Data Integrity

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

November 2019

Data Integrity

What is that?

What is “Fixity”?

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

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?
- 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.

(*) 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.

Hashcode Examples

- CRC =
  `4294967295`
- MD5 =
  `d41d8cd98f00b204e9800998ecf8427e`
- SHA256 =
  `e3b0c44298fc1c149afbf4c8996fb92427ae41e4649b934ca495991b7852b855`

Speaker notes

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

Fixity creation

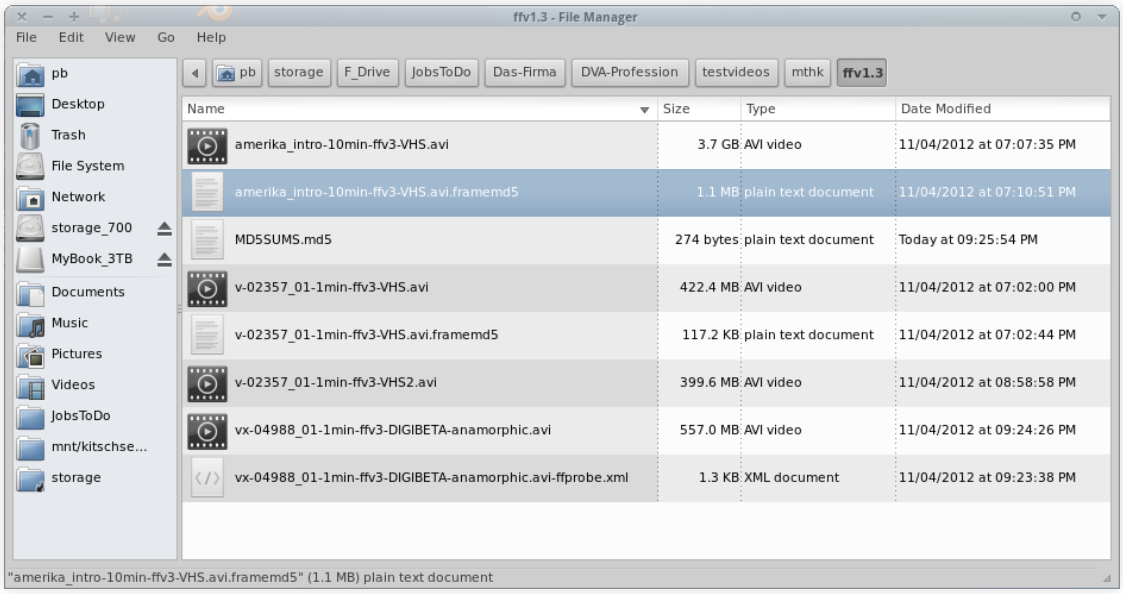
> *Generate hashes as early as possible in a file's lifecycle.*

Different levels

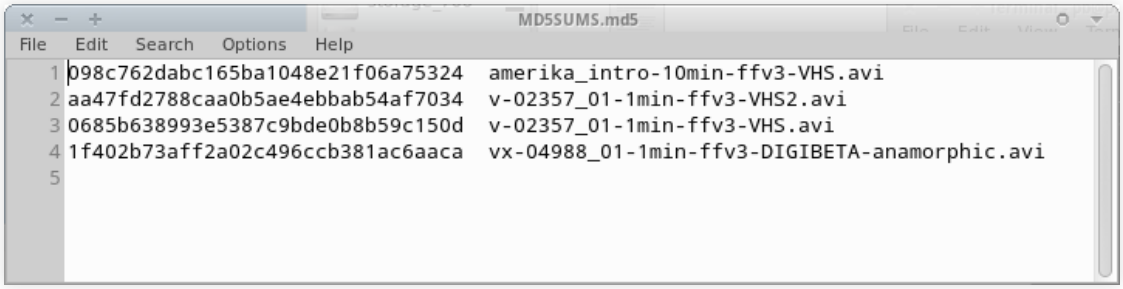
- Filesystem
- File (=data)
- Content (=payload)

Level 1

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



Level 2



Speaker notes

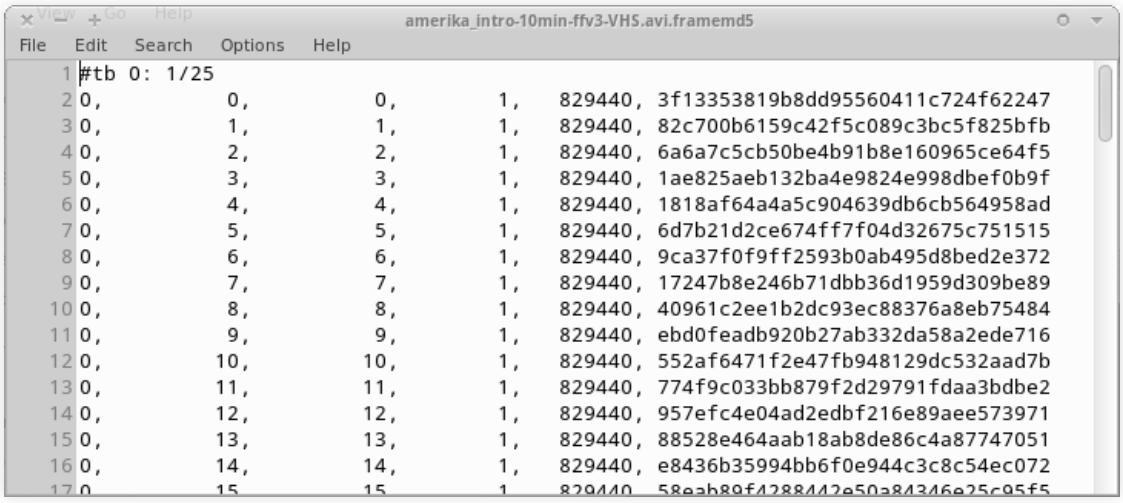
Hashcode manifests are simple plain text files where each line represents a file and its hash. This is also called fixity information.

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

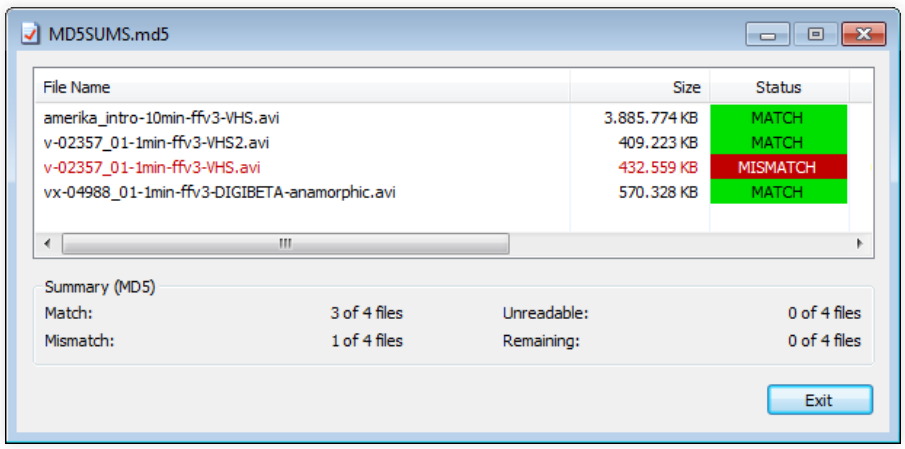


Some Tools

HashCheck

GUI to handle hashcodes (Windows only).

Website: code.kliu.org/hashcheck



LoC BagIt "Bags"

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

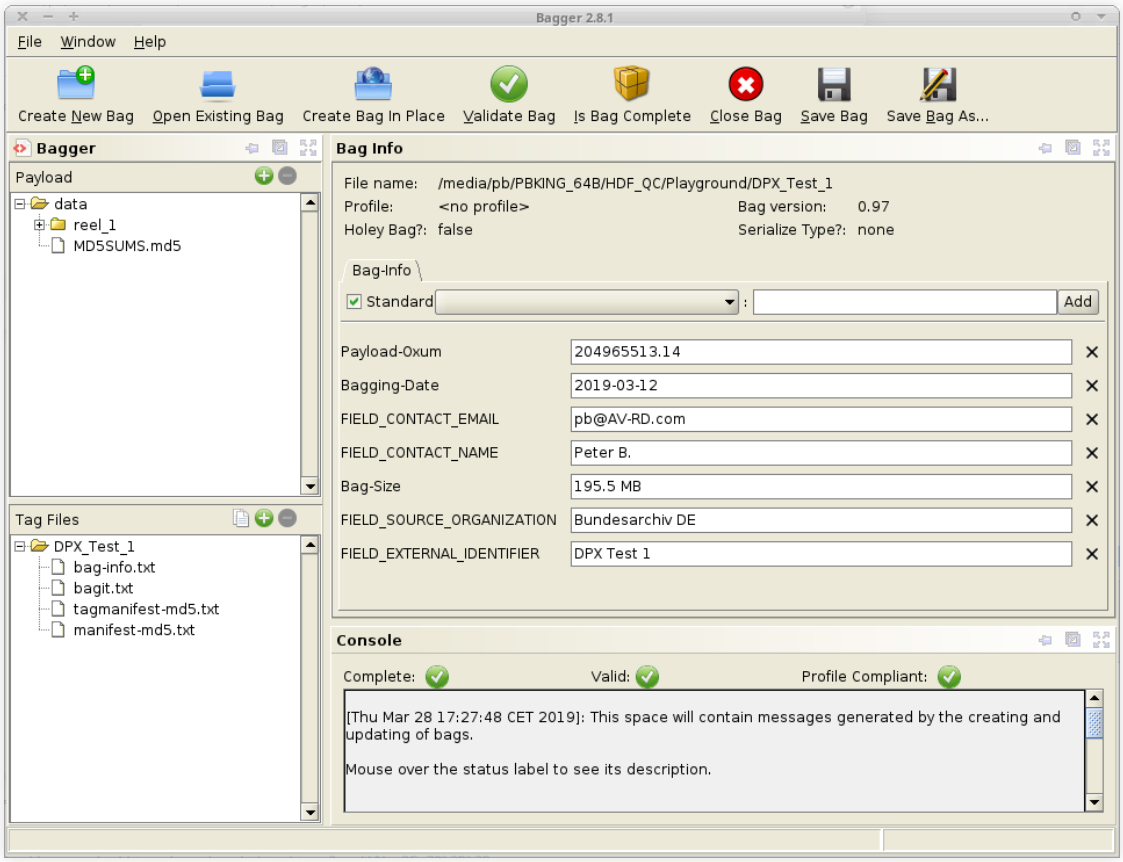
- Intro at 'digitalpreservation.gov'
- Same on Youtube

Bagger

A GUI for handling BagIt bags.

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

Bagger


Bagger 2.8.1
File Window Help
Create New Bag
Open Existing Bag
Create Bag In Place
Validate Bag
Is Bag Complete
Close Bag
Save Bag
Save Bag As...
Bagger
Payload
data
reel_1
MD5SUMS.md5
Tag Files
DPX_Test_1
bag-info.txt
bagit.txt
tagmanifest-md5.txt
manifest-md5.txt
Bag Info
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
Bag-Info
Standard
Add
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
Console
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

Data Integrity Playtime!

Speaker notes

Only a single bit of this file has flipped (=broken). How would you find out which one? How would you fix it?

Comments?

Questions?