Data Integrity

Peter Bubestinger-Steindl
(p.bubestinger@av-rd.com)

December 2020

Data Integrity?

Speaker notes

- What is that?
- How do you identify if a file is *intact*?
- How do you identify duplicates?

What is "Fixity" information?

Speaker notes

Fixity information is metadata that can be used to check/verify that binary data has not (been) changed. This can be used to make sure that files were copied properly from A-to-B, or retrieved bit-exactly the way they were stored, etc.

Fixity information is directly linked to so called "hashcodes".

Hashcodes

raw.txt

> *"This is a raw text file."*

Hashcodes

raw.txt

> *"This is a raw text file."*

```
MD5 = b3a243d2443037a783c8799fe2c4926a
```

Speaker notes

Hashcode = A fixed size number that's like a fingerprint for data.

A "hash" or "hashcode" is the result of a mathematical algorithm that produces something like a fingerprint number for the input data provided. With the intention for any source *not* to map to an identical number. That would be called a "hash collision": 2 different sources mapping to the same hash value/number.

To keep the numbers short(er), they are usually written in hexadecimal (0..9, A..F).

The above example is the hashcode for the string "This is a raw text file."

Hashcodes

raw.txt

"This is a raw text file.□"

Hashcodes

raw.txt

> *"This is a raw text file.□"*

```
MD5 = 7096384353da7d8cb59b1395e63d1250
```

Speaker notes

Even though only a simple space character was added to the string from before, the resulting MD5 hashcode is completely different than before. That's good!

This allows to quickly and securely identify even the smallest deviation in the source data. Even a small change like a single character - or even a binary bit. This way, a mismatching hashcode will tell you if your data is either *exactly* the way it was - or if *anything* has changed.

Hashcodes

raw.txt

> *"this is a raw text file."*

Hashcodes

raw.txt

> *"this is a raw text file."*

```
MD5 = a94a15d1b72bbfee7997bf237cf0347e
```

Speaker notes

Now, the case of the first letter "T" was changed to "t": Different character = different hashcode. Again: Good! :)

Hashcodes

raw-text.txt

> *"this is a raw text file."*

Hashcodes

raw-text.txt

> *"this is a raw text file."*

```
MD5 = a94a15d1b72bbfee7997bf237cf0347e
```

Speaker notes

Now, the filename - *not* the content - was changed. This has no effect on the hashcode, as the hash only depicts the content. The filename is outside, on the filesystem level. The hashcode does not include the name of a file.

Different algorithms

- CRC
- MD5
- SHA .. 1 .. 2 .. 256 .. SHA512
- XXHASH
- WTF?

Speaker notes

There are different hashing algorithms. In a nutshell:

- Shorter hash = faster
  but higher collision chance
  = less secure

- Longer hash = slower
  but lower collision chance
  = more secure

For data integrity verification, short hashes are perfectly fine.

Since hashing algorithms are also used for security purposes (digital signatures), MD5 was said to be "broken". This is only true for security/signature purposes. It is still perfectly valid for checking data integrity.

"xxHash" is relatively new, and is the only "Non-cryptographic hash function" in the above list, designed for speed and not security. It is becoming more common for fixity checks in A/V production, but yet it's still rather the exception.

Hashcodes are fixed-length numbers that are generated in a way that if even a single bit in the source data changes, that number will be completely different. They are often written in hexadecimal, therefore including the characters 0-9 and A-F.

If you have a hashcode for a set of data, it can be used to verify the bit-exact integrity of that data, by calculating the same-algorithm hashcode again and comparing them. If they're identical, the data is intact. If two distinct sets of data have the same hash, it's called a "hash collision. Hash algorithms are designed particularily to keep the chance for collision as low as possible.

Anyways: hashcodes are *a must* for safe data transfer and integrity checks.

In case someone has heard that MD5 is broken, fear not: For plain checking of file transfer or stored data integrity, MD5 is sufficient. It was cryptographically "broken" - which is relevant for security, but not for data integrity checking.

Important: Hashcodes are not sufficient for proving authenticity! In case you need to deal with important originals/documents and you need to make sure they're originals and not forged, etc. please check out this:

- Digital Signatures
- Blockchain

Digital signatures are good, but could be signed with a date in the past (backdating).

If this is a concern, you may consider blockchain mechanisms: Blockchain cipher became popular for digital currency (like Bitcoin), but it can also be used to proof data authenticity and avoid backdating.

AFAIK there are currently no systems that implement this productively yet, but some have already researched into prototyping blockchain use for storage.

Hashcode Examples

- CRC =
  `4294967295`
- MD5 =
  `d41d8cd98f00b204e9800998ecf8427e`
- SHA256 =
  `e3b0c44298fc1c149afbf4c8996fb92427ae41e4649b934ca495991b7852b85`
- xxHash =
  `e4c191d091bd8853`

Speaker notes

Algorithms in order of complexity/size: > CRC ------ MD5 ------ SHA256 ------ SHA512

Speed matters when it comes to calculating hashes for several hundreds of TB or PB of data.

Execution speed depends on the actual implementation of the algorithm. Even if a simpler algorithm may be faster in theory, it may not make a difference if the implementation isn't speed-optimized. However, speeding up hashing becomes more and more interesting e.g. for transfer of digital cinema files, because validating these data amounts may currently be a bottleneck.

Different hashcodes algorithms/implementations may have significantly different runtimes. When dealing with large quantities of data, this may matter.

MD5 is the most popular one around: Well known and widely supported by different applications/systems, etc.

Anyone transferring lots of uncompressed film? You may want to look at xxHash. It's designed for speed.

When?

> *Generate fixity information as early as possible in a file's lifecycle.*

Different levels

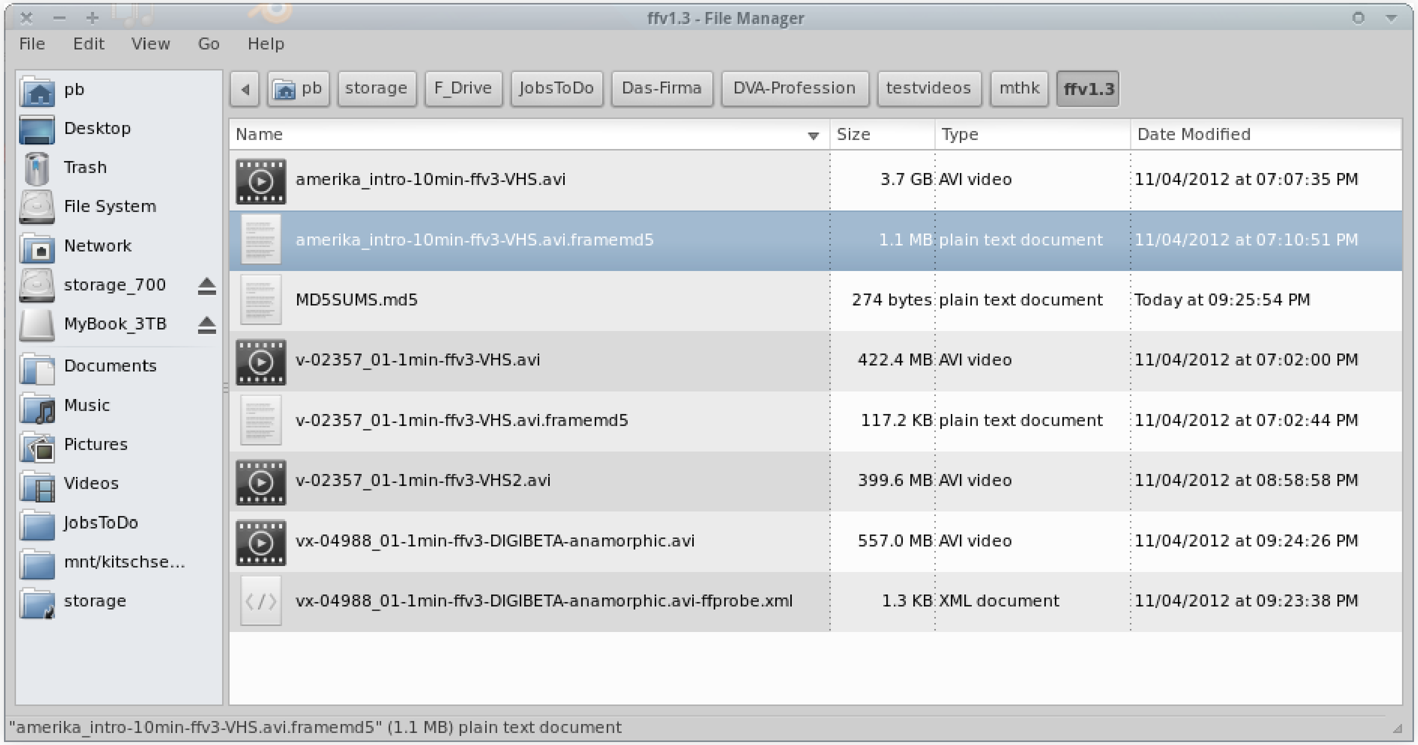
- Filesystem
- File (=data)
- Content (=payload)

Level 1

Filesystem Listing

```
Linux / MacOS
$ ls -la > dirlist.txt
```

```
Windows:
C:\> dir /s /a > dirlist.txt
```



Name	Size	Type	Date Modified
amerika_intro-10min-ffv3-VHS.avi	3.7 GB	AVI video	11/04/2012 at 07:07:35 PM
amerika_intro-10min-ffv3-VHS.avi.framemd5	1.1 MB	plain text document	11/04/2012 at 07:10:51 PM
MD5SUMS.md5	274 bytes	plain text document	Today at 09:25:54 PM
v-02357_01-1min-ffv3-VHS.avi	422.4 MB	AVI video	11/04/2012 at 07:02:00 PM
v-02357_01-1min-ffv3-VHS.avi.framemd5	117.2 KB	plain text document	11/04/2012 at 07:02:44 PM
v-02357_01-1min-ffv3-VHS2.avi	399.6 MB	AVI video	11/04/2012 at 08:58:58 PM
vx-04988_01-1min-ffv3-DIGIBETA-anamorphic.avi	557.0 MB	AVI video	11/04/2012 at 09:24:26 PM
vx-04988_01-1min-ffv3-DIGIBETA-anamorphic.avi-ffprobe.xml	1.3 KB	XML document	11/04/2012 at 09:23:38 PM

"amerika_intro-10min-ffv3-VHS.avi.framemd5" (1.1 MB) plain text document

Speaker notes

For preservation, it's good practice to document these file/folder properties too. Especially when receiving personal collections. They can be used to tell more about a file than you'd think:

- date/time context: Which year? epoch? How old was the author then? Where (in his life) was the author when creating this?
- Ownership / access rights : How private/public was this? For what reason?
- Flags: Why was it hidden? Oh! It was meant to be run as program. Interesting...

By default, date/time format may not be sufficient/suitable for preservation and/or exchange. Therefore make sure that date/time are displayed in a format tht is exactly interpretable.

GNU/Linux systems offer the ability to format it in ISO8601, which is great:

```
$ ls -la --time-style=full-iso
```

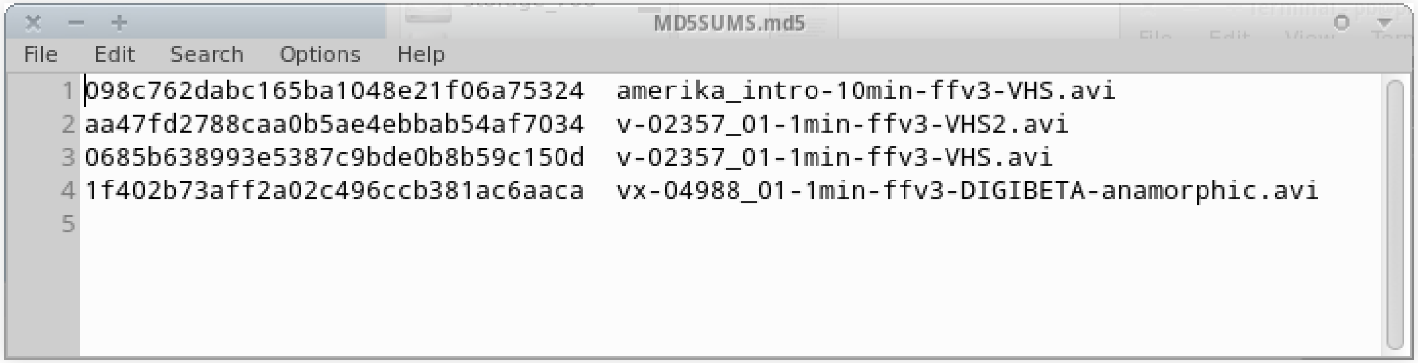
Nice trick: If you want to transfer files from A-to-B, and want to make sure that its timestamps are preserved, you can pack it in a ZIP file: If the application allows it, disable compression (or use TAR).

This wraps a layer around the files, so that their timestamps are now stored as tech-MD inside the archive file.

Level 2

Per File

```
Linux / MacOS
$ md5sum *.* > MD5SUMS.md5
```



```
098c762dabc165ba1048e21f06a75324  amerika_intro-10min-ffv3-VHS.avi
aa47fd2788caa0b5ae4ebbab54af7034  v-02357_01-1min-ffv3-VHS2.avi
0685b638993e5387c9bde0b8b59c150d  v-02357_01-1min-ffv3-VHS.avi
1f402b73aff2a02c496ccb381ac6aaca  vx-04988_01-1min-ffv3-DIGIBETA-anamorphic.avi
```

Speaker notes

Hashcode manifests are simple plain text files where each line represents a file and its hash. This is also called fixity information.

There are different tools to generate/validate hashcode manifests.

The most basic one is "md5sum", which is available by default on *nix systems. For example:

- Single file:

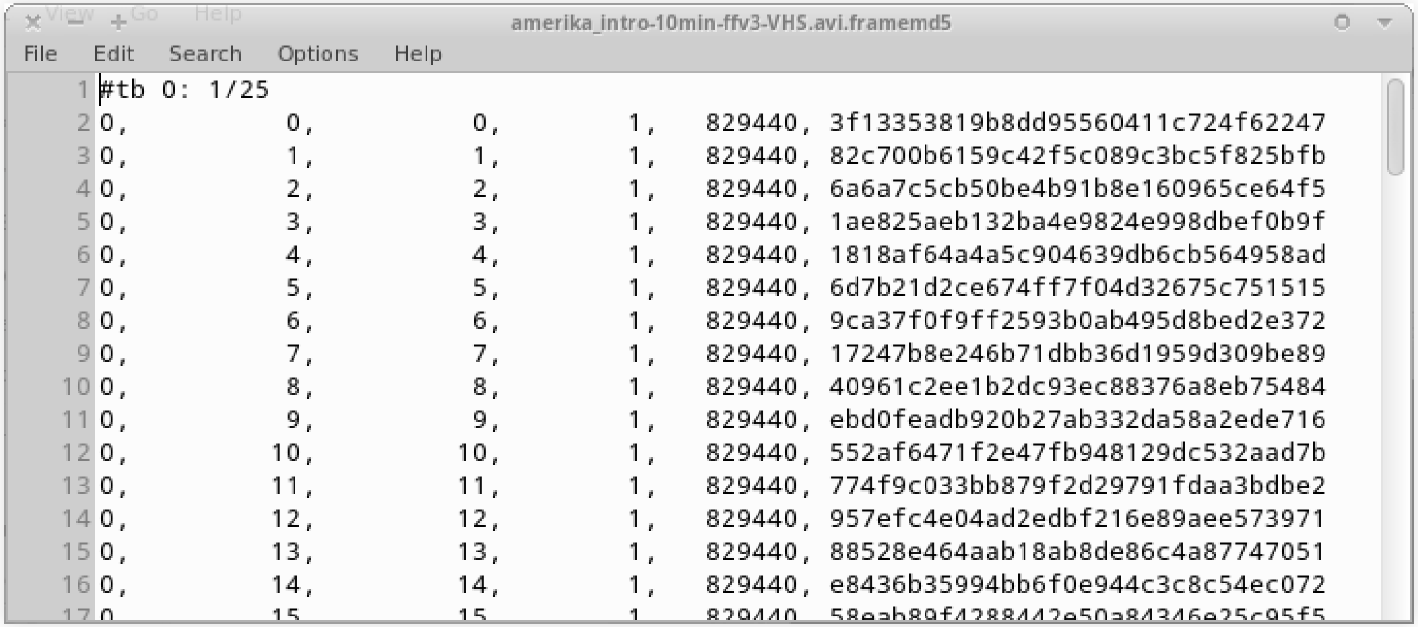
  ```
  $ md5sum my_file.txt > my_file.txt.md5
  ```

- Multiple files:

  ```
  $ md5sum *.mkv > MD5SUMS.md5
  ```

Level 3

Inside: Content Hash



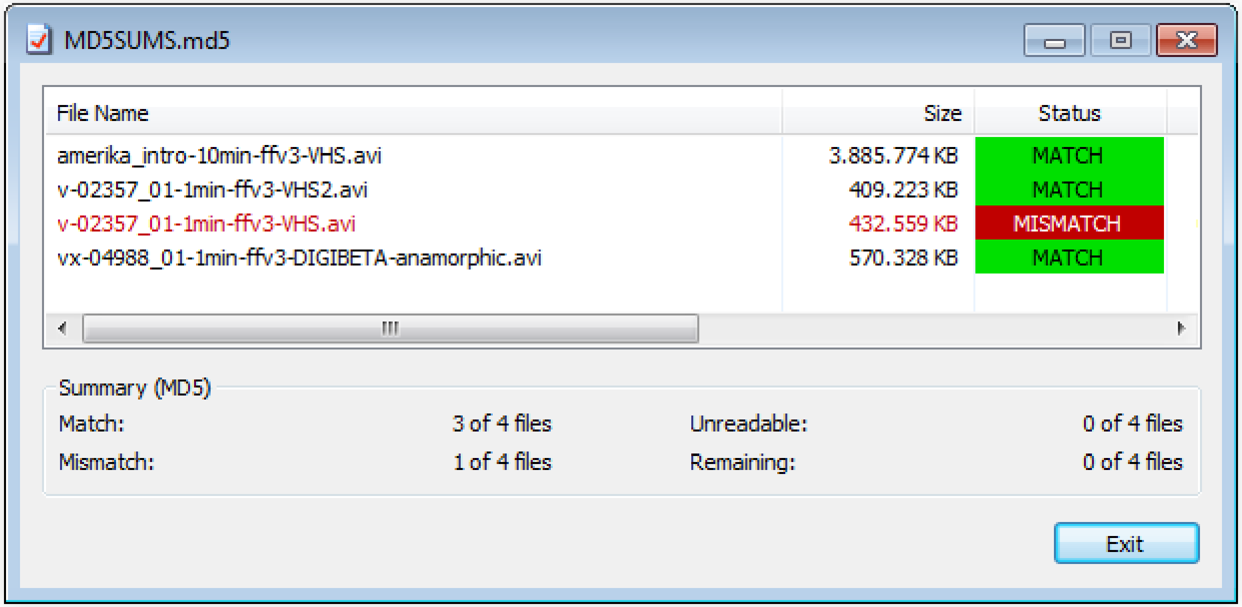
```
#tb 0: 1/25
0,          0,          0,        1,   829440, 3f13353819b8dd95560411c724f62247
0,          1,          1,        1,   829440, 82c700b6159c42f5c089c3bc5f825bfb
0,          2,          2,        1,   829440, 6a6a7c5cb50be4b91b8e160965ce64f5
0,          3,          3,        1,   829440, 1ae825aeb132ba4e9824e998dbef0b9f
0,          4,          4,        1,   829440, 1818af64a4a5c904639db6cb564958ad
0,          5,          5,        1,   829440, 6d7b21d2ce674ff7f04d32675c751515
0,          6,          6,        1,   829440, 9ca37f0f9ff2593b0ab495d8bed2e372
0,          7,          7,        1,   829440, 17247b8e246b71dbb36d1959d309be89
0,          8,          8,        1,   829440, 40961c2ee1b2dc93ec88376a8eb75484
0,          9,          9,        1,   829440, ebd0feadb920b27ab332da58a2ede716
0,         10,         10,        1,   829440, 552af6471f2e47fb948129dc532aad7b
0,         11,         11,        1,   829440, 774f9c033bb879f2d29791fdaa3bdbe2
0,         12,         12,        1,   829440, 957efc4e04ad2edbf216e89aee573971
0,         13,         13,        1,   829440, 88528e464aab18ab8de86c4a87747051
0,         14,         14,        1,   829440, e8436b35994bb6f0e944c3c8c54ec072
0,         15,         15,        1,   829440, 58eab89f4288442e50a84346e25c95f5
```

Some Tools

HashCheck

GUI to handle hashcodes (Windows only).

Website: code.kliu.org/hashcheck



MD5SUMS.md5

File Name	Size	Status
amerika_intro-10min-ffv3-VHS.avi	3.885.774 KB	MATCH
v-02357_01-1min-ffv3-VHS2.avi	409.223 KB	MATCH
v-02357_01-1min-ffv3-VHS.avi	432.559 KB	MISMATCH
vx-04988_01-1min-ffv3-DIGIBETA-anamorphic.avi	570.328 KB	MATCH

Summary (MD5)

Match:	3 of 4 files	Unreadable:	0 of 4 files
Mismatch:	1 of 4 files	Remaining:	0 of 4 files

LoC BagIt "Bags"

> *"Bags have built-in inventory checking, to help ensure that content transferred intact."*

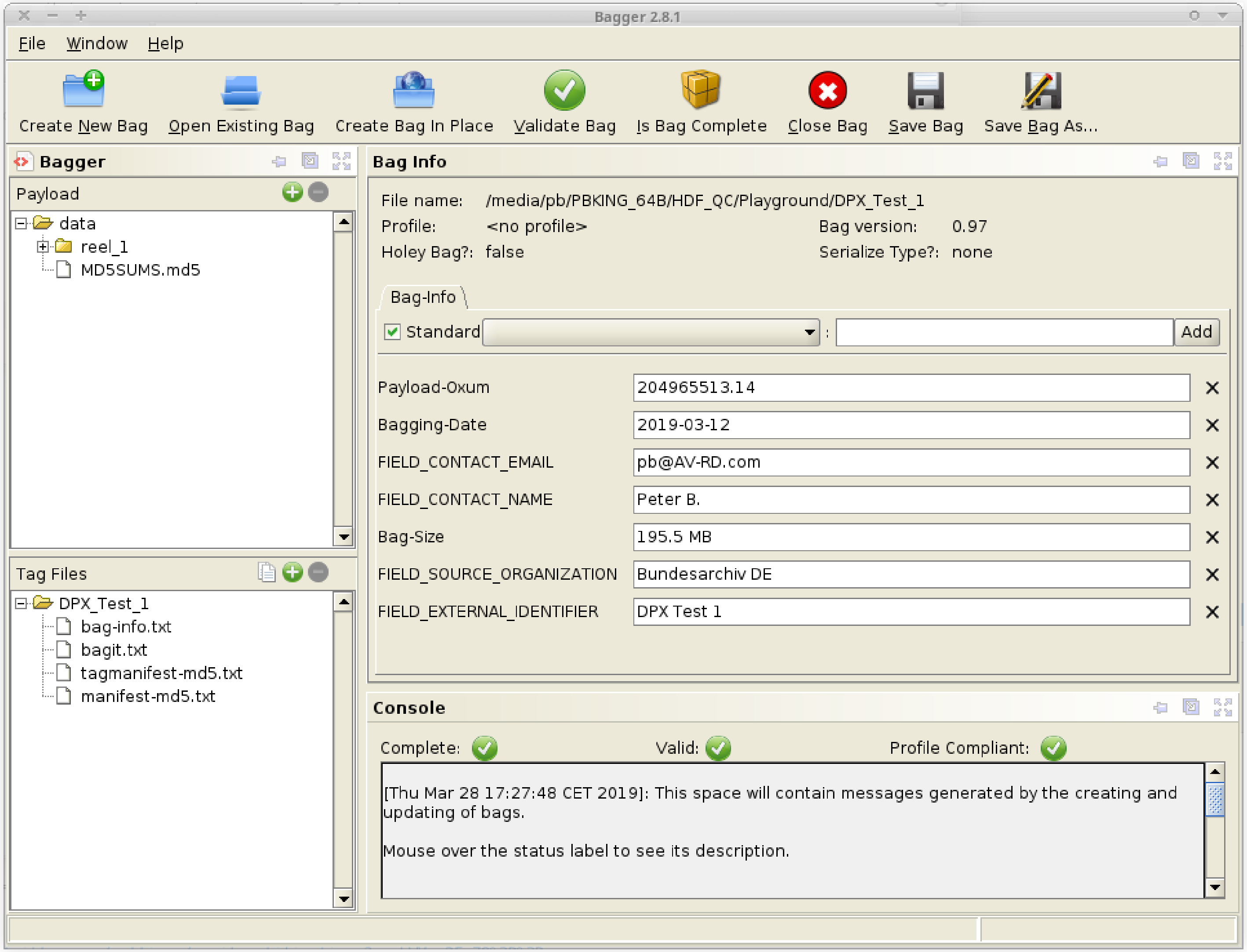
- Intro at 'digitalpreservation.gov'
- Same on Youtube

Bagger

A GUI for handling BagIt bags.

- Website: github.com/LibraryOfCongress/bagger
- Cross platform release (Java)
- Open Source License

Bagger



File name: /media/pb/PBKING_64B/HDF_QC/Playground/DPX_Test_1

Profile: <no profile>

Bag version: 0.97

Holey Bag?: false

Serialize Type?: none

☑ Standard

Payload-Oxum	204965513.14
Bagging-Date	2019-03-12
FIELD_CONTACT_EMAIL	pb@AV-RD.com
FIELD_CONTACT_NAME	Peter B.
Bag-Size	195.5 MB
FIELD_SOURCE_ORGANIZATION	Bundesarchiv DE
FIELD_EXTERNAL_IDENTIFIER	DPX Test 1

Complete: ✓ Valid: ✓ Profile Compliant: ✓

```
[Thu Mar 28 17:27:48 CET 2019]: This space will contain messages generated by the creating and
updating of bags.

Mouse over the status label to see its description.
```

Hashcode use: When?

- Ingest into preservation environment
- Periodically in storage/backup
- During transfers or access
- Deduplification

Comments?

Questions?