

Free CLI Turns Engineering Standards Into Executable Checks Across Every Repo

punt-kit reads a team's conventions from markdown documents and enforces them as deterministic audits, scaffolding, and cross-repo rollouts

Same engine projects through four surfaces — terminal CLI, Claude Code plugin, MCP server, and REST API — so humans, AI agents, and CI systems all enforce the same rules

Press Release

SAN FRANCISCO — October 1, 2026 — Punt Labs today released punt-kit, a free, open-source development lifecycle engine that reads engineering standards from markdown documents and enforces them as executable checks across every repository. Unlike linters that check syntax or developer portals that require a dedicated platform team, punt-kit gives small engineering teams (5–50 engineers) a single tool that audits compliance, scaffolds new projects, and rolls out standard changes across multiple repos. The same engine projects through four surfaces — a terminal CLI, a Claude Code plugin, an MCP server for any AI agent, and a REST API for CI and dashboards — so every actor in the development loop enforces the same rules (see [FAQ 1](#)).

Problem

Engineering teams with multiple repositories maintain standards in wikis, READMEs, and the heads of senior engineers. Enforcement happens through code review — a process that consumes an average of four hours per week per engineer on rote standards checks alone [1]. When a standard changes, someone manually updates every repo. When a new project starts, someone copies config files from the last project and hopes they are current [2]. For multi-repo organizations — the norm for teams that do not want Google-scale monorepo infrastructure — configuration drift is the default state [3].

AI coding agents make this worse. A study of 253 CLAUDE.md files found that these manifests are optimized for code execution, not organizational conventions — only 8.7% include security guidance [4]. Each new agent session begins with no memory of what came before [5], and static context files have no feedback loop to detect when the repo has drifted from the standards they describe [6] (see [FAQ 9](#)).

Solution

punt-kit treats standards as data, not documents. Teams write their conventions in markdown — badge ordering, CI template structure, release pipeline patterns, naming rules — and punt-kit reads those documents, extracts checkable assertions, and runs them against any repository. A single command, `punt audit`, checks a repo against all applicable standards and reports every violation with its source rule. A second command, `punt init`, scaffolds a new project that satisfies those standards from the start. A third, `punt rollout`, applies a standard change to every repo in the org at once (see [FAQ 10](#)).

The engine projects through four surfaces: a terminal CLI (`punt`), a Claude Code plugin (`/punt`), an MCP server that any AI agent can call, and a REST API for CI pipelines and dashboards. All four surfaces call the same engine, so the same checks run whether a human types `punt audit`, an AI agent calls the MCP tool, or a CI job hits the API. The agent session gets the standards as context automatically, and `/punt preflight` validates compliance before every commit (see [FAQ 18](#)).

Customer Quote

“Before punt-kit, I spent every Monday morning auditing our twelve repos by hand — checking that CI workflows matched our template, that badges were in the right order, that new projects had the right pyproject.toml settings. I would find three or four repos that had drifted, fix them, and know they would drift again by next month. Now I run `punt audit` across the org in thirty seconds and `punt rollout` fixes everything that has drifted. That Monday morning is gone.”

— **Maria Chen**, Engineering Lead at a 20-person startup

Getting Started

Getting started with punt-kit takes three steps. First, install with `pip install punt-kit` (or `uv tool install punt-kit`). Second, run `punt audit` in any repository to see how it measures against the default standards. Third, customize the standards directory to match the team’s conventions. Within five minutes, a team has a baseline compliance report for every repo. No account, no configuration server, no credit card.

Spokesperson Quote

“We built punt-kit because we kept solving the same problem by hand across our own repos — auditing badge order, checking CI templates, rolling out config changes one repo at a time. The insight was that every one of those checks was a testable assertion, and if we treated our standards as data instead of documents, we could automate the entire loop. The second insight was that the engine should project through every surface — CLI, plugin, MCP server, REST API — so the same checks run whether a human, an AI agent, or a CI pipeline is doing the work. The engine reads your rules; it does not decide what ‘good’ looks like. That means any team, using any tool, can bring their own conventions and get the same enforcement.”

— **J. Freeman**, Creator, Punt Labs

Call to Action

punt-kit is available today at github.com/punt-labs/punt-kit and on PyPI as `punt-kit`. The tool is free and open-source under the MIT license. Documentation, standards templates, and a quickstart guide are at punt-labs.github.io/punt-kit.

Frequently Asked Questions

External FAQs

Q1: What is punt-kit and who is it for?

punt-kit is a development lifecycle engine that reads engineering standards from a directory of markdown documents and enforces them as executable checks. It is designed for engineering leads and platform-minded engineers at teams of 5–50 people who maintain multiple repositories and want their standards enforced automatically — not just documented. The engine projects through four surfaces: a CLI (punt) for terminal use, a Claude Code plugin (/punt) for AI coding sessions, an MCP server so any AI agent can call the same checks, and a REST API for CI pipelines and dashboards (see [FAQ 6](#)).

Q2: How is this different from projen, Backstage, or Trunk?

Each of these tools solves a narrower problem:

- **projen** [7] enforces project configuration through TypeScript project types. It is the strongest prior art for “standards as code across repos.” punt-kit differs in three ways: it reads standards from existing markdown rather than requiring TypeScript; it integrates with AI coding agent sessions as a first-class citizen; and it targets teams that already have standards in docs, not greenfield CDK infrastructure teams.
- **Backstage** [8] is a developer portal for large enterprises (3,000+ adopters, but adoption stalls below 10% in most non-Spotify organizations). It requires a dedicated full-time team to run. punt-kit targets 5–50 engineer teams that cannot staff a platform team.
- **Trunk** [9] is a metalinter that orchestrates 100+ linting tools with unified output. It covers the linting layer. punt-kit covers the layer above it: project structure, CI templates, release pipelines, naming conventions, and cross-repo rollout.

No existing tool unifies audit, scaffold, rollout, and release under a single standards-as-data model with AI agent integration (see [FAQ 10](#)).

Q3: How do I get started?

Install with `pip install punt-kit` or `uv tool install punt-kit`. Run `punt audit` in a repository. Review the findings. Customize the standards directory if the defaults do not match your team’s conventions. Most teams see their first compliance report within five minutes.

Q4: How much does it cost?

punt-kit is free and open-source under the MIT license. There are no paid tiers, no usage limits, and no telemetry. The tool runs entirely on your machine; it does not phone home (see [FAQ 16](#)).

Q5: What happens to my data?

punt-kit runs locally. It reads files from your repositories and your standards directory. It does not transmit data to any external server. The MCP server and REST API run on your machine (or your infrastructure); findings are never sent elsewhere. When used as a Claude Code plugin, the standards context is provided to the AI agent session but never stored by punt-kit itself.

Q6: Why does punt-kit have four surfaces (CLI, plugin, MCP, API)?

Standards enforcement only works if every actor in the development loop uses the same checks. A human typing in a terminal needs a CLI. An AI coding agent in Claude Code needs a plugin. A different AI agent (Cursor, Windsurf, Codex) needs an MCP server it can call. A CI pipeline or internal dashboard needs a REST API. All four call the same engine — the checks, the standards, and the findings are identical regardless of which surface invoked them. This projection pattern means adopting punt-kit does not lock a team into any single agent or workflow.

Q7: Can I use punt-kit with my own standards instead of the defaults?

Yes. The engine and the standards pack are separable. punt-kit ships with the punt-labs conventions as the default, but any team can point it to their own standards directory — a local folder, a git repository, or a URL. The engine reads the rules; it does not decide what “good” looks like (see [Feature 4](#)).

Internal FAQs

Value & Market

Q8: What is the total addressable market?

Bottom-up estimate (with caveats):

The platform engineering services market was valued at \$7.19 billion in 2024 and is projected to reach \$40.17 billion by 2032, growing at 23.99% CAGR [10]. Gartner predicts that 80% of large software engineering organizations will have platform engineering teams by 2026, up from 45% in 2022 [11]. Google Cloud found that 55% of global organizations have already adopted platform engineering, with 86% saying it is essential to realizing the full business value of AI [12].

Honest gap: These numbers describe large organizations with dedicated platform teams — the opposite of punt-kit’s target segment. No public data source directly measures how many 5–50 engineer teams maintain multiple repos with codified standards. GitHub hosts 3+ million organizations, but there is no breakdown by team size or standards maturity.

Hypothesis: Smaller teams have the same standards-enforcement pain but cannot afford Backstage or a dedicated platform engineer. punt-kit’s bet is that this segment is underserved precisely because existing solutions require enterprise-scale investment. This hypothesis is unvalidated and is the highest-priority item to test (see FAQ 12).

Q9: What evidence do we have that customers want this?

At hypothesis stage, we have no primary customer evidence. What we have instead:

- **Dogfooding:** punt-kit was built to solve a real problem across 14 repositories in the punt-labs org. The tool exists because the manual alternative was unsustainable.
- **Academic evidence:** Four arXiv papers from 2025–2026 validate the context problem. A study of 283 development sessions found that single-file agent manifests “do not scale beyond modest codebases” [13]. A study of 2,303 context files found security specified in only 14.5% [14]. Agent sessions without context files consume 28.64% more runtime and 16.58% more tokens [15].
- **Industry signals:** Code review consumes an estimated 4 hours per week per engineer on standards enforcement that could be automated [1]. 88% of developers cite negative impacts from AI-generated code [16]. 23% of developers use AI agents at least weekly and that number is growing [17].
- **Competitor existence:** projen, Trunk, Copier, and Backstage each solve a slice of this problem, confirming market demand for at least partial solutions.

The missing evidence is direct conversations with 5–50 engineer teams outside punt-labs confirming that they prioritize this problem highly enough to adopt a new tool.

Q10: Who are the competitors and why will we win?

Tool	What It Does Well	Gap punt-kit Fills
projen	Standards-as-code for project config across N repos [7]	Requires TypeScript; AWS-centric; no AI agent integration; high learning curve
Backstage	Developer portal with golden path scaffolding [8]	Requires dedicated team to run; stalls at under 10% adoption outside Spotify
Trunk	Metalinter orchestrating 100+ tools [9]	Covers linting only; no scaffolding, roll-out, or release management
Copier	Template scaffolding with lifecycle updates [18]	No enforcement/audit; no AI integration; template-only
MegaLinter	Open-source CI linting orchestrator	CI-focused; no scaffolding or release; similar scope to Trunk

punt-kit's structural advantage is the combination of four properties no competitor has together: (1) standards from existing markdown, not new TypeScript or YAML; (2) unified lifecycle (audit + scaffold + rollout + release) under one data model; (3) a projection architecture that serves CLI, plugin, MCP, and REST from the same engine; (4) agent-agnostic integration — any tool that speaks MCP can enforce standards, not just Claude Code. If a competitor copies one of these, the remaining three still differentiate.

Q11: What is the customer acquisition strategy?

Primary channel: Claude Code plugin marketplace. punt-kit is already published on the punt-labs marketplace. Engineers who use Claude Code discover `/punt audit` in their workflow and adopt the CLI for terminal use.

Secondary channels:

- PyPI discovery (`pip install punt-kit`)
- GitHub search and stars (open-source visibility)
- Blog posts and conference talks on “standards as executable assertions”
- Word of mouth from dogfooding across punt-labs projects

Conversion assumption: Free, open-source, no signup friction. Adoption is limited by awareness and by whether the problem is painful enough to motivate a tool switch, not by pricing barriers.

Q12: What is the next step to validate the vision?

Interview 15–20 engineering leads at Series A/B companies with 5–50 engineers and 5+ repositories. The interview should answer three questions:

1. Is standards enforcement a top-3 pain point for you?
2. How do you currently propagate standard changes across repos?
3. Would you adopt a CLI that audits and fixes standards compliance automatically?

If fewer than 8 of 20 interviewees identify this as a top-3 pain, the target segment hypothesis is wrong and we should either broaden the customer definition or narrow to a more specific persona.

Technical

Q13: What are the major technical risks?

1. **Standards parsing ambiguity.** Markdown is not a schema language. Extracting checkable assertions from prose requires either structured conventions in the markdown or LLM-assisted parsing. Mitigation: the engine uses deterministic parsing first and falls back to LLM only for genuinely ambiguous cases (see [Feature 2](#)).
2. **Cross-repo rollout safety.** Automatically modifying files across N repos risks breaking things at scale. Mitigation: dry-run mode previews every change before applying; rollout never force-pushes or merges without green CI (see [Feature 8](#)).
3. **Claude Code plugin API stability.** The plugin hooks and slash commands depend on Claude Code's extension API, which is evolving. Mitigation: punt-kit already ships a working plugin (v0.2.0) and tracks API changes as part of ongoing development.

Q14: What dependencies exist on other teams or systems?

punt-kit depends on Claude Code for plugin distribution and on PyPI for CLI distribution. Both are stable, public infrastructure. There are no dependencies on internal services, proprietary APIs, or third-party SaaS. The LLM-powered reconciliation feature (`/punt reconcile`) requires an Anthropic API key, but all deterministic features work without one.

Q15: What is the estimated development timeline?

punt-kit v0.1.0 shipped February 2026 with `/punt audit`, `/punt reconcile`, `/punt init`, and `/punt pii`. The vision document describes ten additional commands. Phased delivery:

- **Phase 1 (shipped):** Audit, reconcile, init, PII scan
- **Phase 2 (Q2 2026):** Preflight, per-standard audit subcommands, doctor
- **Phase 3 (Q3 2026):** Release automation, changelog, status
- **Phase 4 (Q4 2026):** Cross-repo rollout, PR workflow
- **Phase 5 (2027):** Autopilot (autonomous backlog execution)

If timeline compresses, Phase 5 is cut. Phase 2 is the minimum viable next milestone; it validates the "standards as data" engine pattern with deterministic, per-rule checks.

Business

Q16: What is the revenue model?

punt-kit is free and open-source with no planned paid tiers. The strategic value is:

- **Ecosystem presence:** punt-kit establishes Punt Labs as a credible voice in developer tooling, driving awareness for adjacent products (quarry, biff).
- **Dogfooding infrastructure:** punt-kit enforces standards across all punt-labs projects, reducing maintenance cost for the org's own tools.
- **Platform engineering credibility:** The standards-as-data approach, if validated, positions Punt Labs to offer consulting or enterprise features later. But monetization is explicitly not the goal for v1.

Q17: What are the key metrics for success?

1. **Adoption:** 50 GitHub stars and 5 non-punt-labs organizations using punt-kit within 6 months of v1.0. Measurement: GitHub stars, PyPI download counts, Claude Code plugin install count.
2. **Retention:** Of adopting orgs, 3+ run punt audit at least weekly after 3 months.
3. **Standards pack diversity:** At least 2 non-punt-labs standards packs contributed or published within 12 months (evidence that the engine/pack separation works).
4. **Decision threshold:** If after 6 months, fewer than 3 non-punt-labs organizations are using punt-kit actively, revisit the customer definition and consider whether the tool should remain internal-only.

Q18: Why now? What has changed?

Three shifts converged:

1. **AI agents are generating more code faster.** 23% of developers use AI agents at least weekly [17], and developers use AI in 60% of their work [19]. More code, generated faster, with no awareness of organizational conventions means standards enforcement scales worse than ever.
2. **Agent context infrastructure is maturing.** CLAUDE.md, AGENTS.md, and Claude Code plugins now provide hooks to inject standards context into every agent session. This infrastructure did not exist 18 months ago.
3. **Platform engineering is going mainstream but tools assume enterprise scale.** 55% of organizations have adopted platform engineering [12], but the leading tools (Backstage, projen) require dedicated teams. Small teams need the same enforcement without the overhead.

Risk Assessment

Risk	Rating	Assessment
Value	Medium	The problem is real and well-evidenced by academic research and industry signals. The risk is whether 5–50 engineer teams consider standards enforcement a top-3 pain point worth adopting a new tool for. No primary customer evidence exists outside dogfooding. The “Why now” argument (AI agents amplifying drift) is strong but untested with the target segment.
Usability	Low	Three-step onboarding with zero configuration. The tool fits into existing terminal and Claude Code workflows rather than replacing them. Time-to-value is under five minutes. The main usability risk is standards parsing: if teams’ existing markdown documents are too unstructured for the engine to parse, the setup cost increases.
Feasibility	Low	Core technology is proven. v0.1.0 is shipped and running across 14 repositories. The team has domain expertise (the tool was built to solve the team’s own problem). The hardest unsolved problem is cross-repo roll-out safety, which is deferred to Phase 4. No dependencies on unproven technology.
Viability	Medium	punt-kit is free with no direct revenue plan. Strategic value (ecosystem presence, dogfooding infrastructure) is real but hard to measure. The risk is opportunity cost: time spent on punt-kit is time not spent on revenue-generating products. Mitigated by the fact that punt-kit directly reduces maintenance cost across all other punt-labs projects.

Feature Appendix

Defines the scope boundary for the product. Every feature is explicitly categorized to prevent scope creep and force prioritization decisions upfront.

Must Do

Features that are essential for launch. Without these, the product does not solve the core problem.

- F1. Standards-as-data engine** — reads a directory of markdown documents, extracts checkable assertions, and runs them against a repository
- F2. Deterministic audit** — per-standard checks (README badges, Python config, CI templates, shell scripts, naming conventions) that exit 0 or 1 with structured findings
- F3. Preflight** — run quality gates and PII scan on staged files before commit — the daily-use entry point
- F4. Custom standards directory** — teams point the engine at their own rules, not just punt-labs defaults
- F5. Claude Code plugin** — slash commands (`/punt audit`, `/punt preflight`) that inject standards context into AI agent sessions
- F6. MCP server** — exposes audit, init, and preflight as MCP tools so any AI agent (Cursor, Windsurf, Codex, custom agents) can enforce the same standards without a plugin

Should Do

Features that meaningfully improve the product but are not launch-blocking. Candidates for fast follow-up.

- F7. Release automation** — bump version, tag, push, and monitor the full pipeline (build, Test-PyPI, test-install, PyPI) in one command
- F8. Cross-repo rollout** — detect which repos need a standard change, preview the delta, apply, commit, push, and report across N repos
- F9. Project doctor** — health check that verifies MCP wiring, plugin installation, quality gate configuration, and beads initialization
- F10. Status dashboard** — combined view of beads in-progress, git state, CI status, and unreleased commits
- F11. REST API** — HTTP interface for CI pipelines and internal dashboards to query audit findings programmatically

Won't Do

Features explicitly excluded. Naming what we will not build prevents scope creep and clarifies the product's identity.

- F12. GUI or web dashboard** — punt-kit is a terminal tool and a Claude Code plugin. A web interface would split focus and target a different user. Teams that want a portal should use Backstage
- F13. CI system replacement** — punt-kit generates and validates CI workflows. It does not replace GitHub Actions, GitLab CI, or any CI runner
- F14. Code parsing or linting** — punt-kit orchestrates linters (ruff, mypy, shellcheck) and validates their configuration. It does not parse source code. Teams that want a metalinter should use Trunk or MegaLinter

F15. Package management — validates distribution config and automates release pipelines; does not replace PyPI, npm, or uv

References

- [1] DeepSource. *Engineering Manager's Guide to Static Analysis*. Quantifies static analysis at 4 hours/week/engineer saved on code reviews. 2025. URL: <https://deepsource.com/blog/engineering-manager-guide-to-static-analysis>.
- [2] GitHub. *Promote Consistency Across Your Organization with Workflow Templates*. Checking many repos for consistency exposes teams to human error and reduced visibility. 2020. URL: <https://github.blog/2020-06-22-promote-consistency-across-your-organization-with-github-actions/>.
- [3] Trunk.io. *Streamlining Development Workflows: The Secret to Taming Multi-Repo Chaos*. Multi-repo consistency requires deliberate effort; configuration drift without automation. 2024. URL: <https://trunk.io/blog/why-you-want-consistent-tooling-across-your-organization>.
- [4] Matteo Ciniselli et al. *On the Use of Agentic Coding Manifests: An Empirical Study of Claude Code*. 253 CLAUDE.md files from 242 repos; manifests optimized for code execution not org conventions; security in 8.7%. 2025. URL: <https://arxiv.org/abs/2509.14744>.
- [5] Anthropic. *Effective Context Engineering for AI Agents*. "Each new session begins with no memory of what came before". 2025. URL: <https://www.anthropic.com/engineering/effective-context-engineering-for-ai-agents>.
- [6] Anthropic. *Manage Claude's Memory — Claude Code Docs*. CLAUDE.md loaded every session but is static with no feedback loop to detect drift. 2025. URL: <https://code.claude.com/docs/en/memory>.
- [7] projen contributors. *projen: Rapidly Build Modern Applications with Advanced Configuration Management*. Synthesized config files should never be manually edited; custom project types can update N repos; AWS-centric, TypeScript-first. 2024. URL: <https://github.com/projen/projen>.
- [8] Spotify Engineering. *Celebrating Five Years of Backstage*. 3,000+ organizations; adoption stalls at under 10% in most non-Spotify orgs; requires dedicated full-time team. 2025. URL: <https://engineering.atspotify.com/2025/4/celebrating-five-years-of-backstage>.
- [9] Trunk.io. *Trunk Code Quality: Automated Code Quality for Teams*. Metalinter running 100+ tools; CI-focused; does not scaffold, manage releases, or integrate with AI agents. 2024. URL: <https://trunk.io/code-quality>.
- [10] SNS Insider. *Platform Engineering Services Market Size to Hit USD 40.17 Billion by 2032*. Market valued at \$7.19B in 2024, projected \$40.17B by 2032, CAGR 23.99%. SNS Insider, 2024. URL: <https://finance.yahoo.com/news/platform-engineering-services-market-size-133000981.html>.
- [11] Gartner. *Platform Engineering Empowers Developers to be Better, Faster, Happier*. 80% of large software engineering orgs will have platform engineering teams by 2026, up from 45% in 2022. 2022. URL: <https://www.gartner.com/en/experts/top-tech-trends-unpacked-series/platform-engineering-empowers-developers>.
- [12] Google Cloud. *New Platform Engineering Research Report*. 55% of global orgs adopted platform engineering; 90% plan to expand; 86% say essential to AI value. Google Cloud, 2024. URL: <https://cloud.google.com/blog/products/application-modernization/new-platform-engineering-research-report>.

- [13] Jonah Katz et al. *Codified Context: Infrastructure for AI Agents in a Complex Codebase*. 283 sessions on a 108,000-line codebase; single-file manifests do not scale; three-component infrastructure for persistent agent context. 2026. URL: <https://arxiv.org/abs/2602.20478>.
- [14] Filipe Calegario et al. *Agent READMEs: An Empirical Study of Context Files for Agentic Coding*. 2,303 context files from 1,925 repos; security in only 14.5% of files; build commands dominate at 62.3%. 2025. URL: <https://arxiv.org/abs/2511.12884>.
- [15] Lucas Larson et al. *On the Impact of AGENTS.md Files on the Efficiency of AI Coding Agents*. 10 repos, 124 PRs; AGENTS.md presence reduces runtime 28.64% and token consumption 16.58%. 2026. URL: <https://arxiv.org/abs/2601.20404>.
- [16] Sonar. *2026 State of Code Developer Survey Report*. 1,149 developers; 88% cite negative AI code impacts; 96% do not fully trust AI-generated code; 42% of committed code is AI-generated. Sonar, 2026. URL: <https://www.sonarsource.com/state-of-code-developer-survey-report.pdf>.
- [17] Stack Overflow. *2025 Stack Overflow Developer Survey*. 84% of developers use or plan to use AI tools; 23% use AI agents at least weekly; 49,000+ respondents. 2025. URL: <https://survey.stackoverflow.co/2025/>.
- [18] Copier contributors. *Comparisons — Copier Documentation*. Lifecycle management: updating existing projects when templates evolve; confirms propagation problem is real. 2024. URL: <https://copier.readthedocs.io/en/stable/comparisons/>.
- [19] Anthropic. *2026 Agentic Coding Trends Report*. Developers use AI in 60% of work; AI agents market projected at \$52.62B by 2030 at 46.3% CAGR. 2026. URL: <https://resources.anthropic.com/2026-agentic-coding-trends-report>.