

```
>>> # ag
regamos la refer
encia a la libreria necesaria  .
..   >>> import PyPDF2
  >>> # abrimos el archivo en si y lo almacenamos en 'archivoPDF'  ...
>>>
archivoPDF = open('manual.pdf', 'rb'
)  >>> # ese archivo abierto lo leemos con PyPDF2 y lo almacenamos en el
objeto 'lectorPDF'  ...  >>>
lectorPDF = PyPDF2.PdfFileRea
der(archivoPDF)
  >>> # ahora procedemos a obtener la pagina 61, instrucciones en castell
ano y almacenamos en objeto
'paginaPDF'  >>>
paginaPDF = lectorPDF.getPage(61)
  >>> # ya tenemos esa pagina en memoria y aplicamos el metodo de extraer
 solamente el texto  >>> paginaPDF.extractText()
  u'2Instrucciones de seguridad\nLea detenidamente estas instrucciones de
 seguridad \nantes de utilizar el producto.\n ADVERTENCIA\nNo coloque la
TV ni el mando a distancia en los siguientes entornos:\nUna ubicaci\xf3n
expuesta a luz solar directa\nUn \xe1rea con mucha humedad, com  >>> # LA
 PAGINA ES LARGA hemos publicado apenas el inicio de ella  ... # NOTAD qu
e el texto esta 'formateado' para lenguaje Python para representar caract
eres unicode
```

Cada línea está debidamente comentada y debemos agregar que para visualizar correctamente el texto de la página que está en formato unicode para Python3 se debe utilizar, simplemente, lo siguiente:

```
print(paginaPDF.extractText())
```

Para Python (2) debemos hacer una función como la siguiente:

```
def MuestraUnicode(texto):   try:     texto = unicode(texto, 'utf-8')   e
xcept TypeError:     return texto    print(MuestraUnicode(paginaPDF.extra
ctText()))
```

Pero no nos desviemos de nuestro objetivo, desgranar un documento PDF.

Extrayendo un rango de páginas a un documento nuevo PDF.

Acá volvemos a reutilizar los conceptos del ejemplo anterior, de nuevo cada línea está comentada indicando claramente qué hace cada comando:

```
>>>              #              Referencia              a              la
    libreria    necesaria         ...            >>>    import    PyPDF2
  >>> # Abrimos el manual al cual lo denominamos origen de datos  ...   >
>>
origenPDF             =             open('manual.pdf',             'rb')
;                                          >>>                        #
                   Leemos                    el                    orig
en         abierto          en          un          objeto          lector
;                          ...                                       >>>
origenPDFlector=                                                 PyPDF2.
PdfFileReader(origenPDF);
                   >>>          #          Creamos          un          o
bjeto      escritor      de      PDF            ...                  >>>
destinoPDF                    =                   PyPDF2.PdfFileWriter()
 >>> # Creamos un objeto escritor de PDF y adicionamos la pagina 60 (inst
ruccioens      en      castellano)            ...                    >>>
paginaOBJ                     =                    origenPDFlector.getPa
ge(60);                      >>>           destinoPDF.addPage(paginaOBJ)
                                 >>>                 #                 Un
ciclo              para              leer              de              la
pagina      61      a      la      80            ...                   >>>
for      num_pag      in      range(61,80):             ...
paginaOBJ                      =                      origenPDFlector.g
etPage(num_pag)                              ...
destinoPDF.addPage(paginaOBJ)
                  ...                            >>>         #         abri
mos     un     fichero     de     destino          ...                 >>>
                                                          destinoPDFar
```

```
chivo                          =                          open('manual_en_c
astellano.pdf'                              ,                          'wb');
                                                 >>>                          #
escribimos                                  el                            ob
jeto       en       el       fichero       de       destino              .
..                                                                       >>>
destinoPDF.write(destinoPDFarchivo);
                        >>>           #           cerramos           los
       archivos       debidamente                ...                   >>>
destinoPDFarchivo.close()  >>> origenPDF.close()  >>>
```

Tendremos así, entonces, un fichero *solamente con las instrucciones en castellano* que podemos abrir con el visor de dcoumentos predeterminado en Ubuntu, en este caso **Evince**. En nuestro caso la salida pesa 1.240.731 bytes (para nosotros los viejitos eso es casi un *diskette 3½ pulgadas*) y, **tercos nosotros como somos,** lo intentamos abrir de nuevo con **LibreOfficeDraw** y cuando va más o menos por la tercera hoja "COLAPSA" la aplicación rellenando cada cuadrito "basura" que le colocaron para ofuscar el documento. Luego de 45 segundos es que se presta a mostrar por pantalla una copia del documento, tal como nosotros hicimos en lenguaje Python. Incluso después de abrir por completo la primera página, si nos ubicamos en la novena, por ejemplo, de nuevo **LibreOfficeDraw** colapsa de nuevo, así que el truco de cargar basura en un **PDF** vaya que funciona en realidad.

Por ahora vamos a recapitualr el método que aprendimos en esta sección, lo que llamamos pseudocódigo:

- Inicio del proceso.
- Colocamos la referencia a la librería necesaria.
- Abrimos el documento PDF en modo lectura binario.
- Creamos un objeto de lectura de la librería y leemos el archivo abierto.
- Creamos un objeto de escritura de la librería.
- Creamos un objeto de página de la librería que lee del documento origen y lo pasamos al documento destino (1 página a la vez con *exactamente todo el contenido de cada página: texto, imágenes, **basura...***).
- Una vez que tenemos el objeto de escritura listo con las páginas que nos interesa, abrimos un archivo destino en modo de escritura binario (lo nombramos como querramos).
- LLamamos al método de escritura del objeto de escritura de la librería.
- Cerramos el archivo destino.
- Cerramos el archivo origen.
- Fin del proceso.

Ofuscando nuestros propios documentos PDF.

Podemos "ofuscar" nuestros propios documentos PDF colocandole, por ejemplo, una licencia de **Creative Commons** como la que acompaña nuestra página web y dejando bien en claro que la información que compartimos es libre pero respetando, al menos, el nombrar la autoría. Para ello podemos crear un documento PDF de una sola hoja que tenga una *marca de agua* que simplemente es un texto en gris claro escrito a 45° sobre la diagonal de la hoja.

Para crear nuestro documento con la marca de agua podemos utilizar rapidamente a **Gimp** y exportamos a PDF colocandole el nombre, muy original eso sí, **marca_de_agua.pdf** y escribir el siguiente código:

```
      #      Solo      las      librerias      necesarias
from                PyPDF2                import                PdfFile
Writer,                                                  PdfFileReader
                #     Creamos     un     objeto     de     escritura
                              salida          =          PdfFileWriter()
                         #     Abrimos     el     documento     origin
al
origenPDF                            =                          PdfFileRea
der(open('origen.pdf',                                          'rb'))
                                      #           Abrimos           nu
estro           documento           creado           con           mar
ca                de                agua
marca_aguaP
DF                           =                          PdfFileReader(ope
n('marca_de_agua.pdf',                                          'rb'))
   # Abrimos la primera hoja del documento marca de agua y lo guardamos e
n         un         objeto         hoja         PDF
hoja_marca_agua                      =                    marca_aguaPDF
.getPage(0)
     # Hacemos un ciclo para COMBINAR cada hoja del origen con la hoja co
n         la         marca         de         agua
for             i             in             xrange(origenPDF.getNum
Pages()):
     # Leemos la pagina en un objeto y luego lo combinamos con 'merge()'
                  pagina    =    origenPDF.getPage(i)
pagina.mergePage(hoja_marca_agua)
                                    #      vamos      agregando      al
     documento     destino                          salida.addPage(pagina)
                                                     #          escribimos
en       nuestro       disco       duro       el       resultado
```

```
with open('destino.pdf', 'wb') as f:     salida.write(f)
```

¿En qué consiste la ofuscación? Podemos escribir con Python un programa que cree una hoja PDF con imágenes aleatorias para luego combinarla con el documento que queremos ofuscar (agregar por capas) y luego hacer la capa con dichas imágenes como invisible (la capa) ***que cualquier visor PDF no la msotrará pero que al extraer las hojas siempre nos llevaremos dichas imágenes y las podremos ver (y hasta colapsar) al programa que hayamos usado para extraerle páginas, ¿Qué tal os parece?***

Eliminando las imágenes del archivo PDF destino.

De la sección "Extrayendo un rango de páginas a un documento PDF" retomamos exactamente todo el código allí escrito y le insertamos la siguiente línea *justo antes de escribir en el disco duro el resultado* (guiense por los comentarios):

```
# eliminamos las imágen
es del archivo destinoPDF
destinoPDF.removeImages()
   # abrimos un fichero de destino
destinoPDFarchivo = open('manual_en_castellano.pdf' , 'wb');
```

Con el comando "**.removeImages()**" logramos quedarnos solamente con el texto y el fichero resultante es de 780.895 bytes ¡una reducción de 37%! (pesaba 1.240.731 bytes). Otros dos comandos útiles de la librería **PyPDF2** son los comandos para eliminar los enlaces web **.removeLinks()** y para eliminar el texto **.removeText()**.

El punto interesante es que, de nuevo, abrimos el archivo resultante con **LibreOfficeDraw** y podemos hacer una copia del documento que puede ser editada *pero por supuesto, sin las imágenes* ***que son las que ocasionan que se "cuelgue" la aplicación.***

Fuentes consultadas.

En idioma castellano:

- "Formato de fichero de documento electrónico para la conservación a largo plazo" en la Asociación Española de Normalización y Certificación.
- "Uniendo y cortando PDF en GNU/Linux" por Elías R.M. en LinuxGnuBlog.org

En idioma francés:

- "Organisation internationale de normalisation".

En idioma inglés:

- "PDF: Three letters that changed the world".
- "John Warnock Biography" at Wikipedia.
- "PostScript" at Wikipedia.
- "Malicious PDFs Execute Code Without a Vulnerability" By Larry Seltzer (March 30, 2010).
- "¿Analyst's View: PDF— is a Pretty Dangerous Format?" by Neil J. Rubenking (April 7, 2010).
- "¿Does PDF stand for Problematic Document Format?" by Mikko at F-Secure Labs.
- "¿What is PDF/A?" at CVision -Smater Document Capture-.
- "PDFA Association".
- "Using PDF/A as a Preservation Format" at New York Archives.
- "PDF/A, PDF for Long-term Preservation" at Library of Congress of United States of América.
- "How to Remove PDF/A Information from a file" at Adobe's Blog.
- "Working with PDF and Word Documents" by Al Sweigart.
- "PyPDF2 1.15" at Python .org
- "Decoding if it's not unicode" at StackOverflow.
- "Evince" at Gnome.org
- "Manipulating PDFs with Python" by Tim Arnold.
- "How to Install MuPDF 1.9 in Ubuntu 16.04 LTS" by Ji Im.
- "MuPDF".
- "PyMuPDF".
- "Extract images of a PDF - optionally by page using PyMuPDF / fitz (Python recipe)" by Jorj X. McKie.
- "PyMuPDF tutorial".
- "An Intro to pyfpdf – A Simple Python PDF Generation Library".