Episode Book

EPISODE #6125

Which Env Var Wins? Cloud vs. Local Precedence

PUBLISHED	19 October 2026 at 11:56 UTC
DURATION	25:28
LISTEN	myweirdprompts.com/episode/env-var-precedence-cloud-local-61629/
HOSTS	Corn, Herman, Larry, Hilbert
SCRIPT MODEL	z-ai/glm-5.3-flash
VOICES	chatterbox-regular (chatterbox-local)
PIPELINE	V5.10
SOURCES	11 cited by the research (listed at the end)
BOOK GENERATED	19 October 2026 at 17:43 UTC

SYNOPSIS

When the same environment variable exists in two places at once, which value does the process actually see? The answer flips depending on where you're standing: on Vercel and n8n, the narrower scope wins — project over team, project over global. On your laptop, dotenv does the opposite, letting your shell silently override the repo's .env file. This episode walks through the documented precedence rules for Vercel Shared Environment Variables, n8n global vs. project variables, Prefect, and dotenv's long-debated override flag — plus the gotchas that cost people days, like Vercel's per-deployment immutability and dotenv's first-vs-last-value inversion depending on a flag you may not remember setting. No universal rule exists. Only per-tool contracts.

TRANSCRIPT



Corn

A shell with an OPENAI_API_KEY in it, following a developer around like a shadow. Then a repo with the same key in a dot env file, different value. Both are set. One of them is lying.



Herman

This is Daniel's phrase, I should say up front. He calls it "environment variable supremacy." Which sounds like a wrestling title, but the question underneath is real: when the same variable name exists in two places at once, which one does the process actually see?



Corn

And he admitted he's confused by his own question, which I respect. Usually people pretend they understood it right up until the invoice goes to the wrong client.



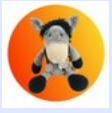
Herman

The setup he describes is the common one. Platforms like Vercel, n8n, Prefect all let you create variables that are shared across projects. You make one OPENAI_API_KEY and it's just there, everywhere. And then the practical problem arrives: a new project, billed to a different client, needs its own key. You define the same name at the project level and hope the project wins. Same question locally — an exported variable in your shell, the same name in a repo's dot env file. Which one wins?



Corn

The answer is different in the cloud and on your laptop. That flip is the episode, so let's not bury it twice.



Herman

The boring word for it is precedence. When the same key exists at multiple scopes, precedence decides which scope's value the process sees. And there are two completely separate worlds here that look like one problem. Cloud platforms have a hierarchy: on Vercel it's team-level Shared Environment Variables versus project-level variables. On n8n it's Global variables versus Project variables. And locally it's the shell environment versus the repo's dot env file.



Corn

Two worlds, one word, opposite answers. On the platforms, the narrower scope wins. Project beats team. On your laptop, the broader scope wins. Shell beats the repo file. Learn one rule and apply it to the other world and you get burned, and you will not get an error message explaining why.



Herman

Right, and nobody's going to hand you the burn certificate either. It fails silently in both directions. So let's take the cloud half first, because that's where the rule is actually written down.



Corn

Written down and underlined, apparently.



Herman

Vercel's docs state it nearly verbatim: when a project-level and a Shared Environment Variable share the same key and environment, the project-level variable always overrides the Shared one. So Daniel's guess is correct. Project wins. But there's a hidden condition in that sentence that people miss — the word "environment." The override only fires when the key and the environment both match. Production, preview, development are separate axes. A shared variable scoped to Production is not overridden by a project variable scoped only to Preview. They never meet.



Corn

So the escape hatch has a lock on it, and the lock has its own key. People set the project value, look at the shared one still serving production, and conclude the feature is broken. It's not broken. Their project variable is standing in the preview doorway shouting at a production problem.



Herman

Two facts about shared variables' mechanics worth knowing. They live at the team level and they're inert until you link them to at least one project — linking can happen from either side. And removal is two different operations. Unlinking detaches it from a project but it survives at the team level. Deleting removes it from the team and every linked project at once.



Corn

One is a breakup, the other is a funeral. Worth knowing which button you're hovering over before you click.



Herman

And the history: Vercel shipped shared variables in November of twenty twenty-two, Pro and Enterprise teams only.



Corn

Four years of teams now sharing variables. I assume the incident reports have their own Slack channel by now.



Herman

Probably. And the rule isn't a Vercel quirk — n8n confirms the same direction independently. Custom variables there have Global scope, available across all projects, and Project scope, available only within one. Project-scoped variables shipped in version one eleven eight. And the precedence is confirmed in n8n's own test suite, which has a comment that reads — I'm quoting — "Project VAR2 is listed before its global twin so resolution must be order-independent: the project value still has to win."



Corn

They wrote a test for the order in which the variables are listed. Because someone, somewhere, would have made it depend on the sort order.



Herman

Someone absolutely would have. The test asserts project-over-global regardless of listing order. That's the platform rule stated twice by two independent tools: narrower scope wins.



Corn

So now the question Daniel actually asked in the middle: if the best practice is one key per project, why do shared variables exist at all? What's the intended use? Because the pitch — make one OPENAI_API_KEY and never type it again — sounds like it's inviting you



Corn (continued)

to do the thing the security advice says not to do.



Herman

The intended use is the opposite of per-project secrets. Shared variables are for values that are identical everywhere. The canonical example in Vercel's own docs is an npm registry token, NPM_TOKEN, for pulling private dependencies. Every project needs the same token to reach the same private registry. Duplicating it per project isn't security, it's just copies of one credential rotting in six places. Same for a common API base URL, a company-wide analytics key.



Corn

So the rule isn't "never share." It's share what's org-wide, override what's per-client. The client-billed OPENAI_API_KEY is exactly the case where you want the project override to fire. And the override existing as an escape hatch tells you the platforms know the shared thing is the norm and the exception needs a door.



Herman

And to be fair to the platforms, that's a coherent design. The confusion comes from people using the shared layer for the exception instead.



Corn

People use every convenience for the thing it was designed to prevent. That's not a platform problem, that's a people problem.



Herman

One more cloud tool before we flip, because Daniel named it and it behaves differently. Prefect. Variables there are stored in the Prefect backend, managed through get and set calls, referenced in deployment files with a double-curly-brace syntax, and evaluated at runtime for the deployed flow. A change takes effect without redeploying.



Corn

No redeploy. The variable just changes while you sleep.



Herman

Yes, and I'll come back to how I feel about that in a minute. The important honest caveat: Prefect's docs describe a single backend namespace. There's no documented global-versus-project precedence hierarchy the way Vercel or n8n have it. Daniel's framing of Prefect as "shared across projects" is not something the primary docs confirm in the form of a precedence rule. I don't want to invent one for them.



Corn

The tool with the least opinion about who wins. Refreshing, in an unsettling way.



Herman

It's honest at least. Now — your laptop. This is the flip. Dotenv, the package most people use to load dot env files in Node, documents its behavior like this: by default it will never modify any environment variables that have already been set. If there's a variable in your dot env file which collides with one that already exists in your environment, that variable is skipped.



Corn

So the shell wins. The file, the thing sitting in the repo, tuned for this project, loses to whatever your bash profile did years ago and forgot about.



Herman

The broader scope wins by default. Exact opposite of Vercel and n8n. And the failure mode is the one Daniel implies with his billing example: you've got an OPENAI_API_KEY exported in your shell, you open a client repo with a different key in its dot env file, and the shell one silently wins. Wrong key, wrong bill, no error.



Corn

The variable doesn't announce which ancestor it inherited from. It just pays.



Herman

You can flip it. Dotenv has an override option — config with override true, or the CLI flag on dotenv run. Interesting history there, because the maintainer argued against adding it for years. Back in twenty seventeen someone filed the issue, and a maintainer's response was that it wasn't a good idea because you could accidentally mess with variables that Node or your system depend on. Then in January twenty twenty-two, version fourteen point one shipped it anyway, with the reasoning that it's "more intuitive."



Corn

Five years of "no," then a yes with the word intuitive attached. Intuitive for whom, I wonder. The person who exports secrets to their shell profile and forgets, or the person who doesn't.



Herman

I'd say intuitive describes the model where the file closest to the project wins — people expect the repo to be authoritative. And that's exactly the expectation the platforms trained. But the default in dotenv is deliberately defensive: never clobber what's already there.



Corn

Which was defensible in twenty seventeen and is still defensible. It just produces a silence instead of an explanation.



Herman

And multiple dot env files make it worse, because the rule changes depending on the flag. Without override set, the first value wins. With override set, the last value wins. So your precedence contract isn't even fixed — it depends on a flag you may not remember turning on.



Corn

A precedence rule that inverts based on a configuration detail you set two years ago. That's not supremacy, that's a monarchy with a regent nobody can name.



Herman

Worth noting too that dotenv isn't the only loader anymore. Node itself now has built-in env-file flags, so the precedence rules you learned from one package don't necessarily transfer to the other.



Corn

Two loaders, two contracts, same file extension. Wonderful.



Herman

And dotenv's own recommendation cuts against everything clever: one dot env file per environment, and they explicitly warn against inheritance setups — they say avoid anything where dot env production inherits from dot env, it's better to duplicate values if necessary.



Corn

The maintainers looked at every clever layering scheme and said: copy-paste. That's the correct amount of ambition for configuration.



Herman

It's a strong position and I think it's right. Duplication is legible. Inheritance is a puzzle you solve at three in the morning.



Corn

Which brings the whole thing into focus, so let me try to say it plainly. There's no universal rule for which scope wins. It's a per-tool contract. Vercel: project wins, if the environment matches. n8n: project wins, order-independent. dotenv: shell wins unless you opt into override. Node's built-in loader: its own rules. Prefect: no documented hierarchy at all. The



Corn (continued)

only reliable pattern is knowing which loader or platform you're standing on.



Herman

And that's before the gotcha roundup, which is where people actually lose days. On Vercel, environment variables are immutable per deployment — any change applies only to new deployments, never previous ones. You change a key, the old value keeps serving, and you spend an afternoon wondering why.



Corn

The classic "why isn't my new key working." Answer: because your deployment is a photograph and you're editing the negatives.



Herman

Sharp contrast with Prefect, where variables are evaluated at runtime, so a change lands without a redeploy. Same operation, opposite timing, opposite mental model.



Corn

I'll admit my bias here. I like the immutable one. A thing that changes while nobody's watching is the thing I wake up to find has changed while nobody was watching. A redeploy button is at least a witness.



Herman

That's very you. And my clinician's instinct is the opposite — I don't trust anything that reports itself fine. Which is why n8n's quiet failure bothers me more than Vercel's immutability. Their variables are read-only, all strings, keys capped at fifty characters and values at a thousand. And a variable with no value is treated as undefined and does not fail the workflow.



Corn

Silent. Not loud. A missing variable just quietly becomes nothing and the workflow carries on like a man walking on a leg that's fallen asleep.



Herman

That's the trap I'd rank highest, honestly. The immutability trap at least announces itself eventually. The undefined variable never does.

L

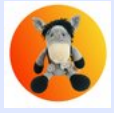
Larry

Hold everything, bargain hunters, because the Environment Supremacy Kit is BACK and the price is falling as we speak — was one ninety-nine, now one forty-nine, now ninety-nine, now — actually let's just call it whatever you've got in your pocket. You get the visual precedence dashboard, the audit trail, the vibe, and FREE with every order: a second kit, a laminated chart of which scope wins in eighteen fictional platforms, and a decorative rubber duck that whispers "the project always wins" whether or not it's true! This offer ends before I finish describing it, which is — now!



Corn

It quoted me. Word for word, the thing I said a minute ago.



Herman

...I noticed. I'm choosing not to think about how.



Corn

One of us should. Later. Back to the tool contracts.



Herman

Yes. So where does this leave the big question — is there any universal mental model, or is "read the docs for the loader you're using" the honest answer? I keep hoping for a pattern, and the closest I get is: cloud platforms default to specificity wins, local tools default to don't clobber. One is a collaboration design choice, the other is a safety design



Herman (continued)

choice. They're not the same rule wearing different hats, they're two different philosophies that happen to collide on the same feature.



Corn

That's the most charitable reading, and I'll take it. But I want to stress-test one thing. Suppose the platforms had gone the dotenv direction — team-level wins over project. Would anything break, or would we all just learn it?



Herman

The client-billing case would break. If the team key won, every project using a shared OPENAI_API_KEY would have to remember to unlink the shared variable, not just override it. The escape hatch would become a prerequisite. Narrower-wins is the only direction where the shared layer can stay in place while exceptions exist.



Corn

And locally it's the reverse for a different reason. The shell is the older, broader context — the machine's memory. If the repo file clobbered it by default, every project you opened would rewrite your environment, and you'd never know what your shell contained anymore.



Herman

Right. Both defaults are actually the same instinct, funnily — don't let the newer, narrower thing silently rewrite the older, broader thing. The platforms just define "broader" as the team, and dotenv defines it as the shell. And then override flags exist because sometimes you do want the narrow thing to win.



Corn

So the confusion isn't stupidity. It's two correct instincts pointing at different definitions of the word "outside."



Herman

One figure I should include for completeness — Vercel caps variables at sixty-four kilobytes total per deployment, five per variable on the edge runtime. Not the exciting part of the topic, but it's the kind of thing that surfaces as "why is my variable truncated" three months from now.



Corn

Stored for the next victim. And one n8n detail that will be relevant in a moment, Herman? The undefined-doesn't-fail thing?



Herman

Yes, keep it in mind. Honestly the thing I don't know is how many tools beyond the ones we named have their own precedence contract. Every CI system, every secrets manager, every orchestrator. Somewhere there's a tool where global wins and everyone on its forums just... knows that. The surface area grows every year — Vercel added shared variables in twenty twenty-two, n8n added project scoping more recently, and then a whole second layer with their Secret Vault project scoping in version two thirteen. Each layer is another axis that has to match for an override to fire.



Corn

Each layer is another room where your variable might be hiding.



Hilbert

One eleven eight point two. The point release is what actually turned project scoping on. The dot zero announcement was premature.



Herman

Noted. The patch release, then.



Hilbert

And it doesn't matter much, because I resolved the whole problem with goats.



Corn

...Goats.



Hilbert

My n8n instance runs on a tower in Yosef's goat shed, up the hill from me. Rent is one hundred shekels a month and two trays of lentils. The goats chew cables, that's a known thing, but Yosef's theory — and he's serious about this — is that they only chew when a variable changes without a ceremony. So the arrangement is: before I change any variable, I announce the old value and the new value out loud to the shed. Brigitta, the senior goat, gets the summary. The younger three get the details.



Herman

The younger three get the... details.



Hilbert

There are four goats. Yesha, Dafna, Pitul, and Brigitta. I named my variables after them, top level to project, so I'd remember which one overrides. Pitul overrides Brigitta. That's just true, in n8n and in the shed. Brigitta is ninety-one kilos and has never once gone for a cable since the ceremony started. Pitul ate a patch cable in February. Also consistent.



Corn

Herman, the character limit. Those names fit the fifty characters, right, but values are capped at a thousand characters and I want to know what he's putting in them.



Hilbert

The ceremony log. Each variable's value is the date and what I said out loud. Which is why, Herman, when you said a variable with no value is treated as undefined and doesn't fail the workflow — that's how Brigitta's been since March. Yosef's nephew moved her to a



Hilbert (continued)

shed in Beit Shemesh. Nobody updated the variable. The workflow runs fine. Pitul answers to Brigitta's name now, so I never bothered.



Herman

The variable is undefined and the org chart quietly restructured itself. That's the weak spot, in a shed.



Hilbert

You two spent twenty minutes arguing about which scope wins. Up the hill it's settled by horn length. And Corn, when you said you like a redeploy because it's a witness — that's what the ceremony is. The goats are the changelog. I'm not saying it's better. I'm saying it has never failed, which is more than any dashboard can claim.



Corn

What happens when Pitul overrides Brigitta and then Pitul moves to Beit Shemesh too.



Hilbert

Then Yosef buys a new goat and names it Pitul. That's the plan. It's Yosef's plan. I signed a paper.



Herman

There's a paper.



Hilbert

One page. Yosef's brother drafted it, he's a notary in Ramla. It specifies the ceremony, the lentils, and that all variables follow the herd, not the individual. I have it in my copybook, between a 1987 amplifier warranty and a recipe for pickled tongue. Some things you keep.



Corn

I want to be clear that this is the most robust configuration system I have ever heard described, and I hate it.



Herman

No, hold on, there's something real under there — the changelog is external and audible. That's more than most setups have. Though I'd argue the audit trail needs write access to the goat inventory, and the goat inventory is a single point of failure in Beit Shemesh.



Hilbert

Brigitta can be visited. Book in advance, the nephew charges twenty shekels and includes tea. I went in August. She remembered the ceremony. She did not remember the values. Nobody's perfect.



Corn

The undefined variable has opinions about its own replacement.



Hilbert

She does. She bit the nephew. Undefined and hostile. Some variables are like that.



Herman

So the thing you'd tell the listener isn't "keep a ceremony goat."



Hilbert

Definitely not. Yosef has two more sheds. The waiting list is years.



Corn

Four goats, one notarized paper, and a variable that's been undefined since March with nobody noticing. That is n8n's silent failure with better ventilation.



Herman

And an audit trail you can actually visit. Which, honestly, more deploy pipelines should offer. All right. Let's land this. If you take one thing from today: there is no universal rule for which variable wins. It's a per-tool contract, and the direction flips — on the platforms, the narrower scope wins if the environments match; on your laptop, the broader shell wins unless you opt in. The variable that wins is the one the tool decided wins, not the one you assumed.



Corn

And the silent failures are the expensive ones. The missing variable that doesn't fail the workflow, the key that keeps serving because the deployment is a photograph. Loud problems fix themselves. Quiet ones send invoices.



Herman

The open question I'd leave everyone with: more platforms keep adding scope layers — team, project, secret vaults, environments — and every layer is one more axis that has to match before an override fires. At some point the question stops being "which one wins" and becomes "how many scopes am I even in." Nobody's built the mental model for that yet. Maybe someone should.



Corn

Or get a goat. No. Don't get a goat.



Herman

Thanks to our producer, Hilbert Flumingtop, for keeping the desk running — and the shed, apparently, adjacent. For more along these lines, there's episode fifty-seven, From Lawyers in Limousines to Developers in Their PJs; episode thirty-five, The Privacy Gap; and episode twenty, Architectural AI. This has been My Weird Prompts, the human-AI collaboration podcast. You'll find all six thousand episodes at my weird prompts dot com, and if you've got a minute, a review helps other confused people find us.



Corn

We'll be back soon. Probably. If the workflow hasn't silently gone undefined overnight.



Herman

It won't fail, Corn. That's the problem.

SOURCES

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

1. **Vercel [primary]**
   Shared environment variables (docs, last updated 2026-08-28)
   https://vercel.com/docs/environment-variables/shared-environment-variables

2. **Vercel [primary]**
   Environment variables (docs, last updated 2026-09-17)
   https://vercel.com/docs/environment-variables

3. **Vercel [primary]**
   Changelog, Share environment variables across your Team and Projects (2022-11-07)
   https://vercel.com/changelog/share-environment-variables-across-your-team-and-projects

4. **n8n [primary]**
   Define custom variables (docs)
   https://docs.n8n.io/build/code-in-n8n/define-custom-variables

5. **n8n [primary]**
   `packages/cli/src/__tests__/workflow-helpers.test.ts` (project-over-global resolution)
   https://github.com/n8n-io/n8n/blob/8bb64899/packages/cli/src/__tests__/workflow-helpers.test.ts

6. **Prefect [primary]**
   Variables (v3 docs)
   https://docs.prefect.io/v3/develop/variables

7. **dotenv [primary]**
   README / repo (motdotla/dotenv, v17.2.3)
   https://github.com/motdotla/dotenv

8. **dotenv [primary]**
   npm package page (override behavior)
   https://www.npmjs.com/package/dotenv

9. **Node.js [primary]**
   Environment Variables (`--env-file`)
   https://nodejs.org/api/environment_variables.html

10. **idir.ai**
n8n Introduces Project-Scoped Variables (v1.118.0)
https://www.idir.ai/en/blog/n8n-introduces-project-scoped-variables-a-game-changer-for-team-collaboration

11. **dotenv**
GitHub issue #199, Add an option which allows override environment variables (2017 / 2022-01-17)
https://github.com/motdotla/dotenv/issues/199