

Archival Storage

Peter Bubestinger-Steindl

(`Peter @ ArkThis.com`)

Considerations

- Storage market focus may deviate from your (=preservation) use case.
- Business “longterm”: 3-5 years (=product lifecycle)
- Cross vendor compatibility? Standards? Documentation?
- Spare parts?
- This market focus affects prices, features and reusability.
- **Who owns your storage = owns your work.**

Speaker notes

- Business focus is usually on high availability, speed and least downtime = fast replacement, not necessarily compatibility or long lasting components.
 - Spare parts: Professional hardware is good, but: Often incompatible with standard components, due to proprietary plugs, pinouts, form-factor, etc.

I've seen cases where whole rack-mounted hardware servers/JBODs had to be replaced because it was not possible to get spare power supply units (PSUs) or even ventilation fans for this proprietary layout.

If you can, get good quality hardware with vendor-neutral replacement possibilities. Available beyond 3-5 years after purchase.

And if you like it or not: Who controls your storage, or may decide what/if/how/when - owns control over your work.

Digital Storage = Layers

- **A Digital File**
- Application required to read/write data
- Operating system
- Hardware it runs on
- Filesystem
- Drive ^(e.g. for tapes, optical media)
- Physical carrier (medium)

Speaker notes

When considering a storage for long-term preservation, at least the following layers/components should be considered:

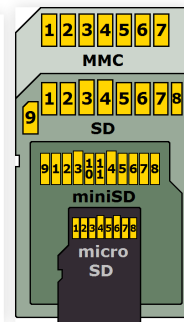
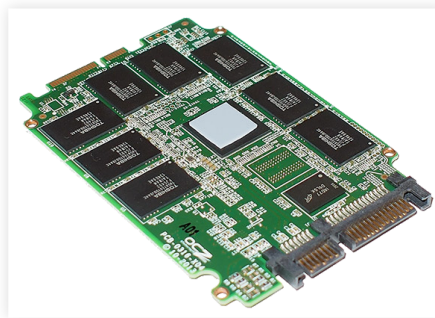
- Application required to read/write data
- Operating system
- Hardware it runs on
- Filesystem
- Drive (e.g. for tapes, optical media)
- Physical carrier (medium)

On top of that (but independent of the storage type chosen) are additional layers of the file itself:

- container format
- video format
- audio format
- metadata format
- ...

Similar principles for selecting a file format apply to selecting a long-term storage solution.

Physical carrier



Speaker notes

Different carrier media have different properties when it comes to error resilience, longevity - and of course: price.

- Magnetic: Susceptible to electro-magnetic issues.
- Optical: light, temperature, warping
- Electronics: Failing capacitors
- Mechanics: Wear and tear
- Supported/obsolete interfaces or drives?

It's usually good to mix storage media to distribute their pros/cons properties.

I won't go into great detail here, as this is possibly covered in many other preservation-storage presentations.

This here is for the interested reader.

Examples for different use cases:

- preservation / backup:
tape (affordable size, good preservation properties, less need for super-fast access)
- access: No need for long-term endurance, so e.g. optical would be sufficient.
- production: HDD-based storage for nearline files - tape for offline storage. Or: tape-robot with HDD cache (same concept, but fully automated)

More properties:

- Data Access Speed?
SSD > HDD > Tape > Optical
- Cost/size ratio?
SSD > HDD > Tape > Optical
- Widespread?
 - SSD, HDD: Excellent!
 - Tape: almost exclusive professional use case Drives not that “easy” to get (for particular LTO generation)
 - Optical: Declining (except for “cold storage” in large data centers)

In greater detail:

Optical Disks

Pros:

- Cost/size ratio: Relatively cheap (cheapest still?)
- Not susceptible to electro magnetic (dis)charge
- No mechanics/electronics in carrier - only in the drive

Cons:

- Error prone
- Suffers easily from material degradation (scratches, data layer falling off, light exposure, temperature, etc...)
- Becoming less and less common = less support, less choice, etc.

HDD: Hard Disk Drive

Pros:

- Reasonable cost/size ratio
- (yet) cheaper than SSD
- well known, well supported

Cons:

- Mechanical wear (moving parts)
- Susceptible to electro magnetic discharge
- Electronics on carrier

Data Tape

Pros:

- Cost/size ratio: Tape cartridge “cheaper” than HDD
- Tape preserves well (and there is much experience with dealing with tape degradation/aging from analogue)
- Supported in “tape libraries” (aka “tape robots”) for removing “tape-jockey” limitations.
- Larger capacity per cartridge than per HDD (yet)

Cons:

- Not common outside certain domains (large institutions, broadcast preservation, ...)
- Therefore support for tapes very “specialized”
- Drives quite expensive (>3000 EUR)

Flash Memory

Pros:

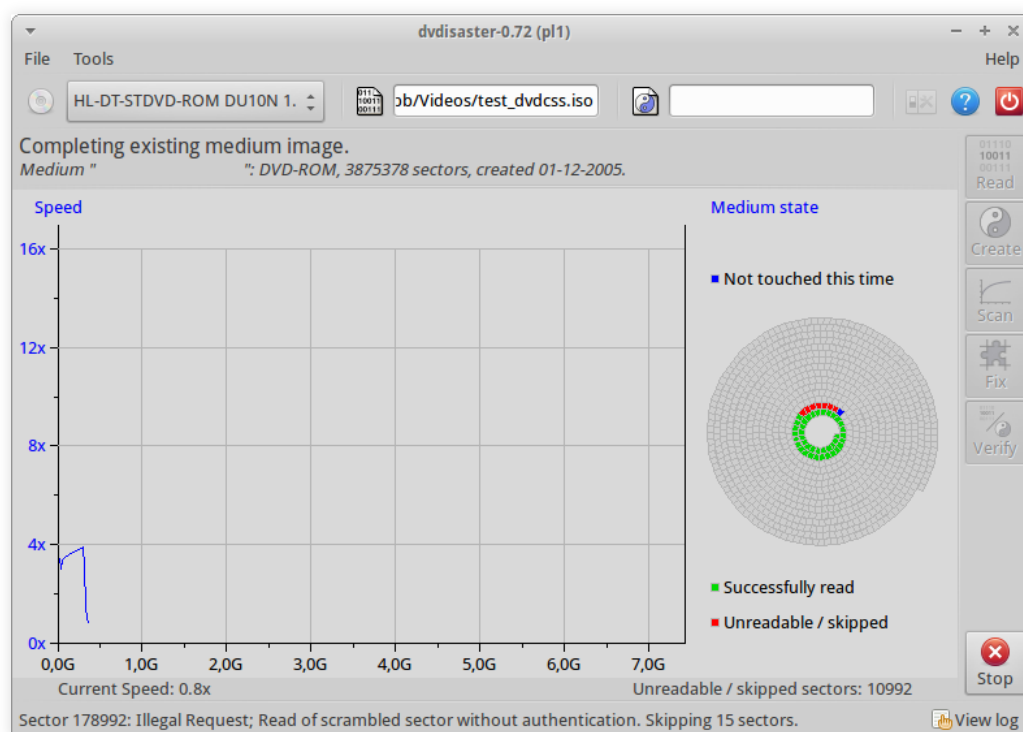
- cost/size ratio: getting cheaper, but still more expensive than HDD
- No moving parts = no mechanical wear
- SD cards: almost no electronics in the carrier
- Speed: SD cards are not sooo fast, whereas SSDs (or [M.2](#)) are extremely fast (compared to HDD).

Cons:

- Currently most expensive (still)
- Lack of long-term experience
- Limited number of write/erase cycles per flash block (becoming less of an issue) = Should not be used for large number of write operations. But preservation storage case is mostly “read”.

More about [Flash memory on Wikipedia](#). SD card image by [Steve Meirowsky \(Wikipedia\)](#) - Own work, CC BY-SA 3.0

Dvd disaster: Optical Rescue.



Speaker notes

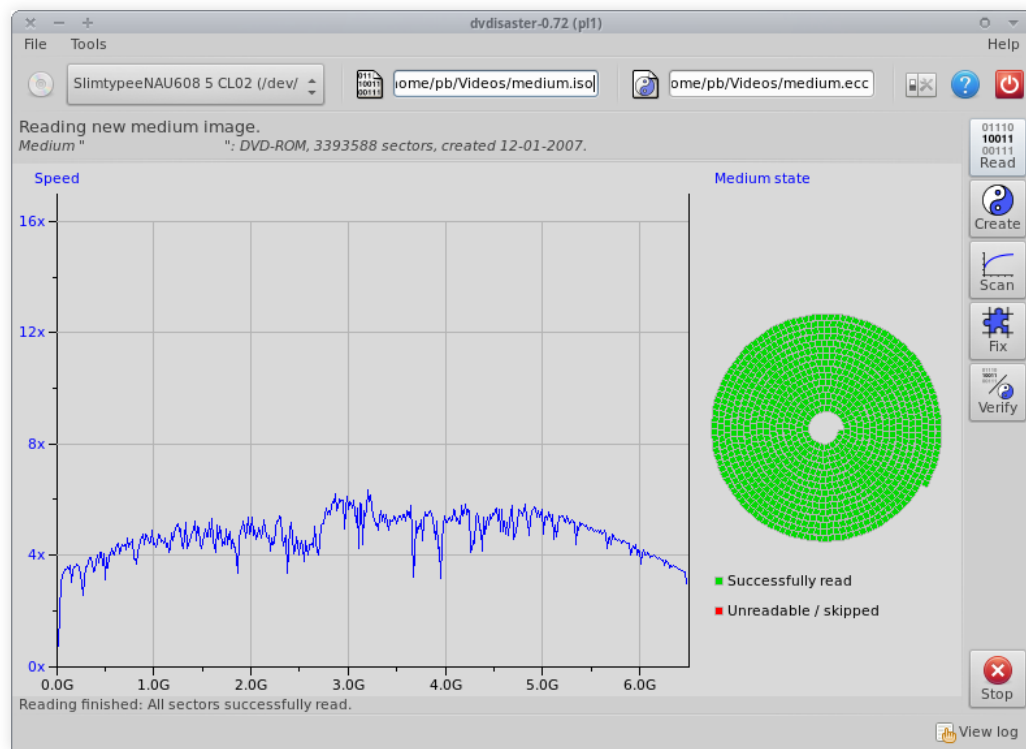
An example of an excellent FOSS-licensed tool for reading optical media (like CDs, DVDs, BluRay) as bit-exact as possible.

[Wikipedia: Dvdisaster](#)

The original author Carsten Gnörlich is not maintaining it anymore, but it is still being maintained by others and included by default in most GNU/Linux distributions.

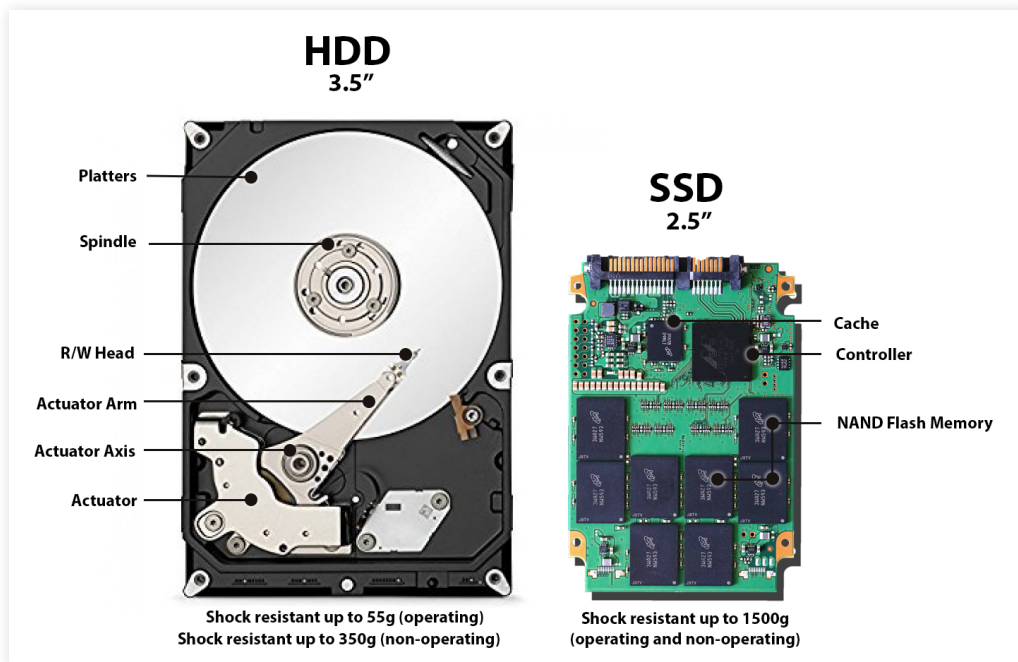
It may be a good idea considering “adopting” this project for DLTP.

Dvdisaster: Success!



Bad shape, Good read.

HDD vs SSD



SSDs: less stuff to break

Speaker notes

HDD:

- (yet still) cheaper cost-per-data ratio
- moving parts = mechanical wear out
- btw: Spinning up is most stressful for the disk. Avoid unnecessary up/down cycles.
- Lots of electronics...
- Faster than tape, slower than SSD

SSD:

- Super fast!
- (yet still) more expensive than HDDs (equal size)
- NO moving parts: good!
- Lots of electronics though...
- Limited number of writes, but:
- Not as relevant for content that is written once, then stays (=archival use case)

Good to know:

- Higher data density = more impact of an error.
Example: 1mm hole in CD vs DVD vs BluRay - or SD vs MicroSD, etc.
- HDD: The longer it is active, the shorter it lives.
- But: *“Hardware that lies, dies.”*
- Be aware if your HDD is “[Shingled \(SMR\)](#)” or not.
- Mixing carrier/storage types: good idea!

Speaker notes

Density:

- Think of a 1mm hole on an optical carrier: CD vs DVD vs BluRay
- All 3 have the same dimension, but different data density.
- When choosing a carrier: Do you really need it to be as small/dense as possible?

Hardware that lies, dies:

Got external HDDs stored on shelves?

You may want to power them up every once in a while (every 1-2 years or so), and copy the data off them before the hardware gets old. Because: Electronics fail (capacitors), lubricants harden out or movable parts get “lazy” or stuck.

Das Werkstatt Rule #2 (German): “Hardware die liegt, stirbt” [Useful “Das Werkstatt” principles](#)

Shingled HDD:

“The more time a shingled disk spends in band compaction, the more it is wasting resources.”

Source: [Classifying Data to Reduce Long-Term Data Movement in Shingled Write Disks](#) This means, SMR disks are active when idle, causing them to potentially wear out faster.

Tape & Drives



LTO tape with a drive

Speaker notes

There's more to say on drive+media storage, but I'll focus on LTO tapes here, since they become more "important".

What may seem obvious is, that even if a tape may last a very very long time: Without a drive, you're pretty much lost.

With LTO-tapes, you also need to consider which "LTO Generation" (LTO-4, LTO-5, ... LTO-9, etc) drive and tape are, since not all LTO drives may read all LTO tapes.

LTO Generations

Imagine you find an old LTO tape...

- Not every drive can read every tape.
- New LTO generation release: ~every 2-3 years.
- < LTO-7: Read=2 gen. / Write=1 gen.
- BUT: LTO-8: Read/write=**1 gen.!**

LTO Generations

Imagine you find an old LTO tape...

- Not every drive can read every tape.
- New LTO generation release: ~every 2-3 years.
- < LTO-7: Read=2 gen. / Write=1 gen.
- BUT: LTO-8: Read/write=**1 gen.!**

Oh, and: Which **filesystem** was it written with...?

LTO improvements include new LTO generations: Higher data density at same form factor. This allows the mechanics (of drives and robots) to stay compatible, with only requiring certain parts to be updated to support a new generation.

The LTO consortium guarantees a certain level of downwards compatibility. This used to be:

- Read: 2 generations down
- Write: 1 generation down

Example: LTO-6 drive should be able to read LTO 6,5,4 - but write only 6 and 5.

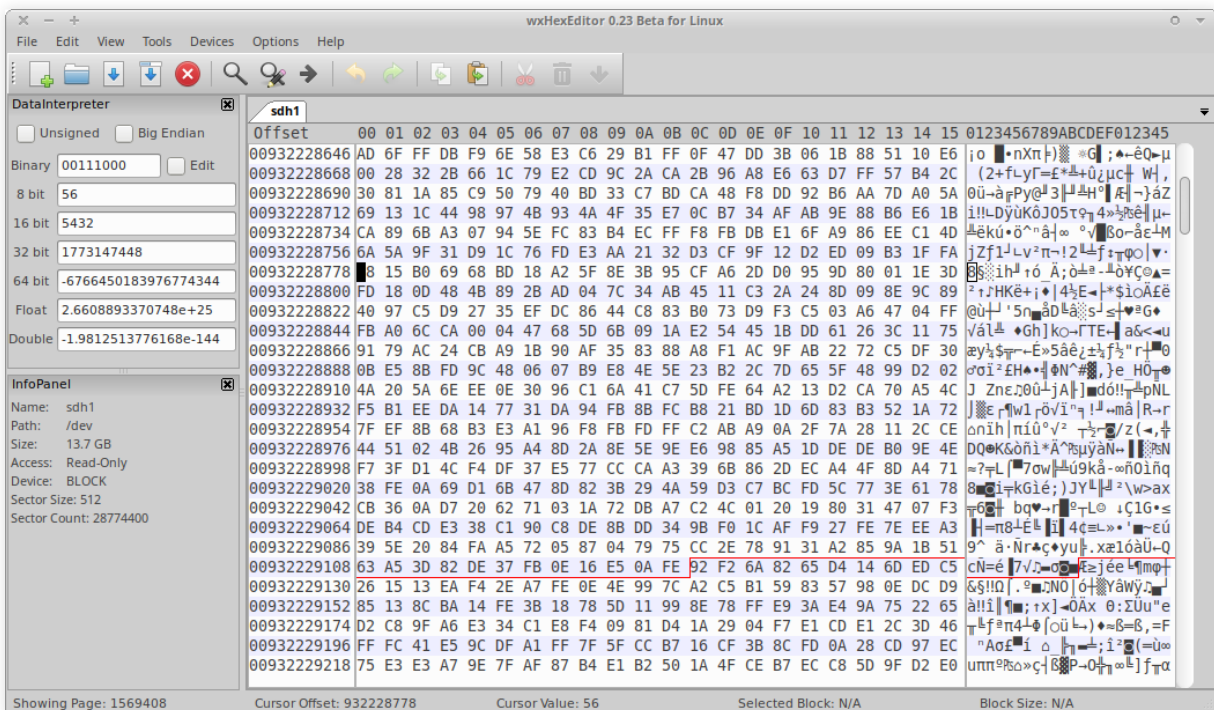
With LTO-8, this was reduced to 1 generation backwards compatibility, so an LTO-8 drive can *not* read LTO-6, only back to LTO-7

New LTO generation release: About every 2-3 years. This must be considered for migration!

This also means that you might want to keep a few drives of different generations in house?

Oh, and if the application you use

The File System



That image shows a disk drive partition as quasi-uninterpreted raw data. This is what your folders and files would look like without a “file system.”

Ever encountered an “unreadable” storage medium? (SD card, USB stick, HDD, etc?) In most cases the reason were broken bits in its Filesystem. The data is often still where it was.

A **File system (FS)** is what forms raw data into a file/folder structure

- Storage without FS = Byte pulp. Like pages of a phone book torn apart, page numbers and index removed - then shuffled.
- Filesystem = A data format Therefore, the same requirements for preservability/accessability apply:
 - Is it documented?
 - Who can read it?
 - Under which terms and conditions?
- Different FS = different formats

Popular File Systems

- **FAT{16,32}** (Microsoft)
- **NTFS** (Microsoft)
- **EXT{2,3,4}** (Linux)
- **ZFS** (Sun)
- **HFS+** (Apple)
- **LTFS** (IBM)

FAT:

- FAT introduced in 1977.
- Widespread, well-known and documented
- Default for SD cards and embedded devices
- FAT32: 32bit limitation (max 4GB filesize, limited max. device size)
- No access permissions (=everyone can read/write everything)
- Very simple and straightforward filesystem
- Non-free / license fees by Microsoft.

NTFS:

- Introduced in 1993.
- max volume size (different implementations): 256 TiB (pre 2016), now 8 PiB (2019 or later)
- Non-free / license fees by Microsoft.

EXT:

- ext2 introduced in 1993.
- ext4 standard for Linux-based OS.
- ext4 and NTFS supported by digital cinema standard. (Because the projectors probably run Linux ;))
- max volume size: 1 EiB (1024 PiB)
- No licensing fees

ZFS: * Introduced in 2005 * More than “just a filesystem”: combines RAID (redundancy, handle harddisk failure) [Logical Volume Management \(LVM\)](#) and filesystem in one. * Integrated integrity checks * (recently) well known among IT administrators of unixoid systems. * Requires slightly more experience to handle (because of complexity. * Very interesting filesystem (also for preservation). Too much to describe here. You may want to read more on Wikipedia :)

HFS+: * HFS Introduced in 1985. * max volume size: 8 EB * Successor of “[HFS](#)” * Officially only supported on Apple OS

LTFS: * LTFS first prototype (Linux and Mac OS X) during 2008/2009 * Only for LTO tapes. * See separate page/slide on LTFS for details.

LTFS: Linear Tape File System

- Open specification = vendor neutral
- Better for preservation, but may not support “convenience” features.
- All implementations must:
 - Correctly read media that was compliant with any prior version.
 - Write media that is compliant with the version they claim compliance with.

Speaker notes

The [Linear Tape File System \(LTFS\)](#) is a good solution to avoid vendor lock-in by tape filesystems. Please make sure that you can read the data written onto the tapes under the following conditions:

- Different drive
- From different vendor
- With (FOSS) software (and/or from different vendor)

Otherwise, you won't really know if you can access your data as technology- and vendor-neutral as desired.

By default, different applications may write data in their own way on LTO tapes. This allows them to implement some non-standard features that often provide more “convenience” by offering additional features. Using such vendor-specific features, instead of LTFS has the downside of reducing the possibilities to read that data under different conditions in the future.

Therefore it is advised to use LTFS for long term preservation use cases.

File system: Disaster relevant?

- Deleted files are still there (maybe fragmented).
- Different FS = different error resilience and recovery options.
- Does it scale? (moving files is when they're most vulnerable...)
- Tools/knowhow to deal with recovery of broken filesystems in your setups?
- **Logical Volume Management (LVM)** snapshots

Speaker notes

- Deleted files: Deleting files is just flagging them in the index and their disk space is marked as available again. The data is still where it was - until a new write operation (=creating new data) uses the same space and overwrites the old data.

Related articles:

- [Get Your Data Back with Linux-Based Data Recovery Tools](#)
- [How to Install and Use TestDisk Data Recovery Tool in Linux](<https://www.tecmint.com/install-testdisk-data-recovery-tool-in-linux/>)
- [Scalpel: Extract lost files from disk/card image](#)
- Different FS have different error resilience and recovery options: Go ask your IT guys! ZFS is pretty awesome, but “simple and well known” is also nice for recovery.
- FS recovery requires understanding the FS format. Got documentation? Source code?
- Logical Volume Management (LVM) Remember VM-snapshots? You can do the same on storage level (below filesystem): [LVM Snapshots](#)
- Does it scale?
 - Total number of files/folders?
 - Length of path?
 - Suitable for NAS/SAN/cluster?
 - [FS vs Blocks vs Object-based?](#)

Failsafe mechanisms



Just to be sure.

Of course: If nothing goes wrong, there's nothing gained from the extra effort of implementing any failsafe mechanisms.

It's like riding a motorbike on a sunny day, with short trousers, sandals, no gloves and no helmet. Wonderful to enjoy! If nothing goes wrong...

It's still great to ride a motorbike, but a few things may make a huuuuge difference in case "something" does happen. :)

S.M.A.R.T.

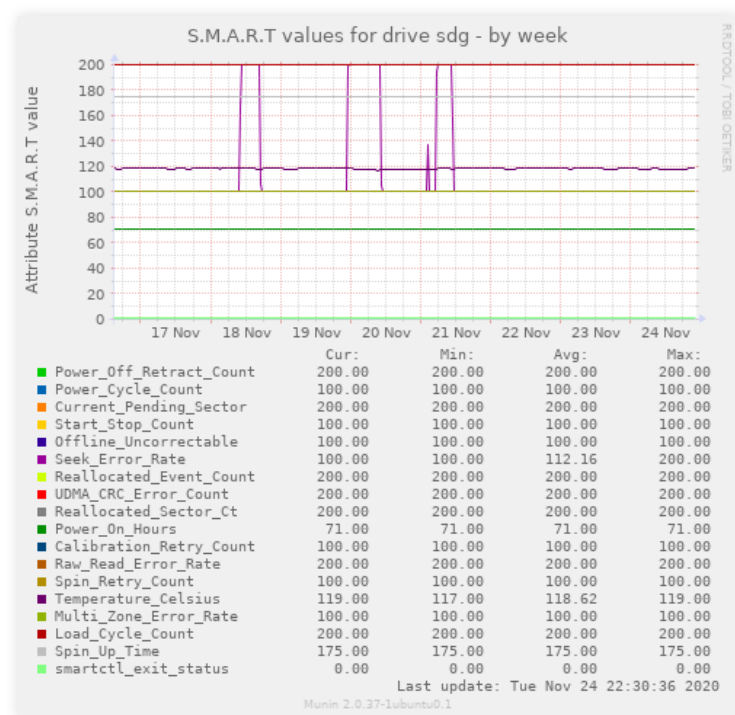
Self-Monitoring, Analysis and Reporting Technology

"[...] is a monitoring system included in computer hard disk drives (HDDs), solid-state drives (SSDs),[1] and eMMC drives. Its primary function is to detect and report various indicators of drive reliability with the intent of anticipating imminent hardware failures."

Source: [Wikipedia: S.M.A.R.T.](#)

- Part of the ATA standard since around 2004.
 - All HDDs these days usually support SMART.
 - Different vendors/drives return different readings.
 - “The meaning and handling of the raw value is a secret of the vendors embedded S.M.A.R.T.-Software on the disk.”
 - It makes sense to monitor these values and alert if values deviate from a certain threshold.

S.M.A.R.T. Graph



- SMART values can be collected over time and drawn as graph. This allows to literally “see” if some reading is “going down”.
 - Generated using [Munin](#), which uses the great [RRDTool](#) to generate graphs.

RAID

Redundant Array of Inexpensive Disks



“[...] is a data storage virtualization technology that combines multiple physical disk drive components into one or more logical units for the purposes of data redundancy, performance improvement, or both.”

- Good 'old' friend: exists since before 1988
 - Here you see a classical hardware RAID chassis.
 - This is just one option.

RAID

- Different **RAID levels**:
 - Stripe = Fast, but dangerous!
 - Mirror = Perfect for OS system disks
 - RAID5/6 = 1 or 2 disks fault tolerance (**parity bits**)
- RAID is *not* a backup.

Data replication is not a Backup: It is not unusual to discover an undesired modification or error only later on.

- Sync:
 - File changes (new, modification) are immediately replicated
 - Unwanted actions, too! (wrong modification or deletion)
- Async:
 - Allows you to go back in time.
 - Allows multiple copies/revisions of the same data.
 - “Long” time between new/modification and the copy.

Sad, but true:

- With > 4TB disks, **RAID-6** may be insufficient...
- Long rebuild times may “burn out” the left-over disks.
- ZFS supports 3 disks parity, but ... future?

Sad, but true:

- With > 4TB disks, **RAID-6 may be insufficient...**
- Long rebuild times may “burn out” the left-over disks.
- ZFS supports 3 disks parity, but ... future?

btw: How does “The Cloud” do it?

Speaker notes

The algorithm RAID-6 uses to provide data redundancy for rebuilding in case of harddisks failing, may have come to an end with HDD capacities beyond about 4TB per-disk.

The reason for that is, that in case of a disk failure, the RAID mechanism reads the parity data from all remaining disks to reconstruct the one that “has gone”. With larger disks, the time for this rebuild to happen is getting longer. During that period, all remaining disks are constantly “stressed” until the required data has been read and the new disk rewritten.

Considering that disk failures more likely occur in RAID arrays that have been running for quite some time, it can be assumed that the other disks may be stressed too hard during that rebuild, causing other disks to fail *due* to the recovery - resulting in a complete data loss.

Cloud Storage

- No internet = no service.
- Make sure your online bandwidth is sufficient.
- Have an exit plan (+contract?) for migration.
- What if their conditions change?
- Where is your data actually stored?
- Does it matter for you? (e.g. legally)
- Again: Consider mixing.

The term “Cloud” in computing came from marketing, and describes nothing else than digital services that already existed for over 20 years - but now had a nicer GUI and focus on convenient usability.

There’s “cloud storage” (e.g. Owncloud/Nextcloud, Dropbox, etc) and “cloud computing/services” (GDocs, etc).

In this context we will only deal with “cloud storage”.

Storing data in large data centers is very convenient. All the same challenges regarding large-scale storage still apply, but they’re someone else’s problem now :)

What’s still relevant for you though is:

- What happens if the provider changes their conditions?
- What happens in case of unstable political situations, war, etc?
- What if you want to migrate your stuff to another storage/provider?
- Does it matter (e.g. legally) if the provider could access your data?

Storage Debate!


Make 2 groups:

- One group for “all in-house”
- One group for “all external”
- Think of 5 “pros” why your scenario is a good choice.
- After 7 min. we'll get back and debate.

Storage Debate: Summary?

- Online/Cloud: For “smaller” files (eg Access copies)
- Large storage: You may want/need to get external support.
- Decide which level of control you have over your data.
- Mixing (again) is a good idea.
- Should we archive outsource all knowhow/skills about digital storage?

Cluster & replication

- Multiple copies on multiple servers (“cluster nodes”) 
- Synchronized (replicated) automatically
- Off-site replication possible
- Replication is *not* a backup!
- What if a node goes down?

Speaker notes

Where RAID provides failover redundancy for individual storage devices, a storage cluster may provide failover redundancy by means of whole PCs.

Each PC is called a “node” and is designed as a component that may fail, while the rest of the cluster can maintain their services. This is how cloud storage/service providers provide transparent failover setups.

Replication is not a backup.

Replication mechanisms try to keep all their data in-sync as quickly and seamlessly as possible. This means that if you delete a file, it is usually not known by any replicating system if that was on purpose or in error.

Therefore, the replication mechanism happily will synchronize this change (=the deletion) to all of its nodes. Voila.

You get it, why this would be a bad backup option? ;)

Erasure Coding

- More parity possible than RAID
- Data can be spread across network cluster nodes.
- Does not suffer from rebuild “burn out”.
- More complex to set up & compute.
- Not that widely known/supported/used yet.
- A FOSS implementation: [MinIO](#)

Speaker notes

“protects data from multiple drives failure, unlike RAID or replication. For example, RAID6 can protect against two drive failure whereas in MinIO erasure code you can lose as many as half of drives and still the data remains safe.”

Source: [MinIO Erasure Code Quickstart Guide](#)

“Compared to data replication, erasure-coding approaches have better performance at reducing storage redundancy and data recovery bandwidth.”

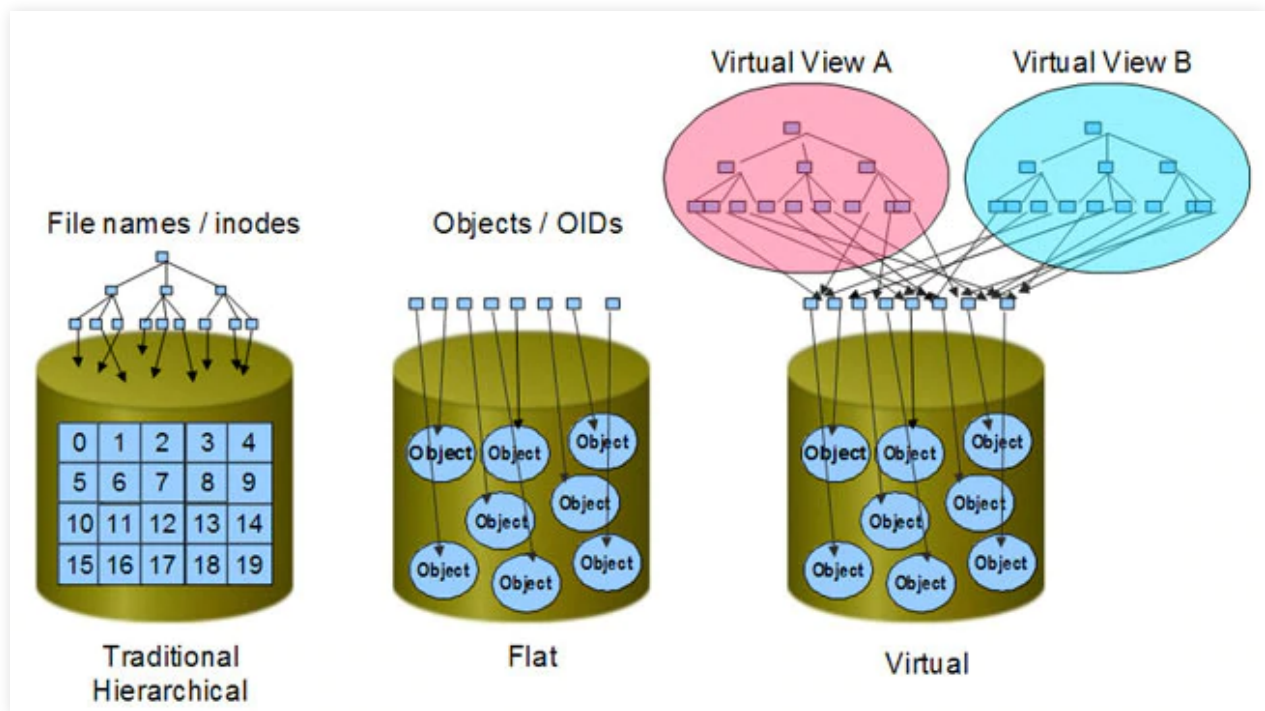
Source: [Reliability Assurance of Big Data in the Cloud \(2015\)](#)

Links:

- [MinIO: Storage usage](#)
- [MinIO: Erasure Code Quickstart Guide](#)
- [Erasure coding vs RAID: Data protection in the cloud era \(ComputerWeekly\)](#)
- [RAID Vs. Erasure Coding \(NetworkComputing\)](#)
- [What is erasure coding? \(VirtuService\)](#)
- [Isilon.com: Erasure Coding \(Isilon\)](#)
- [Erasure Coding vs. RAID: Which Is Right and When? \(ITProToday\)](#)
- [Wikipedia: Erasure code](#)

Object storage

Beyond classical, hierarchical filesystems



Object storage

- Files become “objects”.
- There are no (actual) folders. (Foldername = Plain Metadata)
- There’s just an identifier.
- Arbitrary metadata may be assigned to each “object”.
- Scales infinitely (theoretically).
- Clusters of block storages with a clever abstraction layer.
- Quite a complex construct of APIs, layers and components. Fascinating.

Speaker notes

You may have been using object based storage on Google Drive or Amazon S3.

Pretty cool, but it handles very differently from any regular file/folder based storage! Please do your homework before deploying this.

It is still pretty new and if done wrong, it can cause more troubles/effort than a regular, hierarchical FS storage.

What is often left out by marketing:

- Underneath, it is still using block storage.
- Just with clever abstraction layer(s).
- Important: Your toolchain needs to be object-storage-aware! Regular, well-known tools cannot easily access object files as it used to be.
- I mention it here, because if done right, it scales.

I'm pretty sure, this is the future - and (at least large scale) storage will be going from classical (filesystem-based) to this (=object based) way.

Object storage = The future?

- If implemented well, the storage itself could be the DAM/MAM.
- Allowing links/relationships between objects = 🤖🎉
🤖🎉
- The object filesystem **is an actual catalogue.**
- Combined with LoD, MD Standards & APIs = Wikidata for/of your files.
- If implemented well...

Speaker notes

There is great potential in Object Based Storage.

It is already in production usage for a few years by major online service providers. Yet, the “catalogue” world, including the WWW are still mostly running and “thinking” in hierarchical filesystems. DAMs and MAMs are a middle ware layer for trying to link files and “other stuff related to them” in a way that solely serves: Search and Retrieval.

Because if you store something, and you’re not retrieving it at least 1 anymore: What have you kept it for?

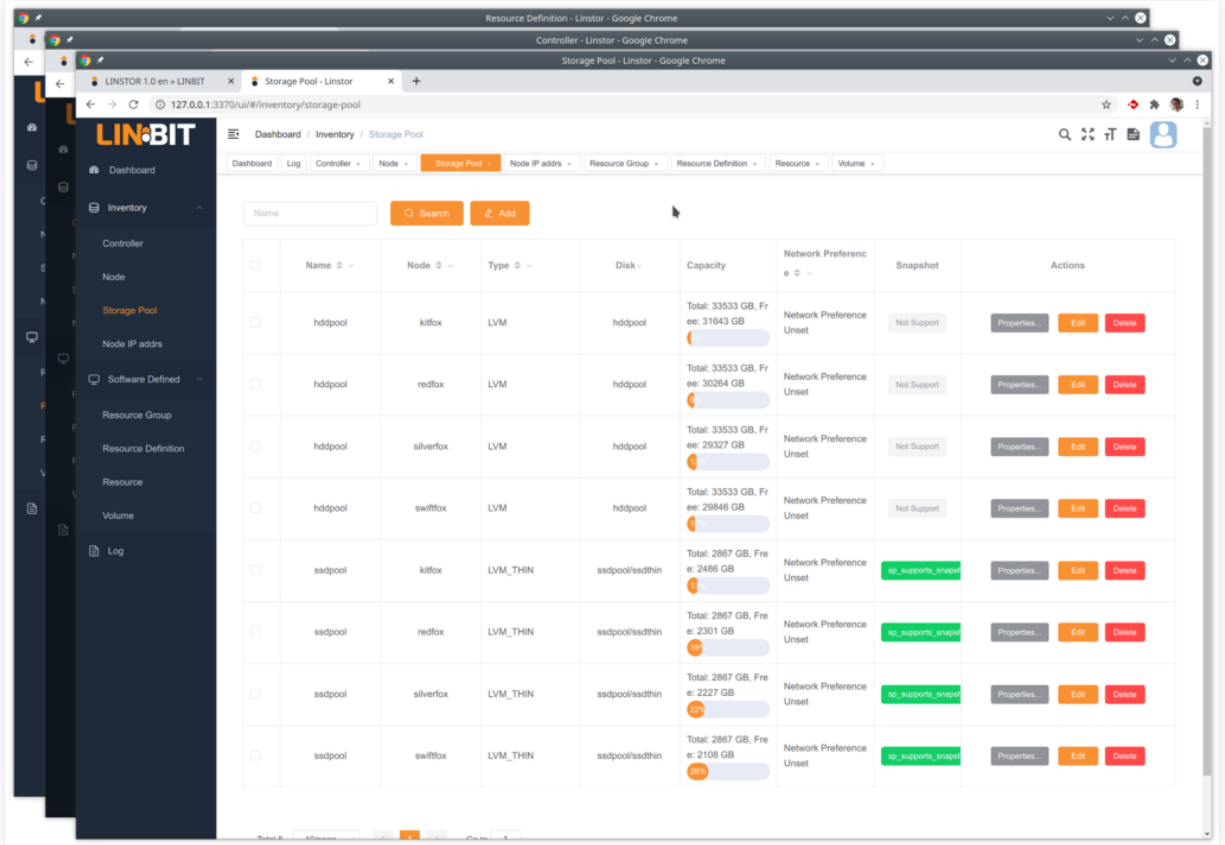
Object storage = The future?

Object based storage may very likely replace hierarchical filesystems, but things need to be rewritten/adapted to properly support it.

It's not yet (really) plug-compatible with existing programs.

Software Defined Storage (SDS)

More “Classical”, yet Supersize-able.



Example screenshot (Linbit SDS)

Speaker notes

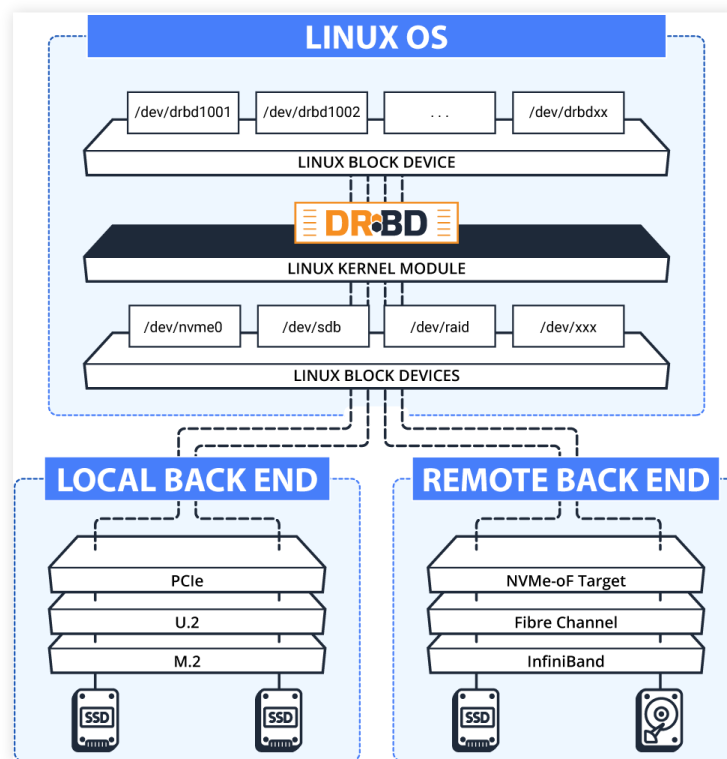
Quote from [Wikipedia](#) "Software-defined storage":

"Software-defined storage (SDS) is a marketing term for computer data storage software for policy-based provisioning and management of data storage independent of the underlying hardware.

Software-defined storage typically includes a form of storage virtualization to separate the storage hardware from the software that manages it.

The software enabling a software-defined storage environment may also provide policy management for features such as data deduplication, replication, thin provisioning, snapshots and backup."

Again: Onions, eh Layers.



Example of abstraction layers

Quote from [Linbit Website](#):

"Since the software simply presents a block device, users do not need to use complex, shared file systems. Because we can use a single node with a filesystem on top of a block device, backing disk failure is transparent to the end user."

Large storage that provides flexibility/scalability, the required features (distribution, replication, backup, etc, etc) - and transparency for the applications accessing the files means stacking up more layers.

Even if it ain't rocket science, it still is far from trivial (especially setting it up, maintaining and and debugging it).

Does it scale?

The image shows one row of Backblaze storage pods in a computing center. This is just to give you an idea that:

- The Cloud literally is just a bunch of computers (and storage media inside them).
- Large volumes of data need large volumes of space. And electricity, knowhow and spare parts and staff.
- However, the underlying mechanisms and applications are designed to scale.

Does it scale?

- What if you run out of disk space?
- Need to copy/move everything?
- Or can you simply add “more space”?
- What about your file/folder positions?
- How to know if everything is still “there” and intact?

In the past with RAID, it was very common that enlarging the storage space meant: get a bigger RAID with bigger disks, then copy everything over and switch to the new storage.

With data sizes currently common for professional AV content, this approach is a very delicate one that puts your data in quite some danger, as the time where data is the most vulnerable is when it's being moved.

Data Integrity?



Fixity information & Hashcodes :)

Hashcodes: fixed size number that's like a fingerprint for data.

- **CRC** =
4294967295
- **MD5** =
d41d8cd98f00b204e9800998ecf8427e
- **SHA256** =
e3b0c44298fc1c149afbf4c8996fb92427ae41e4649b934ca495991b7852b8f
- **xxHash** =
e4c191d091bd8853

Speaker notes

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 particularly to keep the chance for collision as low as possible.

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

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.

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.

Do it early, do it continuously.

- Create hashcodes early in the lifecycle.
- Use them when transferring files.
- Storage: Ongoing integrity checks matter.
- Consider runtimes & interference with daily work.

Ongoing integrity checks on your storage: Know where your collection is at.

It also makes sense to verify your backup that way.

Consider runtimes:

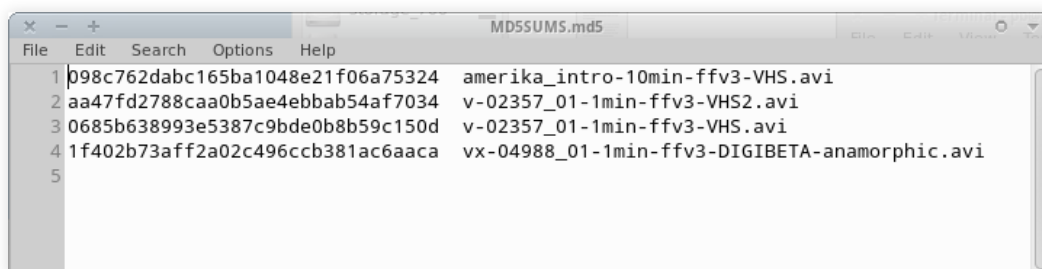
- In order to generate/verify a hashcode, the whole file must be read at least once.
- The chosen hashcode algorithm and its implementation may have impact on processing speed.
- Do some tests, calculations to find a sweet spot for how often you can do integrity checks.
- Consider throttling throughput data to avoid interference with daily work.

Real world example:

1. Archival Institution had hashcodes (ingest system created them), but never used them. They told me: "We don't need to verify hashcodes, we have no problems" So I checked. More than 1000 items had invalid hashcodes. Since they were never checked before, it is unknown when or what caused them - and also their impact on the material.

It's good practice to document the result of integrity checks somewhere. It helps finding issues in your setup, or at least narrowing things down.

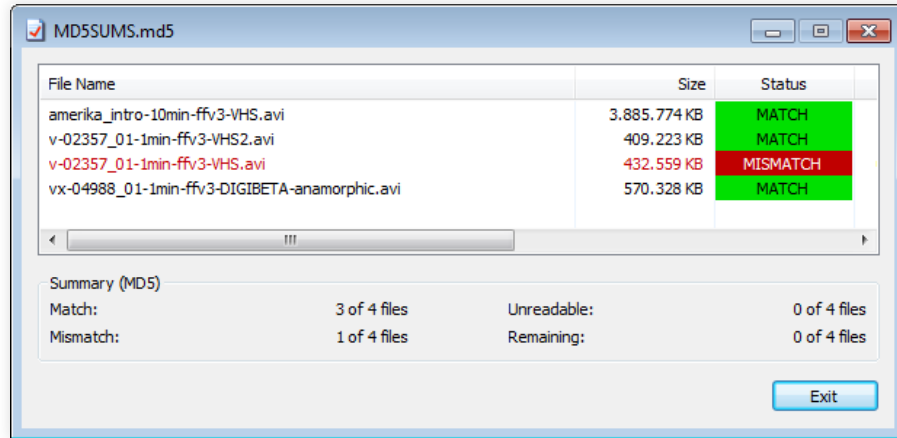
Hashcode manifest



As simple as that. Plain text.

Example for an MD5 file manifest

Using (=validating) hashcodes



HashCheck: GUI tool showing data validation

Speaker notes

Some tools

- [Fixity \(AVP\)](#)
- [HashCheck](#)
- [QuickHash GUI](#)
- ...

Our storage has integrity checks built in!

- Good! 🎉
- But it can only do so as soon as data has arrived.
- Did you verify that the data got there intact? 🤔

There are storage systems that have (continuous) integrity checks built in. This is very good! However, it still seems to be common to simply copy files there without checking if they have ever arrived without transfer errors.

If the whole transfer chain was fine, no problem - if there was any read/transfer error: Your storage could not tell you.


If a hashcode already exists at the source, you can use this to verify bit-proof transfer.

Exercise

The “cursed” digital archive folder.

What hashcode manifests can tell you beyond “*it’s borken*”, and how to interpret them.

Speaker notes

- Make groups of 2-3 people each.
- Download the fixity example zip:
http://download.av-rd.com/pub/exercises/fixity/hashcode-exercise_1a.zip
- Extract the files.
- Compare the manifests with the “reference” and try to spot issues (=differences).
- Try to find out what caused these issues.
- You may want to use a “diff” tool (e.g. [MELD](#) )

Backups

The 3-2-1 Backup Rule

- Keep at least **three** copies of your data.
- Store **two** backup copies on different devices or storage media.
- Keep at least **one** backup copy offsite.

Speaker notes

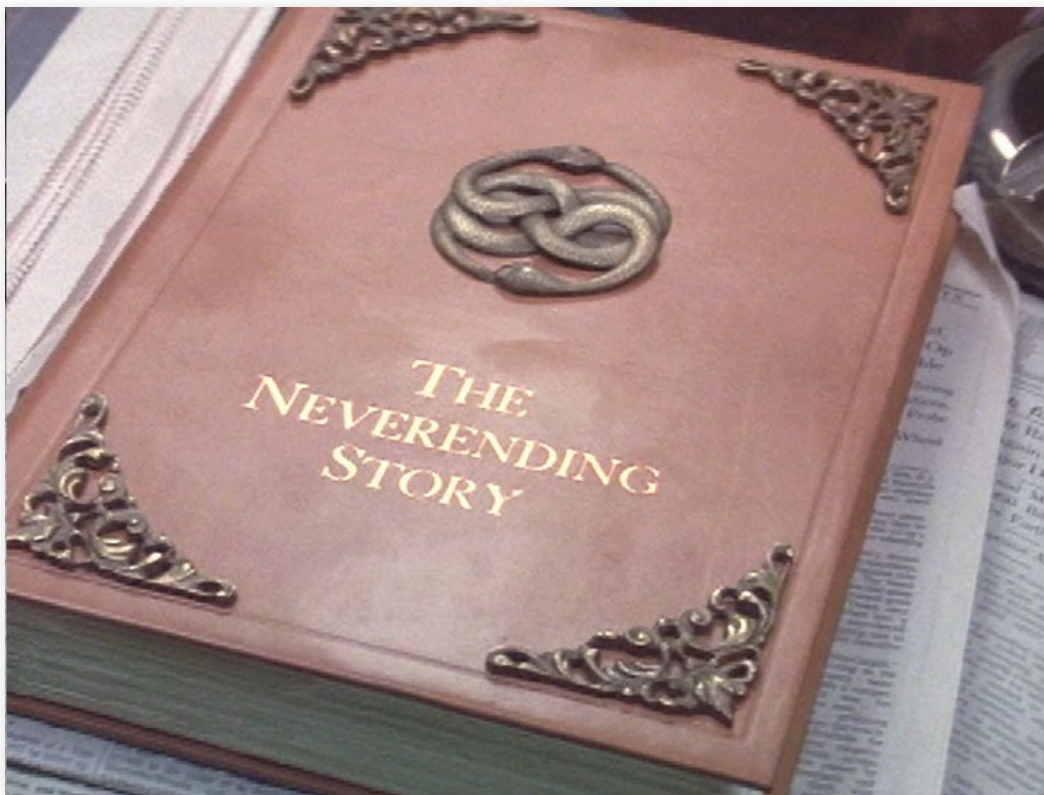
That's not a law, it's a rule :) It's good however to try to implement as much of it as you can.

If you can only do "2-1", it's better than not trying at all.

Backup integrity?

- btw: Is your backup copy (still) intact...?
- Did you check? ;)
- Tape: Backup drives on shelf vs robot?
- Handout: [Backup checklist](#)

Migration?



- There is no final carrier.
- There is no evergreen format.

Therefore fact: Any data must sooner or later be migrated.

Eternal Migration

- Always consider how to get your data out of any system before you get into it.
- Or at least while it's still running.
- As vendor- and technology neutral as possible.
- Try that before you buy.
- Include migration in your planning.

It's like brushing your teeth or cleaning things: It has to be done and is part of the natural life cycle. Don't avoid it - include it.



Which Storage is Right for YOU?



Must	Should	Could	Won't
------	--------	-------	-------

_____	_____	_____	_____
-------	-------	-------	-------

_____	_____	_____	_____
-------	-------	-------	-------

_____	_____	_____	_____
-------	-------	-------	-------

_____	_____	_____	_____
-------	-------	-------	-------

Choose a use-case and try to phrase your wishes and requirements.

Now, according to the MoSCoW method, write down which features you:

- must have
- should have
- could have
- won't have (this time)

And check which storage option provides them, then draw a dot in the corresponding circle area.

It makes perfect sense to fill these lists even if you're not a technician. Having such a list is then an excellent entry point for consulting someone to help you, fill and translate your demands into a proper "solution".

Storage: Summary

- There's more than just *one right* solution...
- Mixing is usually a good idea.
- RAID6 might be at its limits :(
- Consider LTFS and LTO generations
- Embrace hashcodes!
- Consider which level of control *you* have/want.
- Have a backup (plan)

- fin -

Questions? Comments?

Peter Bubestinger-Steindl

Peter @ [ArkThis.com](https://arkthis.com)

Storage Terms Collection

- **S.M.A.R.T.** Self-Monitoring, Analysis and Reporting Technology
- **NAS** Network Attached Storage
- **SAN** Storage Area Network
- **RAID** Redundant Array of Inexpensive Disks
- **Object Storage**
- **JBOD** Just a Bunch Of Disks
- **SDS** Software Defined Storage

Speaker notes

You don't have to become a storage expert, but It's good to have heard these terms and roughly know what they mean. This selection of terms is by faaar not complete, but these are quite likely to appear in a DLTP context.