filedriller

Marrying Siegfried and the National Software Reference Library

Steffen Fritz
German Literature Archive
Germany
steffen.fritz@dla-marbach.de
https://orcid.org/0000-0002-9853-8903

Abstract – Working with huge collections of unstructured data is a common yet still challenging task in digital preservation. This paper presents a tool for finding irrelevant files in large data sets to spot the relevant. The tool builds on two well-known and frequently used applications, respectively data sets and combines and extents them in a meaningful way.

Keywords – format identification, tool, automation, siegfried, NSRL

Conference Topics – Scanning the New Development

I. Introduction

In addition to more than 1.5 million books, journals, media carriers, publishing archives and objects, the German Literature Archive Marbach is home to more than 1400 independent individual archives. While the vast majority of the latter are analog, the number of digital estates is constantly increasing.

In addition to manuscripts and libraries, the estates of writers, scientists and artists include more and more personal computers, hard drives and external data carriers, containing highly diverse file types. In some rare cases relatives or colleagues of the testator can support the archivists and librarians during the structuring and rough assessment of data. But even in these cases, the vast amount of information, stored and created during a lifetime, must be prepared. Where possible, the source objects to be sifted must be reduced.

Which data are relevant? And how can relevance be defined in this context? Files that are common in off-the-shelf products like software libraries, wallpapers or operating system files are mostly less interesting in terms of studying the author's works. These assets are probably not important for the geneses from a nontechnical view. Of course, this means not that such files should be neglected in terms of long-term digital archiving. In this paper a tool is presented that supports an automatic preassessment of large amounts of data. It returns a list of files that might be relevant for further investigations.

The tool is named filedriller and is under development at the German Literature Archive in Marbach. It is built on top of siegfried[1] and the National Software Reference Library (NSRL)[2].

A. Siegfried

Siegfried is a file format identification tool using the PRONOM, freedesktop.org's MIME-info and Library of Congress's format description documents as well as tika-mimetype signatures[1]. The signatures are stored in a signature file, used by siegfried for pattern matching.

The development started in 2014 following the release history on Github. Siegfried is written in golang and is published under the Apache license 2.0. It is a command line tool and available on all major hardware platforms and operating systems[2]. The executable is named *sf*. Siegfried works on files and can walk down directories recursively. An example clarifies the usage and shows the output in the default format to standard out:

```
$ ~/Documents> sf filedriller.png
---
siegfried   : 1.9.1
scandate    : 2021-05-27T14:53:08+02:00
signature   : default.sig
created     : 2020-10-06T19:13:40+02:00
identifiers :
  - name    : 'pronom'
    details : 'DROID_SignatureFile_V97.xml;
              container-signature-20201001.xml'
---
filename : 'filedriller.png'
filesize : 21600
modified : 2021-05-27T00:04:37+02:00
errors   :
matches  :
  - ns      : 'pronom'
    id      : 'fmt/12'
    format  : 'Portable Network Graphics'
    version : '1.1'
```

[1] https://www.itforarchivists.com/categories/siegfried
[2] https://www.nist.gov/itl/ssd/software-quality-group/national-software-reference-library-nsrl

17th International Conference on Digital Preservation
iPRES 2021, Beijing, China.

DOI: 10.1145/nnnnnnn.nnnnnnn

```
    mime    : 'image/png'
    basis   : 'extension match png;
              byte match at [[0 16] [37 4]
              [21588 12]] (signature 3/3)'
    warning :
---
```

Siegfried is not only a command line tool but also a library that can be used in other golang projects. While not all functions are exported, i.e. importable, the format identification capabilities are available.

B. *National Software Reference Library*

The NSRL is a huge software corpus that consists of more than 140 millions of files from commercial products and software packages[3]. The collection is split in legacy and modern sets. The modern set contains applications created in or after 2010 while the legacy set contains applications created in or before 2009. The NSRL also provides databases that hold "metadata about the files that make up those software packages". These metadata sets are called NSRL Reference Data Sets (RDS) and are available via a free download[4].

The metadata are stored in csv files and can easily be parsed. In the files are file names, associated MD5 and SHA-1 hash sums, file sizes, vendors, supported operating systems and more information stored. The useful property is the availability of hash sums of off-the-shelf products.

The decision if a file is relevant is based on the thesis that if the hash sum of a file is in the NSRL, it is probably not interesting for a discussion of the contents. Files not in the NSRL are therefore of interest.

II. Filedriller and Redis

A. *Redis*

Filedriller is written in golang and uses siegfried's format identification methods and the signature file, compiled and provided by its authors. It also has to have the information if the hash sum of a file is in the NSRL. While just the hash sums are needed for this task, the csv file is still 1 GB in size. Also a file does not scale for a lot of requests in a short time. Therefore the SHA-1 hash sums were parsed from the RDS and fed to a redis docker container as keys while the value for each key is *TRUE*. The container formed the basis for a docker image that can be pulled from Docker Hub[3].

A bash script to create an import with current data is available on Codeberg[4]. The script can be used to create a redis NSRL instance serving SHA-1 hashes, even without the need for docker. The script fetches the modern RDS set, extracts all SHA-1 sums with awk, builds a redis protocol file and imports it using *redis-cli*. A sample of the protocol file can be seen in the following excerpt, all hash sums are abbreviated:

```
SET 0000001FFEF4BE31...B53AEAA3E4684D85 TRUE
SET 00000079FD7AAC9B...50750E1F50B27EB5 TRUE
SET 000000F694CA836D...EB5E2724338B422D TRUE
SET 000001169CF30652...459D9E167B132C06 TRUE
```

To be able to consume such services, filedriller implements a redis client. Server address and port are configurable via runtime flags. If the redis server flag is omitted filedriller does not check if a file is in the NSRL.

B. *Filedriller*

Filedriller is a command line tool that recursively walks a directory tree and creates a file list. In the next step it iterates over the list and performs for every entry a file identification and creates and gathers additional metadata. These are file sizes, PRONOM UIDs, hash sums and file entropies. It also generates UUID version 4 strings for unambiguous referencing.

A diagram showing the components and their communication paths can be seen in Figure 1.

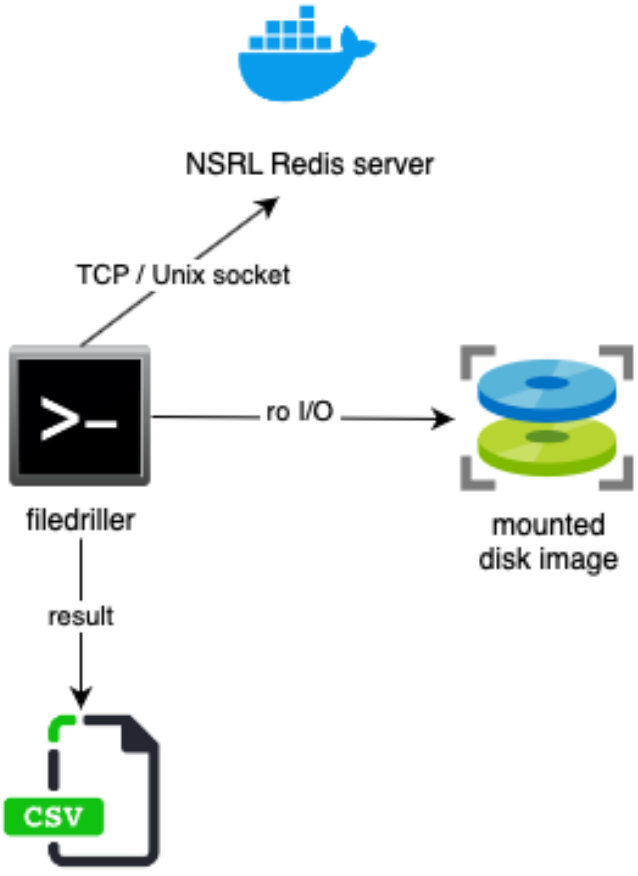

Figure 1: Architecture

[3] https://hub.docker.com/r/ampoffcom/nslredis
[4] https://codeberg.org/steffenfritz/nslredis

C. Installation and example

Filedriller can be downloaded as a precompiled fat binary, it therefore has no external dependencies. There are executables for Linux, macOS and Windows. They can be downloaded from Github[5]. The binary is named *friller*. Friller can also be compiled from source with

```
go get github.com/dla-marbach/filedriller
          /cmd/friller.
```

The compilation is placed in $GOPATH/bin. If this is in your search path you can execute *friller* from any location on the host system.

A common setup consists of a read-only filesystem, mounted to the host system running friller and a local or remote redis instance. On the first run friller checks if a signature file is locally available. It expects the file in the same directory where itself is located. If not found it downloads a mirrored version from Github[6].

A run of filedriller with the mandatory *-i* flag and optional *-s* flag, i.e. the redis server address, looks like the following. Output is written to info.csv, logs.txt and errorlogs.txt in the same directory. All file names are configurable via runtime flags.

```
$ ~/> friller -i /mnt/imagefile -s 10.10.0.10
2021/05/27 22:22:56 info: friller started
2021/05/27 22:22:57 info: Created file list.
   Found 54962 files.
2021/05/27 22:22:57 info: Started file format
   identification
54962 / 54962 [------------] 100.00% 2546 p/s
2021/05/27 22:23:19 info: Inspected 54962 files.
2021/05/27 22:23:19 info: Creating output file
2021/05/27 22:23:19 info: Writing output
    to info.csv
2021/05/27 22:23:19 info: Output written
    to info.csv
2021/05/27 22:23:19 info: Log file written
    to logs.txt
2021/05/27 22:23:19 info: Error log file written
    to errorlogs.txt
2021/05/27 22:23:19 info: friller ended
$ ~/>
```

A complete call may look like this:

```
$ ~/> friller -i /mnt/imagefile
    --redisserv localhost
    --redisport 6378
    --algo sha256
    --entropy
    --output myoutput.csv
    --errlog myerror.log
    --log my.log
```

The csv result can be imported easily and processed by third-party tools. This could be spreadsheet applications, OpenRefine[7] or SQL databases. The schema of the file has the following fields:

```
Filename, SizeInByte, Registry, PUID, Name,
Version, MIME, ByteMatch, IdentificationNote,
SHA256, UUID, inNSRL, Entropy
```

The naming of the tenth column, in the example *SHA256*, depends on the used fixity function.

It is not always desired to write log files or examine multiple files at once. There are also use cases where several files are examined at once, but the results are to be processed immediately and individually. For these cases friller provides the *-f* flag. This writes the result directly to standard out:

```
$ ~/> friller -i testbinary
```

The output for this example file has the format of a single line in a regular output of friller:

```
"testbinary","10751680","pronom","fmt/693",
"Mach-O","64bit","", "byte match at 0, 4
(signature 2/2)","","fc86eb3b3f12b500b...",
"b037aae2-dffa-43ef-8cdf-aba6e3019683",,
```

III. Conclusion and next steps

Filedriller is a simple tool to inspect large quantities of data. It can assist archivists by giving a first assessment to decide which files are worth a more thorough look. It helps to optimize workloads on more sophisticated applications, e.g. tools that are indexing and analyzing content. Next steps are to optimize the code and make filedriller faster by using goroutines where possible. It is also planned to extent its functionality and create a graphical user interface.

References

[1] *Siegfried, Homepage*. [Online]. Available: https://www.itforarchivists.com/siegfried/.

[2] *Siegfried code repository, Github*. [Online]. Available: https://github.com/richardlehane/siegfried/releases.

[3] T. Owens, *The Theory and Craft of Digital Preservation*. Baltimore, MD: Johns Hopkins University Press, 2018, pp. 168–170.

[4] *Nsrl introduction, Webpage*. [Online]. Available: https://www.nist.gov/itl/ssd/software-quality-group/national-software-reference-library-nsrl/about-nsrl/nsrl-introduction.

[5] https://github.com/dla-marbach/filedriller/releases/

[6] https://github.com/dla-marbach/filedriller/tree/main/third_party

[7] https://openrefine.org