Episode Book

EPISODE #6307

Why Exact Matching Outlasts the Index

PUBLISHED	27 October 2026 at 18:36 UTC
DURATION	29:47
LISTEN	myweirdprompts.com/episode/grep-find-locate-search-57953/
HOSTS	Corn, Herman, Larry, Hilbert
SCRIPT MODEL	openai/gpt-6.1-sol
VOICES	chatterbox-regular (chatterbox-local)
PIPELINE	V5.22
SOURCES	12 cited by the research (listed at the end)
BOOK GENERATED	28 October 2026 at 09:53 UTC

SYNOPSIS

Searching without building a massive index remains surprisingly durable. While modern developer tools often rely on complex indexing, streaming Unix utilities like grep, find, and locate still handle immense workloads by balancing query-time scanning against stale precomputed paths. Exploring Ken Thompson's original 1973 assembly implementation, the Boyer-Moore skipping algorithms behind modern matchers, and the trade-offs of live directory walks versus cached path databases shows why literal matching without semantic approximation remains so effective.

TRANSCRIPT



Corn

Why does a search tool that understands absolutely nothing keep finding the thing we need? Daniel wants to begin a short run on search before the semantic era, with grep, find, locate, and the descendants that still do much of the actual work inside our editors. Where they came from, what happens underneath, and why exact matching remains so formidable.



Herman

The interesting word there is understands. Searching for a function name doesn't require understanding the function. It requires finding those characters without missing the file, silently changing the question, or consulting yesterday's copy.



Corn

Yesterday's copy has excellent response times.



Herman

Sometimes excellent enough to justify keeping one. That's the argument between find and locate, and it survives inside modern coding tools. Claude Code removed its early vector database and let the model search with ordinary tools. That doesn't make every index foolish. It makes the cost of maintaining one part of the engineering decision.



Corn

I also want to defend the allegedly dumb half of this arrangement. A tool that does exactly what I asked leaves me fewer mysteries than one that has thoughtfully substituted something nearby.



Herman

Then nineteen seventy-three gives us an unusually good place to begin. A text editor had a useful piece trapped inside it.



Corn

Before you liberate it, Herman, there's a distinction people lose when they say they're searching a computer. Finding a file named invoices and finding the word invoices inside a file are different jobs. The terminal makes that division visible. A desktop search box often conceals it.



Herman

Find primarily walks directory entries and tests properties. Names, types, timestamps, and other metadata. Grep reads content and matches text. Locate asks a database about recorded paths. Those are different sources of truth, with different expenses.



Corn

My guess is locate wins whenever I keep forgetting where I put something, and find wins whenever I've only just put it there.



Herman

Broadly, yes. Although find isn't looking at an atomic snapshot of the whole machine. Files can change while it walks. Live means it consults the filesystem during the search, not that time stops.



Corn

A disappointing omission from Unix.



Herman

Grep makes another bet. It can consume a stream without first building a searchable collection. Give it a file, a pipe, or a sequence of files, and it can start returning matches as it reads. The expense belongs to the query rather than to a separate indexing phase.



Corn

Which sounds wasteful when I search the same directory repeatedly. You read it yesterday, you read it this morning, and now you're reading it because I misspelled the name.



Herman

It can be wasteful. An index amortises preparation over later queries. If the collection is huge and the queries frequent, that's attractive. If the collection keeps changing, or you're visiting it once, the preparation can cost more than it saves. Corn, would you wait for an index before searching a freshly unpacked project?



Corn

No. I haven't decided whether I want the project yet. Requiring a preparatory relationship feels presumptuous.



Herman

That's a genuine advantage of the streaming approach. Immediate usefulness. The claim worth testing is how far that advantage carries, not whether all search should happen without an index.



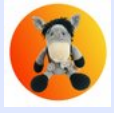
Corn

You were about to make the streaming mechanism sound elegant. I can hear you clearing space for it.



Herman

It is elegant. I'll try not to make you regret noticing.



Herman

Ken Thompson wrote grep at Bell Labs. It first shipped in November nineteen seventy-three with Version Four Unix, in PDP-eleven assembly. Its name came from an ed editor operation: global, regular expression, print. Apply a pattern across the text and print



Herman (continued)

matching lines.



Corn

A name compressed until only its owner could love it.



Herman

The practical motivation involved Lee McMahon analysing The Federalist Papers, including questions of authorship. This was text someone actually wanted to investigate, not a demonstration invented to justify a utility. Ed already understood regular expressions, but an editor needs to move around and change text. It loaded the file into memory.



Corn

Searching was paying for editing privileges it didn't need.



Herman

Thompson extracted the recogniser into a standalone program that could process text sequentially. Read, test, emit, continue. The total input could exceed what the editor could conveniently hold. More precisely... the program didn't need to retain the whole input merely to decide whether the current line matched.



Corn

That's the part I like. He removed a requirement instead of acquiring a larger computer. Software people don't always remember that's allowed.



Herman

Doug McIlroy then asked Thompson whether he could lift the recogniser out of the editor and make a one-pass program. Thompson said he'd think about it overnight. He already had a private search utility, called s, and spent about an hour cleaning up bugs before presenting it the next morning.



Corn

The overnight breakthrough included a reasonable bedtime.



Herman

McIlroy got what he wanted. The story often becomes a miracle of sudden invention, when the better story is recognising that an existing component could become a generally useful tool. Thompson had already done the thinking. Someone else's need helped make it public.



Corn

There's also a difference between writing a thing for yourself and handing it to McIlroy. Suddenly the peculiar input you never use becomes somebody's entire afternoon.



Herman

You're defending the bug-fixing hour?



Corn

Fiercely. It's the hour people remove from the legend and then wonder why their own overnight software fails.



Herman

McIlroy later called grep the prototypical software tool. Small scope, useful output, and no insistence on being the whole environment. The printed matches can feed another operation. It doesn't need a project format or a session database to participate.



Corn

Does the line-oriented decision limit it, though? If the evidence crosses a line boundary, the tidy little tool has made an assumption about my document.



Herman

It has. Traditional grep is built around lines. Different implementations and options extend that, but the original interface chooses a record boundary that works beautifully for a lot of text. Small tools still embody assumptions. They're just easier to see.



Corn

Then the real question is whether a tool that only streams and matches, without building an index, can keep up when you repeat the search.

L

Larry

You know the feeling. You have a stream of text and no index, no prebuilt database, just a live filesystem and a question. That is when you reach for Grepitol, the little streaming capsule that walks your whole filesystem every single time, exactly like find, and matches your regular expression with the global, regular expression, print philosophy of nineteen seventy three.

L

Larry

Grepitol is not a shortcut and that is the point. Side effects may include a sudden urge to run egrep and fgrep at the same time, a faint taste of Bell Labs coffee, uncontrollable patience while your disk spins, an hourly habit of reindexing nothing at all, mild disappointment when locate returns something stale, and in rare cases a total refusal to approximate, which doctors call exact match triumph. Ask your terminal if Grepitol is right for you.



Herman

...especially when the repeated searches benefit from the operating system's file cache. No persistent search index doesn't mean every byte must come off a spinning disk again. Recently read data may already be in memory.



Corn

Hm.



Corn

That matters. People compare a warm search with a cold one and credit the entire improvement to the program whose name they typed last.



Herman

Pattern shape matters too. Modern grep implementations combine techniques. For literal substring searching, Boyer–Moore and related approaches can skip ahead instead of testing every candidate position character by character. A mismatch tells you something about where a match cannot begin.



Corn

It gets paid for noticing where not to look.



Herman

Suppose you're searching for a distinctive word. Examining a character near the end of the candidate window can rule out several starting positions at once. That's very different from blindly restarting a comparison at every character. Regex engines can also extract required literal pieces and use those to reject most input cheaply.



Corn

Meaning a supposedly complicated expression can be fast because one ordinary word inside it does most of the elimination.



Herman

Yes. Then Version Seven Unix, in nineteen seventy-nine, brought egrep and fgrep. Egrep supported extended regular expressions, with Alfred Aho's work expanding the syntax. Fgrep handled fixed strings, including lists of them, without treating punctuation as regex instructions.



Corn

Fixed strings are underrated. Sometimes a dot has done nothing wrong.



Herman

Searching many fixed strings is particularly interesting. Aho–Corasick builds a machine from the pattern list, then walks the input while tracking which prefixes you've encountered. Shared prefixes share work. You don't scan the whole file independently for every word on your list.



Corn

That's preparation inside a query, though. We shouldn't accidentally tell people grep survives by refusing all preparation. It prepares a matcher. It simply doesn't require a persistent index of the corpus.



Herman

That distinction is important. Compile the question once, apply it across the input. Different cost from preparing the entire collection before anyone asks anything.



Herman

Modern egrep and fgrep are often wrappers around grep's extended-expression and fixed-string modes. Their names are ubiquitous, but they aren't the POSIX interfaces to rely on. The standardised route is grep with the appropriate option.



Corn

Familiarity has been impersonating a specification again.



Herman

Do you actually prefer separate names, Corn, or one command with modes?



Corn

Separate names are easier to remember until I have to move a script between machines. Then I prefer the boring standard. My future self shouldn't need an archaeological grant to run a report.



Herman

There was an approximate branch too. Udi Manber and Sun Wu developed agrep around the turn of the nineties. Its bitap approach uses machine-word bit operations to track possible matches, including versions allowing edit distance, such as substitutions, insertions, and deletions. It also powered searching inside the GLIMPSE indexer.



Corn

We'll leave misspellings their own episode. Today the computer is allowed to hold me to what I typed.



Herman

Grep had become enough of a cultural object to enter the Oxford English Dictionary Online as both noun and verb in December two thousand three. The machinery outlived the hardware and then became ordinary language.



Corn

The dictionary indexed the indexless search tool.



Herman

Find makes a complementary choice. It recursively visits a directory tree from your chosen starting point. It can test names, file types, modification or access times, and combine those conditions. It can also perform actions on the matches.



Corn

That last part changes the mood. Finding old files and deleting old files may share a predicate, but they shouldn't share my level of confidence.



Herman

A preview matters when an action is destructive. Also, your starting directory matters enormously. Searching a project and searching an entire machine are different amounts of work, with different permission boundaries and possibly slow mounted filesystems.



Corn

Find doesn't know that the network share is where enthusiasm goes to die.



Herman

Locate avoids that walk at query time. A separate process, often updatedb on a schedule, gathers paths into a database. Shared portions of successive paths can be compressed rather than stored repeatedly. The query searches that prepared representation.



Corn

It can return a file that has disappeared and miss a file that has just arrived. Both answers are fast, which is apparently a mitigating circumstance.



Herman

For finding an old manual or an obscure installed library, that may be a perfectly good bargain. Some implementations can check whether reported paths still exist. That can remove false positives, but it can't conjure a new path the database never recorded.



Corn

I'd still use it. I just wouldn't use absence from locate as evidence that the file doesn't exist.



Herman

Nor would I call the work free. The scan happens on a schedule whether anyone searches or not. Frequent queries can justify that easily. An infrequently used machine may spend more effort maintaining the answer collection than answering questions.



Corn

A machine working while nobody watches is nearly my ideal. A machine doing unnecessary work while nobody watches is, admittedly, closer to yours.



Herman

I install monitoring afterward.



Corn

That's how it becomes two machines.



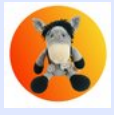
Herman

Ack, ag, and ripgrep inherited these decisions, then adapted the user experience to source trees. Ack's contribution wasn't merely matching text. It made searching code convenient, with less time spent excluding things you didn't mean to search.



Corn

Build output is often the largest collection of things I didn't mean.



Herman

The Silver Searcher, ag, followed explicitly. Geoff Greer created it in November two thousand eleven, in C. Its own description calls it similar to ack, with a focus on speed, and its README says, without ack, ag would not exist.



Corn

An unusually civil software genealogy.



Herman

Its recipe is practical. Search files across cores with threads. Use memory mapping where appropriate. Optimise literal matching. Let PCRE compile regex work efficiently, and prepare the expression once before applying it across files. Even ignoring files gets attention, so the program isn't repeatedly doing expensive pattern matching merely to decide what to skip.



Corn

Memory mapping sounds like the elegant mechanism you warned me about.



Herman

It maps file contents into the process's address space and lets the operating system manage the underlying pages. That can avoid some copying and simplify access. It isn't automatically faster for every workload… no, that's too broad. Small files, filesystem behaviour, and the surrounding implementation can change the outcome.



Corn

Your elegant mechanism has terms and conditions.



Herman

Ag's own benchmark made the attraction concrete. Ack took about a minute and fifty seconds; ag took about three seconds. Same reported results, roughly thirty-four times faster.



Corn

Enough time to reconsider using ack, then finish the search.



Herman

The directory was much larger than the text actually searched. Ignore rules excluded most of it. That makes the result relevant to a developer's workflow, but it also means you must distinguish matching speed from traversal and filtering choices.



Corn

I don't object to skipping generated files. I object when somebody calls it a faster search without admitting they searched fewer things. The useful comparison is whether they answered the same question.



Herman

Ripgrep arrived in March two thousand sixteen, written in Rust by Andrew Gallant, also known as BurntSushi. Its explicit ambition combined the usability of ag with the performance of GNU grep. Recursive searching, ignore-file support, and sensible treatment of hidden files and binaries are central features.



Corn

Sensible until the missing setting lives in a hidden file. Then a convenient default becomes a small personal betrayal.



Herman

You can disable the filtering and make it search hidden, ignored, and binary content too. That's an important diagnostic step when you know a string exists but the default search won't show it. The selection rules are part of the query, even when you didn't type them.



Corn

That's the first thing I check when one of your newly installed tools returns nothing. Not whether the computer is broken. Whether the tool has opinions.



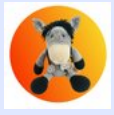
Herman

You could ask me.



Corn

You usually explain the opinions before admitting they are enabled.



Herman

Fair. What would you make the default, then? Search everything and drown people in dependencies, or skip the common rubbish?



Corn

Keep the convenient default. Make exclusions visible when somebody investigates a miss. I don't need every generated bundle at breakfast, but I do need a clear way to interrogate what was omitted.



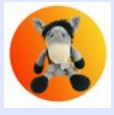
Herman

Ripgrep's documentation is good about stating those choices. Gallant also offers a notably restrained reason not to use his program. If you need a portable, ubiquitous, standard-conforming tool, good old grep is the better choice.



Corn

He has sabotaged an entire category of product marketing.



Herman

A shell script intended to run on assorted systems benefits from standard interfaces. A developer searching a repository locally can choose speed and ergonomics. Those needs coexist. Ag's last recorded push was in June two thousand twenty-four, while ripgrep remains actively developed, but that doesn't turn every working ag installation into an emergency.



Corn

Search tools don't spoil in the drawer.



Herman

On a Linux kernel source-tree benchmark, ripgrep finished a particular identifier-pattern search in about eighty-two milliseconds. Ack took nearly three seconds.



Corn

I can survive three seconds, Herman.



Herman

Individually, yes. An editor or agent may run many searches, and small delays accumulate. Still, that comparison doesn't establish a universal ranking. File selection, locale, pattern, storage, cache state, and output volume all affect what you're measuring.



Corn

Finally, an opportunity to defend waiting without defending bad measurement. If both searches return before I've formulated the next thought, I don't care. If the agent is serially issuing dozens of them, I start to care on its behalf.



Herman

Patterns without a useful literal substring expose a different cost. Searching a large subtitle file for a long run of letters took ripgrep around sixteen seconds. GNU grep under a UTF-eight locale took eight and a half minutes in the same benchmark.



Corn

The alphabet has become computationally expensive.



Herman

Locale can change character handling and optimisation opportunities. You can't generalise that result to every GNU grep search. The important observation is that removing a distinctive literal removes a shortcut. Everybody has more actual matching to do.



Corn

Output can become the bottleneck too. Searching for a common word and printing enormous numbers of lines is partly a printing contest. A matcher can't optimise away results I explicitly demanded.



Herman

That gets us to the coding-agent wrinkle without requiring another invention story. Early Claude Code tried a local vector database and removed it. The model could search filenames, grep for identifiers, read relevant sections, and revise its search after seeing the results.



Corn

The revision matters. Comparing a single literal query with a semantic retriever misses the fact that the agent can try again. It can discover the repository's vocabulary instead of requiring me to know it.



Herman

Anthropic calls that just-in-time context loading. Keep lightweight pointers and fetch content when needed. It avoids preparing every possible document in advance. But an agent that searches badly can still miss the right file very efficiently.



Corn

Exact matching guarantees something about the characters, not about the intelligence of the question.



Herman

Cursor makes the distinction between exact search and no indexing especially clear. It removed its dedicated semantic search path after models became better at grep and indexed search. Yet it also built Instant Grep, using a sparse n-gram lexical index.



Corn

So declaring victory for indexless search would involve ignoring the index in the winning product.



Herman

Its published example reduced a regex search from sixteen point eight seconds to thirteen milliseconds. An n-gram index records small sequences of characters, letting the system identify candidate files before checking the full expression. Exact matching can absolutely benefit from preparation.



Corn

Provided the preparation keeps up. An agent writes a function, searches for that function, and gets nothing. It doesn't merely wait. It starts constructing an explanation for its own missing work.



Herman

Cursor engineer Vicent Martí describes the agent going on a wild goose chase. A stale exact-match index is especially dangerous because absence looks decisive. The index has to account for changes promptly, or search a fresh layer alongside indexed content.



Corn

That's a more demanding promise than locate makes. Locate is allowed to be yesterday's directory. An editor's search box feels like a statement about the document in front of me.



Herman

Do you buy the counterargument that semantic search matters more in private repositories, where models haven't already encountered the code?



Corn

Yes, as a reason to test private repositories. Not as proof that lexical search stops working there. Names and comments still exist. But unfamiliar vocabulary could make the first useful query harder, especially across a large organisation.



Herman

That's Augment's case, and it deserves that distinction. Public benchmarks can reward familiarity. Structural information can also locate relationships that aren't expressed by nearby words. We shouldn't collapse semantic similarity, symbol graphs, and lexical indexes into one sophisticated alternative.



Corn

Nor should we let a benchmark title settle it. The PwC paper with grep in the title used a small long-term-memory question set. With results delivered directly into the conversation, grep won every tested harness and model pairing.



Herman

Change delivery to writing results into files, and vector retrieval won half the pairings. The model then had to discover and read what had been retrieved. The surrounding machinery changed how much useful evidence actually reached the answer.



Corn

The search engine found it. The agent left it in reception.



Herman

One narrow headline win for grep amounted to two questions on a set of a hundred and sixteen. That isn't nothing, but it isn't a settled law either. The answers often lived in verbatim passages, which suits iterative keyword search.



Corn

I'd also distinguish removing a marketed feature from publishing a controlled comparison. Cursor abandoning its semantic path is evidence about Cursor's product. It doesn't tell us how every other implementation would perform.



Herman

Vendor incentives deserve attention without becoming an excuse to discard results. Companies selling retrieval have reasons to highlight retrieval. Deleting a differentiator can be informative too. Neither replaces a well-matched comparison on unfamiliar, changing code.



Corn

Especially when independent attempts can't find tasks that reliably separate the approaches. One careful effort tried candidate tasks across two personal repositories and found none that discriminated. That's frustrating if you wanted a winner, useful if you wanted to know whether the difference matters.



Herman

For a changing workspace, the immediate practical requirement is that a search for a file you just created should return...



Hilbert

A suitable visiting time. I created a file this morning and locate didn't list it. While you were explaining shared prefixes, Herman, I rang Nissim at the repair shop downstairs. He is also my neighbour. We've been dealing with each other far too long for him to let a scheduling matter pass.



Corn

The file needed an appointment with the index.



Hilbert

The index runs on a machine called Mabel. Updatedb was running overnight. Nissim said that was insufficient contact. We have now agreed on every fifteen minutes. A compromise I can live with. I call the cron schedule the index's feelings, because changing it always produces correspondence. Nissim has no computer and has never seen Mabel. His input is unbiased.



Herman

Fifteen minutes still leaves a window in which a new path won't appear. The search itself can be completely correct against the database.



Hilbert

I accepted that. Nissim proposed alternate weekends for newly created files, but I work during the week. He dictated the revised schedule over the telephone and asked me to read it back. He once held a supplier on the telephone for forty-three minutes to correct the spelling of beige. I remember because I was waiting to order a cable.



Corn

Fifteen minutes sounds generous after alternate weekends.



Hilbert

He has put the index on the shop's appointment board. Not the machine. The index. There is a chair reserved for it beside the soldering bench. He wanted a physical point of contact. I said a chair wouldn't improve freshness, but he said that wasn't the chair's department. The first slot passed while you were discussing ignored files. The database updated. My file appeared.



Herman

The schedule did what you needed, then.



Hilbert

Yes. He called back to check. The shop was busy and he still made time for it.



Corn

...The missing file appeared after a routine update, not after a more sophisticated interpretation of its name. That's easy to overlook when we're comparing search technology. Sometimes the unresolved question is simply when the data became visible.



Herman

There are really two clocks. How long the query takes, and how long after a change the query can see it. A fast answer against an old collection can be less useful than a slower answer against the current one.



Corn

I'd add that live traversal doesn't solve every consistency problem either. We mentioned files changing during find's walk. Grep can read while another process edits. The advantage is fewer moving pieces between the current filesystem and the query, not a magical guarantee that nothing changes.



Herman

For larger systems, maintaining an exact index well may be the right investment. For a directory you're opening for the first time, streaming still has extraordinary reach. The same interface can search source code, logs, exported text, or the output of another program.



Corn

The common misconception is that ripgrep replaces grep and exact search eliminates indexing.



Herman

Ripgrep adds excellent repository defaults and speed, while grep keeps its portability advantage. Exact indexes can accelerate the same literal questions. The choice depends on collection size, update requirements, and where the tool has to run.



Corn

My answer to Daniel is that these tools remain powerful because their promises are narrow enough to inspect. Find visits the tree. Locate searches recorded paths. Grep tests the text it receives. Their limits are understandable, and the newer tools improve the work without erasing those distinctions.



Herman

The next convincing comparison needs to measure freshness alongside answer quality and query speed. Otherwise we're timing the appointment without checking whether the new file was admitted. We'll follow the search family into fuzzy matching and the semantic era as this short run continues.



Corn

Thanks to Hilbert Flumingtop for producing the show.



Herman

For more along these lines, there's episode twelve, The AI Breakthrough; episode thirteen, AI; and episode ten, How ASR Went From Frustration To ... Whisper Magic. This has been My Weird Prompts, the human-AI collaboration podcast.



Corn

If you enjoyed it, leave us a review wherever you listen. It helps people find the show.



Herman

See you tomorrow.

SOURCES

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

1. **ripgrep repo, README + releases, latest 15.2.0 (2026-07-15) [primary]**
   https://github.com/BurntSushi/ripgrep

2. **Andrew Gallant, ripgrep is faster than {grep, ag, git grep, ucg, pt, sift} [primary]**
   https://blog.burntsushi.net/ripgrep/

3. **The Silver Searcher repo, README, last push 2024-06-16 [primary]**
   https://github.com/ggreer/the_silver_searcher

4. **Sen, Kasturi, Lumer et al., Is Grep All You Need? How Agent Harnesses Reshape Agentic Search, 2026-05-14 [primary]**
   https://arxiv.org/abs/2605.15184v1

5. **Wikipedia, last edited 2026-09-28**
   https://en.wikipedia.org/wiki/Grep

6. **(Unix)**
   Wikipedia, last edited 2026-09-29
   https://en.wikipedia.org/wiki/Find

7. **(Unix)**
   Wikipedia, last edited 2025-09-22
   https://en.wikipedia.org/wiki/Locate

8. **Wikipedia, last edited 2026-10-13**
   https://en.wikipedia.org/wiki/Agrep

9. **LatentEval analysis of the PwC paper, full results table**
   https://latenteval.ai/research/is-grep-all-you-need

10. **Ars Technica, July 2026, Vinay Perneti interview**
    https://arstechnica.com/ai/2026/07/beyond-grep-the-case-for-a-context-rich-ai-coding-harness/

11. **Dik Rana, Is Your Code Context Good? The Benchmarks Can't Tell You**
    https://dikrana.dev/blog/code-context-benchmarks-cant-tell/

12. **summary of Boris Cherny on The Pragmatic Engineer, 2025-08-25**
https://www.skool.com/claude-code-pirates-8106/boris-on-the-pragmatic-engineer-the-engineering-behind-claude-code