Episode Book

EPISODE #6708

Why Grep Is Not Enough for Search

PUBLISHED	20 November 2026 at 14:27 UTC
DURATION	25:47
LISTEN	myweirdprompts.com/episode/grep-ripgrep-inverted-index-23722/
HOSTS	Corn, Herman, Dorothy
SCRIPT MODEL	qwen/qwen4-max
VOICES	omnivoice
PIPELINE	V5.36
BOOK GENERATED	20 November 2026 at 21:20 UTC

SYNOPSIS

A scan across 22 million words takes less than a tenth of a second with modern tools like ripgrep, yet full-text retrieval still requires an entirely different data structure. Exploring the mechanics of grep, regular expressions, and benchmarks across source code and subtitles reveals where linear scanning hits a wall. The discussion covers why relational B-tree indexes fail on natural text and how the inverted index solves structured queries like negation and phrase ranking.

TRANSCRIPT



Corn

Daniel wants to build a search engine over our own archive. Not actually build one, he's careful about that, but walk the whole thing from nothing and use six thousand-odd episodes of transcripts as the test pile. Where do you start, when does reading every file stop being good enough, and what is the first real structure you'd put underneath it.



Herman

The inverted index, which he already knows, so the interesting half is everything before it and everything after.



Corn

Right. Tens of millions of words of two brothers talking. Somebody should be able to find things in there.



Herman

This is part one of five, by the way. We're going to take the primitives in order and not skip ahead.



Corn

Five episodes on finding words. I've waited my whole life for this.



Herman

You've waited your whole life for most things.



Corn

Slowly, and with results.



Herman

Let me put the corpus on the table honestly first, because the numbers are a mess and the mess is useful. Our own data page says twenty-two point six million words, five thousand three hundred and twenty-nine episodes. That's dated September. The Hugging Face card for the transcripts dataset says six thousand five hundred and two episodes and six hundred thirteen thousand speaker turns. There's another artifact in between at six thousand and four.



Corn

Same archive. Four different sizes.



Herman

Same archive, four different days. It rebuilds daily from the production database. And the docs say so outright, they warn that a figure measured against main cannot be reproduced tomorrow, and if you want a number that stays put you pin a snapshot or a commit.



Corn

So before we've indexed anything we've already got a reproducibility problem.



Herman

Which is exactly why an index is interesting here. A scan is a measurement of a moment. An index is a snapshot you can hold still and argue about.



Corn

Fine. Now tell me the thing I think you're going to say, which is that twenty-two million words is a lot of text and grep will choke on it.



Herman

No. That's the part I want to lead with, actually, because it's backwards. Twenty-two million words is roughly a hundred and thirty to a hundred and fifty megabytes of plain text. That's it. That's a large email attachment.



Corn

A hundred and fifty megabytes.



Herman

With ripgrep, a literal pattern over a gigabyte of English subtitles runs in about a quarter of a second. Scale that down to our pile and you're well under a tenth of a second to read every word we have ever said.



Corn

Under a tenth of a second.



Herman

And I want to flag that as arithmetic, not a measurement. Nobody has benchmarked a podcast transcript corpus of this shape. The subtitle benchmarks are the closest thing, right order of magnitude, different kind of text. But the conclusion isn't fragile. Grep is already fast enough for us.



Corn

Then the index isn't buying speed.



Herman

It isn't. It's buying the ability to ask questions a scan can't answer. Ranking. Negation across a whole collection. Phrase queries you don't pay for again every time. Counting how many episodes mention something without touching every byte again. Capability, not throughput.



Corn

Good. Because I was going to spend the episode defending grep and you've taken that away from me.



Herman

You can still defend it. It deserves it.



Corn

The name is the algorithm, isn't it. Grep comes out of the ed command, g slash re slash p. Global, regular expression, print.



Herman

That's the whole tool. Loop over lines, match a pattern, print the ones that match. Ken Thompson wrote it in a night, more or less, because somebody asked for the ed feature as a standalone thing.



Corn

A utility named after its own instruction sequence. Nothing has been named that honestly since.



Herman

And the regular expression part goes back further than Unix. Stephen Cole Kleene formalized regular languages in the fifties, purely as mathematics, no computer involved. It sat there for fifteen years before anybody pointed it at a text file.



Corn

Fifteen years of theory waiting for a log file.



Herman

Then it splits. POSIX syntax and Perl syntax diverge through the eighties and nineties, and now you've got two dialects that look identical until they very much aren't. Which is why every search tool eventually grows a flag to switch engines.



Corn

Ripgrep has that.



Herman

Andrew Gallant's pitch for it is that it combines the usability of The Silver Searcher with the raw performance of GNU grep. Written in Rust. Recursive directory walking, respects your gitignore, filters by file type, Unicode on by default, PCRE2 if you opt in, compressed files if you ask, and arbitrary preprocessing filters so you can pipe PDFs through a text extractor before matching.



Corn

You installed it the week it shipped, I assume.



Herman

I installed it the week it shipped and broke my own dotfiles inside a day, because I aliased grep to rg and about nine scripts that depended on grep's exact exit behaviour stopped working.



Corn

Nine scripts.



Herman

Somewhere around nine. I stopped counting when I found the one that had been silently returning nothing for a week.



Corn

A tool that reports itself fine while doing nothing. Your favourite kind of patient.



Herman

It's the symptom nobody mentions that tells you what's wrong. Anyway. The numbers, because they're surprising in both directions. On the Linux kernel source tree, searching for PM underscore RESUME, ripgrep took nought point one three seconds. Grep took one point one five.



Corn

Nearly nine times.



Herman

Nearly nine. On a gigabyte of English subtitles looking for Sherlock Holmes, ripgrep nought point two six eight, grep nought point five one six. Barely twice.



Corn

Same tools, same idea, wildly different margin.



Herman

Wildly. There's a later run on a nine hundred and sixty-two megabyte file with multiple patterns where ripgrep came in at a hundred and thirty-two milliseconds against grep at one point eight seven seconds. Fourteen times. And the summary line on the benchmark page is honest about it, the margin ranges from about one point three times to thirty-six times depending on the tool and the pattern.



Corn

Thirty-six being the marketing number and one point three being the one you get.



Herman

One point three being the one I got on subtitles, yes. The reason the kernel tree is so dramatic is file count. Thousands of small files, and ripgrep parallelizes across them and skips the ones your gitignore excludes. Grep walks them serially and reads all of them.



Corn

Our archive is thousands of small files.



Herman

Which is the good case for ripgrep and the bad case for grep. Six and a half thousand transcripts, none of them enormous.



Corn

Then why would I ever use grep.



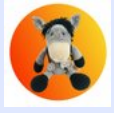
Herman

Because Gallant says so himself, which I love. His own anti-pitch section says if you need a portable and ubiquitous tool, ripgrep works on Windows and macOS and Linux but it is not ubiquitous and conforms to no standard, and the best tool for that job is good old grep.



Corn

The author writing his own limitations section. Rare.



Herman

He also says, plainly, I'm hearing reports from some people that rg isn't as fast as I've claimed on their data. And there's a whole discussion thread titled a case where rg is slower than grep.



Corn

Named that himself, did he.



Herman

Someone else opened it. He engaged with it for pages. There's a real edge case in there about memory mapping, where tools searching many files at once are generally slower with mmap, not faster, which is the opposite of what everybody assumes.



Corn

Everybody including you, last spring, in this room.



Herman

Including me, last spring, in this room. I was wrong then and I'm wrong now about different things.



Corn

So we've got a scanner that reads our entire life's work in a blink. Where does it actually fall apart? Because it clearly does or nobody would have built anything else.



Herman

Ask it a question with a shape. Which episodes mention spectrum allocation but never mention Ofcom.



Corn

It reads everything twice.



Herman

It reads everything, finds the matches for the first term, then for each of those files it has to go back and check the absence of the second. Two passes minimum, and it's doing that from scratch every single time you ask. Change one word in the query and the whole cost repeats.



Corn

Whereas an index has both lists sitting there already.



Herman

Both lists sitting there, sorted, and you walk them against each other. The work of finding out which documents contain a word has been done once, at index time, and paid for.



Corn

And counting. If I want to know how many episodes have ever said the word anteater...



Herman

You don't want to know that.



Corn

I very much want to know that, and I want the number to be zero.



Herman

With a scan you read a hundred and fifty megabytes to produce one integer. With an index you look up one term and read the length of a list.



Corn

Zero, ideally.



Herman

It's not zero, Corn, you bring it up.



Corn

Preemptively. That's different.



Herman

There's a lovely historical precedent for exactly this path, by the way. agrep, approximate grep, Udi Manber and Sun Wu, late eighties into ninety-one. It picks among several string-searching algorithms including bitap, which is built on Levenshtein distance, so it tolerates typos.



Corn

Grep that forgives you.



Herman

And the same engine became the search component inside GLIMPSE, which is an indexer. They hit precisely the wall we're describing, and their answer wasn't to throw grep away. It was to keep the matching engine and put an index in front of it so it only had to look at candidate files.



Corn

Thirty-five years ago somebody made the same argument we're making and shipped it.



Herman

And the architecture is still right. Filter cheaply with an index, then do the expensive exact matching on the survivors.



Corn

Before we build anything, I want to know why a normal database index doesn't just do this. Postgres has indexes. I've used them. They make things fast.



Herman

They make a different thing fast. A B-tree index stores an ordered mapping from row values to row locations. Ordered is the operative word. It's brilliant for range queries and exact lookups, give me every order between March and June, give me the customer with this exact ID.



Corn

Because it's sorted, so you can jump.



Herman

Because it's sorted. But the value it's sorting is the whole field. Nobody searches for the entire contents of a Wikipedia article. You search for a word inside it. A B-tree over a transcript column indexes the transcript as one enormous string, which is useless, because the thing you want is somehere in the middle.



Corn

It's an index of documents. I want an index of what's in them.



Herman

That's the inversion, and it's literally where the name comes from. A forward index maps document to contents. An inverted index maps content to locations. Word goes in, list of places comes out.



Corn

And that's the most used data structure in retrieval, isn't it. Not one option among many.



Herman

It's the default everywhere. Also decades older than the web, which people forget. Mainframe database systems in the seventies, ADABAS, Model two oh four, built on inverted list architectures. Web search inherited it rather than inventing it.



Corn

Now it is time for another edition of Dorothy's History of Toast.

D

Dorothy

It is me, dear, with Dorothy's History of Toast.

D

Dorothy

The first pop-up toaster for the home, the Toastmaster, went on sale in nineteen twenty-six.

D

Dorothy

You would think that would be the start of something lovely, but it mostly reminds me that things do not work out, dear, they just pop up and then they are done.

D

Dorothy

Anyway. Have you had your breakfast? I have left some bread out for you and I am going to put the kettle on. Bye bye now.



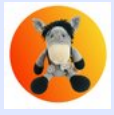
Herman

Morning, Mum.



Corn

Two variants of the index, you said. Record level and word level.



Herman

Record level is the cheap one. For each word, a list of which documents contain it. That's all. Word level additionally stores where in the document each occurrence sits, and that's what makes phrase queries possible.



Corn

Because "spectrum allocation" as a phrase means those two words next to each other, not merely both present.



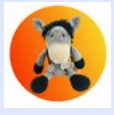
Herman

Both present in a forty-minute transcript is nearly meaningless. Two words appearing somewhere in four thousand words tells you almost nothing. Adjacent tells you everything. But you pay for positions in space and in build time.



Corn

How much space.



Herman

Considerably more. Every occurrence now carries an offset instead of just a document appearing once in a list. A word that shows up eighty times in one episode is one entry in a record-level index and eighty entries in a positional one.



Corn

Which takes us to the words that show up eighty times in every episode.



Herman

Yes. There's a from-scratch implementation on complex.codes that builds a tiny toy index over three documents, and you can see the problem immediately in their printed output. The word "do" has fourteen postings. Across three documents. "The" has six.



Corn

Three documents and "do" is already the largest thing in the index.



Herman

Scale that to six and a half thousand transcripts and the posting list for "the" is longer than most of the documents it points into. You've built a data structure whose biggest entries are the ones nobody will ever search for.



Corn

Every tutorial has that moment where the elegant hashmap meets the word "and".



Herman

And that's the whole reason stopwords, compression and skip lists exist. They're not optimizations somebody added later for fun. They're responses to a problem you can see in a three-document example.



Corn

Walk me through the pipeline properly, then. What actually happens when I hand you six thousand transcripts.



Herman

Four stages. Documents, analyze, index, query. Analyze is tokenization, you turn the string into tokens. The primitive version is lowercase the entire business and pull out every run of word characters with a regex.



Corn

Which is wrong in ways I can already guess.



Herman

It's wrong in ways you can guess and ways you can't. Hyphens split, so "long-term" becomes two tokens. Apostrophes vanish or split depending on your pattern. Hebrew gets handled however your definition of a word character happens to feel that day.



Corn

That last one matters here. Half our proper nouns are transliterated inconsistently.



Herman

Constantly. And the tokenizer is where you decide whether Jerusalem and jerusalem are the same term, whether a hyphen joins or separates, whether a number is a word. Every one of those decisions is invisible at query time and completely determines what you can find.



Corn

The unglamorous half.



Herman

The half that decides whether the glamorous half works. Then index, which is the hashmap. Word maps to a list of document identifier and token position pairs. Then query, where you turn the user's little expression into set operations over those lists. Intersection for AND, subtraction for negation, adjacency check on positions for phrases.



Corn

Eighty lines of Python, according to one write-up I've seen.



Herman

Alex Molas did it in about eighty lines and he's candid that the title cheats, because the crawler and the API and the templates are extra. But the search engine proper really is under eighty. His index is a nested dictionary, word to document to count. Four lines to add a document.



Corn

Counts, not positions.



Herman

Counts, so no phrase queries in his version, and he's fine with that because he wanted ranking rather than phrases. Different tradeoff, deliberately chosen.



Corn

You'd take positions.



Herman

I'd take positions, because for transcripts specifically the interesting queries are phrases. People quote us. Somebody wants the episode where you said sloths invented pizza.



Corn

Somebody wants that episode for evidence purposes.



Herman

They'd need adjacency, otherwise they get every episode where both words appear anywhere, which given your habits is a substantial fraction of the archive.



Corn

The pizza claim is load-bearing.



Herman

Now, what a real one looks like underneath, because the gap between eighty lines and Lucene is instructive. Lucene's index has two parts. A terms dictionary, which is the vocabulary, and the postings, which are the lists. The dictionary is loaded into memory as a finite state transducer, which is a compressed automaton that shares prefixes and suffixes between terms.



Corn

So the vocabulary itself is squeezed.



Herman

Squeezed hard, and it doesn't necessarily load every term. The postings stay on disk and they're compressed with real aggression. Document identifiers are delta encoded, so instead of storing five thousand, five thousand and twelve, five thousand and nineteen, you store five thousand, twelve, seven.



Corn

Small numbers compress better.



Herman

Much better, and then they're packed into fixed blocks of a hundred and twenty-eight integers, all encoded at the same bit width. If the widest delta in the block needs seven bits, all hundred and twenty-eight use seven bits, which sounds wasteful until you realize it means the machine can unpack them without branching. Leftover deltas that don't fill a



Herman (continued)

block go in as variable-length integers.



Corn

A hundred and twenty-eight because it fits nicely somewhere.



Herman

Because it fits nicely in cache and in the vector registers, yes. It's a hardware-shaped number, not a mathematical one.



Corn

And the skip lists.



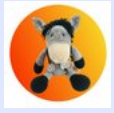
Herman

Skip lists are how you avoid reading a posting list you don't need. In the classic Lucene format the default skip interval was sixteen. If you want to jump forward a hundred documents, you read six skip entries and then four actual entries instead of a hundred.



Corn

Ten reads instead of a hundred.



Herman

Roughly. And that's what makes AND queries fast, because intersecting two lists is mostly jumping past the parts where they can't possibly meet. The Jira ticket where this got designed has a line I like, the skip interval is always a compromise. Smaller interval, faster skipping, bigger index.



Corn

Sixteen being somebody's judgement call that stuck for twenty years.



Herman

Somebody's judgement call that stuck. There's no theorem that says sixteen.



Corn

Here's the bit I actually care about, though. Our corpus changes every day. Does the index have to be rebuilt from nothing every morning?



Herman

No, and this is the design decision I find beautiful. Lucene's index is append-only and made of immutable segments. When you commit, it writes a new segment containing the new documents. It never edits the old ones. A search covers all segments and merges the results.



Corn

Never edits. So a correction to an old transcript...



Herman

Gets handled as a deletion marker plus a new document, and the actual removal happens later during a merge. The old bytes sit there until then.



Corn

Which means yesterday's answer is still available while today's is being written.



Herman

Exactly the property you want for a daily rebuild. You index the new episodes into a fresh segment, the existing six thousand stay untouched and searchable throughout, and the lot never goes down.



Corn

And the snapshot problem from the top of the episode solves itself, almost. Each commit is a pinned state of the world.



Herman

Each commit is a pinned state. Which is more than the live dataset gives you.



Corn

The other unglamorous thing. Our own data dictionary says several fields are well under half populated, and the gaps aren't random, they follow eras of the show.



Herman

That sentence should be tattooed on every search engineer. A field that's empty for the back catalogue is missing data, not a measured zero.



Corn

Meaning if I search for episodes with no guest, I get four thousand hits that mean nothing.



Herman

You get four thousand hits that mean the field didn't exist yet. And your index will happily report them as matches, because an index has no opinion about what absence means. Median episode length went from about three thousand words to about four thousand two hundred across the last two years as well, so even the documents themselves changed shape mid-collection.



Herman

Which reminds me. I'm putting a card index in for the flat. Proper one, oak drawers, the sort you buy from a library clearance. Starting Sunday. I've priced the cards at eleven pounds a thousand from a supplier in Kent and I want four thousand to begin with.



Corn

For what.



Herman

Correspondence. Letters, postcards, the gas board, everything paper that comes into this flat. One card per item, filed under the significant words, and on each card I write the drawer, the folder and the position within the folder. Not the contents. Where the thing physically is.



Corn

You want a word index that resolves to a shelf.



Herman

I want to be able to stand up and put my hand on it. Knowing a letter exists is worthless. I had a dispute with the records office at the municipality over a document they said I'd never submitted, and it ran eleven weeks because I could prove it existed and not prove where it was. Man called Dov Berenstein handled it, decent fellow, kept apologizing for the form. He wasn't the problem. The problem was their intake desk logged receipt without location.



Corn

Eleven weeks over a filing convention.



Herman

Eleven weeks. Dov used to hold the phone away from his mouth when he sighed, which I always appreciated. Anyway. Mine won't do that. Drawer, folder, position, on every card, in pencil so it can move.



Corn

Pencil.



Herman

Pencil. And I'll be adding the new cards for the backlog first, because there's about two hundred items in the sideboard that were never indexed at all.



Corn

The strangest part of that is the pencil and you've said it like it's nothing. Why pencil.



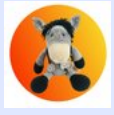
Herman

Ink.



Corn

Right.



Herman

No, that's the answer, ink smudges when you handle a card four hundred times. I learned that doing stock ledgers at a warehouse in Ashdod, briefly, a long time ago. Pencil doesn't lift.



Corn

Two hundred items in the sideboard, four thousand cards, and a man in Kent who's about to have a very good week.



Herman

Eleven pounds a thousand. It's cheaper than the eleven weeks.



Corn

The thing that stays with me is the position within the folder. He's not indexing letters, he's indexing where letters are, and those come apart the moment anybody tidies a drawer.



Herman

Which is the same problem as a document identifier that points at a file somebody moved. The index stays perfectly correct and perfectly useless.



Corn

Ours rebuilds daily, so at least the pointers get refreshed. His survive on pencil and discipline.



Herman

Positions cost you storage and they cost you maintenance. That's the tradeoff nobody puts in the tutorial, because in the tutorial nothing ever moves.



Corn

Six thousand transcripts that never move, versus two hundred letters that do.



Herman

And the moment you allow the collection to change, you need the immutable-segment machinery, the deletion markers, the merges. All of that exists because paper gets rearranged and files get renamed. The engineering is downstream of things moving.



Corn

I'd have said the engineering was downstream of things being numerous.



Herman

Numerous is easy. A hundred and fifty megabytes is nothing. Moving is hard.



Corn

Then the honest version of where we've landed is that we spent thirty minutes discovering our archive is small.



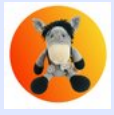
Herman

Small and awkward. Which is most collections.



Corn

The index tells us which episodes contain a word. It has no idea which of those episodes anybody would actually want to read first.



Herman

That's part two. Ranking and scoring, and why counting occurrences turns out to be a surprisingly defensible place to start.



Corn

Something to look forward to.



Herman

Part two, then. Ranking.



Corn

Bring the pencil.