



(/)

digital preservation - file formats

PDF processing and analysis with open- source tools

06 September 2021



Plumbers Tool Box (<https://www.flickr.com/photos/130648318@N06/42662053232>) by pszz (<https://www.flickr.com/photos/130648318@N06/>) on Flickr. Used under CC BY-NC-SA 2.0 (<https://creativecommons.org/licenses/by-nc-sa/2.0/>).

Over the years, I've been using a variety of open-source software tools for solving all sorts of issues with PDF documents. This post is an attempt to (finally) bring together my go-to PDF analysis and processing tools and commands for a variety of common tasks in one single place. It is largely based on a multitude of scattered lists, cheat-sheets and working notes that I made earlier. Starting with a brief overview of some general-purpose PDF toolkits, I then move on to a discussion of the following specific tasks:

- Validation and integrity testing
- PDF/A and PDF/UA compliance testing
- Document information and metadata extraction
- Policy/profile compliance testing

- Text extraction
- Link extraction
- Image extraction
- Conversion to other (graphics) formats
- Inspection of embedded image information
- Conversion of multiple images to PDF
- Cross-comparison of two PDFs
- Corrupted PDF repair
- File size reduction of PDF with hi-res graphics
- Inspection of low-level PDF structure
- View, search and extract low-level PDF objects

How this selection came about

Even though this post covers a lot of ground, the selection of tasks and tools presented here is by no means meant to be exhaustive. It was guided to a great degree by the PDF-related issues I've encountered myself in my day to day work. Some of these tasks could be done using other tools (including ones that are not mentioned here), and in some cases these other tools may well be better choices. So there's probably a fair amount of selection bias here, and I don't want to make any claims of presenting the "best" way to do any of these tasks here. Also, many of the example commands in this post can be further refined to particular needs (e.g. using additional options or alternative output formats), and they should probably best seen as (hopefully useful) starting points for the reader's own explorations.

All of the tools presented here are published as open-source, and most of them have a command-line interface. They all work under Linux (which is the main OS I'm using these days), but most of them are available for other platforms (including Windows) as well.

PDF multi-tools

Before diving into any specific tasks, let's start with some general-purpose PDF tools and toolkits. Each of these are capable of a wide range of tasks (including some I won't explicitly address here), and they can be seen as "Swiss army-knives" of PDF processing. Whenever I need to get some PDF processing or analysis done and I'm not sure what tool to use, these are usually my starting points. In the majority of cases, at least one of them turns out to have the functionality I'm looking for, so it's a good idea to check them out if you're not familiar with them already.

Xpdf/Poppler

Xpdf (<https://www.xpdfreader.com/>) and Poppler (<https://poppler.freedesktop.org/>) are both PDF viewers that include a collection of tools for processing and manipulating PDF files. Poppler is a fork of this software, which adds a number of unique tools that are not part of the original Xpdf package. The tools included with Poppler are:

- **pdfdetach**: lists or extracts embedded files (attachments)
- **pdffonts**: analyzes fonts
- **pdfimages**: extracts images
- **pdfinfo**: displays document information
- **pdfseparate**: page extraction tool
- **pdfsig**: verifies digital signatures
- **pdftocairo**: converts PDF to PNG/JPEG/PDF/PS /EPS/SVG using the Cairo (<https://www.cairographics.org/>) graphics library
- **pdftohtml**: converts PDF to HTML
- **pdftoppm**: converts PDF to PPM/PNG/JPEG images
- **pdftops**: converts PDF to PostScript (PS)
- **pdftotext**: text extraction tool
- **pdfunite**: document merging tool

The tools in Xpdf are largely identical, but don't include *pdfseparate*, *pdfsig*, *pdftocairo*, and *pdfunite*. Also, Xpdf has a separate *pdftopng* tool for converting PDF to PNG images (this functionality is covered by *pdftoppn* in the Poppler version). On Debian-based systems the Poppler tools are part of the package *poppler-utils*.

Pdfcpu

Pdfcpu (<https://pdfcpu.io/>) is a PDF processor that is written in the Go language. The documentation explicitly mentions its main focus is strong support for batch processing and scripting via a rich command line. It supports all PDF versions up to

PDF 1.7 (ISO-32000).

Apache PDFBox

Apache PDFBox (<https://pdfbox.apache.org/>) is an open source Java library for working with PDF documents. It includes a set of command-line tools (<https://pdfbox.apache.org/2.0/commandline.html>) for various PDF processing tasks. Binary distributions (as JAR ([https://en.wikipedia.org/wiki/JAR_\(file_format\)](https://en.wikipedia.org/wiki/JAR_(file_format))) packages) are available here (<https://pdfbox.apache.org/download.html>) (you'll need the "standalone" JARs).

QPDF

QPDF (<http://qpdf.sourceforge.net/>) is "a command-line program that does structural, content-preserving transformations on PDF files".

MuPDF

MuPDF (<https://www.mupdf.com/>) is "a lightweight PDF, XPS, and E-book viewer". It includes the mutool (<https://www.mupdf.com/docs/index.html>) utility, which can do a number of PDF processing tasks.

PDFtk

PDFtk (<https://www.pdflabs.com/tools/pdftk-server/>) (server edition) is a "command-line tool for

working with PDFs” that is “commonly used for client-side scripting or server-side processing of PDFs”. More information can be found in the documentation (<https://www.pdflabs.com/docs/pdftk-man-page/>), and the command-line examples page (<https://www.pdflabs.com/docs/pdftk-cli-examples/>). For Ubuntu/Linux Mint users, the most straightforward installation option is the “pdftk-java” Debian package. This is a Java fork of PDFtk¹.

Ghostscript

Ghostscript (<https://www.ghostscript.com/>) is “an interpreter for the PostScript language and PDF files”. It provides rendering to a variety of raster and vector formats.

The remaining sections of this post are dedicated to specific tasks. As you will see, many of these can be addressed using the multi-tools listed in this section.

Validation and integrity testing

PDFs that are damaged, structurally flawed or otherwise not conformant to the PDF format specification can result in a multitude of problems. A number of tools provide error checking and integrity testing functionality. This can range from limited structure checks, to full (claimed) validation against the filespec. It’s important to note that none of the

tools mentioned here are perfect, and some faults that are picked up by one tool may be completely ignored by another one and vice versa. So it's often a good idea to try multiple tools. A good example of this approach can be found in this blog post by Micky Lindlar (<https://openpreservation.org/blogs/trouble-shooting-pdf-validation-errors-a-case-of-pdf-hul-38/>).

Validate with Pdftcpu

The Pdftcpu command-line tool has a `validate` command (<https://pdftcpu.io/core/validate>) that checks a file's compliance against PDF 32000-1:2008 (https://www.adobe.com/content/dam/acom/en/devnet/pdf/pdfs/PDF32000_2008.pdf) (i.e. the ISO version of PDF 1.7). It provides both a “strict” and a “relaxed” validation mode, where the “relaxed” mode (which is the default!) ignores some common violations of the PDF specification. The command-line is:

```
pdftcpu validate whatever.pdf
```

The “strict” mode can be activated with the `-m` option:

```
pdftcpu validate -m strict whatever.pdf
```

Validate with JHOVE

JHOVE (<http://jhove.openpreservation.org/>) is a file format identification, validation and characterisation tool that includes a module for PDF validation. It is widely used in the digital heritage (libraries, archives) sector. Here's a typical command-line example (note that you explicitly need to invoke the *PDF-hul* module via the `-m` option; omitting this can give unexpected results):

```
jhove -m PDF-hul -i whatever.pdf
```

Check out the documentation (<https://jhove.openpreservation.org/modules/pdf/>) for more information about JHOVE's PDF module, and its limitations.

Check integrity with QPDF

The `--check` option of QPDF (see above) performs checks on a PDF's overall file structure. QPDF does not provide full-fledged validation, and the documentation (<http://qpdf.sourceforge.net/files/qpdf-manual.html>) states that:

“ *A file for which `-check` reports no errors may still have errors in stream data content but should otherwise be structurally sound*

Nevertheless, QPDF is still useful for detecting various issues, especially in conjunction with the `--verbose` option. Here's an example command-line:

```
qpdf --check --verbose whatever.pdf
```

Check for Ghostscript rendering errors

Another useful technique is to process a PDF with Ghostscript (rendering the result to a “nullpage” device). For example:

```
gs -dNOPAUSE -dBATCH -sDEVICE=nullpage whatever.pdf
```

In case of any problems with the input file, Ghostscript will report quite detailed information. As an example, here’s the output for a PDF with a truncated document trailer:

```
**** Error:  An error occurred while reading an XRE
F table.
**** The file has been damaged.  This may have been
caused
**** by a problem while converting or transferring t
he file.
**** Ghostscript will attempt to recover the data.
**** However, the output may be incorrect.
**** Warning:  There are objects with matching obje
ct and generation
**** numbers.  The output may be incorrect.
**** Error:  Trailer dictionary not found.
           Output may be incorrect.
No pages will be processed (FirstPage > LastPage).

**** This file had errors that were repaired or ign
ored.
**** Please notify the author of the software that
produced this
**** file that it does not conform to Adobe's publi
shed PDF
**** specification.
```

Check for errors with Mutool info

command

Running Mutool (part of MuPDF, see above) with the `info` command returns information about internal pdf resources. In case of broken or malformed files the output includes error messages, which can be quite informative. Here's an example command-line:

```
mutool info whatever.pdf
```

Check for errors with ExifTool

ExifTool (<https://exiftool.org/>) is designed for reading, writing and editing meta-information for a plethora of file formats, including PDF. Although it does not do full-fledged validation, it will report error and warning messages for various read issues, and these can be useful for identifying problematic PDFs. For example, here we use ExifTool on a PDF with some internal byte corruption:

```
exiftool corrupted.pdf
```

Result:

```
ExifTool Version Number      : 11.88
File Name                    : corrupted.pdf
Directory                    : .
File Size                    : 87 kB
File Modification Date/Time  : 2022:02:07 14:36:47+
01:00
File Access Date/Time       : 2022:02:07 14:37:11+
01:00
File Inode Change Date/Time  : 2022:02:07 14:36:59+
01:00
File Permissions             : rw-rw-r--
File Type                    : PDF
File Type Extension          : pdf
MIME Type                    : application/pdf
PDF Version                  : 1.3
Linearized                   : No
Warning                       : Invalid xref table
```

In this case the byte corruption results in an “Invalid xref table” warning. Many other errors and warnings are possible. Check out this blog post by Yvonne Tunnat (<https://openpreservation.org/blogs/pdf-validation-with-exiftool-quick-and-not-so-dirty/>) which discusses PDF “validation” with ExifTool in more detail.

Other options

- VeraPDF (<https://verapdf.org/>) can provide useful information on damaged or invalid PDF documents. However, VeraPDF is primarily aimed at validation against PDF/A (<https://en.wikipedia.org/wiki/PDF/A>) and PDF/UA (<https://en.wikipedia.org/wiki/PDF/UA>) profiles, which are both subsets of ISO 32000 (<https://en.wikipedia.org/wiki/PDF>) (which defines the PDF format’s full feature set). As a result, VeraPDF’s validation output can be

somewhat difficult to interpret for “regular” PDFS (i.e. documents that are not PDF/A or PDF/UA). Nevertheless, experienced users may find VeraPDF useful for such files as well.

- Several online resources recommend the *pdfinfo* tool that is part of Xpdf and Poppler for integrity checking. However, while writing this post I ran a quick test of the tool on a PDF with a truncated document trailer² (which is a very serious flaw), which was not flagged by *pdfinfo* at all.

PDF/A and PDF/UA compliance testing with VeraPDF

PDF/A (<https://en.wikipedia.org/wiki/PDF/A>) comprises a set of ISO-standardized profiles that are aimed at long-term preservation. PDF/UA (<https://en.wikipedia.org/wiki/PDF/UA>) is another ISO-standardized profile that ensures accessibility for people with disabilities. These are not separate file formats, but rather profiles within ISO 32000 that put some constraints on PDF’s full set of features. VeraPDF (<https://verapdf.org/>) was originally developed as an open source PDF/A validator that covers all parts of the PDF/A standards. Starting with version 1.18, it also added support for PDF/UA. The following command lists all available validation profiles:

```
verapdf -l
```

Result:

```
1a - PDF/A-1A validation profile
1b - PDF/A-1B validation profile
2a - PDF/A-2A validation profile
2b - PDF/A-2B validation profile
2u - PDF/A-2U validation profile
3a - PDF/A-3A validation profile
3b - PDF/A-3B validation profile
3u - PDF/A-3U validation profile
ua1 - PDF/UA-1 validation profile
```

When running VeraPDF, use the `-f` (flavour) option to set the desired validation profile. For example, for PDF/A-1A use something like this³:

```
verapdf -f 1a whatever.pdf > whatever-1a.xml
```

And for PDF/UA:

```
verapdf -f ua1 whatever.pdf > whatever-ua.xml
```

The documentation (<https://docs.verapdf.org/cli/validation/>) provides more detailed instructions on how to use VeraPDF.

Document information and metadata extraction

A large number of tools are capable of displaying or extracting technical characteristics and various kinds of metadata, with varying degrees of detail. I'll only highlight a few here.

Extract general characteristics with `pdftinfo`

The *pdftinfo* tool that is part of Xpdf and Poppler is useful for a quick overview of a document's general characteristics. The basic command line is:

```
pdftinfo whatever.pdf
```

Which gives the following result:

```
Creator:      PdfCompressor 3.1.32
Producer:     CVISION Technologies
CreationDate:  Thu Sep  2 07:52:56 2021 CEST
ModDate:      Thu Sep  2 07:53:20 2021 CEST
Tagged:       no
UserProperties: no
Suspects:     no
Form:         none
JavaScript:   no
Pages:        1
Encrypted:    no
Page size:    439.2 x 637.92 pts
Page rot:     0
File size:    24728 bytes
Optimized:    yes
PDF version:  1.6
```

Extract metadata with Apache Tika

Apache Tika (<https://tika.apache.org/>) is a Java library that supports metadata and content extraction for a wide variety of file formats. For command-line use, download the *Tika-app* runnable JAR from here (<https://tika.apache.org/download.html>). By default, Tika will extract both text and metadata, and report both in XHTML

format. Tika has several command-line options that this behaviour. A basic metadata extraction command is (you may need to adapt the path and name of the JAR file)):

```
java -jar ~/tika/tika-app-2.1.0.jar -m whatever.pdf >
whatever.txt
```

Result:

```
Content-Length: 24728
Content-Type: application/pdf
X-TIKA:Parsed-By: org.apache.tika.parser.DefaultParser
X-TIKA:Parsed-By: org.apache.tika.parser.pdf.PDFParser
access_permission:assemble_document: true
access_permission:can_modify: true
access_permission:can_print: true
access_permission:can_print_degraded: true
access_permission:extract_content: true
access_permission:extract_for_accessibility: true
access_permission:fill_in_form: true
access_permission:modify_annotations: true
dc:format: application/pdf; version=1.6
dcterms:created: 2021-09-02T05:52:56Z
dcterms:modified: 2021-09-02T05:53:20Z
pdf:PDFVersion: 1.6
pdf:charsPerPage: 0
pdf:docinfo:created: 2021-09-02T05:52:56Z
pdf:docinfo:creator_tool: PdfCompressor 3.1.32
pdf:docinfo:modified: 2021-09-02T05:53:20Z
pdf:docinfo:producer: CVISION Technologies
pdf:encrypted: false
pdf:hasMarkedContent: false
pdf:hasXFA: false
pdf:hasXMP: true
pdf:producer: CVISION Technologies
pdf:unmappedUnicodeCharsPerPage: 0
resourceName: whatever.pdf
xmp:CreateDate: 2021-09-02T07:52:56Z
xmp:CreatorTool: PdfCompressor 3.1.32
xmp:MetadataDate: 2021-09-02T07:53:20Z
xmp:ModifyDate: 2021-09-02T07:53:20Z
xmpMM:DocumentID: uuid:2ec84d65-f99d-49fe-9aac-bd0c1ff
f5e66
xmpTPg:NPages: 1
```


Tika offers several options for alternative output formats (e.g. XMP and JSON); these are all explained here (<https://tika.apache.org/2.1.0/gettingstarted.html>) (section “Using Tika as a command line utility”).

Extract metadata with ExifTool

ExifTool (<https://exiftool.org/>) is another good option for metadata extraction. Here’s an example:

```
exiftool whatever.pdf
```

Result:

```
ExifTool Version Number      : 11.88
File Name                    : whatever.pdf
Directory                    : .
File Size                    : 24 kB
File Modification Date/Time   : 2021:09:02 12:23:32+
02:00
File Access Date/Time        : 2022:02:07 15:04:11+
01:00
File Inode Change Date/Time   : 2021:09:02 15:27:38+
02:00
File Permissions              : rw-rw-r--
File Type                    : PDF
File Type Extension           : pdf
MIME Type                     : application/pdf
PDF Version                   : 1.6
Linearized                   : Yes
Create Date                   : 2021:09:02 07:52:56+
02:00
Creator                       : PdfCompressor 3.1.32
Modify Date                   : 2021:09:02 07:53:20+
02:00
XMP Toolkit                   : Adobe XMP Core 5.6-c
017 91.164464, 2020/06/15-10:20:05
Metadata Date                 : 2021:09:02 07:53:20+
02:00
Creator Tool                  : PdfCompressor 3.1.32
Format                       : application/pdf
Document ID                   : uuid:2ec84d65-f99d-4
9fe-9aac-bd0c1fff5e66
Instance ID                   : uuid:28d0af59-9373-4
358-88f2-c8c4db3915ed
Producer                     : CVISION Technologies
Page Count                    : 1
```

ExifTool can also write the extracted metadata to a variety of output formats, which is explained in the documentation.

Extract metadata from embedded documents

One particularly useful feature of Tika is its ability to deal with embedded documents. As an example, this file (<https://github.com/openpreserve/format->

corpus/blob/master/pdfCabinetOfHorrors/digitally_signed_3D_Portfolio.pdf) is a PDF portfolio (<https://helpx.adobe.com/acrobat/using/overview-pdf-portfolios.html>), which can contain multiple files and file types. Invoking Tika with the `-J` (“output metadata and content from all embedded files”) option results in JSON-formatted output that contains metadata (and also extracted text) for all for all files that are embedded in this document:

```
java -jar ~/tika/tika-app-2.1.0.jar -J digitally_signed_3D_Portfolio.pdf > whatever.json
```

Elaborate feature extraction with VeraPDF

Although primarily aimed at PDF/A validation, VeraPDF (<https://verapdf.org/>) can also be used as a powerful metadata and feature extractor for any PDF file (including files that don't follow the PDF/A or PDF/UA at all!). By default, VeraPDF is configured to only extract metadata from a PDF's information dictionary, but this behaviour can be easily changed by modifying a configuration file, which is explained in the documentation (<https://docs.verapdf.org/cli/config/#features.xml>). This enables you to obtain detailed information about things like Actions, Annotations, colour spaces, document security features (including encryption), embedded files, fonts, images, and much more. Then use a command line like⁴:

```
verapdf --off --extract whatever.pdf > whatever.xml
```

VeraPDF can also be used to recursively process all files with a .pdf extension in a directory tree, using the following command-line (here, *myDir* is the root of the directory tree):

```
verapdf --recurse --off --extract myDir > whatever.xml
```

The VeraPDF documentation (<https://docs.verapdf.org/cli/feature-extraction/>) discusses the feature extraction functionality in more detail.

Policy or profile compliance assessment with VeraPDF

The results of the feature extraction exercise described in the previous section can also be used as input for policy-based assessments. For instance, archival institutions may have policies that prohibit e.g. PDFs with encryption or fonts that are not embedded. This can also be done with VeraPDF. This requires that the rules that make up the policy are expressed as a machine-readable Schematron (<https://en.wikipedia.org/wiki/Schematron>) file. As an example, the Schematron file below is made up of two rules that each prohibit specific encryption-related features:

```
<?xml version="1.0"?>

<sch:schema xmlns:sch="http://purl.oclc.org/dsdl/schematron" queryBinding="xslt">
  <sch:pattern name="Disallow encrypt in trailer dictionary">
    <sch:rule context="/report/jobs/job/featuresReport/documentSecurity">
      <sch:assert test="not(encryptMetadata = 'true')">Encrypt in trailer dictionary</sch:assert>
    </sch:rule>
  </sch:pattern>

  <sch:pattern name="Disallow other forms of encryption (e.g. open password)">
    <sch:rule context="/report/jobs/job/taskResult/exceptionMessage">
      <sch:assert test="not(contains(., 'encrypted'))">Encrypted document</sch:assert>
    </sch:rule>
  </sch:pattern>

</sch:schema>
```

A PDF can subsequently be tested against these rules (here in the file “policy.sch”) using the following basic command-line:

```
verapdf --extract --policyfile policy.sch whatever.pdf
> whatever.xml
```

The outcome of the policy-based assessment can be found in the output file’s *policyReport* element. In the example below, the PDF did not meet one of the rules:

```
<policyReport passedChecks="0" failedChecks="1" xmlns:vera="http://www.verapdf.org/MachineReadableReport">
  <passedChecks/>
  <failedChecks>
    <check status="failed" test="not(encryptMetadata = 'true')" location="/report/jobs/job/featuresReport/documentSecurity">
      <message>Encrypt in trailer dictionary</message>
    </check>
  </failedChecks>
</policyReport>
```

More examples can be found in my 2017 post [Policy-based assessment with VeraPDF - a first impression \(/2017/06/01/policy-based-assessment-with-verapdf-a-first-impression\)](#).

Text extraction

Text extraction from PDF documents is notoriously hard. This post (<https://filingdb.com/b/pdf-text-extraction>) gives a good overview of the main pitfalls. Tim Allison's excellent Brief Overview of the Portable Document Format (PDF) and Some Challenges for Text Extraction (<https://irsg.bcs.org/informer/wp-content/uploads/OverviewOfTextExtractionFromPDFs.pdf>) provides a more in-depth discussion, and this really is a must-read for anyone seriously interested in this subject. With that said, quite a few tools are available, and below I list a few that are useful starting points.

Extract text with pdftotext

The *pdftotext* tool that is part of Poppler and Xpdf is a good starting point. The basic command-line is:

```
pdftotext whatever.pdf whatever.txt
```

The tool has lots of options to fine-tune the default behaviour, so make sure to check those out if you're looking for. Note that the available options vary somewhat between the Poppler and Xpdf versions. The documentation of the Poppler version is available here (<https://manpages.debian.org/stretch/poppler-utils/pdftotext.1.en.html>), and here is the Xpdf version (<https://www.xpdfreader.com/pdftotext-man.html>).

Extract text with PDFBox

PDFBox is also a good choice for text extraction. Here's an example command (you may need to adapt the path to the JAR file and its name according to the location and version on your system):

```
java -jar ~/pdfbox/pdfbox-app-2.0.24.jar ExtractText w  
hatever.pdf whatever.txt
```

PDFBox also provides various options, which are documented here (<https://pdfbox.apache.org/1.8/commandline.html#extracttext>).

Extract text with Apache Tika

I already mentioned Apache Tika

(<https://tika.apache.org/>) in the metadata extraction section. Tika is also a powerful text extraction tool, and it is particularly useful for situations where text extraction from multiple input formats is needed.

For PDF it uses the PDF parser of PDFBox (see previous section). By default, Tika extracts both text and metadata, and reports both in XHTML format. If needed, you can change this behaviour with the

`--text` option:

```
java -jar ~/tika/tika-app-2.1.0.jar --text whatever.pdf  
f > whatever.txt
```

Again, an explanation of all available options is available here (<https://tika.apache.org/2.1.0/gettingstarted.html>) (section “Using Tika as a command line utility”).

Batch processing with Tika

The above single-file command does not scale well for situations that require the processing of large volumes of PDFs⁵. In such cases, it’s better to run Tika in batch mode. As an example, the command below will process all files in directory “myPDFs”, and store the results in output directory “tika-out”⁶:

```
java -jar ~/tika/tika-app-2.1.0.jar --text -i ./myPDFs  
/ -o ./tika-out/
```

Alternatively, you could use TikaServer. A runnable JAR is available here (<https://tika.apache.org/download.html>). To use it, first start the server

using:

```
java -jar ~/tika/tika-server-standard-2.1.0.jar
```

Once the server is running, use cURL (<https://en.wikipedia.org/wiki/CURL>) (from another terminal window) to submit text extraction requests:

```
curl -T whatever.pdf http://localhost:9998/tika --header "Accept: text/plain" > whatever.txt
```

The full TikaServer documentation is available here (<https://cwiki.apache.org/confluence/display/TIKA/TikaServer>).

Yet another option is Tika-python (<https://github.com/chris-mattmann/tika-python>), which is a Python port of Tika that uses TikaServer under the hood (resulting in similar performance).

Link extraction

When extracting (hyper)links, it's important to make a distinction between the following two cases:

1. Links that are encoded as a “link annotation”, which is a data structure in PDF that results in a clickable link
2. Non-clickable links/URLs that are just part of the body text.

The automated extraction of the first case is

straightforward, while the second case depends on some kind of lexical analysis of the body text (typically based on regular expressions). For most practical applications the extraction of both types is desired.

Extract links with pdfx

The `pdfox` (<https://www.metachris.com/pdfox/>) tool is designed to detect and extract external references, including URLs. Its URL detection uses lexical analysis, and is based on RegEx patterns written by John Gruber (<https://gist.github.com/gruber/8891611>). The basic command line for URL extraction is:

```
pdfox -v whatever.pdf > whatever.txt
```

I did some limited testing with this tool in 2016. One issue I ran into is that `pdfox` truncates URLs that span more than one line (<https://github.com/metachris/pdfox/issues/21>). As of 2021, this issue hasn't been fixed so far, which seriously limits the usefulness of this (otherwise very interesting) tool. It's worth mentioning that *pdfox* also provides functionality to automatically download all referenced PDFs from any PDF document. I haven't tested this myself.

Other link extraction tools

- Some years ago Ross Spencer wrote a link

extraction tool that uses Apache Tika
(<https://github.com/httppreserve/tikalinkextract>).
There's more info in this blog post
(<https://openpreservation.org/blogs/hyperlinks-in-your-files-how-to-get-them-out-using-tikalinkextract/>).

- Around the same time I wrote this simple extraction script (<https://gist.github.com/bitsgalore/aab680a9bccfc5496948b776ee06397c>) that wraps around Apache Tika and the xurl (<https://github.com/mvdan/xurls>) tool. I used this to extract URLs from MS Word documents, but this should probably work for PDF too (I haven't tested this though!).

Image extraction with *pdfimages*

PDFs often contain embedded images, which can be extracted with *pdfimages* tool that is part of Xpdf/Poppler. At minimum, it takes as its arguments the name of the input PDF document, and the “image-root” which is actually just a text prefix that is used to generate the name of the output images. By default it writes its output to one of the Netpbm (<https://en.wikipedia.org/wiki/Netpbm>) file formats, but for convenience you might want to use the `-png` option, which uses the PNG format instead:

```
pdfimages -png whatever.pdf whatever
```

Output images are now written as “whatever-000.png”, “whatever-001.png”, “whatever-002.png”, and so on. The `-j`, `-jp2`, `-jbig2` and `-ccitt` switches can be used to store JPEG, JPEG2000, JBIG2 and CCITT images in their native formats, respectively (or use `-all`, which combines all of these options).

Conversion to other (graphics) formats with pdftocairo

The *pdftocairo* tool (Xpdf/Poppler) can convert a PDF to a number of (mostly graphics) formats. The supported output formats are PNG, JPEG, TIFF, PostScript, Encapsulated PostScript, Scalable Vector Graphics and PDF. As an example, the following command will convert each page to a PNG image:

```
pdftocairo -png whatever.pdf
```

List embedded image information with pdfimages

The *pdfimages* tool is also useful for getting an overview of all embedded images in a PDF, and their main characteristics (width, height, colour, encoding, resolution and size). just user the `-list` option as shown below:

```
pdfimages -list whatever.pdf
```

This results in a nice table like this:

```

page   num  type   width height color  comp bpc  enc in
terp
-----
-----
    1    0 image   1830  2658  gray    1    1  jbig2
no
    1    1 image    600   773  gray    1    8   jpx
no

page   object ID x-ppi y-ppi size ratio
-----
    1     16  0   301   301   99B 0.0%
    1     17  0   300   300  17.9K 4.0%
```

Conversion of multiple image files to PDF

Losslessly convert raster images to pdf with img2pdf

The `img2pdf` (<https://gitlab.mister-muffin.de/josch/img2pdf>) tool converts a list of image files to PDF. Unlike several other tools (such as ImageMagick), it does not re-encode the source images, but simply embeds them as PDF objects in their original formats. This means that the conversion is always lossless. The following example shows how to convert three JP2 (JPEG 200 Part 1) (<http://fileformats.archiveteam.org/wiki/JP2>) images:

```
img2pdf image1.jp2 image2.jp2 image3.jp2 -o whatever.pdf
```

In the resulting PDF, each image is embedded as an image stream with the JPXDecode (JPEG 2000) filter.

PDF comparison with Comparepdf

The Comparepdf (<http://www.qtrac.eu/>)⁷ tool compares pairs of PDFs, based on either text or visual appearance. By default it uses the program exit code to store the result of the comparison. The tool's command-line help text explains the possible outcomes:

“ A return value of 0 means no differences detected; 1 or 2 signifies an error; 10 means they differ visually, 13 means they differ textually, and 15 means they have different page counts

For clarity I used the `-v` switch in the examples below, which activates verbose output. To test if two PDFs contain the same text, use:

```
comparepdf -ct -v=2 whatever.pdf wherever.pdf
```

If all goes well the output is either “No differences detected” or “Files have different texts”.

To compare the visual appearance of two PDFs, use:

```
comparepdf -ca -v=2 whatever.pdf wherever.pdf
```

In this case the output either shows “No differences detected” or “Files look different”.

Repair a corrupted PDF

Sometimes it is possible to recover the contents of corrupted or otherwise damaged PDF documents. This thread on Super User (<https://superuser.com/questions/278562/how-can-i-fix-repair-a-corrupted-pdf-file>) mentions two useful options.

Repair with Ghostscript

```
gs -o whatever_repaired.pdf -sDEVICE=pdfwrite -dPDFSETTINGS=/prepress whatever_corrupted.pdf
```

Repair with pdftocairo

A second option mentioned in the Super User thread is *pdftocairo*, which is part of Xpdf and Poppler:

```
pdftocairo -pdf whatever_corrupted.pdf whatever_repaired.pdf
```

It's worth adding here that the success of any repair action largely depends on the nature and extent of

the damage/corruption, so your mileage may vary. Always make sure to carefully check the result, and keep a copy of the original file.

Repair with Pdftk

Finally, *pdftk* can, according to its documentation (<https://www.pdflabs.com/docs/pdftk-cli-examples/>), “repair a PDF’s corrupted XREF table and stream lengths, if possible”. This uses the following command line:

```
pdftk whatever_corrupted.pdf output whatever_repaired.pdf
```

Reduce size of PDF with hi-res images with Ghostscript

The following Ghostscript command (source here (<https://askubuntu.com/questions/113544/how-can-i-reduce-the-file-size-of-a-scanned-pdf-file/256449#256449>)) can be useful to reduce the size of a large PDF with high-resolution graphics (note that this will result in quality loss):

```
gs -sDEVICE=pdfwrite \  
-dCompatibilityLevel=1.4 \  
-dPDFSETTINGS=/ebook \  
-dNOPAUSE -dQUIET -dBATCH \  
-sOutputFile=whatever_small.pdf whatever_large.pdf
```

Reduce size of PDF with hi-res

images with ImageMagick

As an alternative to the above Ghostscript command (which achieves a size reduction mainly by downsampling the images in the PDF to as lower resolution), you can also use ImageMagick (<https://imagemagick.org/>)'s *convert* tool (<https://imagemagick.org/script/convert.php>). This allows you to reduce the file size by changing any combination of resolution (`-density` (<https://imagemagick.org/script/command-line-options.php#density>) option), compression type (`-compress` (<https://imagemagick.org/script/command-line-options.php#compress>) option) and compression quality (`-quality` (<https://imagemagick.org/script/command-line-options.php#quality>) option).

For example, the command below (source here (<https://askubuntu.com/questions/113544/how-can-i-reduce-the-file-size-of-a-scanned-pdf-file/469255#469255>)) reduces the size of a source PDF by re-encoding all images as JPEGs with 70% quality at 300 ppi resolution:

```
convert -density 300 \  
        -compress jpeg \  
        -quality 70 \  
        whatever_large.pdf whatever_small.pdf
```

If the `-density` value is omitted, *convert* resamples all images to 72 ppi by default. If you don't want that, make sure to set the `-density` value to the resolution

of your source PDF (see the section “List embedded image information with pdftimages” on how to do that).

Even though ImageMagick’s *convert* tool uses Ghostscript under the hood, it doesn’t preserve any text (and probably most other features) of the source PDF, so only use this if you’re only interested in the image data!

Inspect low-level PDF structure

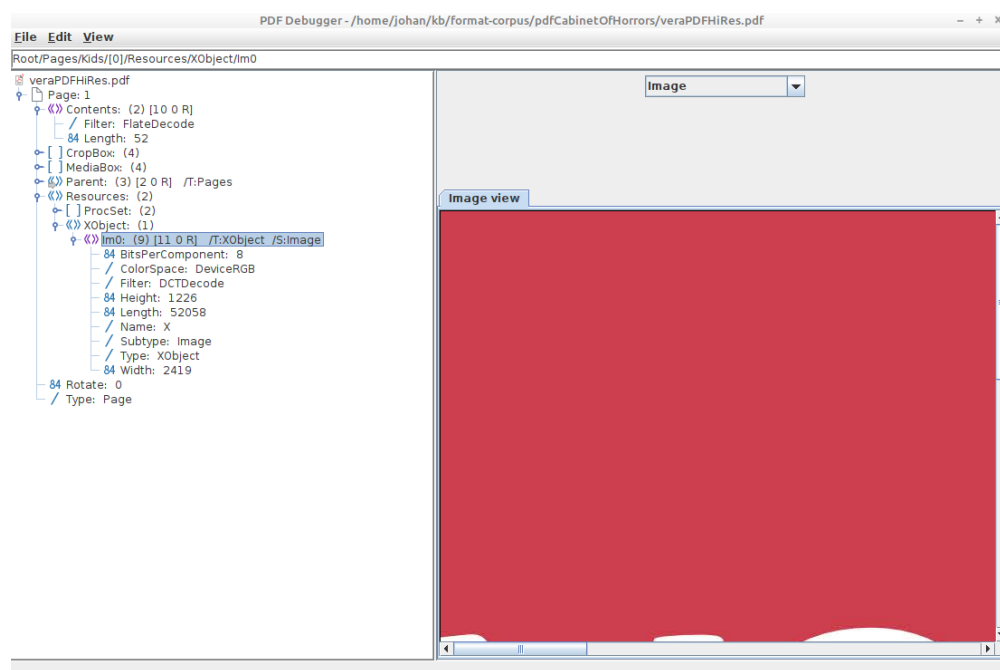
The following tools are useful for inspecting and browsing the internal (low-level object) structure of PDF files.

Inspect with PDFBox PDFDebugger

PDFBox includes a “PDF Debugger”, which you can start with the following command:

```
java -jar ~/pdfbox/pdfbox-app-2.0.24.jar PDFDebugger w  
hatever.pdf
```

Subsequently a GUI window pops up that allows you to browse the PDF’s internal objects:



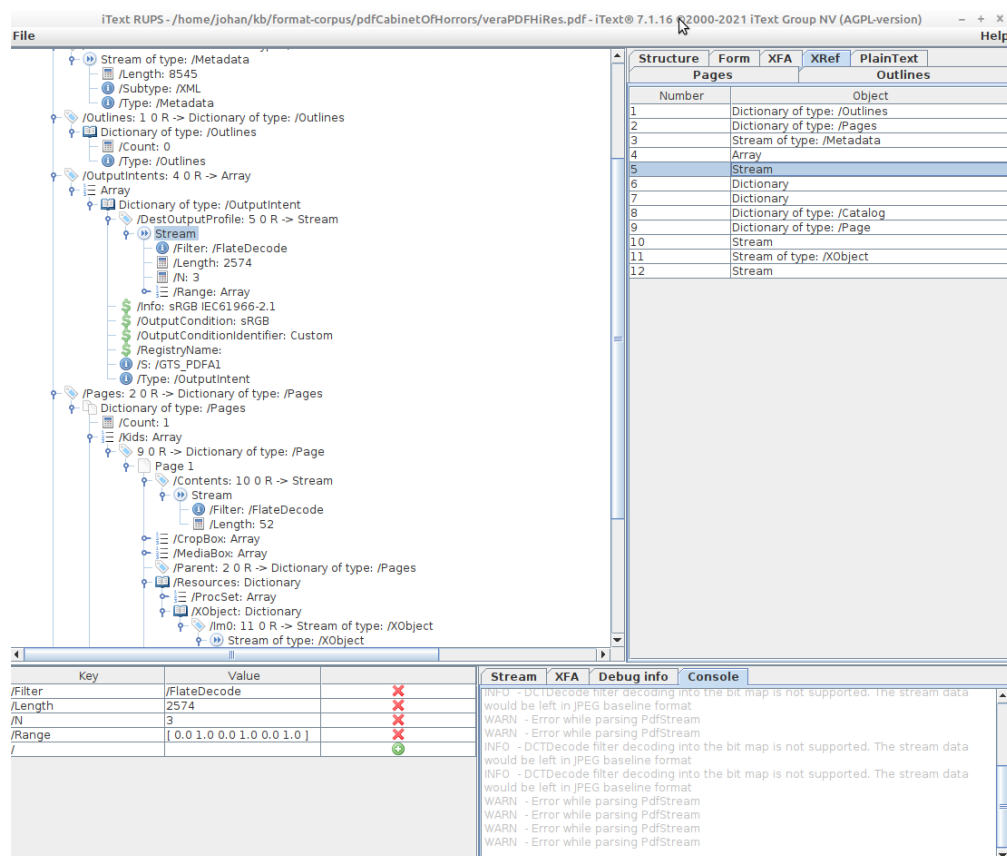
Screenshot of PDFBOX PDFDebugger.

Inspect with iText RUPS

The itext RUPS (<https://github.com/itext/i7j-rups>) viewer provides similar functionality to PDF Debugger. You can download a self-contained runnable JAR here (<https://github.com/itext/i7j-rups/releases/latest>) (select the “only-jars” ZIP file). Run it using:

```
java -jar ~/itext-rups/itext-rups-7.1.16.jar
```

Then open a PDF from the GUI, and browse your way through its internal structure:



Screenshot of *iText RUPS*.

View, search and extract PDF objects with mutool show

Mutool's *show* command allows you to print user-defined low-level PDF objects to stdout. A couple of things you can do with this:

- Print the document trailer:

```
mutool show whatever.pdf trailer
```

Result:

```
trailer
<<
/DecodeParms <<
  /Columns 3
  /Predictor 12
>>
/Filter /FlateDecode
/ID [ <500AB94E8F45C149808B2EEE98528B78> <431017E49
5216040A953126BB73D0CD4> ]
/Index [ 11 10 ]
/Info 10 0 R
/Length 47
/Prev 24426
/Root 12 0 R
/Size 21
/Type /XRef
/W [ 1 2 0 ]
>>
```

- Print the cross-reference table:

```
mutool show whatever.pdf xref
```

Result:

```
xref
0 21
00000: 0000000000 00000 f
00001: 0000019994 00000 n
00002: 0000020399 00000 n
00003: 0000020534 00000 n
::
etc
```

- Print an indirect object by its number:

```
mutool show whatever.pdf 12
```

Result:

```
12 0 obj
<<
/Metadata 4 0 R
/Pages 9 0 R
/Type /Catalog
>>
endobj
```

- Extract only stream contents as raw binary data and write to a new file:

```
mutool show -b whatever.pdf 151 > whatever.dat
```

This command is particularly useful for extracting the raw data from a stream object (e.g. an image or multimedia file).

More advanced queries are possible as well. For example, the mutool manual (<https://mupdf.com/docs/manual-mutool-show.html>) gives the following example, which shows all JPEG compressed stream objects in a file:

```
mutool show whatever.pdf grep | grep '/Filter/DCTDecod
e'
```

Result:

```
1 0 obj <</BitsPerComponent 8/ColorSpace/DeviceRGB/Fil
ter/DCTDecode/Height 516/Length 76403/Subtype/Image/Ty
pe/XObject/Width 1226>> stream
18 0 obj <</BitsPerComponent 8/ColorSpace/DeviceRGB/Fi
lter/DCTDecode/Height 676/Length 149186/Subtype/Image/
Type/XObject/Width 1014>> stream
19 0 obj <</BitsPerComponent 8/ColorSpace/DeviceRGB/Fi
lter/DCTDecode/Height 676/Length 142232/Subtype/Image/
Type/XObject/Width 1014>> stream
24 0 obj <</BitsPerComponent 8/ColorSpace/DeviceRGB/Fi
lter/DCTDecode/Height 676/Length 192073/Subtype/Image/
Type/XObject/Width 1014>> stream
25 0 obj <</BitsPerComponent 8/ColorSpace/DeviceRGB/Fi
lter/DCTDecode/Height 676/Length 141081/Subtype/Image/
Type/XObject/Width 1014>> stream
```

Final remarks

I intend to make this post a “living” document, and will add more PDF “recipes” over time. Feel free to leave a comment in case you spot any errors or omissions!

Update on Hacker News topic

Someone created a Hacker News topic on this post (<https://news.ycombinator.com/item?id=33145498>). The comments mention some additional tool suggestions that look useful. I might add some of these to a future revision.

Further resources

- Moritz Mähr, “Working with batches of PDF files”, The Programming Historian 9 (2020)

- (<https://doi.org/10.46430/phen0088>)
- PDF tools in Community Owned Digital Preservation Tool Registry (COPTR)
(<https://coptr.digipres.org/index.php/PDF>)
- Policy-based assessment with VeraPDF - a first impression (/2017/06/01/policy-based-assessment-with-verapdf-a-first-impression)
- What's so hard about PDF text extraction?
(<https://filingdb.com/b/pdf-text-extraction>)
- Tim Allison, "Brief Overview of the Portable Document Format (PDF) and Some Challenges for Text Extraction" (<https://irsg.bcs.org/informer/wp-content/uploads/OverviewOfTextExtractionFromPDFs.pdf>)
- Yvonne Tunnat, "PDF Validation with ExifTool – quick and not so dirty"
(<https://openpreservation.org/blogs/pdf-validation-with-exiftool-quick-and-not-so-dirty/>)
- Micky Lindlar, "Trouble-shooting PDF validation errors – a case of PDF-HUL-38"
(<https://openpreservation.org/blogs/trouble-shooting-pdf-validation-errors-a-case-of-pdf-hul-38/>)
- Hacker News topic on this post
(<https://news.ycombinator.com/item?id=33145498>)

Revision history

- 7 September 2021: added sections on metadata extraction and Tika batch processing, following suggestions by Tim Allison.

- 8 September 2021: added section on inspecting low-level PDF structure with iText RUPS, as suggested by Mark Stephens; added sections on PDFtk as suggested by Tyler Thorsted; corrected errors in *pdftocairo* and *gs* examples.
- 9 September 2021: added section on image to PDF conversion.
- 27 January 2022: added reference to Tim Allison's article on PDF text extraction.
- 7 February 2022: added sections on Exiftool, and added reference to Yvonne Tunnat's blog post on PDF validation with ExifTool.
- 10 October 2022: added update on and link to Hacker News topic on this post.
- 28 November 2022: added reference to Micky Lindlar's blog post on trouble-shooting PDF validation errors.
- 16 February 2023: added section on reducing PDF file size with ImageMagick's *convert* tool.

1. The Debian package of the “original” PDFtk software was removed from the Ubuntu repositories (<https://www.joho.se/2020/10/01/pdftk-and-php-pdftk-on-ubuntu-18-04-without-using-snap/>) around 2018 due to “dependency issues”. ↩

2. Command line: `pdftinfo whatever.pdf` ↩

3. In this example output is redirected to a file; this is generally a good idea because of the

amount of XML output generated by
VeraPDF. ↩

4. The `--off` switch disables PDF/A validation.

Output is redirected to a file (recommended because, depending on the configuration used, VeraPDF can generate a *lot* of output). ↩

5. This is because a new Java VM is started for each processed PDF, which will result in poor performance. ↩

6. Of course this also works for metadata extraction, and both text and metadata extraction can be combined in one single command. As an example, the following command will extract both text and metadata, including any embedded documents:

```
java -jar ~/tika/tika-app-2.1.0.jar -J --text -i ./myPDFs/ -o ./tika-out/ ↩
```

7. On Debian-based systems you can install it
using `sudo apt install comparepdf`. ↩

- Apache-Tika
- JHOVE
- PDF
- preservation-risks
- VeraPDF

← Previous (/2021/02/24/towards-a-preservation-workflow-for-mobile-apps)

Next → (/2021/09/24/on-the-significant-properties-of-spreadsheets)

Comments



([https://github.com](https://github.com/markee174)

/markee174)markee174 ([https://github.com](https://github.com/markee174)
/markee174) wrote:

We are big fans of Rups ([https://github.com](https://github.com/itext/i7j-rups)
/itext/i7j-rups) for looking at the structure of
PDf files

2021-09-08T08:50:25Z ([https://github.com](https://github.com/bitsgalore/bitsgalore.github.io/issues/76#issuecomment-915044600)
/bitsgalore/bitsgalore.github.io/issues
/76#issuecomment-915044600)



([https://github.com](https://github.com/bitsgalore)

/bitsgalore)bitsgalore ([https://github.com](https://github.com/bitsgalore)
/bitsgalore) wrote:

@markee174 I just added a section on RUPS,
thanks for the suggestion!

2021-09-08T15:29:06Z ([https://github.com](https://github.com/bitsgalore/bitsgalore.github.io/issues/76#issuecomment-915341461)
/bitsgalore/bitsgalore.github.io/issues
/76#issuecomment-915341461)



([https://github.com](https://github.com/markee174)

/gettalong)gettalong (<https://github.com/gettalong>) wrote:

The HexaPDF cli utility (<https://hexapdf.gettalong.org/documentation/reference/hexapdf.1.html>) falls into the same category as qpdf, pdftk and the like.

2022-10-10T15:17:15Z (<https://github.com/bitsgalore/bitsgalore.github.io/issues/76#issuecomment-1273470872>)



(<https://github.com/gollux>)gollux

(<https://github.com/gollux>) wrote:

Some years ago, I wrote paperjam (<https://mj.ucw.cz/sw/paperjam/>), which can rearrange pages within a PDF, make booklets, do n-up printing, crop pages, and many other operations. It is based on libqpdf.

2022-10-10T15:42:02Z (<https://github.com/bitsgalore/bitsgalore.github.io/issues/76#issuecomment-1273504056>)



(<https://github.com/jsnmrs>)jsnmrs

(<https://github.com/jsnmrs>) wrote:

I built PDFcheck (<https://jsnmrs.github.io>)

/pdfcheck/) as a fast, local gut check on PDF accessibility considerations.

Drag and drop any number of PDFs onto the page for a client-side (local) read and report on PDF metadata.

2022-10-12T01:23:40Z (<https://github.com/bitsgalore/bitsgalore.github.io/issues/76#issuecomment-1275464703>)



([https://github.com](https://github.com/ItsIgnacioPortal)

/ItsIgnacioPortal)ItsIgnacioPortal

(<https://github.com/ItsIgnacioPortal>) wrote:

@bitsgalore you might be interested in adding 5f0ne/pdf-examiner (<https://github.com/5f0ne/pdf-examiner>): It provides an overview of the inner file structure of a PDF.

2022-11-18T00:36:29Z (<https://github.com/bitsgalore/bitsgalore.github.io/issues/76#issuecomment-1319398252>)

Post comment (Github) (<https://github.com/bitsgalore/bitsgalore.github.io/issues/76>)



(<https://www.bitsgalore.org/about.html>)

About

(<https://www.bitsgalore.org/about.html>)

Search



Tags

- ▶ Android
- ▶ Apache-Preflight
- ▶ Apache-Tika
- ▶ APK
- ▶ Debian
- ▶ digital-dark-age
- ▶ digital-preservatic day
- ▶ disk-imaging
- ▶ diskimgr
- ▶ DROID
- ▶ e-depot

- ▶ emulation
- ▶ EPUB
- ▶ EPUBCheck
- ▶ Fido
- ▶ FITS
- ▶ FLAC
- ▶ floppy-disks
- ▶ format-identification
- ▶ geodata
- ▶ GW-BASIC
- ▶ HFS
- ▶ High-Sierra
- ▶ iOS
- ▶ IPA
- ▶ iromlab
- ▶ ISO-9660
- ▶ isolyzer
- ▶ JHOVE
- ▶ JHOVE2
- ▶ JP2
- ▶ jpeg-2000
- ▶ jpylyzer
- ▶ magic
- ▶ Microsoft
- ▶ omimgr
- ▶ OneDrive
- ▶ optical-media
- ▶ packaging
- ▶ PDF

- ▶ preservat
risks
- ▶ python
- ▶ Quattro-
Pro
- ▶ rant
- ▶ schematr
- ▶ Siegfried
- ▶ significan
properties
- ▶ spreadsh
- ▶ tapeimgr
- ▶ tapes
- ▶ TIFF
- ▶ Twitter
- ▶ UDF
- ▶ unix-
file
- ▶ VeraPDF
- ▶ virtualiza
- ▶ WAVE
- ▶ web-
archaeolog
- ▶ web-
archiving
- ▶ XS4ALL
- ▶ ZIP

Archive

▼ 2023

January

Writing
yet
another

workflow
tool for
imaging
portable
media
(/2023
/01/23
/writing-
a-workflow-
tool-
for-
imaging-
portable-
media)

- ▶ 2022
 - ▶ 2021
 - ▶ 2020
 - ▶ 2019
 - ▶ 2018
 - ▶ 2017
 - ▶ 2016
 - ▶ 2015
 - ▶ 2014
 - ▶ 2013
 - ▶ 2012
 - ▶ 2011
 - ▶ 2010
-

Issues

Report a
problem
with this
site
([https://github.com
/bitsgalore
/bitsgalore.github.io](https://github.com/bitsgalore/bitsgalore.github.io))

/issues)

Hackers

Hall of

Fame

(<https://www.bitsgalore.org>

/hackers-

hall-of-

fame.html)

Social

Mastodon

(digipres.club)

(<https://digipres.club>

/@bitsgalore)

Twitter

(<https://twitter.com>

/bitsgalore)

Feeds

RSS

(<https://www.bitsgalore.org>

/rss.xml)

ATOM

(<https://www.bitsgalore.org>

/atom.xml)

© 2023 Johan van der Knijff. All content on this blog is licensed under a Creative Commons Attribution 4.0 International License (<http://creativecommons.org/licenses/by/4.0/>), unless indicated otherwise. Created using Jekyll Bootstrap (<http://jekyllbootstrap.com>) and Twitter Bootstrap (<https://getbootstrap.com/>).