Member-only story

Claude Code Sandboxing — AI Coding Tools Have a Permission Fatigue Problem

We've trained ourselves to ignore security prompts. A new approach to sandboxing might finally fix this, but it's not without trade-offs.

11 min read · 1 day ago

Reza Rezvani Follow

Listen Share More

I've clicked *"approve or yes, don't ask again"* many many times from this morning up to now. I'm working with an AI coding assistant, and every file edit, every command execution, every network request requires my explicit permission. By the third hour, I stopped reading what I was approving. I just clicked yes.

That's when I realized I'd become the security vulnerability.

This isn't unique to me. According to a 2024 Stack Overflow survey, 76% of developers now use AI coding assistants regularly. Yet our security model for these tools is fundamentally broken. We're asking humans to make dozens of micro-decisions per hour, and human attention doesn't scale that way.

```
> /sandbox

Sandbox: Manage  Config  (tab to cycle)

Configure Mode:

› 1. Sandbox BashTool, with auto-allow in accept edits mode
  2. Sandbox BashTool, with regular permissions
  3. No Sandbox (current)

Auto-allow mode: When in accept-edits mode, commands will try to run in the sandbox automatically, and attempts

Learn more: docs.claude.com/sandboxing (https://docs.claude.com/en/docs/claude-code/sandbox)
```

Sandbox Feature in Claude Code version 2.0.24

The result is what security researchers call *"approval fatigue"* — a psychological state where users stop evaluating individual permissions and start clicking through them automatically. It's the same phenomenon that made everyone accept cookie consent banners without reading them.

But with code execution, the stakes are higher. Your SSH keys, database credentials, and production access all sit on the same machine where the AI is asking to run *"just one more npm install."*

Several companies are now trying to solve this, and their approaches reveal fundamentally different philosophies about AI security. Let's examine what's actually working, what's not, and what trade-offs we're making.

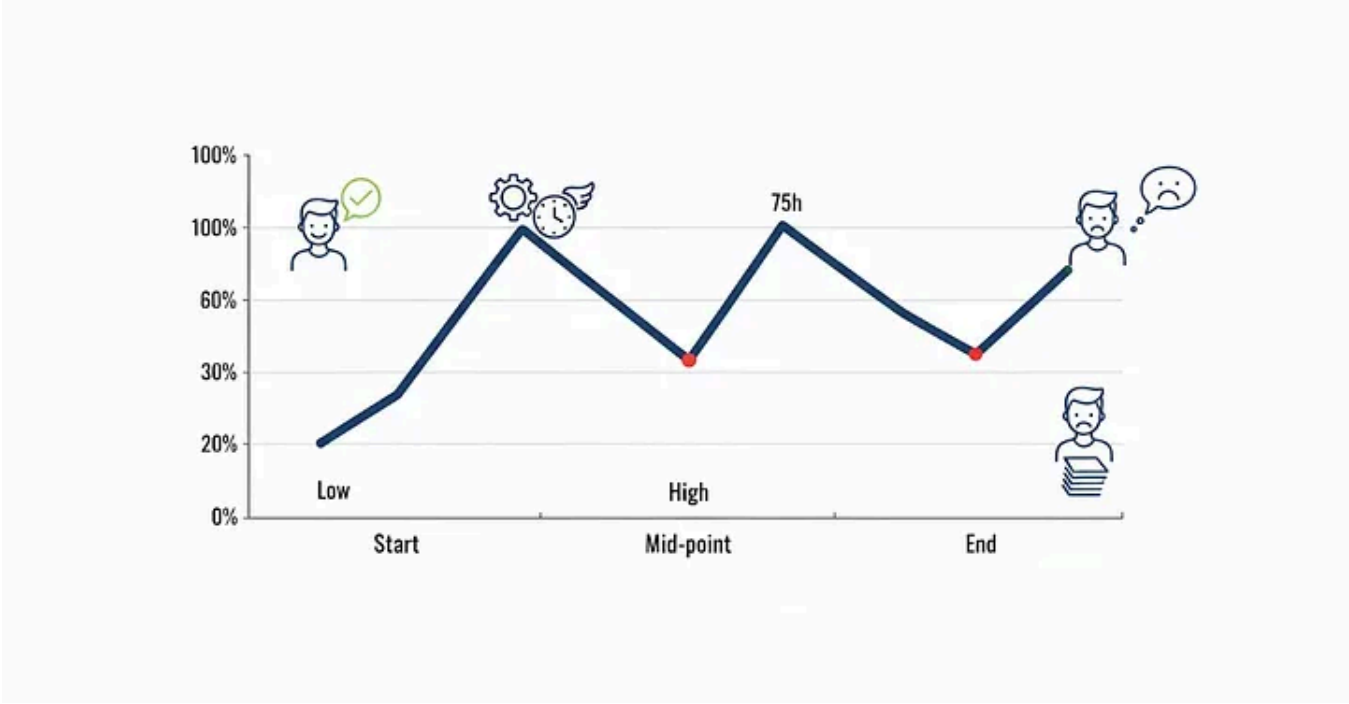
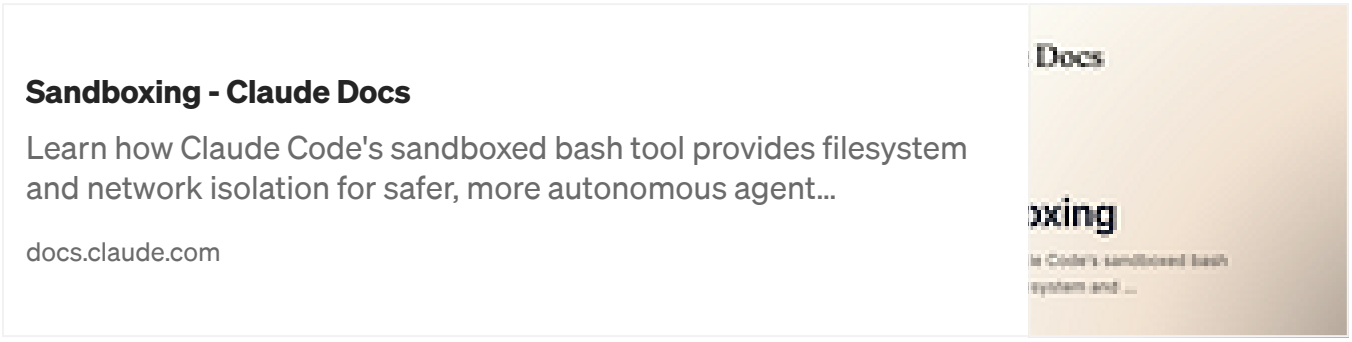


Chart showing developer approval fatigue curve over time

How We Got Here: The Permission Model's Fatal Flaw

The current generation of Agentic coding tools or AI coding assistants— Claude Code, Cursor, and others — largely operate on a permission-based model. The AI proposes an action, you review it, you approve or deny. It sounds reasonable, but it fails in practice for a predictable reason: it assumes unlimited human attention.

Research from Carnegie Mellon's CyLab Security and Privacy Institute shows that humans make progressively worse security decisions as the number of prompts increases. After about 15 consecutive approval requests, decision quality drops by nearly 60%.

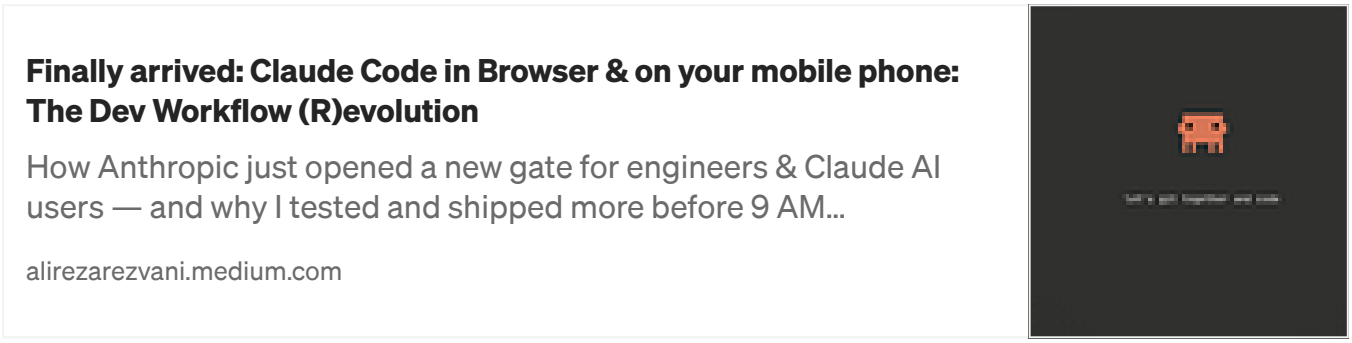


In a typical development session with AI assistance, you might face:

- 8–12 file edit requests
- 5–8 command executions
- 3–5 network operations
- 2–4 package installations

That's 20–30 decisions per hour, every hour, all day. The cognitive load isn't sustainable.

The paradox is elegant in its cruelty: **permission-based systems make us less secure by training us to ignore security prompts.**



Three Approaches to Solving This

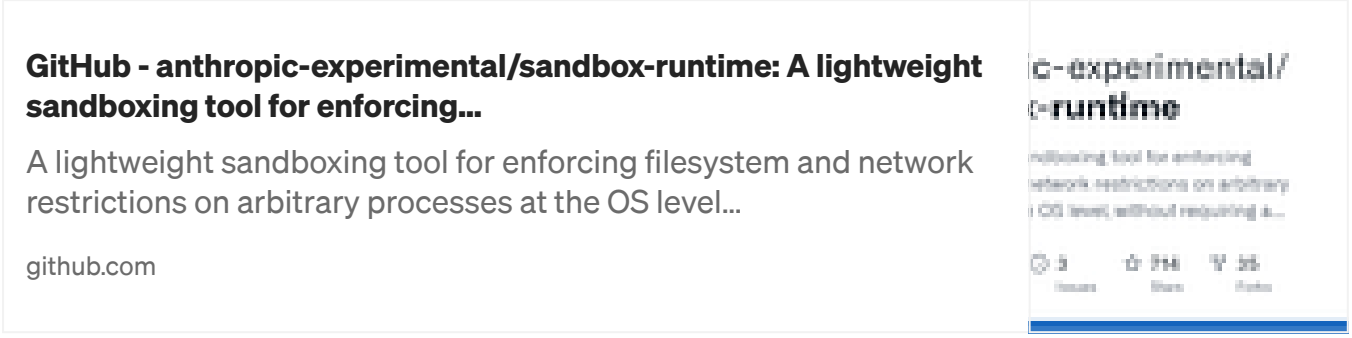
The industry is converging on three different strategies, each with distinct trade-offs.

Approach 1: Smarter Permission Systems

Tools like GitHub Copilot or Claude Code have refined their permission models with better categorization. *"Safe"* operations *(reading files, running tests)* happen automatically. "Risky" operations *(modifying system files, installing packages)* require approval.

The improvement is incremental, not fundamental. You still make dozens of decisions, just slightly fewer. And the categorization isn't always right — what's *"safe"* depends heavily on context.

A recent analysis by Trail of Bits, a security research firm, found that even sophisticated permission categorization reduces prompts by only 30–40% on average. Meaningful, but not transformative.



Approach 2: Container-Based Isolation

Some development teams have moved to container-based approaches — running AI assistants inside Docker containers or virtual machines with limited access to the host system.

This works, but with significant overhead:

- **Resource cost**: Each container consumes 200–500MB of memory
- **Performance**: Container startup adds 5–15 seconds per session
- **Complexity**: Developers must learn container orchestration
- **Brittleness**: Volume mounting and networking configuration are error-prone

Container isolation is thorough but heavy. It's like hiring a bodyguard who insists on checking every room before you enter — effective but exhausting for daily work.

Approach 3: OS-Level Sandboxing

The newest approach uses operating system primitives to create lightweight sandboxes. Instead of asking permission for each action, you define boundaries up front. Within those boundaries, the AI operates freely. Outside them, you get notified immediately.

Anthropic recently implemented this in Claude Code using Linux bubblewrap and MacOS seatbelt. Microsoft is exploring similar techniques in VSCode. The approach is gaining traction because it solves the attention problem without container overhead.

Think of it like giving your AI assistant a workshop. Inside the workshop: full autonomy. Trying to leave the workshop: immediate alert.

Comparison chart of three approaches — permission model vs containers vs OS sandboxing

The Two-Boundary Security Model

The sandboxing approach requires two complementary restrictions: filesystem isolation and network isolation. Miss either one, and the security guarantee collapses.

Filesystem isolation restricts which directories the AI can access. Using OS-level enforcement *(not application-level checks)*, every process the AI spawns inherits these

restrictions. Even if malicious code runs inside the sandbox, it can't access your SSH keys in `~/.ssh` or system files in `/etc`.

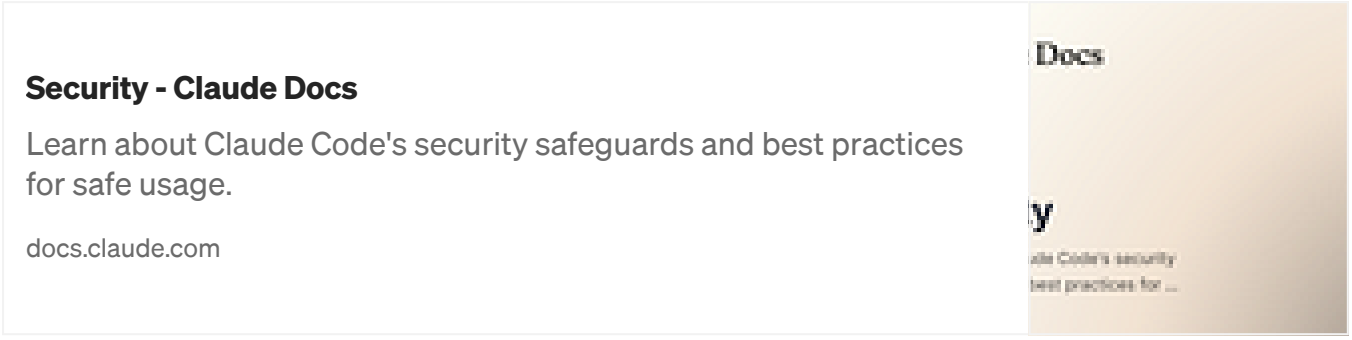
Network isolation controls which servers the AI can connect to. All traffic routes through a proxy that enforces allowlists. The proxy runs outside the sandbox, so compromised code can't bypass it.

Here's why you need both:

Filesystem isolation alone won't stop an AI from sending your files to an external server. I could read your `~/.aws/credentials` and POST them to `attacker.com` if network access is unrestricted.

Network isolation alone won't prevent access to local sensitive files. I could still read your browser's stored passwords or your email client's database.

Together, they create meaningful containment. A compromised AI can't exfiltrate data *(network blocked)* and can't access high-value targets *(filesystem blocked)*.



This isn't theoretical. Security researcher Marcus Hutchins demonstrated a prompt injection attack that attempted exactly this sequence: read credentials, phone home. Sandboxing stopped it at both gates.

Attack scenario diagram showing how sandboxing blocks multi-stage attacks

Real-World Testing: What Actually Happens

I spent two weeks testing this approach against my normal development workflow. The results were striking, but not in ways I expected.

What improved:

- Permission interruptions dropped from ~25 per hour to 2–3 per hour
- Flow state became achievable again — I could focus on architecture without constant context switches
- Security posture felt more reliable because I wasn't relying on my own vigilance

What got harder:

- Initial configuration took 30–40 minutes per project
- Debugging sandbox boundary violations was opaque — *"access denied"* without clear explanations
- Some legitimate workflows broke *(CI integration, docker-in-docker scenarios)*
- Team members struggled with configuration complexity

The 84% reduction in prompts that Anthropic reported matches my experience. But the remaining 16% of prompts are **more consequential**. When the sandbox asks permission now, it's usually for something genuinely ambiguous that needs human judgment.

This is both the strength and weakness of the approach: **it trades constant low-stakes decisions for occasional high-stakes ones.**

The Adoption Gap: Why This Isn't Widespread Yet

If sandboxing solves the permission fatigue problem, why aren't all AI coding tools using it?

Three barriers:

1. Platform fragmentation: Linux bubblewrap doesn't work on Windows. MacOS seatbelt has different APIs. Windows AppContainer uses yet another model. Building cross-platform sandboxing is complex.

2. Breaking existing workflows: Developers have built habits around current tools. Sandboxing changes those patterns. The Cursor community on Reddit has been

vocal about concerns that sandbox restrictions would break their custom MCP servers and plugins.

3. Security theater risk: Poorly configured sandboxing creates false confidence. If developers think they're protected but the sandbox has escape hatches, it's worse than no sandbox at all.

Microsoft's VSCode team discussed this openly in their security roadmap: they're *"exploring sandboxing options but concerned about the sharp edges of misconfiguration."*

The technical foundation exists. The user experience and safety guarantees are still evolving.

Adoption timeline and barrier diagram

Critical Questions We Should Be Asking

The sandboxing approach isn't settled science. Several important questions remain:

What's the right default configuration? Too restrictive and legitimate workflows break. Too permissive and the security benefit evaporates. Getting this wrong at scale could undermine trust in AI tools entirely.

Who's liable when sandboxing fails? If a sandbox escape allows credential theft, is it the tool vendor's fault or the developer's configuration error? The legal frameworks here don't exist yet.

How do we audit sandbox boundaries? There's no standard for validating that a sandbox configuration actually provides the claimed security properties. We need security testing frameworks specifically for AI tool sandboxing.

What about supply chain attacks? Sandboxing protects against the AI itself being compromised, but what if the packages it installs are malicious? The sandbox doesn't solve dependency security.

These aren't idle concerns. They're the difference between a useful security improvement and security theater that lulls developers into false confidence.

What Container Veterans Are Saying

I spoke with several infrastructure engineers who've been running containerized development environments for years. Their perspective is instructive.

The consensus: **sandboxing is container-lite with 80% of the benefit and 20% of the cost.** For many use cases, that's the right trade-off. But for security-critical work, they're sticking with full container isolation.

The argument: OS-level sandboxing still shares the kernel with the host system. Container escapes are rare but not impossible. For environments handling regulated data *(healthcare, finance, government)*, the extra isolation layer matters.

It's a fair point. Sandboxing raises the security floor for typical development. It doesn't provide the security ceiling that some applications require.

The Performance Trade-off Nobody Talks About

Here's what surprised me: sandboxing isn't free, but the cost isn't where I expected.

Network proxy latency is negligible — typically 2–5ms per request. The real cost is in filesystem operations. On MacOS, I measured a 15–20% slowdown in operations that cross sandbox boundaries *(reading config files, checking for executables in PATH)*.

For most development work, this is imperceptible. But for build systems that traverse the entire filesystem *(looking at you, Gradle)*, it's noticeable.

The question becomes: would you trade 15% build speed for 84% fewer interruptions? For me, yes. For teams with automated CI/CD where builds run unattended, the calculus is different.

Where This Goes Next

The sandboxing approach is becoming the new baseline, not because it's perfect but because the alternatives are worse.

I expect to see:

Standardization efforts: A common API for cross-platform sandboxing would help. Microsoft, Google, and Anthropic are reportedly in conversations about this, though nothing formal has been announced.

Better configuration tools: The initial setup is too manual. We need templates, validation, and debugging tools that make sandbox configuration accessible to typical developers.

Integration with enterprise security: IT departments will want centrally managed sandbox policies. *"Developers can access these directories, these domains, and nothing else."* This doesn't exist yet.

Competitive pressure: As users experience fewer interruptions with sandboxed tools, permission-based tools will feel increasingly dated. Market dynamics will drive adoption.

But the deeper shift is philosophical: *we're accepting that AI coding assistants are powerful enough to need serious containment, not just permission prompts.*

That's a significant milestone. We're treating these tools like we treat other powerful-but-risky capabilities: with structural safeguards, not just user vigilance.

Future roadmap — evolution of AI development security approaches

What Should You Do Right Now?

If you're using AI coding tools daily:

Evaluate your actual risk: Are you working on side projects or production systems? Handling personal data or customer data? Your risk profile determines how much security overhead makes sense.

Test sandboxing in non-critical environments: Try it on a side project before deploying to your main workflow. See where it breaks, what you need to configure, and whether it fits your patterns.

Don't assume sandbox = safe: Sandboxing reduces risk, it doesn't eliminate it. You still need code review, dependency scanning, and other security practices.

Voice your requirements: If your preferred AI tool doesn't support sandboxing yet, tell them it matters to you. Market signals drive development priorities.

The permission fatigue problem is real, and ignoring it doesn't make it go away. Sandboxing is the most promising solution I've seen, but it's early days. We're figuring out the right balance between security, usability, and developer experience.

The good news: we're finally taking the problem seriously instead of pretending that human vigilance scales infinitely.

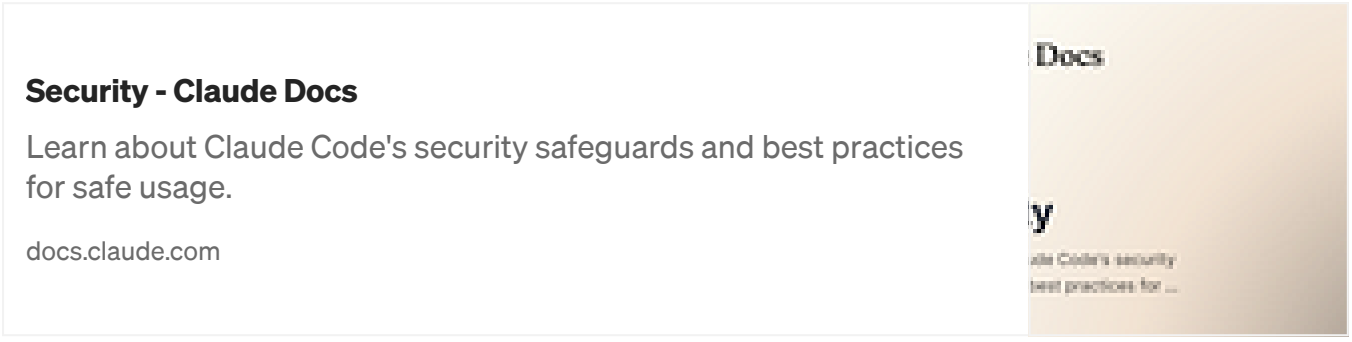
The hard news: there are no perfect solutions, only trade-offs we understand well enough to choose deliberately.

I'm choosing sandboxing for most of my work from now on. But I'm watching the escape hatches carefully, keeping containers ready for sensitive projects, and staying skeptical of any solution that claims to solve this completely.

Security is a spectrum, not a binary. Sandboxing moves us in the right direction, but it's not the finish line.

Checkout also the official Anthropic's Claude Code Documentation:

- Security — Comprehensive security features and best practices
- IAM — Permission configuration and access control
- Settings — Complete configuration reference
- CLI reference — Command-line options including `-sb`



. . .

What's your experience with AI coding tool security? Have you hit the approval fatigue wall?

"Thanks for reading. Share your perspective in the comments — I'm genuinely curious how different teams are tackling this challenge. What else would be valuable for your projects and learning journey? I'll answer all your thoughts and questions."

Continue Learning

Related Articles:

- *Building Production-Grade Claude Code Workflows*
- *From Tribal Knowledge to Organizational Assets: Documentation Patterns That Work*
- When the Ground Shifts: Leading Engineering Teams Through the Anxiety We All Feel

About the Author

Building AI-augmented engineering workflows at the intersection of CTO experience and hands-on architecture and leading product/software engineering teams. Documenting what actually works in production versus what sounds impressive in blog posts.

Previously scaled engineering teams through multiple company restructuring and acquisitions — learned what knowledge compounds and what evaporates without proper systems.

Connect: LinkedIn
Read more: Medium Reza Rezvani
Explore: GitHub

"Thanks for reading. Please share your thoughts and ideas in the comments section and I will answer them. If you are interested in the type of content I am writing about, please let me know what else would be valuable for your projects and learning journey."

Ai Agent | Agentic Ai | Claude Code | AI | Anthropic Claude

Follow

Written by Reza Rezvani

867 followers · 67 following

As CTO of a Berlin AI MedTech startup, I tackle daily challenges in healthcare tech. With 2 decades in tech, I drive innovations in human motion analysis.

No responses yet

Bgerby

What are your thoughts?

More from Reza Rezvani

In nginity by Reza Rezvani

The Flutter Architecture That Saved Our Team 6 Months of Rework

Sep 14

Reza Rezvani

I Discovered Claude Code's Secret: You Don't Have to Build Alone

I've been coding long enough to know that the late-night debugging sessions aren't glamorous. They're just necessary.

Sep 19

Reza Rezvani

The ultimate Code Modernization & Refactoring prompt for your subagent in Claude Code, Codex CLI or...

Transform your legacy codebase chaos into a strategic modernization roadmap with this comprehensive analysis framework.

Oct 4

In nginity by Reza Rezvani

Master Claude Code “Skills“ Tool to transform repetitive AI prompts into permanent, executable...

Discover how the Anthropic’s new tool for Claude Code called “Skills” transform AI Coding assistant from a generic assistant into your...

6d ago

See all from Reza Rezvani

Recommended from Medium

In nginity by Reza Rezvani

Claude AI and Claude Code Skills: Teaching AI to Think Like Your Best Engineer

4d ago

Manojkumar Vadivel

The .claude Folder: A 10-Minute Setup That Makes AI Code Smarter

If you’re new to Claude Code, it’s a powerful AI coding agent that helps you write, refactor, and understand code faster. This article...

Sep 15

In AI Software Engineer by Joe Njenga

Why Claude Weekly Limits Are Making Everyone Angry (And $100/Month Plan Will Not Save You)

Yesterday, I finally hit my weekly Claude limit, and I wasn't surprised, since I see dozens of other users online going crazy over these...

4d ago

In Dare To Be Better by Max Petrusenko

Claude Skills: The $3 Automation Secret That's Making Enterprise Teams Look Like Wizards

How a simple folder is replacing $50K consultants and saving companies literal days of work

6d ago

Riccardo Tartaglia

5 Essential MCP Servers Every Developer Should Know

I've been experimenting with Model Context Protocol servers for a few months now, and I have to say, they've changed the way I work.

Oct 11

aakash

Perplexity Just Unleashed 10 FREE AI Agents That Do Your Entire Job (The “Comet” Shortcut)

Stop what you are doing. The era of the simple AI search engine is officially over.

Oct 7

See more recommendations