biff

Team communication for engineers who never leave the terminal

February 2026 | Jim Freeman | punt-labs

Press Release

Summary

San Francisco, CA — Today, punt-labs announced **biff**, an open-source communication tool that lets software engineers message teammates, share context, and coordinate work without leaving their 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 `/mesg @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.

The cost is not hypothetical. Research consistently shows that a single context switch costs 15–25 minutes of deep work recovery. 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 in February 2026: "The terminal is becoming the new center of gravity, as developers prompt fleets of agents across multiple terminal windows at once." 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

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

```
/plan "refactoring the auth layer"
/mesg @kai "auth module is ready for review"
```

```
/cr @kai
/hive @kai @eric @jim
/wall "I'm pulling the auth changes into the feature branch"
```
...thirty minutes of focused work later...
```
/hive off
```

Every command implies intent. There are no channels to monitor, no threads to catch up on, no emoji reactions to parse. Communication is pull-based: 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 unread count. You decide when to engage. When you need deeper collaboration, `/pair @kai` invites a teammate to send input to your Claude Code session — and they can only join when you explicitly consent. When a problem needs three people for thirty minutes, `/hive` creates a temporary group that dissolves when the work is done. The full command reference, organized by priority tier, is in Appendix A.

Because biff speaks MCP, it does not distinguish between a human session and an agent session. An autonomous coding agent can join a `/hive`, announce progress via `/wall`, or `/mesg` a human when it needs a decision. Agents can coordinate with other agents the same way. Biff is not just communication for engineers — it is the communication layer for the entire hive of humans and agents building software together.

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 `/biff off` when I'm deep in a problem and `/biff on` when I come up for air. My teammates can `/finger @priya` to see what I'm on without interrupting. 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 one command: `pip install biff-mcp`. Biff registers itself as an MCP server in your Claude Code session. Team configuration lives in a `.biff` file committed to your git repository — it contains the relay URL and whitelisted member IDs. 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 — `talk`, `wall`, `finger`, `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-mcp`. The core tool is free and always will be. For teams that want hosted relay, presence guarantees, and audit logs, biff teams will be available in Q3 2026.

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 eleven slash commands covering messaging, presence, collaboration, and code review — 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.

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-mcp`. Biff auto-registers as an MCP server in your Claude Code session. If your repo has a `.biff` file (committed by a teammate), biff picks up the relay URL and team roster automatically. 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?

Messages are end-to-end encrypted in transit. In the open-source self-hosted mode, messages route through your own infrastructure and nothing is stored by punt-labs. In the hosted relay mode (biff teams), messages are encrypted at rest and subject to a retention policy you configure. Biff never reads, 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: What does "pairing" mean? Can someone control my session?

When a teammate invokes `/pair @you`, they are requesting to send input to your Claude Code session — not the other way around. You see the request and explicitly accept or decline. If you accept, the teammate can type prompts that are sent to your Claude, but you retain full control and can see everything they send. You can end the pairing at any time. No one can read your session, execute commands, or modify your files without your explicit, per-session consent.

This is different from `/talk`, which is a simple real-time text conversation between two terminals — no session access involved.

Q: Can agents use biff?

Yes. Because biff is an MCP server, any MCP-compatible agent can use it the same way a human does. An autonomous coding agent can join a `/hive`, broadcast status via `/wall`, send artifacts via `/send`, or message a human when it needs a decision. Agents can also communicate with other agents — coordinating work, handing off tasks, or requesting reviews. Biff does not distinguish between human and agent sessions. If it speaks MCP, it can participate.

Internal FAQs

Value & Market

Q: What is the total addressable market?

The immediate market is engineers using AI coding assistants inside terminals. GitHub reports over 100 million developers on the platform. AI-assisted coding adoption is growing rapidly — GitHub Copilot alone surpassed 1.8 million paid subscribers by late 2024, and Claude Code launched to strong adoption in early 2025. 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 autonomous agents become standard members of engineering teams, the number of sessions that need coordination grows faster than headcount.

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"[1] describes Anthropic's internal engineering culture where teams "swarm around campfires" and "the whole team is pair programming at once" — all happening inside Claude Code sessions.

[1] https://steve-yegge.medium.com/the-anthropic-hive-mind-d01f768f3d7b

He reports engineers achieving 10–100x productivity gains, but notes that "you need full transparency at all times, at their speeds, or 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.[2] His thesis: "the entire software ecosystem is being bottlenecked by a manual system of production" and "the terminal is becoming the new center of gravity." 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: 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.

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. 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, validated).** A spike validated that Claude Code acts on `notifications/tools/list_changed` sent over SSE: when the biff server updates a tool description (e.g., "Check messages (1 unread: @kai about auth)"), the model sees the change and can proactively mention it to the user. The full chain — message arrival, tool re-registration, SSE notification, model awareness — has been tested end-to-end. The remaining MCP spec maturity risk is mitigated by wrapping all MCP interactions behind an internal abstraction layer.

[2] https://entire.io/blog/hello-entire-world/

2. **User-visible notification gap (medium risk).** Dynamic tool descriptions are visible to the AI model but not directly to the human user in the Claude Code UI. There is no badge, popup, or inline notification for the user. Mitigation: the terminal status bar is a configurable shell command that can read local files. Biff will write unread state to a file, and the status bar script displays a live count. This is a proven pattern but requires per-terminal setup.
3. **Message relay infrastructure (low risk).** Routing messages between engineers on different machines requires a relay server. This is well-understood infrastructure (WebSocket/SSE relay). The self-hosted option reduces dependency on punt-labs infrastructure.
4. **Security of pairing (medium risk).** The `/pair` feature, which allows one engineer to send input to another's Claude Code session, requires careful permission design. Mitigation: explicit per-session consent, all input visible to the session owner, all interactions logged, session owner retains kill switch. This is scoped to Phase 3, 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 `pip install biff-mcp` — the configuration is already there. The `/plan`, `/finger`, and `/who` commands provide value to individual users even without teammates. The `/send` and `/cr` commands provide enough value in pairs to bootstrap adoption within a team.

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 Python packaging infrastructure (PyPI). There are no dependencies on other punt-labs products. The relay server is self-contained and can be self-hosted.

Q: What is the estimated development timeline?

A technical spike has validated the core architecture: HTTP transport with SSE for server-push notifications, dynamic tool descriptions via `notifications/tools/list_changed`, and end-to-end message awareness in Claude Code. The architecture is proven; what remains is implementation.

Phase 1 (4 weeks) Core MCP server with `/mesg`, `/who`, `/plan`, `/finger`, `/biff on/off`. Local relay for same-machine communication. Single-user value from `/plan` and session awareness. Architecture validated by spike.

Phase 2 (4 weeks) Network relay for cross-machine messaging. `/wall`, `/hive`, `/send`, `/cr`. Team communication functional end-to-end.

Phase 3 (4 weeks) `/talk` real-time conversation, `/pair` with permission model. End-to-end encryption. Security audit. Note: the spike found that direct subprocess handoff is not viable in MCP; `/talk` will use model-mediated message exchange.

Phase 4 (4 weeks) Hosted relay service (biff teams). Admin controls, audit logs, retention policies. Billing integration.

Total: approximately 16 weeks to full product with hosted offering. Phase 1 delivers usable value and can be released as a public beta.

Q: What is the scaling story?

The relay server is a stateless WebSocket router. Horizontal scaling is straightforward — add more relay nodes behind a load balancer. The bottleneck at scale is presence: broadcasting `/who` status across thousands of users requires a gossip protocol or presence service. This is a solved problem (XMPP, Matrix) and is not needed until biff reaches thousands of concurrent users.

Business

Q: What is the revenue model?

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

- Managed relay with 99.9% uptime SLA
- End-to-end encryption with key management
- Audit logs and compliance exports
- Team admin controls (permissions, onboarding, offboarding)
- Priority support

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 exceeds 5–10 people 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: 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 describes Anthropic engineers who "never leave their terminal sessions."

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.** Biff has `/hive` for temporary groups that dissolve when the work is done, not channels that accumulate forever. There is no concept of a persistent room.
- **Message history or search.** Messages are ephemeral by default. If you need a record, use `/send` to send an artifact to a durable store (git, GitHub Issues). 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.

Risk Assessment

Four Risks Assessment

Value

Medium risk. Strong circumstantial evidence (CLI adoption trend, MCP ecosystem growth, Yegge's observations about AI-speed collaboration) but no direct customer validation yet. 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 artisanal approach — building for ourselves first — provides an honest signal: if we do not use it daily, no one else will either.

Usability

Low risk. The slash-command interface mirrors patterns engineers already use (git commands, Unix utilities, Claude Code slash commands). Installation is a single `pip install`. 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 (11 commands), the names are intuitive (borrowed from Unix), and `/help` documents everything inline.

Feasibility

Low risk (validated). A technical spike has confirmed the core architecture end-to-end: HTTP transport with SSE, dynamic tool descriptions via `notifications/tools/list_changed`, and model-visible message awareness in Claude Code. The MCP notification mechanism — the highest-uncertainty component — works as designed. The remaining technology (WebSocket relay, message routing, file-based storage) is well-understood. The highest-risk component is the `/pair` feature, which requires careful permission design for session input sharing. This is scoped to Phase 3, giving time for the security model to mature. No component requires 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 hosted relay's value proposition (uptime, encryption, compliance) is clear for teams above 5–10 people. 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 priority tier using the MoSCoW framework. **Must Have** commands ship in Phase 1 and deliver standalone value. **Should Have** commands ship in Phase 2 and enable team workflows. **Could Have** commands ship in Phase 3 and require additional architecture work identified during technical spikes.

Must Have — Phase 1: Core Communication

Command	Origin	Description
`/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.
`/who`	BSD `who`	List all active sessions. Shows who is online, their availability (`/biff on/off`), and their current plan.
`/finger @user`	BSD `finger`	Read a user's plan and current status. Non-intrusive — the target is not notified.
`/mesg @user "text"`	BSD `mesg`	Send a one-way message to a user. Asynchronous — the recipient reads it when they choose. Core validated by technical spike.
`/biff on | off`	BSD `biff`	Control whether you receive messages. When off, incoming messages are queued until you turn it back on.

Should Have — Phase 2: Team Communication

Command	Origin	Description
`/wall "text"`	BSD `wall`	Broadcast a message. If you are in a `/hive`, the message goes to hive members. Otherwise it goes to your full team.
`/send @user`	—	Send a diff, file, or code snippet to a user with full context. The recipient receives the artifact inline in their session.
`/cr @user`	—	Request a code review. Automatically attaches the relevant changes from your current branch. The reviewer receives the diff and can respond with comments.
`/hive @a @b @c`	—	Create a temporary group with the named users. All members see each other's messages. Use `/wall` to broadcast within the hive. `/hive off` dissolves the group.

Could Have — Phase 3: Real-Time & Pairing

Command	Origin	Description
`/talk @user`	BSD `talk`	Open a real-time bidirectional conversation. The technical spike found that direct subprocess handoff is not viable in MCP; this command will use model-mediated message exchange instead.
`/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. Highest security design complexity.