Trainer-Spickzettel für Trainingslager: "PDF-KungFoo mit Ghostscript & Co."

kurt.pfeifle@mykolab.com

04. September 2015,
MRMCD Darmstadt

Contents

Ein paar der HTML-Folien zur Einführung zeigen

(Trainer-Vorstellung)

Auf Beamer/PDF-Folien verweisen...

(Thema: PDF-Syntax)

Erstellung einer PDF im Text-Editor “live” vorführen

(dabei nebenher die diversen CLI-Tools demonstrieren, die zur Analyse oder Modifikation von PDFs nützlich sind)

(dabei zugleich jeweils auf die bereits seit Jahren vorhandenen, äußerst nützlichen, jedoch wenig bekannten Parameter hinweisen…)

Die folgenden Tools vorstellen und ihre Verwendung demonstrieren

(Mein Spickzettel ist ab hier Englisch. Aber das kann ich ja gut genug, um das folgende zu verstehen :-)

A Bag of Useful Tools

When hand-coding PDF files (or manually manipulating PDFs which are already existing), it is useful to be friends with a little arsenal of command line tools for testing and other purposes. Here is a list of tools which we regularly use. Most of them are available for all major OS platforms (Windows, OSX, Linux):

- **`pdfinfo`** (the most recent versions based on Poppler – forked from the original XPDF codebase – have new features not supported by the venerable XPDF versions). This utility reports general metadata about a PDF.
- **`pdfimages`**. This utility reports various details about images embedded in PDFs. It can also extract them.
- **`pdffonts`**. This utility reports various details about the fonts used by a PDF.
- **`pdfdetach`**. This utility reports if a PDF file makes use of the official feature that allows to embed other files within a PDF. It can also extract a copy of the attached files from the PDF.
- **`pdfresurrect`**. This utility reports if a PDF makes use of the offical feature that allows to make *“incremental updates”* of the document. It can also restore previous versions of the file.
- **`pdftk`**. This utility can …
- **`qpdf`**. This utility is, according to its self-description *“a command-line program that does structural, content-preserving transformations on PDF files.” It is extremely valuable to un-compress streams and object streams contained in a PDF that you want to understand, debug or modify.
- **`Ghostscript`**. This utility can process PostScript and PDF and convert them into a lot of different graphics and printer-understandable raster formats. It doubles up as a PostScript and PDF viewer also. Furthermore, it create and modify PDF files from PostScript and PDF input.
- **`zathura`**. This utility is a very fast and lightweight PDF viewer. (Additional plugins extend it to a viewer for PostScript, DjVu and CB files.) It has a very limited GUI; instead of mouse and menue buttons, it is to be controlled from the keyboard.
- **`mupdf`**. This utility is another lightweight PDF viewer. (It can also display XPS files.) It has a very limited GUI; instead of mouse and menue buttons, it is to be controlled from the keyboard.
- **`mutool`**. This utility is a sibling to `mupdf`. Not a PDF viewer, but a little toolbox with useful sub-commands: `clean` (re-writes PDF files), `extract` (extracts font and image resources), `show` (displays internal PDF objects), `poster` (splits large PDF pages into smaller tiles) and `info` (displays a PDF’s metadata).
- **`podofo{box,color,countpages,crop,encrypt,gc,img2pdf,imgextract,impose,incrementalupdates,merge,pages`**
  This family of utilities can …
- **`peepdf`**. `peepdf.py` is a Python suite of toolto explore PDF files. It was initially created to help find out if a PDF contains harmful contents or not. It is however extremely useful beyond PDF malware research, because it helps to explore, study, investigate and understand PDF file structures in general.
- **`pdfid.py`**. This utility can …
- **`pdf-parser.py`**. This utility can …

Specific hints

In case you are not yet familiar with these tools, you should get soon. However, here are a few specific hints. These are beneficial also to "old timers" who are using them since many years already (because they may have missed the updated features we now hint at).

pdfinfo

1. To see if the file includes an `XMP` version of its metadata, use:

```
pdfinfo -meta the.pdf
```

2. To see all page boxes (MediaBox, CropBox, TrimBox, ArtBox) used by the file (or what page boxes are implicitely used because they are undefined), use:

```
pdfinfo -box the.pdf
```

   Be aware, that this form of the command examines only the first page, and the command output reflects this. As you know, PDF documents may have different page sizes within the same file. So take a look at the next tip.

3. To see page-related info about different (or all) pages, use the `-f <N> -l <M>` parameters. The following command retrieves the page box info related to pages 11–15:

```
pdfinfo -f 11 -l 15 -box the.pdf
```

4. To print the contents of an embedded JavaScript, use:

```
pdfinfo -js the.pdf
```

   Note: If it is a malicious JavaScript which the originator wanted to hide from the users by applying certain obfuscation techniques, the `-js` key will very likely not work.

pdfimages

1. Older versions of this utility could only *extract* embedded images. Recent Poppler versions (not the XPDF versions though!) have now the `-list` parameter:

```
pdfimages -list the.pdf
```

   This command will print a list of images which are used inside the PDF file with various additional info: the page number where the image appears, the image dimensions, their compression ratio, the PDF object ID of the image, their color depth, the number of color componenents.

2. One notable detail about the previous hint: since `pdfimages -list` returns the respective PDF object ID, it is worth to check if various images listed for the PDF use *identical* ID numbers!

   Because identical object IDs for different image numbers mean: the PDF makes re-use of that image (it is very efficiently constructed) multiple times, showing it on different locations – but it is embedded only once. This is not always the case – older versions of OpenOffice/LibreOffice embedded a page background image once per page, creating very large output PDFs. This is no longer the case (at least with the 4.3/4.4 releases of LibreOffice).

3. Take note of the fact that the `-f <N> -l <M>` command line params does also work for `pdfimages`.

pdffonts

1. This utility prints a list of metadata about the fonts used by a PDF file. If the column headed `uni` does not show up a `yes` for a specific font, it may be difficult or even impossible to extract the text (either by *copy'n'paste'* or with the help of `pdftotext`). All you get may be unreadable gobble-di-gook.

2. A new feature in recent Poppler versions (again: not available for XPDF-based `pdffonts`!) is the `-subst` parameter. It prints

```
pdffonts -subst the.pdf
```

If the tool returns an empty table, then the PDF may have

(a) either all fonts embedded
(b) no fonts in use at all

Note: whatever substitute that tool reports for non-embedded fonts is not true for Acrobat or other PDF viewers. These may use a different font substitution method (Acrobat frequently generates a *MultipleMaster* font "on the fly" for use in place of a non-embedded one). The tool's reported substitute font is only applicable for those programms which make use of the *FreeType* font engine.

pdfresurrect

pdfdetach

pdftk

Ghostscript

1. If you want to generate (or re-generate) a PDF which does not use real fonts for text glyphs, but instead small vector shapes, you can use `-dNoOutputFonts` on the command line (starting with GS release v9.15):

```
gs -o out.pdf -sDEVICE=pdfwrite -dNoOutputFonts input.pdf
```

2. If you are on Windows, you'll have different Ghostscript executables:

   - **`gswin32c.exe`** and **`gswin64c.exe`** : 32-bit variant of the 'console' version of Ghostscript. Note the **`c`** in the names.
   - **`gswin32.exe`** and **`gswin64.exe`** : 32-bit/64-bit variants of the 'GUI' version of Ghostscript. (The GUI is only an extra window that opens to report `<stdout>` and `<stderr>` messages, and which serves to receive keyboard input if you use Ghostscript interactively). Note there is no **`c`** in the names of the GUI versions.

zathura

mupdf

mutool

podofo{box,color,countpages,crop,encrypt,gc,img2pdf,imgextract,impose,incrementalupdates

peepdf

1. `peepdf` can be used to apply and unapply different `/Filter` settings to streams. Supported *encoding* filters (for applying to an un-encoded stream) are: asciihex (alias: ahx), lzw, flatedecode (alias: fl). Supported *decoding* filters (for decoding to an encoded stream) are: asciihex (alias: ahx), ascii85 (alias: a85), lzw, flatedecode (alias: fl), runlength (alias: rl), ccittfax (alias: ccf).
2. Not implemented filters (in either direction) are: jbig2 (`/JBIG2Decode`), dct (`/DCTDecode`), jpx (`/JPXDecode`). There is a Google Summer of Code 2015 opportunity for a willing student to implement missing filters for PeepDF.

pdfid.py

pdf-parser.py

qpdf

1. Be aware: `qpdf` is not perfect! Though it works extremely well for most practical cases, there are some things where its *"content preserving transformation"* fails:

- It cannot preserve "layers" (in PDF parlance: `OCG`, optional content groups) in the output PDF. If you let `qpdf` transform a PDF file containing layers, the output will have flattened them all into one.
- It does not know how to handle *incremental updates* of PDFs. If you let `qpdf` transform a PDF file containing incrementally updated versions the output will reflect the last file version only.

2. For me personally, qpdf' ist most useful when it comes to the following three tasks:

   - Quick-check the PDFs internal structure by running:

     ```
     qpdf --check the.pdf
     ```

   - Return the correct `xref` table values by running:

     ```
     qpdf --show-xref the.pdf
     ```

   - Unpack as many internal structures as possible by running:

     ```
     qpdf --qdf --object-streams=disable the.pdf unpacked.pdf
     ```

[… TO BE COMPLETED …]
