biff

Team communication for engineers who never leave the terminal

February 2026 | Jim Freeman | punt-labs | v2.7

Press Release

Summary

San Francisco, CA — August 2026 — Today, punt-labs announced **biff 1.0**, an open-source communication tool that lets software engineers message teammates, check presence, see what their agents are doing, and broadcast to their team — all without leaving the terminal. Unlike Slack and Discord, which demand a separate window and constant attention, biff runs inside the engineer's existing Claude Code session as MCP-native slash commands. Engineers type `/write @kai` the same way they type `git push` — in flow, with intent, and then move on.

Problem

Software engineers using AI-assisted coding tools like Claude Code are experiencing a step change in productivity. Entire features that once took days now take hours. But every time these engineers need to coordinate with a teammate — asking a question, sharing a diff, requesting a code review — they context-switch to Slack or Discord, tools designed for managers and all-day-online knowledge workers, not for makers in deep focus. Worse, those tools show *everything* — every channel, every project, every server — regardless of which codebase is open in the terminal right now.

The cost is not hypothetical. Gloria Mark's research at UC Irvine found that a single interruption costs an average of 23 minutes and 15 seconds before a worker returns to the original task.[1] For an engineer shipping three features a day with AI assistance, three Slack interruptions can erase an entire feature's worth of productive time. Slack's always-on presence model compounds the problem: even *checking* for messages breaks flow, and the social expectation of quick replies creates anxiety that degrades concentration even when no message arrives.

As Thomas Dohmke, former CEO of GitHub, observed when launching Entire: "The terminal has become the new center of gravity on our computers again" — and the best engineers can now run a dozen agents at once.[2] The engineers building the future of software — the ones spending their entire day inside a terminal, collaborating with AI agents, iterating at speeds that would have seemed absurd two years ago — have no communication tool that matches how they actually work.

Solution

[1] Mark, G., Gudith, D., & Klocke, U. (2008). The Cost of Interrupted Work: More Speed and Stress. *Proc. CHI 2008*, ACM.

[2] Dohmke, T., "Hello Entire World," Entire blog, February 2026, https://entire.io/blog/hello-entire-world/

Biff brings communication into the terminal as slash commands borrowed from the BSD Unix utilities that shipped with every workstation in the 1980s — `write`, `wall`, `finger`, `who`, and `mesg` — updated for the MCP era. A typical morning might look like this:

```
/plan “refactoring the auth layer”
/write @kai “auth module is ready for review”
/wall “pulling the auth changes into the feature branch”
...thirty minutes of focused work later...
/read
```

Every command implies intent. There are no channels to monitor, no threads to catch up on, no emoji reactions to parse. Communication is repo-scoped and pull-based: when you open a project, you see only the teammates and messages for *that* project — not the ten others on the back burner. When someone sends you a message, your session notifies you — your AI assistant mentions it naturally (“You have 2 unread messages from @kai. Want me to read them?”) and your terminal’s status bar shows a live indicator — `kai:tty1(3)` in bold yellow when you have unreads, `kai:tty1(0)` when caught up, `kai:tty1(n)` when messages are off. You decide when to engage. The full command reference is in Appendix A.

Because biff speaks MCP, agents participate alongside humans as natural members of the team. An autonomous coding agent can `/plan` what it is working on, `/write` a human when it needs a decision, and show up in `/who` alongside everyone else. Each agent gets a distinct identity via `/tty` sessions, and `/who` shows host and directory per session so you can see where each agent is working. Workflow hooks tie into your development process: claiming a task auto-sets your plan, creating a PR triggers a `/wall` announcement. All messages are end-to-end encrypted via NaCl/libsodium, so the relay never sees message contents. Biff is designed for the humans who work alongside agents, not for agents that work without humans — the same intentional, human-scale communication vocabulary that made BSD utilities great, updated for teams where some of the members happen to be AI.

Customer Quote

> *“I used to mass-quit my Slack channels once a month out of frustration and then guiltily rejoin them. With biff, I just set* `/mesg n` *when I’m deep in a problem and* `/mesg y` *when I come up for air. My teammates can* `/finger @priya` *to see what I’m on without interrupting. Now I run three agents alongside my own session —* `/who` *shows all four of us, each with its own plan, and when one of them needs a decision it just* `/write`*s me. Last week I shipped more in three days than I used to ship in two weeks, and I didn’t miss a single important message.”*
>
> — **Priya Chandrasekaran**, Senior Engineer at a Series B startup

Getting Started

Getting started with biff takes three commands: `pip install biff`, then `biff install` to register the MCP server and slash commands, then `biff doctor` to verify the setup. Restart Claude Code, and you are live. Biff ships with a shared relay on Synadia Cloud with end-to-end encryption, so there is zero infrastructure to provision. Team configuration lives in a `.biff` file committed to your git repository — it only needs a team member list; the relay URL defaults to the shared relay. Your display name is resolved automatically from your GitHub identity. When you clone a repo that has a `.biff` file, biff automatically connects you to the right team. There is no account to create, no workspace to configure, and no browser to open. Type `/who` to see your teammates. Within five minutes of installation, you can send your first message, see your team's presence, and set your plan.

Spokesperson Quote

> "We built biff because we watched the best engineers in the world disappear into their terminals — shipping at speeds that make the old sprint-and-standup cadence look quaint — and then lose hours every day to communication tools designed for a different era. Slack was built for the open-office, always-online workplace. Biff is built for the deep-focus, AI-accelerated one. We didn't add a chat feature to the terminal. We resurrected the Unix communication vocabulary — `write`, `wall`, `finger`, `who`, `mesg` — because those commands understood something Slack forgot: communication should be purposeful, not ambient."
>
> — **Jim Freeman**, Founder, punt-labs

Call to Action

Biff is open source and available today at github.com/punt-labs/biff. Install with `pip install biff`, then run `biff install` and `biff doctor`. The core tool, end-to-end encryption, and the shared relay are free and always will be. For larger teams that need uptime guarantees, audit logs, and admin controls, biff teams is available as a paid tier.

Frequently Asked Questions

External FAQs

Q: What is biff and who is it for?

Biff is a terminal-native communication tool for software engineers who use Claude Code or other MCP-compatible systems. If you spend your working day in a terminal and resent switching to a browser to talk to your team, biff is for you. It provides eight slash commands covering messaging, presence, broadcast, session identity, and availability control — all without leaving your session. See Appendix A for the full command reference.

Q: How is biff different from Slack?

Slack is a workplace chat platform designed around channels, threads, and continuous presence. It assumes you are watching. Biff is a communication tool designed around intent and focus. It assumes you are working. In Slack, the default state is "available and monitoring." In biff, the default state is "heads down; interrupt me if it matters." Slack is optimized for managers who need visibility. Biff is optimized for engineers who need concentration.

Structurally, biff runs inside your existing development environment as MCP slash commands. There is no separate app, no browser tab, no notification center. Communication happens where your code already lives. And because team configuration lives in a `.biff` file committed to the repo, communication is scoped to the project you are working on right now. Slack shows you every channel from every project simultaneously; biff shows you only the teammates and messages relevant to the code in your terminal.

Q: How is biff different from Discord?

Discord is a community platform built around persistent voice and text channels. It works well for open-source communities and gaming. Biff is a team tool built around directed messages and explicit intent. Discord is for hanging out. Biff is for getting things done. Biff also runs natively inside your AI coding session — there is no alt-tab.

Q: How do I get started?

Run `pip install biff`, then `biff install` to register the MCP server and slash commands, then `biff doctor` to verify the setup. Restart Claude Code. Biff ships with a shared demo relay on Synadia Cloud, so there is no infrastructure to provision. If your repo has a `.biff` file (committed by a teammate), biff picks up the team roster automatically. Your display name is resolved from your GitHub identity. Type `/who` to see your team. No account creation, no workspace to configure. You are communicating within five minutes.

Q: What happens to my messages? Is communication private?

All direct messages are end-to-end encrypted using NaCl/libsodium. The relay stores only ciphertext it cannot read. Biff follows a POP mail model: messages are held only until the recipient reads them, then discarded. Biff is not a system of record. The shared relay is hosted on Synadia Cloud; teams that want data sovereignty can self-host the same NATS server and route all traffic through their own infrastructure. Biff never analyzes or trains on your messages.

Q: Does biff work without Claude Code?

Biff is built on the Model Context Protocol (MCP), which is an open standard. Any MCP-compatible client can use biff. Claude Code is the primary target today because it has the largest population of engineers living in the terminal. As other MCP clients mature, biff will work there too.

Q: Will biff support real-time pairing or live conversation?

Real-time features are on the roadmap but not yet shipped. `/talk` would open a live text conversation between two terminals. `/pair` would allow a teammate to send input to your Claude Code session, with explicit per-session consent, full visibility for the session owner, and a kill switch at any time. Both features require careful security design and are scoped to a future phase.

Q: Can agents use biff?

Yes. Because biff is an MCP server, agents participate alongside humans as natural members of the team. An autonomous coding agent can `/plan` what it is working on, `/write` a human when it needs a decision, and show up in `/who` alongside everyone else. Each agent gets a distinct identity via `/tty`, targetable via `/write @user:tty`. `/who` shows host and directory per session so you can see where each agent is working. Workflow hooks automatically set plans and announce milestones.

Biff is designed for hybrid teams — humans and agents working together — not for pure agent-to-agent coordination. If you need agents to coordinate autonomously without human involvement (shared memory, high-throughput task dispatch, swarm orchestration), dedicated tools like claude-flow and other agent frameworks are better suited for that. Biff serves the human-scale communication layer: the place where humans see what their agents are doing, and agents can reach a human when they need one.

Internal FAQs

Value & Market

Q: What is the total addressable market?

The immediate market is engineers using AI coding assistants inside terminals. GitHub's Octoverse 2025 report counts over 180 million developers on the platform, with 36 million joining in the past year alone.[3] AI-assisted coding adoption is accelerating — GitHub Copilot reached 20 million cumulative signups by mid-2025 and 4.7 million paid subscribers by late 2025.[4] Claude Code launched in early 2025 as the primary MCP client. The subset of engineers who work primarily in the terminal with MCP-compatible tools is small today (estimated tens of thousands) but growing fast as MCP becomes the standard protocol for AI tool integration.

Bottoms-up: if 50,000 engineers adopt biff in year one, and 20% of them are on teams that convert to the paid hosted relay at $10/user/month, that represents $1.2M ARR. The real opportunity scales in two dimensions: as MCP adoption grows, the human addressable market multiplies; and as engineers run more agents alongside their own sessions, each human

[3] GitHub, "Octoverse 2025," `https://github.blog/news-insights/octoverse/octoverse-2025/`

[4] Microsoft Q2 FY2026 earnings; GitHub blog, "Octoverse 2025."

represents a growing number of sessions that benefit from biff's visibility and communication layer.

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

Four categories of evidence:

1. **Behavioral signal:** The explosive growth of CLI-native tooling (Claude Code, Cursor's terminal mode, Warp, Ghostty) demonstrates that engineers are consolidating their workflow into the terminal. The ecosystem of MCP servers, Claude Code plugins, and slash commands exists precisely because engineers want to *stay in flow*. Communication is the last major activity that forces them out.
2. **Cultural signal:** Steve Yegge's February 2026 article "The Anthropic Hive Mind"[5] profiles engineering teams where "the whole team is pair programming at once" and everyone sees each other's work in real time. He reports engineers achieving 10–100x productivity gains but argues this requires full transparency at AI speeds — otherwise "nobody will ever see what you are doing." This is exactly the problem biff solves: visibility and coordination at AI speed, inside the tool.
3. **Investment signal:** Thomas Dohmke, former CEO of GitHub, launched Entire in February 2026 with a $60M seed round — the largest seed investment ever for a developer tools startup.[6] His thesis: "the terminal has become the new center of gravity on our computers again." Entire focuses on the code context layer (how AI changes get reviewed and governed). Biff focuses on the human coordination layer (how engineers communicate while working at AI speed). The same tectonic shift creates both opportunities.
4. **Historical precedent:** The original BSD Unix included `write`, `talk`, `wall`, `finger`, and `mesg` as standard utilities because the designers understood that engineers who share a system need lightweight, intentional communication. Those tools died when engineers left the terminal for GUIs. Now that engineers are returning to the terminal, the need has returned with them.

Q: How will engineers discover biff?

Three channels, in order of expected impact:

1. **Viral loop via `.biff` file.** When a team member commits a `.biff` file to a shared repo, every engineer who clones that repo gets the team roster and relay configuration automatically — the project carries its own communication context. The install path is two commands. This is the primary growth mechanism: one adopter seeds an entire team, and every clone of the repo is a distribution event.
2. **Developer community launch.** Hacker News, dev Twitter/Bluesky, and the Claude Code plugin marketplace. The Unix nostalgia angle ("biff is back") and the anti-Slack positioning generate organic interest. Target: 500 installs in the first week.
3. **Content and conference talks.** Blog posts on terminal-native communication, talks at developer conferences. The open-source model means the project itself is the marketing — every GitHub star, every fork, every `.biff` file committed is a distribution event.

[5] Yegge, S., "The Anthropic Hive Mind," Medium, February 2026, https://steve-yegge.medium.com/the-anthropic-hive-mind-d01f768f3d7b

[6] Dohmke, T., "Hello Entire World," Entire blog, February 2026, https://entire.io/blog/hello-entire-world/

The real cost of acquisition is time, not money. Roughly 80% of the go-to-market effort is founder time: writing blog posts, creating interest within personal networks, answering questions on forums, giving conference talks, and nurturing early adopters through the first-week onboarding. The remaining 20% is modest spend on conference sponsorships and targeted ads. The product itself is free and the viral loop is built into the protocol (`.biff` commits with the repo), so the monetary CAC is near zero — but the time investment to reach critical mass on each team is the true bottleneck.

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

Slack is the dominant team communication tool but is increasingly resented by engineers for its interruption-driven model. Slack could build a CLI client, but its revenue depends on engagement metrics (messages sent, channels monitored, time-in-app) that are fundamentally opposed to biff's focus-first model. Slack's incentives make it structurally unable to optimize for fewer interruptions.

Discord serves developer communities well but is not a team coordination tool. Its voice-channel model does not map to directed, intentional engineering communication.

GitHub has discussions, issues, and PR comments — all web-based, all requiring a browser. The `gh` CLI covers some workflows but offers no real-time messaging or presence.

Zulip and **Mattermost** are open-source Slack alternatives with the same fundamental model: channels, threads, browser or desktop app. They solve the vendor-lock-in problem but not the context-switch problem.

Claude Code (and other MCP hosts) could build messaging natively. This is the most credible competitive threat. Mitigation: biff is open source and protocol-level, not tied to a single host. Anthropic's incentive is to grow the MCP ecosystem, not to build every tool themselves. If Claude Code ships native messaging, biff's MCP protocol can interoperate with it rather than compete.

Agent coordination frameworks (claude-flow, hive-mind, swarm orchestrators) solve a different problem: pure agent-to-agent coordination without human involvement. They optimize for shared memory, high-throughput task dispatch, and machine-speed orchestration. Biff does not compete with these tools — it complements them. A team might use claude-flow for agent-to-agent task routing and biff for the human-scale layer: seeing what agents are doing, messaging teammates, and having agents reach a human when they need one.

Biff wins by refusing to play Slack's game. It is not a chat app with a CLI client. It is a communication protocol native to the environment where engineers already work, designed for hybrid teams of humans and agents. The moat is the MCP ecosystem: as more tools adopt MCP, biff's integration surface grows without additional development.

Technical

Q: What are the major technical risks?

1. **MCP notification mechanism (low risk, shipped).** The full chain — message arrival, tool re-registration, `notifications/tools/list_changed` over SSE, model awareness — is shipped and working in production. When a message arrives, the model sees the updated tool description (e.g., "Check messages (1 unread: @kai about auth)") and proactively mentions it to the user. All MCP interactions are wrapped behind an internal abstraction layer for spec-maturity resilience.

2. **User-visible notification gap (low risk, shipped).** Dynamic tool descriptions are visible to the AI model but not directly to the human user in the Claude Code UI. Biff ships status bar integration (`biff install` includes setup) showing `user:tty(N)` with three states: caught up, unreads (bold yellow), and messages off (`n`). Combined with model-level awareness via tool descriptions, the notification gap is addressed through two complementary channels.
3. **Message relay infrastructure (low risk, shipped).** The NATS relay is shipped and tested against hosted NATS on Synadia Cloud. Biff ships with a shared demo relay so teams can start with zero infrastructure. The relay follows a POP mail model: messages are stored only until the recipient reads them, then discarded. Teams that want data sovereignty can self-host the same single-binary NATS server.
4. **Security of pairing (medium risk, future).** The `/pair` feature, which would allow one engineer to send input to another's Claude Code session, requires careful permission design. This is scoped to a future phase, giving time for the security model to mature.
5. **Adoption chicken-and-egg (high risk).** Biff is only useful if your teammates also use it. Mitigation: the `.biff` file committed to the repo means the second person on a team only needs to clone, run `uv tool install -editable .` and `biff install` — the configuration is already there. The demo relay eliminates infrastructure setup. The `/plan`, `/finger`, and `/who` commands provide value to individual users even without teammates.

Q: What dependencies exist on other teams or systems?

Biff depends on the MCP protocol (maintained by Anthropic), Claude Code as the primary host client (also Anthropic), and NATS for the message relay. There are no dependencies on other punt-labs products. The relay server is a single NATS binary and can be self-hosted.

Q: Where does data live?

All data lives on the NATS relay. The relay serves three durability tiers:

Persistent: `/plan` status is stored on the relay. Like the original BSD `~/.plan` file, it survives sessions, disconnects, and restarts. `/finger` reads the persistent plan and combines it with live presence.

Store-and-forward: `/write` follows a POP mail model — messages are held on the relay until the recipient runs `/read`, then discarded.

`/who` and `/mesg y/n` are live presence state. `/wall` (team broadcast) is on the roadmap.

Today, messages traverse the relay in cleartext. The `.biff` file in the repository contains the team member list and optional relay configuration. It is the root of trust: adding your username to `.biff` and committing it proves you have commit access to the repo, which is what makes you a team member. End-to-end encryption ships with biff 1.0 and will make the relay unable to read message contents.

Q: What is the estimated development timeline?

Shipped Core MCP server with `/write`, `/read`, `/who`, `/plan`, `/finger`, `/tty`, `/mesg y/n`, and `/last`. NATS-based relay for cross-machine messaging on Synadia Cloud. POP-model store-and-forward for `/write`. TTY sessions (each agent gets a distinct identity, targetable via `/write @user:tty`). Enriched presence (`/who` shows 7 columns: NAME, TTY, IDLE, S, HOST, DIR, PLAN; `/finger` shows host and directory per session). Session history (`/last` shows login/logout records modeled after Unix `last(1)`, with three-layer logout detection). Per-session status bar showing `user:tty(N)` with three states: caught up, unreads (bold yellow), and messages off (`n`). `/mesg n` suppresses the unread count on the status line. Display names from GitHub identity. Released and in use.

Next: Team Coordination `/wall` (team broadcast). Plan auto-expand (`/plan biff-bf8` shows the task title). Workflow hooks (claiming a task auto-sets plan, creating a PR triggers `/wall`). Project opt-in (`/biff y` enables coordination per project).

Future: Security End-to-end encryption (NaCl/libsodium), GitHub identity and auth, per-repo NATS credentials.

Future: Real-Time `/talk` for live conversation, `/pair` for session sharing with explicit consent.

Future: Hosted Managed relay service (biff teams). Admin controls, audit logs, retention policies. Billing integration.

Q: What is the agentic coordination vision?

Biff serves the human+agent collaboration layer, not the pure agent-to-agent coordination layer. These are different problems with different design constraints.

Pure agent coordination — shared memory, high-throughput task dispatch, swarm orchestration — is well served by dedicated tools like claude-flow and other hive-mind frameworks. Those tools optimize for machine-speed coordination where no human is in the loop. Biff does not compete with them and is not trying to.

What biff provides is visibility and communication for hybrid teams: humans and agents working together on the same codebase. Because biff is repo-scoped, you see only the humans and agents working on *this* project — not every team across your organization. Today, TTY sessions (`/tty`) give each agent a distinct identity — one user running three agents shows three entries in `/who`, each targetable via `/write @user:tty`. Enriched presence shows host and directory per session so you can see where each agent is working. An agent can `/plan` what it is doing, `/write` a human when it needs a decision, and show up in `/who` alongside everyone else. `/last` provides an audit trail of all session activity — who logged in, when, and for how long — mirroring Unix `last(1)`. The next phase adds plan auto-expand (`/plan biff-bf8`) to surface what each agent is working on without anyone looking it up, and workflow hooks to automate coordination overhead: claiming a task auto-sets your plan, creating a PR triggers a `/wall` announcement.

The design choice is intentional. Biff's Unix communication vocabulary — `write`, `finger`, `who`, `wall`, `mesg` — is human-scale and purposeful by design. Pure agent swarms need shared memory and high-throughput coordination. Hybrid teams need human-scale communication with intent: the ability to see what your agents are doing, reach any of them directly, and have them reach you when it matters.

Q: What is the scaling story?

The relay is a NATS JetStream instance — a single Go binary that handles 100,000+ messages per second on one node. Horizontal scaling is straightforward: NATS supports clustering with Raft-based replication. The bottleneck at scale is presence: broadcasting `/who` status across thousands of users requires a gossip protocol. This is a solved problem and is not needed until biff reaches thousands of concurrent users. For the POP-model message relay, NATS JetStream consumers with acknowledge-and-delete provide exactly the retrieve-and-discard semantics biff requires.

Business

Q: What is the revenue model?

Open core. The biff CLI tool, self-hosted relay, and shared demo relay are free and open source (MIT license). Revenue comes from **biff teams**, a hosted relay service priced at $10/user/month for teams above the free tier:

- Free tier: up to 5 team members on the shared demo relay
- Paid tier: unlimited team members with 99.9% uptime SLA
- Audit logs and compliance exports
- Team admin controls (permissions, onboarding, offboarding)
- Priority support

The paid tier is about team size and operational guarantees. This model is proven by GitLab, Sentry, PostHog, and other developer-tools companies. Engineers adopt the free tool; their companies pay for the managed version when the team grows and needs reliability guarantees.

Q: What are the key metrics for success?

Weekly active users Target: 1,000 WAU within 6 months of launch. Measures adoption.

Messages per user/week Target: 15+ messages/week for active users. Measures engagement without rewarding noise — biff's intentional model should show *fewer* messages per user than Slack but higher signal-to-noise.

Team conversion rate Target: 10% of teams with 3+ active free users convert to biff teams within 90 days.

Retention (D30) Target: 40% of installers still active after 30 days. CLI tools typically see high install-and-forget rates; 40% retention indicates genuine workflow integration.

Decision threshold: if WAU has not reached 500 within 4 months of public launch, revisit the go-to-market strategy. If D30 retention is below 20%, the product is not sticky enough and needs fundamental rethinking.

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

Ship to 10 external beta teams within 60 days. Each team must have at least 3 members using biff as their primary coordination tool for a real project. Measure D30 retention and messages-per-user-per-week. Conduct post-trial interviews to understand what worked, what was missing, and whether users would pay for the hosted tier.

If D30 retention is below 20% across beta teams, the product is not solving a real problem and needs fundamental rethinking. If retention is above 40% but conversion interest is zero, the free tier may be too generous or the paid tier insufficiently differentiated.

Q: What does the P&L look like at steady state?

Assumptions (labeled): 10,000 paid users at $10/user/month = $1.2M ARR. Infrastructure costs are approximately $2,000/month (NATS relay hosting on Synadia Cloud, monitoring, CI/CD). Support costs are minimal at this scale given the CLI-native user base and open-source community. Gross margin exceeds 95%.

Break-even requires approximately 200 paid users ($2,000/month revenue vs. $2,000/month infrastructure). The path to break-even is short because infrastructure costs are near-constant up to tens of thousands of users — NATS is designed for high-throughput messaging with minimal per-message overhead. The primary cost at scale is engineering headcount, not infrastructure.

Q: Why now? What has changed?

Three things changed simultaneously, and the convergence is visible in real dollars and real behavior:

1. **Engineers returned to the terminal.** After two decades of IDE dominance, AI coding tools brought engineers back to the command line. Claude Code, Cursor's terminal mode, and the MCP ecosystem mean the terminal is once again the primary workspace for a growing population of engineers. Dohmke calls this "the new center of gravity" and bet $60M on it. Yegge profiles engineering teams achieving 10–100x productivity gains by working entirely inside terminal sessions with AI agents.
2. **MCP created a standard protocol.** Before MCP, building a communication tool inside a coding environment meant integrating with every editor separately. MCP provides a single integration point that works across any compatible client. Biff integrates once and works everywhere MCP does.
3. **AI-accelerated engineers outgrew Slack.** When engineers ship 3–5x faster, the communication overhead that was tolerable at the old pace becomes the bottleneck. Slack's interruption model was designed for a world where shipping a feature took a sprint. In a world where shipping a feature takes a morning, Slack's model actively harms productivity.

Q: What are we not building?

Biff is not building:

- **Persistent channels or temporary groups.** No channels, no rooms, no `/hive`. Communication is directed: you message a person or broadcast to the team. There is no concept of a persistent or temporary room.
- **Artifact sharing (`/send`).** Sending diffs, files, or code snippets inline is a compelling idea but out of scope for now. Use git, GitHub Issues, or PR comments for artifact exchange.
- **Inline code review (`/cr`).** Requesting and conducting code review inside the terminal is future work. GitHub's review workflow remains the system of record.
- **Message history or search.** Messages are ephemeral by default. Biff is not a system of record.
- **Video or voice.** Biff is text. If you need a video call, use the tool built for video calls.
- **Project management.** No tasks, no boards, no sprints. Biff is communication, not coordination. Use beads, Linear, or GitHub Issues for project tracking.
- **A mobile app.** Biff is for engineers at their terminals. If you are on your phone, you are not in the target workflow.

Q: What are the most likely ways biff fails?

Five scenarios, in descending order of likelihood:

1. **The anti-Slack niche is too small.** Engineers who hate Slack still use it because their PM is on it. Biff's repo-scoped, engineer-only model may not generate enough standalone value to overcome Slack's network effects. Kill signal: D30 retention below 20% across beta teams.

2. **Claude Code ships native messaging.** Anthropic adds `/write` and `/who` to Claude Code directly, making biff redundant. Mitigation: biff is cross-client (any MCP host), cross-machine (NATS relay), and repo-scoped — capabilities an embedded feature would not prioritize. But if Anthropic ships 80% of biff's value natively, the remaining 20% may not justify a separate install.
3. **Time-to-critical-mass is too long.** Each team needs at least 2–3 members before biff provides value. If onboarding friction or the "install biff, then convince a colleague" loop takes weeks instead of days, teams abandon before reaching critical mass.
4. **The relay becomes a liability.** A security incident or prolonged outage before the team is large enough to maintain the infrastructure erodes trust permanently. Today's cleartext-over-TLS model is honest but fragile — a single relay compromise exposes every message.
5. **Agents outgrow the communication model.** If engineering teams shift to fully autonomous agent fleets faster than expected, the hybrid human+agent model becomes a niche within a niche. Pure agent coordination tools (A2A, CrewAI, LangGraph) would serve the dominant use case, and biff's human-centric design becomes a liability rather than a differentiator.

Risk Assessment

Four Risks Assessment

Value

Medium risk. Strong circumstantial evidence (CLI adoption trend, MCP ecosystem growth, Yegge's observations about AI-speed collaboration) and early artisanal validation — the builders use biff daily for their own coordination. The risk is not "do engineers dislike Slack?" (they do) but "is the pain severe enough to adopt a new tool?" Slack's network effects are powerful — even engineers who hate Slack use it because their team does. Biff must provide enough standalone value (`/plan`, `/finger`) to bootstrap adoption before network effects kick in. The hybrid positioning — designed for humans who work alongside agents, not for pure agent coordination — is a strength: it keeps biff focused on its core audience (human engineers) while pure-agent tools serve a different need. The risk is that the hybrid framing is too narrow if the market shifts toward fully autonomous teams faster than expected, or too broad if engineers do not yet run enough agents for the hybrid story to resonate. The artisanal signal is positive: the tool is in daily use by its creators in a hybrid human+agent workflow. The remaining risk is whether that signal generalizes beyond the builders.

Usability

Low risk. The slash-command interface mirrors patterns engineers already use (git commands, Unix utilities, Claude Code slash commands). Installation is four commands (`git clone`, `uv tool install -editable .`, `biff install`, `biff doctor`). Time-to-value is under five minutes. The primary usability risk is discoverability: new users need to learn the command vocabulary. Mitigation: the vocabulary is small (8 shipped commands), the names are intuitive (borrowed from Unix), and the tool descriptions are self-documenting within Claude Code. The secondary usability risk is the empty-room experience: a solo installer sees no teammates and may not persist long enough for network effects to form. Mitigation: the `.biff` file viral loop means most users arrive on a team that already has at least one other member.

Feasibility

Low risk (shipped). The core architecture is not just validated — it is shipped and in production. The MCP notification mechanism, NATS JetStream relay (tested against hosted NATS on Synadia Cloud), dynamic tool descriptions, model-visible message awareness, and status bar integration are all working. The notification gap — the highest-uncertainty component at design time — is addressed through two channels: model-level awareness via tool descriptions and user-level awareness via the status bar. The highest-risk future component is `/pair`, which requires careful permission design for session input sharing. End-to-end encryption (NaCl/libsodium) ships with biff 1.0. No shipped component required novel technology.

Viability

Medium risk. The open-core model is proven in developer tools, but biff's intentional low-volume communication model means engagement metrics will look different from typical SaaS. Investors and boards accustomed to Slack-style engagement numbers may question whether "15 messages per user per week" constitutes a healthy product. The counter: biff's value is measured in time saved, not messages sent. The free-to-paid conversion trigger is team size (free for up to 5, paid beyond that), which aligns with natural adoption: one engineer installs, teammates follow, and the team hits the threshold organically. The risk is reaching that critical mass of team adoption quickly enough to demonstrate the conversion funnel.

Appendix A: Command Reference

Commands are organized by status: **Shipped** commands are released and working today. **Next** features are the immediate roadmap focus. **Future** features are planned but not yet scheduled. **Won't Do (for now)** features are out of scope.

Shipped: Core Communication

Command	**Origin**	**Description**
`/write @user "text"`	BSD `write`	Send a one-way message to a user. Asynchronous — the recipient reads it when they choose via `/read`.
`/read`	BSD `from`	Check your inbox. Shows unread messages with sender, timestamp, and content.
`/finger @user`	BSD `finger`	Read a user's plan, current status, host, and directory. Non-intrusive — the target is not notified.
`/who`	BSD `who`	List all active sessions with 7 columns: NAME, TTY, IDLE, S (message status), HOST, DIR, and PLAN. Shows who is online, where they are working, and what they are doing.
`/plan "text"`	BSD `.plan`	Set your status message. Visible to anyone who runs `/finger` or `/who`. Update it as your work changes. Provides single-user value without teammates.
`/tty "name"`	BSD `tty`	Name the current session. Each agent gets a distinct identity — one user with 3 sessions shows 3 entries in `/who`, targetable via `/write @user:tty`.
`/mesg y | n`	BSD `mesg`	Control whether you receive messages. When off (`n`), incoming messages are queued until you turn it back on (`y`), and the status bar shows `(n)` instead of the unread count.
`/last`	BSD `last`	Show session login/logout history. Displays who logged in, from where, when, and for how long. Active sessions are marked "still logged in". Three-layer logout detection: sentinel-based (graceful shutdown), orphan detection (crash recovery), and KV watcher (TTL expiry).

Next: Team Coordination

Feature	Description
`/wall "text"` (team broadcast)	Broadcast a time-sensitive announcement to all terminals. Visible immediately on the status bar with duration-based expiry. Does not go to inboxes — it is a banner, not a message.
Plan auto-expand	`/plan biff-bf8` auto-expands to show the task title. Everyone sees what you are working on without looking it up.
Workflow hooks	Claiming a task auto-sets your plan. Creating a PR triggers a `/wall` announcement.
Project opt-in	`/biff y` enables the coordination workflow per project via `AGENTS.md`.

Future: Network & Security

Feature	Description
E2E encryption	NaCl/libsodium (Curve25519 + XSalsa20-Poly1305). Messages encrypted with recipient's public key; relay stores ciphertext it cannot read.
GitHub identity & auth	Authenticate users via GitHub identity. Per-repo NATS credentials for team isolation.

Future: Real-Time & Pairing

Command	Origin	Description
`/talk @user`	BSD `talk`	Open a real-time bidirectional conversation via model-mediated message exchange.
`/pair @user`	—	Invite a user to send input to your Claude Code session. Requires explicit acceptance. The session owner sees all input and retains full control.

Won’t Do (for now)

Feature	Rationale
`/send @user` (artifact sharing)	Sending diffs, files, or code snippets inline. Compelling but out of scope — use git or GitHub for artifact exchange.
`/cr @user` (code review)	Inline code review in the terminal. GitHub’s review workflow remains the system of record.
`/hive @a @b @c` (temporary groups)	Temporary multi-person groups that dissolve when work is done. Directed messaging (`/write`) and broadcast (`/wall`, on roadmap) cover the coordination need without the complexity of group lifecycle management.