Episode Book

EPISODE #5827

LTFS Explained: Tape That Acts Like a Disk

PUBLISHED	10 October 2026 at 10:05 UTC
DURATION	25:58
LISTEN	myweirdprompts.com/episode/ltfs-tape-disk-archive/
HOSTS	Corn, Herman, Hilbert
SCRIPT MODEL	deepseek-flash
VOICES	chatterbox-regular (chatterbox-local)
PIPELINE	V5.2
SOURCES	15 cited by the research (listed at the end)
BOOK GENERATED	19 October 2026 at 17:40 UTC

SYNOPSIS

LTFS turns LTO tape into something that browses like a disk, using a clever two-partition trick and a plain-XML index. But the physics of tape never went away: append-only writes, unreclaimable deletes, and a random seek measured at around 37 seconds. This episode walks through what LTFS actually is under the hood, then asks the practical question — if you had hundreds of terabytes of PDFs to make searchable, how would you build it? Disk caching, tape libraries, and hierarchical storage management all enter the picture, along with the honest verdict on whether LTFS can be a foundation for a document archive or whether sequential access always wins.

TRANSCRIPT



Corn

Daniel's been reading about tape again.



Herman

Oh no.



Corn

He has. And this time he's found the thing that makes tape dress up like a disk. LTFS. He wants the whole picture. What it actually is under the hood, how a medium that can only move forwards gets to pretend it's a directory tree. Then he wants the practical version. Suppose you've got hundreds of terabytes of PDFs and documents, and you want the whole lot browsable and retrievable through a server with LTO underneath it. How do you build that, what does disk caching and a tape library and hierarchical storage management actually do in the picture, and what does it feel like to use. And then the honest question at the end. Compared to just writing backup sets and restoring them when you need them, is LTFS really a foundation for a searchable document archive, or is sequential access always going to win in the end.



Herman

That's the right question to end on.



Corn

It's the only one that matters, really. Everything before it is plumbing.



Herman

Well. Let's start with the plumbing, because the plumbing is clever. LTFS is not just a format. It's a format and a driver together. It's an ISO standard now, and it started life at IBM. They demonstrated it at the National Association of Broadcasters show in two thousand nine, and released it publicly in two thousand ten with HP, Quantum and the LTO Consortium behind it. The reference implementation is open source under a BSD license. You run it on Linux and macOS through FUSE, which is the framework that lets a filesystem live in user space instead of the kernel. Windows gets a FUSE-like equivalent.



Corn

And the trick is the partitions.



Herman

The trick is the partitions. LTO-5 is where it arrives. Until then, a tape was one long ribbon and that was that. LTO-5 says the cartridge has two partitions, and you get to address them separately. The small one is the Index Partition. The big one is the Data Partition. The index partition holds the XML index, the metadata, the directory structure. The data partition holds the actual content of your files. And critically, the index partition can be rewritten without touching the data partition at all.



Corn

So you've got a little bookkeeping area at the front of the tape that you can keep open and scribble in.



Herman

That's the whole thing. And it's plain XML. Human-readable. File names, timestamps down to the eight-digit sub-second field, sizes, and then the part that matters, the extent lists. An extent list maps a byte range in the file to a block location on the tape. That's how the filesystem knows where a file physically lives. And there's no Unix user IDs, no permission bits, no Windows access control lists. That's deliberate. Everything platform-specific is left out so that the cartridge means the same thing on any system that can read it.



Corn

Which is the sales pitch, right? The tape describes itself.



Herman

You don't need the software that wrote it. You don't need a database. You need an LTFS driver and a tape drive, and if all else fails, the index is XML, so a determined person with a text editor can find the block and go get it. That was the point. The 2010 paper from David Pease's group at IBM put it plainly. Data on an individual tape couldn't be recovered without external databases and proprietary systems, and that's a serious problem when you're recovering from a catastrophe. The goal was to make data tape an equal member of the family of portable storage devices.



Corn

A thumb drive with a robot arm.



Herman

A thumb drive with a very patient robot arm. Yes.



Corn

So explain the part I always trip over. Why can't you just write to the middle of a tape? Why does everything have to append?



Herman

The head writes on a shingled path. It erases a wider track than it records. If you tried to go back and overwrite one block in the middle, you'd wipe the neighbors on either side. So the medium is append-only by physics, not by policy. LTFS doesn't change that. What it does is pretend. When you save a file, LTFS doesn't try to write it where the old version lived. It appends the new version at the end of the data partition and updates the index to point at the new extents. The old copy is still sitting there, physically untouched, just unreachable. The filesystem layer is what makes that feel like an edit.



Corn

And browsing.



Herman

Browsing is free, and this is the bit people underestimate. The index is read once when you mount the tape and then cached in memory or on local disk. Directory listings, searching for a filename, checking timestamps, none of that moves the tape. You're browsing a local cache that happens to describe a tape. That's why the experience doesn't feel like tape at all until you actually open something.



Corn

And old index copies?



Herman

Because the tape is append-only, every time the index changes, the previous version gets written into the data partition with a back-pointer to it. So you can roll a tape back. You can say, show me this cartridge as it was on Tuesday, and LTFS walks the chain backwards. There's also a redundant copy of the index at the very end of the data partition, so if the index partition gets damaged you can recover from the tail.



Corn

You'd never design that from scratch. You'd design it from the constraint.



Herman

Entirely. That's the whole personality of the format. Every feature is a workaround that got promoted to a specification. Deletes don't free anything, either.



Corn

Right, tell me about deletes.



Herman

Deleting a file marks the extents as unavailable. The bytes stay on the tape forever. The only way to reclaim that space is to reformat the cartridge and start again. So the capacity you bought is not the capacity you get back after a few years of churn. Tape isn't a workspace. It's a place things sit.



Corn

The little file trick. That one's good.



Herman

Small files can be stored in the index partition itself. When you format the tape you set criteria, either by size or by name pattern, and anything that matches gets parked in the index partition. Because the index partition is cached at mount, those files are just there. No tape motion at all. The classic case is media and entertainment. You've got a huge MXF video file, and next to it a tiny index file, a few hundred kilobytes, that describes the shot. You put the little file in the index partition and the giant file in the data partition. Someone browsing the tape opens the index file instantly and knows what the enormous thing next to it is, without the drive ever having to wind to it.



Corn

And how big is the index partition on LTO-5?



Herman

About thirty-seven and a half gigabytes out of one and a half terabytes. Two wraps minimum. The data partition is whatever's left after the guard wraps. So you've got a decent little cache at the front of every cartridge.



Corn

Library Mode.



Herman

Library Mode is where it stops being a tape and starts being an archive. When you've got a changer or a library with hundreds of slots, LTFS-LE presents all of them as folders under a single mount point. Every cartridge is a folder. And because it caches every tape's index to disk, you can list and search the contents of the entire library without mounting a single tape. You can grep across six hundred cartridges and the drive never moves. Then when you open a file, the robotics go and fetch the right cartridge and the drive reads it.



Corn

And how long does that take.



Herman

This is the number that decides the whole episode. Worst case on LTO-5, ninety to a hundred seconds. The measured average random seek in the Pease benchmark was around thirty-seven seconds. And it's independent of file size. A two-kilobyte text file and a two-gigabyte video cost the same seek.



Corn

Thirty-seven seconds to open a small file.



Herman

Give or take. It's a robot pulling a cartridge off a shelf and loading it into a drive. That's what the wait is.



Corn

So the browsing is disk-speed and the opening is shelf-speed.



Herman

Exactly the split. And the benchmark bears that out. One gigabyte files wrote at about a hundred and thirty-three mebibytes per second and read back at a hundred and thirty-two. One mebibyte files wrote at ninety-three and read at a hundred and thirty-three. Which looks fine, until you remember that the seek is thirty-seven seconds and it's per access, not per byte.



Corn

Then there's the constraint I keep circling back to. One tape, one folder.



Herman

Because the index describes one cartridge. A directory tree in the index is a tree that lives on that piece of media. There's no concept built into LTFS of a file that starts on cartridge one and finishes on cartridge four. Which is fine if you're shipping a finished project to a client. It's fatal if you're trying to present a single logical archive.



Corn

So how would you actually build the thing Daniel's describing? Hundreds of terabytes of PDFs.



Herman

You would not build it on bare LTFS. You'd build it on hierarchical storage management, and LTFS would be the layer at the bottom that nobody sees. The canonical example is IBM's Storage Archive Enterprise Edition. It sits on top of Spectrum Scale, which is their clustered filesystem, and it migrates files out of that namespace and onto tape when they go cold, and recalls them when someone opens them. From the user's perspective there's one namespace with a persistent view of the data and the tape is invisible. It scales to about five hundred petabytes with TS1155 drives and a couple of libraries, and it can keep up to three replicas of every file, with WORM tape support if you need the writes to be immutable.



Corn

The metadata is what makes it work.



Herman

Metadata on disk is the whole trick. QStar's volume-spanning product is a good example to look at, because they say it openly. They use a disk cache to store all the file locations, on disk as well as on the media, and they keep file metadata on disk and on the LTFS media both. That's what lets you search an archive of thousands of tapes. You are not searching the tapes. You're searching a database on a disk that knows which tape holds what, and the tapes are just where the bytes live.



Corn

And the spanning.



Herman

QStar's spanning is what turns the many-folders problem into one share. Tens, hundreds, thousands of cartridges appear as a single network share that grows as you add media. New tape gets added automatically when the old one fills. The folder you're browsing is imaginary. The files are on cartridges scattered across a library.



Corn

There's no open source version of that.



Herman

There isn't. That's the honest gap. If you want to reassemble a file that spans multiple tapes, you need either the object storage system or a specialized spanning tool, and there is no standalone open source LTFS product that does it. PoINT says so fairly bluntly. DIY spanning is not a thing you can install.



Corn

The library itself.



Herman

The library is the robots. Bar-code scanning identifies which cartridge is in which slot, and the automation pulls the right one and loads it into a drive on demand. IBM's Diamondback holds up to one thousand five hundred and forty-eight LTO-9 cartridges, which is twenty-seven petabytes native in a single unit. That's the machine you'd be putting in front of the storage manager.



Corn

And the experience for the person using it.



Herman

Browsing is instant, because it's the disk cache. Searching across the whole archive is instant, for the same reason. Opening a document that's cold is a tape load, so seconds to a couple of minutes, plus the drive read. For a PDF archive where most of the material is cold, that's fine. Reading a contract from 2009 is not a latency-sensitive operation. What's not fine is anything interactive. You can't do random access on tape. You can't scatter-gun little reads across a shelf of cartridges and expect anyone to be happy about it.



Corn

And that gets us to the small file thing.



Herman

PDFs are small files. That's the problem. A maintained GitHub issue from February last year, number four ninety-six, reports a hundred files of two kilobytes each taking between twenty-nine and thirty-six minutes to write. Not to read. To write. And in the same report, four-kilobyte files transferred instantly. The maintainer, piste-jp, explains why. LTFS does support partially updating a file, but every partial update creates more scattered extents, and the more scattered extents a file has, the slower it reads back. It degrades in a way that compounds. And that issue was closed last June with no fix. The pathology is still there.



Corn

That's not a performance number. That's a rejection.



Herman

It's a rejection of the PDF archive premise as stated. To get anywhere near acceptable throughput with a mass of small files, PoINT's guidance is that you have to read them sequentially, in the order they were written. Otherwise, and this is their line, the reading process will take days.



Corn

Days.



Herman

Days. And PoINT's 2024 critique goes further than that. They argue LTFS is a bad foundation for object storage at all. The forced index updates, the file-mark overhead, the alignment to block boundaries, all of it causes significant loss of read and write performance and inefficient use of capacity, and small files are the worst case. They also point out there's no file versioning, the filename rules are restrictive, and that the interchangeability benefit, the entire reason LTFS exists, becomes practically meaningless once you have a large number of tapes, because nobody can find the individual media without the management software anyway.



Corn

That's the contradiction the whole episode turns on. The format exists so you don't need the software. And at scale you always need the software.



Herman

It evaporates exactly where you'd need it most. Which is not a knock on the format. It's a limit on the promise.



Corn

The other side of the ledger. Security guard with a warning label.



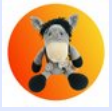
Herman

Archiware's line is the one to keep. Anyone expecting a mounted LTFS tape to behave like a very large USB thumb drive will be disappointed. Operating systems hang, they show errors, and anything that reads ahead for a preview or a thumbnail causes delays nobody predicted.



Corn

And the comparison to just tarring things up and restoring them.



Herman

Conventional tape use writes backup or archive sets. You're writing tar, or a proprietary format like TSM or Archiware P5, and you keep an external database that maps every file to its blocks on tape. The advantage is that the backup software can do things LTFS can't. Cloning, parallelization, spanning a single enormous file across multiple tapes, several jobs running at once, bare-metal recovery. All of that is built on the assumption that there's a catalogue doing the work. The disadvantage is the one LTFS was invented to fix. If you lose the database, the data is on the tape and effectively unreadable. You've got petabytes of bytes and no way to know which byte is which file.



Corn

So LTFS trades catalogue power for self-description.



Herman

It trades one for the other. And the trade is real in both directions. But here's where I'd land on it. LTFS is good as a transport and interchange format. Media and entertainment, film archives, a production house shipping a finished project to a distributor, that's where it shines. The tagline the LTO people like, bandwidth by the box. A flat-rate postal box holds twenty-eight LTO tapes, which is around forty-two terabytes, for fifteen dollars, and it arrives in about three days. That's roughly one point three gigabits per second. You cannot beat that with a network link and a budget.



Corn

You can't beat it with anything.



Herman

And the economics are absurd in the other direction too. Long-term SATA disk against LTO-4 tape runs about twenty-three to one on cost. The energy ratio is as high as two hundred and ninety to one. Tape is rated for thirty years. If you're storing cold bytes at scale, there's nothing close.



Corn

So the verdict.



Herman

As a foundation for a large searchable document archive, LTFS works only when it's wrapped in commercial HSM software that keeps the metadata on disk and uses LTFS as the tape format underneath. Which means the searchability is coming from the disk, not the tape. Bare LTFS gives you one folder per tape and a small-file pathology that directly undercuts the thing Daniel's asking about. The self-describing promise is real, and it's most valuable precisely when you're small enough not to need it.



Corn

Hold on. I want to push on that, because I think you're being too neat about it.



Herman

Go on.



Corn

You said the searchability comes from the disk and not the tape. Fine. But the reason the archive survives at all is that the tape doesn't need the disk. If the database dies, you lose the search. You don't lose the documents.



Herman

That's fair.



Corn

So it's a division of labour, not a bait and switch. The disk does the finding. The tape does the keeping. And the reason the tape can do the keeping for thirty years is that it doesn't need anything to read it.



Herman

That's better than what I said. I'll take that.



Corn

Don't get comfortable. The small file thing still kills the premise as stated, and I don't think any amount of architecture fixes it. If a hundred two-kilobyte files take half an hour to write, a document archive is the worst possible workload for this format. You'd be better off packing the PDFs into containers and giving up the per-file browsability, at which point you've reinvented the backup set with extra steps.



Herman

You've reinvented the backup set with extra steps and a nicer folder icon.



Corn

Right. Now, before we wrap, the thing that didn't fit.



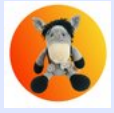
Herman

The MXF moe file. That's my favourite detail in the whole spec. In a film workflow, you've got a camera original that's enormous, and next to it a tiny sidecar file, a few hundred kilobytes, that describes the shot, the timecode, the reel, all the metadata the editor needs. The whole design of the partition split exists so that little file can live in the index partition and be instantly readable while the gigantic one sits in the data partition, never touched until someone actually needs the footage. It's a two-file relationship driving an entire storage format.



Corn

A whole industry's workflow, built around one very small file next to one very large one.



Herman

And it's why the format succeeded where it did. Not because it made tape into a disk. Because it made the important part of the tape into something you could read in a second.



Corn

There's something I keep turning over. The self-describing promise and the management software both being necessary.



Herman

Say more.



Corn

If you need the software to find the tape, and the tape exists so you don't need the software, then the value of the format is really a bet on which failure you think is more likely. A dead company, or a dead database. LTFS is insurance against the first. The backup set is insurance against nothing, it just outsources the risk to whoever maintains the catalogue.



Herman

Which is a real question for anyone planning thirty years out.



Corn

And it gets harder as the generations go up. If LTO-10 is thirty terabytes a cartridge, then a hundred-petabyte archive is a few thousand cartridges. At a few thousand cartridges, the self-describing property is a nice fact about each individual tape and completely useless as a retrieval mechanism. You can't walk a shelf.



Herman

The capacity growth makes the small files worse, too. It's the same index partition on a much bigger cartridge, indexing a much bigger pile of small files.



Corn

Bigger tape, same narrow throat.



Hilbert

Hewlett-Packard StorageWorks Ultrium 1840. Fifteen hundred dollars, refurbished, two thousand and nine.



Corn

Herman, you want to take that one.



Herman

LTO-4. Go ahead, Hilbert.



Hilbert

I didn't buy it. I priced it. A small regional newspaper up in the north wanted their archive moved. Back issues, photographs, the whole run. They had it on a pair of drives that were making a noise, and the editor wanted it on something permanent. So I spent a week doing the numbers on LTFS, and I gave him a written cost. Eighteen thousand pounds, six hundred tapes, three years to migrate.



Corn

Six hundred tapes for a small regional newspaper.



Hilbert

That's what I said. He didn't blink. Turns out the paper had been sold the year before to a group, and the group had been buying up local titles for a decade, and what the editor called the archive was about four hundred terabytes of scanned back issues and negatives across thirty-one titles. Nobody at the paper knew the number until I asked for it.



Herman

So it wasn't a small paper.



Hilbert

It was a small paper with a large landlord. The group's publisher owned the building the paper sat in, and he was my landlord at the time as well, because he owned the unit above the shop I was renting. He'd come down on a Thursday and ask how the tapes were going.



Corn

Did you do the job?



Hilbert

I did not do the job. I quoted, and the publisher took the quote to his brother-in-law, who ran an IT firm, and the brother-in-law said he could do it for half. So I walked. Two years later I got a call about the tapes.



Herman

From the publisher?



Hilbert

From the editor. The brother-in-law's system had written an index partition so full it stopped accepting new entries. Every tape in the cabinet had been formatted with the default and the default wasn't enough for the file counts they had. Nine hundred thousand scanned pages, most of them under sixty kilobytes. The index overflowed around tape forty, and after that the software kept writing data to the data partition that nothing could locate. Reformatting was the only fix, and reformatting loses the generations, so every rollback point they had went.



Corn

So they lost the ability to go back.



Hilbert

They lost the ability to prove anything about the sequence of the archive. The bytes were there. The story of the bytes was gone.



Herman

Which is exactly the PoINT objection, isn't it. The index is a fixed slice and the file count is not.



Hilbert

The file count is not. I told them at the quote stage. The daughter of the editor asked me, at the time, whether you could just make the index partition bigger, and I said you can set it at format time and you cannot set it after. That's the transaction. You pick the size before you know what you're storing.



Corn

And the cat.



Hilbert

The library was a four-slot changer in a converted stationery cupboard, and the stationery cupboard was in the same room as the accounts department, and the accounts department had a cat because of mice. The cat urinated on the changer. Once, on a Friday, over a bank holiday weekend. The changer never spun up again. Insurance wouldn't cover it because the policy didn't list livestock.



Herman

What happened to the tapes?



Hilbert

They went to another vendor and got read back with someone else's software, most of them salvaged, the last ninety or so gone. I kept one cartridge. It's on the shelf at home. Still sealed. Still has the index on it. I don't have a drive that reads LTO-4 anymore, so I can't tell you what's on it, and neither can anyone else.



Corn

You priced a job, you didn't get it, and you still have one of the tapes.



Hilbert

I asked for it. They were going to bin the damaged cartridges, and I said I'd take one. It's a good paperweight. It's a solid object.



Herman

That's the whole lesson, isn't it. The tape outlasted the newspaper.



Hilbert

The tape outlasted the newspaper, the drive, the man who bought it, and the cat.



Corn

And the index partition that would have told us what was on it.



Hilbert

The index is on there. There's just nothing that can read it.



Corn

So play that forward. If the tape survives the machine that reads it, and the index survives only as long as someone maintains a driver, then the interchangeability story is really a story about how long anyone bothers.



Herman

Which is a thirty-year question and a twenty-year format.



Corn

And it gets harder as generations go up. If the small-file problem scales with cartridge capacity, then the whole premise of using this as a browsable document archive gets worse over time, not better. Which means the case for LTFS is going to get narrower, not wider, as the years go on.



Herman

And the object storage crowd is already saying it. Flat namespace, metadata in an index, content addressable. That's the modern archive. If you want a searchable archive with a disk front end, the honest answer is that LTFS is the last mile and something else does the walking.



Corn

The sequential nature of the tape doesn't change. The index just hides it from you long enough to forget.



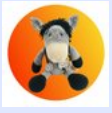
Herman

Right up until you open something.



Corn

Thanks to Hilbert Flumingtop for producing, and for the paperweight.



Herman

For more along these lines, there's episode eleven seventy-seven, The Race Against the Digital Dark Age; episode thirty-five, The Privacy Gap; and episode fifty-two oh three, Why LTO Tape Still Backs Up the Cloud. This has been My Weird Prompts. Send us your own prompt on Telegram at t dot me slash MWP listener bot, and we'll be back soon.



Corn

See you then.

SOURCES

What the research for this episode read before the script was written. Primary sources first.

1. **SNIA [primary]**
   Linear Tape File System (LTFS) Format Specification landing page
   https://www.snia.org/tech_activities/standards/curr_standards/ltfs

2. **SNIA [primary]**
   LTFS Format v2.5 Technical Position
   https://www.snia.org/sites/default/files/technical-work/ltfs/release/SNIA-LTFS-Format-v2.5-Technical-Position.pdf

3. **SNIA [primary]**
   David Pease, Linear Tape File System deck (2025-03)
   https://www.snia.org/sites/default/files/2025-03/DavidPease_LinearTape_File_System.pdf

4. **Pease, Amir, Villa Real, Biskeborn, Richmond, Abe [primary]**
   The Linear Tape File System, MSST 2010
   https://msstconference.org/MSST-history/2010/Papers/MSST/Pease.pdf

5. **GitHub [primary]**
   LinearTapeFileSystem/ltfs reference implementation (README, configure.ac, BSD license)
   https://github.com/LinearTapeFileSystem/ltfs

6. **IBM Documentation [primary]**
   IBM Storage Archive Enterprise Edition functions (last updated 2026-06-15)
   https://www.ibm.com/docs/en/storage-archive-ee/1.3.5?topic=overview-storage-archive-functions

7. **IBM [primary]**
   Storage Deep Archive 1.1.0 documentation (Diamondback library, 27 PB)
   https://www.ibm.com/docs/en/SSHZT2E_1.1.0/pdf/storage-deep-archive-1.1.0-documentation.pdf

8. **QStar [primary]**
   LTFS Volume Spanning for Tape Libraries datasheet (2017)
   https://www.qstar.com/download/QStar_LTFS_Volume_Spanning_for_Tape_Libraries_A4_1017.pdf

9. **Wikipedia**
   Linear Tape File System (spec history table, ISO/IEC 20919, limitations)
   https://en.wikipedia.org/wiki/Linear_Tape_File_System

10. **IBM Spectrum Scale User Group**
    Tiering to tape with IBM Spectrum Archive deck (up to 500 PB)
    https://www.spectrumscaleug.org/wp-content/uploads/2019/05/SSUG19UK-Day-2-B05-Tiering-to-tape-with-IBM-Spectrum-Archive.pdf

11. **PoINT Software & Systems**
    Why LTFS Is a Bad Choice for Tape-Based Object Storage Systems (2024-03-12)
    https://point-blog.de/2024/03/why-ltfs-is-a-bad-choice-for-tape-based-object-storage-systems/

12. **Archiware Blog**
Understanding LTFS, What it's for and how we use it (2019-01-17)
https://blog.archiware.com/blog/de/understanding-ltfs-what-its-for-and-how-we-use-it/

13. **LTO.org / Media Technology Market Partners**
LTFS Hits the Mark in Media & Entertainment (2012)
https://www.lto.org/wp-content/uploads/2014/06/2012_MTMP-LTFS-Hits-the-Mark.pdf

14. **GitHub**
LTFS issue #221, Slow read performance (2020, closed 2024)
https://github.com/LinearTapeFileSystem/ltfs/issues/221

15. **GitHub**
LTFS issue #496, Extremely slow write performance for files 4kb (2025, closed 2025-06-02)
https://github.com/LinearTapeFileSystem/ltfs/issues/496