Episode Book

EPISODE #5705

Tape Isn't a Pool: How LTO Really Works on Linux

PUBLISHED	3 October 2026 at 14:23 UTC
DURATION	30:53
LISTEN	myweirdprompts.com/episode/lto-tape-linux-integration/
HOSTS	Corn, Herman, Hilbert
SCRIPT MODEL	deepseek-flash
VOICES	chatterbox-regular (chatterbox-local)
PIPELINE	V5.2
BOOK GENERATED	19 October 2026 at 17:39 UTC

SYNOPSIS

Daniel asked a simple question: once you've bought the tape drive, how does tape actually get integrated into a Linux server? The answer starts with a surprise — there is no driver to install, because LTO support has lived in the mainline kernel for decades. But the deeper answer is that tape refuses the mental model most of us bring from ZFS and LVM. A library does not appear to the operating system as a single expandable block device. There is no pool that grows when you add a cartridge. Each cartridge is an independent volume with a fixed capacity, and "pool" in backup software means a catalog, not a device. Along the way: the st versus nst device distinction that quietly destroys archives, why tar's default block size can make a modern LTO drive run 24x slower than it should, what LTFS actually buys you and why it isn't packaged for Debian, and why the real hardware compatibility question is the SAS card, not the drive.

TRANSCRIPT



Corn

There's a line in the Linux tape driver documentation that stops you cold the first time you read it.



Herman

One sentence. In the middle of a very dry manpage.



Corn

"There is no corresponding block device."



Herman

Which sounds like a footnote. It is not a footnote. It is the entire episode.



Corn

Here's what Daniel sent in. He's been thinking about the LTO episode we did, the one about tape as a medium and why the real barrier for a normal person is the cost of the drive rather than the tape itself. His point is that once you've decided to spend the money, tape is not exotic. Inside the IT supply world it's a completely ordinary commodity. What he wants to look at is the layer above the hardware. How tape actually gets integrated into a computer system.

Herman

He built us a hypothetical. A Linux server with a small internal SSD. One cron job runs daily, finds the new podcast episodes, writes them to tape. A second job archives Python function logs. Then he asks the questions.



Corn

First, how is a tape pool exposed to the operating system as read-write storage. Second, what packages and drivers do you actually need. Third, how much of this works out of the box on Ubuntu or Debian. And fourth, the one that turns out to be the real subject. Since tape is physical, how do you get notified when the current cartridge crosses a storage threshold, so somebody remembers to add the next one. Does an entire tape library show up as a virtual block device that just grows when you add cartridges, or do you provision a pool to a fixed size before you start?



Herman

And that last one is where the whole thing lives or dies. Because most listeners, if they've used ZFS or LVM, have a mental model of storage as something that expands. You add a disk, the pool gets bigger, the filesystem is oblivious.



Corn

Tape does not do that.



Herman

Tape does the opposite of that.



Corn

So the driver situation first, since it's the part that sounds hard and isn't.



Herman

This is the revelation of the episode for me. LTO support on Linux is part of the mainline kernel. It's in there through the SCSI tape module, and the reason it's so mature is that IBM contributed heavily to it. So the honest answer to "what drivers do I need" is none. You do not install a driver for a tape drive. The kernel already has one, and it's been there for decades.



Corn

Which is not the way most people expect this to go. The expectation with specialized storage hardware is a vendor disk, a downloaded package from a support portal, a license key.



Herman

No. The tape driver in Linux is one of the oldest pieces of the storage stack. What shows up is a character device. You get something like slash dev slash st zero, and if you have more drives, st one, st two, numbered from zero. There's a parallel set of devices for the SCSI generic driver, the sg devices, which is how the operating system talks to the changer and the drive at a lower level.



Corn

And the st devices are the actual interface.



Herman

The st devices are how you write to tape. The driver takes control of every device it detects that identifies itself as sequential access, which is the SCSI device class tape falls into. It uses major device number nine, and each physical drive gets eight minor numbers. Now the eight minor numbers matter more than they sound like they do, because they're the single most common way people accidentally destroy an archive.



Corn

Explain.



Herman

For each drive you get a set of devices. There's the principal device, st zero, and there's the no-rewind variant, nst zero. The difference is what happens when you close the file. On the plain st device, closing the file sends a rewind command to the drive. So your tape spools all the way back to the beginning. On the no-rewind device, it just stops where it is. Which means if you have a backup that runs in stages, and you write to the wrong device between stages, the second stage lands on top of the first one at the start of the tape.



Corn

And the first stage is gone.



Herman

The first stage is gone, and there is no warning, because the drive did exactly what it was told. That's the rewind-on-close behavior. It is a feature for single-shot archives and a landmine for anything else.



Corn

The manpage is very clear that the driver doesn't know anything about specific drives either.



Herman

It doesn't. It explicitly does not support any particular tape drive brand or model. What it does is probe the drive at startup and then hand control over to the drive's own firmware. So the capabilities, the block sizes, the compression, the cleaning behavior, all of that comes from the hardware itself. Since kernel two point six, the driver also exports the devices under sys class scsi tape, so you can find them without guessing which sg number is the tape.



Corn

So the test for whether a drive works is embarrassingly simple.



Herman

You look for a device in slash dev slash tape slash by id. If there's an nst device sitting in there, the kernel has already found your drive and the driver is already attached. That's it. You haven't installed anything.



Corn

The caveat is the card.



Herman

The caveat is the host bus adapter, the SAS controller the drive is plugged into. The tape driver is in the kernel, but a SAS card that Linux doesn't have a driver for is a card Linux doesn't have a driver for. Which in practice means buying a known-good card. People gravitate toward the LSI and Broadcom parts, because those have been in servers for years and the drivers are rock solid. But that is the actual hardware compatibility question. It is not the tape drive. It is the card between the drive and the motherboard.



Corn

So what do you install?



Herman

Three things, and only one of them is arguably required. mt-st, which gives you the mt command, that's the tape control tool, it's how you query status, set block sizes, turn compression on and off, rewind, retension. lsscsi, which lists the SCSI devices so you can see what the kernel thinks is attached. And sg3-utils, which gives you sg_inq, a little utility that asks a device what it is. Those are a single apt command. That's the whole install.



Corn

And yet some people do none of that and the drive still works.



Herman

The drive still works. You can echo a file at the device and it will go onto tape. What you lose is your ability to control anything. Without mt you can't ask what's on the tape, you can't set the block size, you can't ask whether the drive is asking for a cleaning cartridge. You're flying blind on a medium where flying blind is expensive.



Corn

Now the software model. There are two of them, and they're different ways of thinking about tape.



Herman

The simple one is tar. You point tar at the device and it writes an archive. And the one thing that everybody gets wrong the first time is the block size. The default block size tar uses is small, and tape drives hate small blocks. If you write to a modern LTO drive in tiny chunks, the drive never gets going. It spends its time waiting for the next block to arrive.



Corn

There's a number attached to this.



Herman

Seven megabytes a second. That's what one guide measured writing to tape with tar's default block handling. Then they set the block size flag to five hundred twelve blocks, and it went to one hundred seventy megabytes a second. Same hardware, same tape, same machine. The only change is how much data tar hands the drive at once. That's not an optimization. That's the difference between tape being unusable and tape being fast.



Corn

Twenty-four times faster.



Herman

Because tape is a streaming medium. The drive wants a continuous river of data, and if you hand it a trickle, it stops, backs up, repositions, starts again. The mechanical work of stopping and restarting is expensive in a way that disk people have no intuition for. A disk that gets a small write just does the small write. A tape drive that gets a slow stream does a dance.



Corn

And the other knob is compression.



Herman

LTO drives compress in hardware by default. For most data that's fine and it's free throughput. But if you already compressed the archive in software, or if you're writing data that doesn't compress, you're asking the drive to do work that produces nothing. So you turn it off with the mt compression command and you manage it yourself. And this matters because compression changes how much fits on the tape, and you need to know how much fits on the tape.



Corn

Then there's the other model, which is LTFS.



Herman

LTFS is a real filesystem. It lives on the tape itself, on a two-partition layout, and the effect is that a cartridge looks like a removable drive. You mount it, you see files, you can drag things on and off. It is the closest tape has ever come to feeling like a disk. You format the cartridge once with mkltfs, then you mount it with the ltfs command pointed at the generic device.



Corn

And it is not installed by default.



Herman

It is not packaged for Debian. The upstream project's own documentation says Debian package, not available yet. So on Ubuntu and Debian you compile it from source. And there's a wrinkle there, which is that the reference implementation used a part of the kernel's proc interface that Debian removed, so the clean version doesn't build. A lot of people end up using the HP variant of LTFS, which is a patched fork that works. That's a real friction point and it's worth knowing before you start.



Corn

There's a warning in the LTFS documentation that sounds like a typo and is not.



Herman

You must not touch any st device while LTFS is mounting a tape. Because opening the st device sends a rewind, and a rewind in the middle of LTFS assembling its index is a corrupted tape. The LTFS tooling wants to talk to the drive through the generic device, and if something else reaches in and rewinds it, the format comes apart. If you must poke the drive manually while LTFS has it, use the no-rewind device. That's the whole reason the no-rewind device exists.



Corn

There's a third path, which is the enterprise backup software.



Herman

Bareos, which is the community fork of Bacula, and Bacula itself. Those use tape as a storage backend and they don't require LTFS at all. They have their own catalog, their own volume management, their own notion of which tape holds which job. If you're running a serious backup environment, that's the road. For a single server with two cron jobs, it's a lot of machinery.



Corn

Which brings us to the actual question. The one the episode is about.



Herman

The capacity question. And here's the answer stated plainly. A tape library does not present to the operating system as a single expandable storage block. There is no device that grows when you add a cartridge. There is no filesystem-level pool that expands. Each cartridge is an independent volume, and the operating system sees one drive, and if you have a robot, a medium changer device for the robot. That's the whole surface area.



Corn

So when you add a cartridge, the filesystem doesn't get bigger.



Herman

Nothing gets bigger. The filesystem doesn't know a cartridge was added. There's nothing for it to notice. The operating system's view of tape is a device you can open and write a sequential stream into, and that stream ends when the medium ends. The word "pool" does not mean what it means in ZFS or LVM. When backup software talks about a tape pool, it means a logical catalog of independent cartridges. It's a list, not a device.



Corn

Which means Daniel's question answers itself in a slightly annoying way.



Herman

It does. There's no provisioning step where you declare a pool of a certain size, and there's also no expansion step where you add a cartridge and the pool grows. Neither of those things exists. What exists is a set of cartridges, each with a fixed capacity decided when it was formatted, and a piece of software that keeps track of which one is currently in the drive and what's on it.



Corn

So capacity is a bookkeeping problem.



Herman

Capacity is a catalog problem. And that's the sentence I'd put on the wall. On disk, capacity is a filesystem problem. The filesystem knows about free blocks and inodes and it grows or it doesn't. On tape, capacity is a catalog problem. Which tape holds what. And that catalog lives in software, in a database or a text file or a human being's memory, not in the operating system.



Corn

Now let's talk about what actually happens when a tape fills up.



Herman

The drive tells you. The driver returns a specific error, ENOSPC, and the manpage language is worth repeating because it's so plain. A write operation could not be completed because the tape reached end of medium. There's a companion condition, end of tape, which is physical end of the media rather than the end of the formatted space, but from your software's point of view it's the same event. The write fails. The job fails. And then nothing happens, because there is nothing that can happen.



Corn

There's no automatic failover to the next cartridge.



Herman

Only if your software does it, and most software doesn't. tar can do it. tar has a multi-volume flag and a tape-length flag. You tell tar the length of a volume, and it spans across cartridges, prompting you to swap. There's a volume number file that tracks where you are so a resumed run knows which tape to ask for. It works. But the manpage has a warning that catches people out, which is that for reliable multi-volume archives the driver has to be configured a particular way. Buffered writes and asynchronous writes have to be turned off. Because if the driver is buffering and you hit end of medium, the tape can be left in a state where the volume boundary isn't where the archive thinks it is.



Corn

So spanning works, conditionally.



Herman

Spanning works, conditionally, with tuning, and for tar specifically. Now here's the thing. LTFS does not span. LTFS mounts one cartridge as one filesystem. When it fills, you unmount it, you take the tape out, you put the next tape in, and you mount that. There is no automatic spanning across a library in the base LTFS model. The SNIA material notes that in library mode you can list and search all the volumes without mounting them, so you can find things. But the mount itself is per cartridge. So you have two models and they fail differently. tar spans and it's opaque. LTFS is transparent and it doesn't span.



Corn

Pick your poison.



Herman

Pick the one that matches how you'll read the data back. If you're restoring a podcast archive, tar is fine. If you're pulling one file out of a dozen cartridges, LTFS is the difference between minutes and a very long afternoon.



Corn

And the capacity of each cartridge is fixed when you format it.



Herman

Fixed at format time. When you create an LTFS filesystem on a blank tape, it's laid down at that tape's native capacity, which means the uncompressed number. And here's the detail that surprises people. The native capacity is typically about half the number on the box. LTO eight, for example, is twelve terabytes native and thirty terabytes with compression. So you format the tape, and LTFS says you have twelve terabytes, and you stare at the box that says thirty.



Corn

Because the thirty assumes you're feeding it compressible data.



Herman

Because the thirty assumes compression, and compression is a property of your data, not your tape. If you're writing already-compressed archives, or video, or encrypted backups, you will get close to the native number and nowhere near the advertised one. And planning a backup schedule around the box number is the mistake that fills your tape at sixty percent of what you budgeted.



Corn

Let's do the threshold notification, because this is the part Daniel specifically asked about and I suspect it's thinner than people expect.



Herman

It is thinner than I expected, and I went looking properly. Here are the building blocks that exist. You have the mt command, which can report status on the drive and the tape. You have tapeinfo, which reports the drive and media state including whether hardware compression is enabled. You have the driver's status query, which returns a set of flags including the cleaning request bit, that's the drive saying it wants a cleaning cartridge. And you have the changer control program, mtx, which moves cartridges between slots and drives and can list what's where.



Corn

So the pieces are all there.



Herman

The pieces are all there. What does not exist, as far as I can find, is a canonical published script that watches a tape, notices it's past some threshold, and emails you. I could not find one. Not a package, not a well-known gist, not a documented recipe. People clearly write these. The building blocks are documented, so anybody competent can assemble them. But there's no standard answer, and I think that's notable, because this is the single most operationally important thing in a small tape setup. The tape filling up isn't a crisis. The tape filling up and nobody noticing is a crisis.



Corn

Because the daily cron job just fails.



Herman

The daily cron job fails, quietly, and writes an error into a log nobody reads. And you find out three weeks later when you try to restore something, and discover that the last three weeks of podcast episodes exist in exactly one place. Tape is supposed to be the second copy. If the second copy stopped existing three weeks ago, you've been running without a backup and you didn't know.



Corn

Which is exactly the failure mode tape is supposed to prevent.



Herman

It's an unforced error. And the fix is a script that checks the drive's status, reads the remaining capacity, and sends an alert at a threshold you choose. Which is a fifteen-line script. But somebody has to write it, and there's no template, and that's a real gap.



Corn

There's one more operational thing that I think people miss, and it's the one that bites.



Herman

The sustained throughput requirement.



Corn

The drive needs to be fed continuously. Around one hundred sixty megabytes a second, give or take, on modern LTO. If the source can't keep up, the drive stalls, backs up, and resumes. And every stall costs you time and wear.



Herman

And here's the cruel part. Your source in Daniel's hypothetical is a cheap SSD. Small consumer SSDs have a cache. They're fast until the cache is full, and then they fall off a cliff, sometimes to a tenth of the rated speed. So the first few gigabytes write beautifully, the drive is delighted, and then the SSD's cache runs out and the write rate collapses, and the tape drive starts doing its stop-and-go dance, and the backup that should take forty minutes takes four hours, or fails outright.



Corn

And it fails at the worst time.



Herman

It fails on the large archive, which is the one you actually care about. The small daily job that writes forty megabytes of podcast episodes sails through, because the SSD cache covers it. The big monthly archive that writes four hundred gigabytes hits the cliff and falls apart. So your testing passes and your production fails, and the difference is scale, which is the least helpful kind of difference.



Corn

So what do you do about it.



Herman

You source from something that can sustain the rate. A spinning disk will actually do better here than a cheap SSD in many cases, because a spinning disk's sequential throughput is honest. It's not fast, but it's sustained. Or you pace the feed, or you use a tool that can read ahead and buffer. But the point is that the tape drive is not the bottleneck. The tape drive is the fastest thing in the chain and it's the one that suffers when the rest of the chain can't keep up.



Corn

There's also the file size thing.



Herman

Tape is bad at small files. The seek time reading back from tape can be measured in tens of seconds, because the drive has to spool to a position. So if you dump ten thousand small files onto an LTFS tape, reading them back one at a time is painful. The advice is to bundle them. Make a tar archive, then put the archive on tape. Then you have one large sequential object, which is what tape is good at, and the filesystem overhead disappears.



Corn

Which is a nice piece of symmetry. tar is the simple model and it's also the right model for small files.



Herman

The simple model is right more often than people expect. LTFS is the one that feels better and behaves worse in the common case.



Corn

Give me the Windows contrast, because I like it.



Herman

The same hardware that works without any installed drivers on Debian is, in the words of one writeup, hopeless on Windows ten and up. Microsoft broke the file copy interface for sequential access devices. The recommendation in the community is simply to use Linux. Not because Linux is ideologically superior, but because the tape support is older, more direct, and hasn't been deprecated out from under the people using it.



Corn

Older means it still works.



Herman

Older means nobody has had a reason to break it. The kernel tape driver is doing the same job it did twenty years ago and doing it the same way, and that's exactly what you want from the piece of software that holds your archive.



Corn

Herman, let me push on the capacity thing one more time, because I want to make sure we're not underselling how strange it is.



Herman

Go ahead.



Corn

On a disk, when I create a filesystem, I'm making a promise about how much space is available. The filesystem knows the size and it enforces it. On tape, the cartridge has a capacity, but nothing in the operating system knows or cares. The cartridge is just a piece of media that ends.



Herman

Right. And the consequence is that the operating system has no opinion about your total archive capacity. It cannot report it. There is no command that says you have forty terabytes of tape in this library. Because that fact isn't a fact about the operating system. It's a fact about a catalog. That's why the backup software exists. Bacula and Bareos aren't doing anything clever with the hardware. They're maintaining a database. A database of which tape holds which backup on which date, and which tapes are free, and which are full. The cleverness is entirely in the bookkeeping.



Corn

And a person can do the same bookkeeping with a spreadsheet.



Herman

A person absolutely can, and plenty of people do, and it works fine until it doesn't. Which is the point. The catalog is as reliable as whoever maintains it. And this is where I'd push back on the framing of Daniel's question slightly. He asks whether the library presents as a virtual block device that expands, or whether you provision a pool. The honest answer is that there's a third option he didn't list, which is that the pool is in your head and in a text file, and that's the one a lot of small setups actually run on.



Corn

Which sounds fragile.



Herman

It is fragile in exactly the way human memory is fragile. It works perfectly for two years and then somebody goes on holiday.



Corn

Hold on. Say the thing about the barcode again.



Herman

Libraries label cartridges with barcodes. The robot reads the barcode to know which tape is in which slot, so it can inventory the library without physically loading every tape. And the label is often the same as the volume label recorded on the tape itself.



Corn

So the physical object and the logical record are meant to match.



Herman

They're meant to match, and the robot is designed to trust the barcode. Which is fine, and which is also the seam where things go wrong, because a barcode can be right while the tape inside the cartridge is not the tape the label claims. If somebody swaps a cartridge and reads the wrong label, the software has no way to know.



Hilbert

The label is usually right. I saw one where it wasn't.



Herman

Go on.



Hilbert

Twenty-three years. That was the whole run of it. Small operation, one library, maybe two hundred cartridges, and it had a card catalog. Actual index cards, in a wooden drawer, one card per tape, with the job name and the date written in pencil.



Corn

Somebody maintained that by hand.



Hilbert

The operator did. One man, and he did it every morning before the first job. The software already knew which tape was where. He knew that too. He wrote it down anyway, because he'd watched the software be wrong twice, and he didn't want to be in the position of explaining to somebody that the tape with the payroll on it was the tape with the telephone directory on it.



Corn

So the catalog was the backup for the catalog.



Hilbert

The card catalog was for when the software catalog was wrong. And it did get used. Twice that I know of. You'd go into the drawer, find the card, and the card would tell you a different slot than the software, and the card would be right, because the card had been written when the tape was physically in the man's hand.



Herman

Which is a kind of confirmation that's actually stronger than a database entry.



Hilbert

It's stronger because it was written while he was holding it. The software writes when the robot moves it, and the robot can be told the wrong thing. The card was written when somebody looked at the cartridge.



Corn

What happened to it.



Hilbert

The card catalog went when the operation moved. The tapes went to a depot somewhere and I assume the cards went out with the rest of it. I have one of the tapes. My brother-in-law gave it to me, it's got his handwriting on the label. I don't have a drive for it. Nobody does, it's a format we stopped making.



Corn

So you can't read it.



Hilbert

I can't read it. He could, at the time. That was the whole thing. Somebody had to be able to read it, and for a few years somebody could. Now nobody can, and the tape is still perfectly good. Thirty year shelf life. It'll outlast all of us and it won't mean anything to anybody. That's the part people don't put in the slides. The medium survives. The ability to use it doesn't.



Herman

That's the honest version of the archive problem.



Hilbert

The card catalog was the same thing. It wasn't about storage. It was about somebody paying attention. And the reason it worked for twenty-three years is that for twenty-three years, the same man came in every morning and paid attention. When he stopped, that was the end of it, not a fire, not a failure. He stopped and nobody replaced him.



Herman

The threshold notification problem isn't a scripting problem.



Hilbert

It's a person problem. You can write a script that emails somebody, but somebody has to read the email, and somebody has to walk over with the next tape. That's how it works. It has always worked that way. The technology is fine.



Corn

Hilbert's point lands somewhere uncomfortable, and I want to sit with it for a second. We spent this whole episode on the mechanism. The character device, the kernel module, the mount, the spanning behavior. And the thing that actually keeps an archive alive is that somebody walks over with the next cartridge and puts a card in a drawer.



Herman

And that's the failure we can't fix with a package.



Corn

It's a interesting place for tape to end up. Because on disk, the system does the work. You don't have to remember anything. You add a disk, the pool grows, the filesystem doesn't care. Tape doesn't do that, and it never will, and I think the argument Hilbert's making is that this is not a bug. If tape were frictionless, you'd treat it like a disk, you'd fill it, you'd forget about it, and you'd discover the problem the same way you'd discover it on a disk, which is when you need the data. The friction drags a human into the loop. And a human in the loop is the difference between an archive and a pile of tapes.



Herman

There's a real tradeoff buried in there, though. Friction is only a feature if the person is paying attention. The same friction that keeps you honest is the friction that lets the job fail quietly for three weeks when the person is on holiday.



Corn

Which is Hilbert's point exactly.



Herman

Right. And I don't think you can have one without the other. The system where the human is essential is the system where the human is a single point of failure. Which means a good setup does both. It keeps the human in the loop, and it makes sure the human failing is loud. The script that emails at a threshold is not a substitute for the person walking over with the tape. It's a way of making sure somebody notices when the person doesn't. And that's the piece I couldn't find documented. The building blocks are all there. The mt status query, the tapeinfo report, the threshold bit. Nobody's packaged the obvious script, and I think that's because the obvious script is arguably the least interesting part of the problem, and also the one most likely to save your archive.



Corn

So if you've solved it, we'd like to hear how.



Herman

Whether it's a cron job, a monitoring check, a systemd timer, whether it emails you or pages you or blinks a light. There's no canonical answer out there and there could easily be one.



Corn

Where I'd leave this is the part about where the software layer actually lives. The hardware question is settled. A tape drive on Linux works without any driver install, works on the same mainline kernel support it's had for two decades, and the same hardware is effectively broken on Windows. That's a striking asymmetry. The work is not in getting the drive seen. The work is in configuring the drive, in understanding that a cartridge is not a block device and never will be, and in building the bookkeeping that tells you which tape holds what and when to change it. That's where a sysadmin earns the money. Not in the installation. In the catalog.



Herman

And the catalog, in the end, can be a database, or a spreadsheet, or a drawer of index cards. Same problem, different medium. The medium is not really the interesting part. The interesting part is that somebody has to decide what happens when it fills up, and there's no software that decides that for you.



Corn

One forward thing, and then we're out. Tape is getting cheaper per gigabyte and disk is getting cheaper per gigabyte and the two lines have crossed back and forth for twenty years. The question worth watching is whether the friction itself becomes the selling point. An archive that you have to maintain deliberately is an archive you can't forget about. Which is either the reason to use tape or the reason to avoid it, depending on what kind of person you are.



Herman

Our thanks to producer Hilbert Flumingtop.



Corn

This has been My Weird Prompts. If you got something out of this dive into the operating system side of tape, send it to somebody who's stared at a full cartridge and wondered what happens next. And if you've built the threshold alert script we couldn't find, email us at show at my weird prompts dot com.



Herman

We'll be back soon.