

Anthropic’s agents outgrew Git: 30,000 merges a day pushed it onto a custom source-control system

Coding Agents Alpha Tracker

2026-10-23

Anthropic’s agents outgrew Git: 30,000 merges a day pushed it onto a custom source-control system

By Coding Agents Alpha Tracker • October 23, 2026

Anthropic says developer infrastructure, not coding, now limits its output, and Modal and Cursor describe their own infrastructure answers. This brief also covers Theo’s harness-vs-MCP explainer, the Agent Skills donation, a secrets-handling pattern for agents after the OpenCode key leak, and Ben Tossell’s overnight build of a 24-subagent pipeline.

At agent scale, the bottleneck moves to infrastructure

Anthropic says “coding is no longer the bottleneck.” What limits it now is “developer infrastructure built for human scale: source control, builds, and CI.” It averaged about 2,500 merged changes a day in May 2026 and more than 30,000 a day by September [1]. The Pragmatic Engineer reports that about 3,000 staff typically run 10–15 parallel Claude agents each, which adds up to about 25 million Git checkouts a day just for CI [1].

Anthropic moved its monorepo off Git onto **Scion**, a closed-source system with server and client written in Rust. It also built **Antique** for code review, with a Go backend and React frontend [1]. Claude was “instrumental” in the migration. The team’s approach was to get about 99% of use cases covered, force the cutover, and have Claude fix whatever actually broke [1]. Anthropic adds that “most teams don’t need to leave git” [1]. The practical lesson for everyone else is to look at whether your CI and checkout costs will hold up once each engineer runs a dozen agents.

Modal’s Erik Bernhardsson offered a related infrastructure idea on Latent

Space. Treat agents as long-running durable workflows: spin up a sandbox for a few commands, shut it down and snapshot it. Forking those snapshots would let an agent explore hundreds of branches in parallel instead of working sequentially [2]. For CI, he suggests running setup once, snapshotting the container, and cloning it so each test runs in its own sandbox. Today every shard repeats the installs [2]. Other Modal practices:

- **Optional code review.** Docs and basic changes don't wait for a human. Bernhardsson says this only works with rate limits everywhere, decoupled systems, no single points of failure and strong test coverage. He adds that agents are now good at writing tests [2].
- **Sandbox-escape red-teaming.** Modal puts an agent in a sandbox and tells it to escape. It has run this for a few months and found nothing so far [2].
- **Agents copy deprecated syntax from their training data,** so Modal now deprecates features more cautiously and updates examples published across the web [2].

Does the harness still matter?

In a talk, **Jediah Katz** cites studies (UC Berkeley, Apple/EPFL) that found the choice of harness didn't significantly change coding-benchmark scores on newer models. In his view, models have absorbed the old harness jobs: exploring the codebase, working persistently with `/goal`, and running tests [3]. The same studies did find significant differences in cost per task. Tuning cache hit rate, token efficiency and model mix cut cost and latency in his team's harness "without any hit to quality" [3].

Two patterns from the talk you can reuse: - **Software factory.** A bug reported in a channel is auto-triaged. A cloud agent fixes it and records before-and-after videos as proof. After the fix merges, another agent watches the production roll-out [3]. - **One "subscriptions" primitive.** Instead of a separate integration for each trigger, agents get one tool to subscribe to events such as Slack posts, CI going red on a watched PR, or a time of day [3].

Theo published a full explainer on the terms. A harness is the tools, the tool-call syntax, a parser that runs the calls, a way to feed results back, and the list of tool descriptions [4]. Claude Code's TUI and GUI are interfaces sitting on top of that harness [4]. His design point: T3 Code lets Claude Code threads spawn Codex subagents through an MCP server. That way the code lives in T3 Code once and doesn't have to be rebuilt as a plugin for every harness [4].

Related: Anthropic donated **Agent Skills** to the Agentic AI Foundation, under the Linux Foundation, which already hosts MCP [5]. It says 70+ platforms have integrated the standard, including ChatGPT, Cursor, GitHub Copilot and VS Code [6]. Geoffrey Huntley agreed with a post arguing that "you don't need hundreds of skills or agents definitions" [7, 8].

Keeping secrets away from agents

According to Theo, the OpenCode breach exposed provider keys that users had stored on OpenCode’s server side. Developers reported \$400 to \$3,000 in abuse within minutes [9]. His advice is to keep keys “very well separated, and easy to revoke” [9]. He also says agents got him through a day of outages: they triaged incidents and “rolled dozens of api keys across various platforms” [10].

1Password for Agents launched with this flow: the agent requests a credential, you approve it, and 1Password autofills the credential directly so the agent never sees it [11]. Kent C. Dodds says Kody can connect the same setup to any agent and restrict exactly how each secret is used [12].

On prompt injection, Simon Willison is “cautiously starting to suspect” it may not be a problem for today’s strongest models. He says it remains a real concern for cheaper, weaker models [13], and the hard part is *proving* a model or harness has solved it [14].

Anthropic and Theorem Labs say they used Claude to prove security properties in more than 100 real-world open-source libraries over a few weeks [15].

How Ben Tossell built his newsletter pipeline

Ben Tossell’s pipeline syncs his X bookmarks to Supabase every 15 minutes and produces previews and drafts on a schedule [16]. The build process is reusable: - He spelled out edge-case classification rules the agent couldn’t infer, such as recognizing a podcast where he was only the guest [16]. - He ran a “bake-off” with six subagents, each given different amounts of personal context and link content. The winner was simply giving the model his last two newsletter sends as examples [16]. - An OpenAI model in the transcribed “6.1 Sol” line split the build across subagents, with end-to-end verification required. He treats OpenAI models as “workhorse” models and uses Opus 5.5 for design. 24 agents worked overnight [16].

When he compared its first draft with his own, the picks were “more or less line for line the same” [16].

Quick tips

- **swyx** calls Opus 5.5 the best work-estimation model he has used. Its estimates reflect agent speed, such as 2–3 hours instead of a human’s 2–3 days [17]. He says this is extremely important for pipelining, where it estimates ETAs for its own fixes while the pipeline is still running [18].
- **Huntley**: if your code is properly instrumented, agent loops can find $O(n)$ hotspots and fix slow performance automatically [19].
- **LangSmith LLM Gateway** (public beta, Plus and Enterprise) adds spend caps, rate limits, fallbacks and redaction for Claude Code, Codex

or Deep Agents Code. You point the agent at the gateway endpoint and change `base_url`; the rest of your code stays the same [20].

Sources

1. The Pulse: Anthropic builds its own source control system
2. Why AI Compute Comes in Rectangles — Erik Bernhardsson, Modal CEO
3. How SpaceXAI is Redefining the Agent Harness
4. X article by @theo
5. Donating Agent Skills to the Agentic AI Foundation
6. X post by @ClaudeDevs
7. X post by @k_cieslak
8. X post by @GeoffreyHuntley
9. Even The NSA Got Involved
10. X post by @theo
11. X post by @1Password
12. X post by @kentcdodds
13. X post by @simonw
14. X post by @simonw
15. X post by @AnthropicAI
16. My newsletter system
17. X post by @swyx
18. X post by @swyx
19. X post by @GeoffreyHuntley
20. LangSmith LLM Gateway: Runtime Controls for Agents