OFFICIAL REFERENCE

SYNLYNK

The OS for Multi-Agent Development

v0.12.0 · Measurement & Reliability

```
$ curl -fsSL synlynk.com/install.sh | sh

✓ synlynk 0.12.0 installed to ~/.synlynk/bin
✓ 4 agents discovered (claude · agy · codex · grok)
✓ SQLite canon · relay broker · daemon ready

→ synlynk init && synlynk exec claude
```

Python 3.8+ | stdlib only | 472 tests | MIT

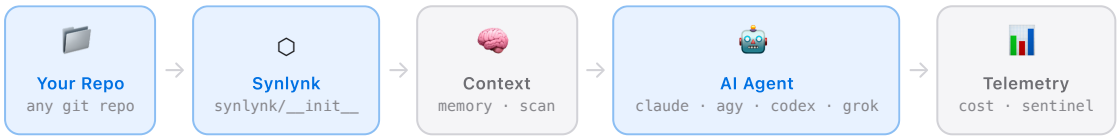
synlynk.com · github.com/nikhilsoman/synlynk · 2026

WHAT IS SYNLYNK?

Your AI agents, working as a team.

Synlynk is a single-package Python CLI that wraps AI agents — Claude, AGY (AntiGravity), Codex — and makes them **project-aware, cost-tracked, and safe to run in parallel**. It injects your project context before every invocation, guards your budget, detects hallucination loops, dispatches background tasks across multiple agents simultaneously, runs as an always-on daemon, and connects your workgroup over a real-time relay.

How it works



Why Synlynk?

Context-Aware by Default

Roadmap, memory, todos, and source architecture injected before every session — no copy-pasting.

Zero Surprise Bills

Budget guard blocks execution the moment you'd hit your limit. Real-time cost tracking every exec.

Hallucination Sentinel

Detects flatline loops, success loops, quota exhaustion, and skipped verification automatically.

Parallel Dispatch

Launch Claude, AGY, and Codex in parallel on different tasks. Monitor all jobs from one CLI.

Team-Ready from Day One

Per-user devlogs with git attribution, one-command onboarding, and a real-time relay channel.

Zero Dependencies

Pure Python 3 stdlib. No pip, no Docker, no build step. Works anywhere Python 3.8+ runs.

25+	472	0	∞
COMMANDS	TESTS PASSING	PIP DEPENDENCIES	PARALLEL AGENT JOBS

GETTING STARTED

Install in 60 seconds.

1 **Clone & install globally**

Installs the package to `~/.synlynk/lib/` and adds `synlynk` to your PATH. No sudo, no pip, no build.

install

```
$ git clone https://github.com/nikhilsoman/synlynk && cd synlynk && ./install.sh
✓ Installed synlynk/ package to ~/.synlynk/lib/
✓ Installed binary to ~/.synlynk/bin/synlynk · PATH updated in ~/.zshrc
✓ synlynk 0.12.0 ready
```

2 **Bootstrap your project — with intelligent doc migration**

In any repo: synlynk finds existing docs (roadmap, memory, README) and migrates them — no blank skeletons.

synlynk init

```
~/my-project $ synlynk init
Scanning repository for existing docs...
✓ project-docs/roadmap.md (migrated from roadmap.md)
✓ project-docs/memory.md (migrated from rxcc_memory.md)
✓ project-docs/todo.md (generated from SQLite stories)
✓ CLAUDE.md · GEMINI.md · AGENTS.md
✓ claude → /usr/local/bin/claude · agy → /usr/local/bin/agy
→ Ready. Try: synlynk exec claude
```

⚡ **SMART DOC MIGRATION**

Synlynk searches at repo root, `project-docs/`, and project-prefixed variants (`rxcc_memory.md`). First file >200 bytes wins and is migrated verbatim. Git history seeds a roadmap as a last resort only.

3 **Run your first context-injected session**

Your project context — roadmap, memory, todos, and source scan — is prepended before the agent launches.

synlynk exec

```
$ synlynk exec claude
📋 Context injected (memory · roadmap · todos · source scan) · 💰 $2.34 / $10.00
claude › Based on your roadmap and source scan, the next step is...
```

🏆 **ACHIEVEMENT UNLOCKED**

Context Master

Your AI agent now knows your entire project — zero onboarding time on every session.

V0.4.0+ · BACKGROUND DISPATCH

Run agents in parallel. In the background.

Dispatch fires an agent as a background job — non-interactive, captured to a log, tracked by the job store. Multiple agents run simultaneously on different tasks. Your terminal stays free.

```
synlynk dispatch

$ synlynk dispatch claude --task "Write unit tests for the billing module"
✓ Dispatched job-a3f2 (claude) running in background
$ synlynk dispatch agy --task "Review billing API for edge cases"
✓ Dispatched job-b7e1 (agy) running in background
```

Monitor all jobs (--watch)

```
jobs

$ synlynk jobs --watch
● job-a3f2 claude running 2m14s
● job-b7e1 agy running 1m58s
```

Tail a job's output

```
logs

$ synlynk logs --job job-a3f2
Writing test_billing.py...
✓ test_charge · ✓ test_refund
→ 12/12 tests passing
```

Scoped Story Dispatch v0.9.0

Link a dispatch to a tracked story for a minimal context slice — story metadata, active tasks, up to 20 domain-filtered source files. 60–80% less token overhead. Control it per-run with `--context-mode none|task|full`.

```
scoped dispatch

$ synlynk dispatch codex --story story-abc123 --context-mode task --task "Fix token extraction"
Context: story metadata + active tasks + 8 domain-matched files (~420 bytes)
Agent framing: ## Task Criteria bullets (Codex-optimised)
✓ Pre-flight gate passed · Dispatched job-c9d4 (codex)
```

🚀 ACHIEVEMENT UNLOCKED

Workgroup Commander

Claude, AGY, and Codex work in parallel — each receiving exactly the context it needs, no more.

💡 **AGENT-SPECIFIC FRAMING**

Synlynk formats prompts per agent: Codex gets `## Task Criteria` bullets; AGY gets a concise `Task:` prefix with 2000-char context; Claude receives the full narrative. Same story — three optimised messages.

SENTINEL & BUDGET GUARD

Your safety net, always on.

Synlynk watches every agent invocation for danger signals — spiralling costs, hallucination loops, quota exhaustion, skipped verification — and fires alerts before they become expensive mistakes.

💰 Budget Guard

Set limits in `.synlynk/config.json`. Synlynk blocks execution the moment you'd exceed them.

config.json

```
{
"limit_usd": 10.00,
"limit_requests": 100,
"project_docs_dir": "project-docs"
}
```

budget pulse

```
── Budget Pulse ──────────────
Cost: $2.34 / $10.00
Requests: 14 / 100
Tokens: 2,840 in · 480 out
```

Sentinel Patterns

Sentinel watches your telemetry log and fires alerts for dangerous patterns automatically.

FLATLINE
Same command failed 3× in a row. Possible hallucination loop.

⚠ SUCCESS_LOOP
Same command succeeded 5× in <10 min. Likely an automated loop burning tokens.

QUOTA_EXHAUSTED
Output matched a known quota-exhaustion phrase. Switch agent or check limits.

⚠ VERIFY_SKIP V0.9.4
Job exited 0 but no test evidence found in the output — work may be unverified.

Managing sentinel alerts

sentinel commands

```
$ synlynk sentinel list
[CRITICAL] FLATLINE `synlynk exec claude` failed 3x — 2026-06-22 09:41
[WARN] VERIFY_SKIP job-c9d4 exited 0, no tests — 2026-06-24 10:02
$ synlynk sentinel clear --code FLATLINE
✓ Cleared FLATLINE alerts
```

ACHIEVEMENT UNLOCKED

Incident Avoided

Sentinel detected a FLATLINE loop before you burned $40 in repeated failed calls.

TEAM MODE

Every teammate. Every agent. In sync.

Team mode adds per-user devlogs, a shared roadmap visible to all agents, one-command onboarding via `synlynk join`, and a pull-before-write guard that prevents doc conflicts in concurrent sessions.

enable team mode

```
$ synlynk init --mode team --org acme --repo api
✓ Team mode enabled · project-docs/devlogs/nikhil.md created
✓ Pull-before-write guard active
```

How teams work with Synlynk

Per-User Devlogs

Each member writes to `devlogs/username.md`. Agents read recent entries from all teammates so every AI session knows what the whole team has been doing.

Pull-Before-Write

Before writing to any shared doc, synlynk auto-pulls from origin. Prevents doc divergence in concurrent sessions and git worktrees.

Attribution in Memory

Decisions in `memory.md` carry `[@username]` markers. Every agent sees who made each decision — accountability without meetings.

Team Status Dashboard

`synlynk team status` shows each teammate's last active session, recent decisions, and open sentinel alerts.

One-command onboarding v0.9.2

A new teammate clones the repo and runs `synlynk join`. Synlynk seeds their devlog from git history, regenerates the agent files, and prints a digest of recent team activity.

1. **Start every session with a pull**
   `git pull` before `synlynk exec` — the guard will remind you if you forget.
2. **Write decisions to memory with attribution**
   Add `[@yourname 2026-06-24]` to any decision line — agents surface these to teammates automatically.
3. **Run consensus decisions**
   `synlynk decide --topic "..."` polls all agents and records the consensus to memory with attribution.

MONOREPOS

One repo. One state database.

Synlynk is monorepo-native. All worktrees for the same git root share a single state database. The scanner skips `node_modules`, `dist`, `build`, `vendor`, `.next`, `target`, and `__pycache__` automatically. All you decide is where to put your docs.

Step 1 — Where do your docs live?

Option A: project-docs/ (default)

Docs in a dedicated folder at the monorepo root. Best for repos that didn't have docs before synlynk.

default

```
~/monorepo $ synlynk init
✓ project-docs/roadmap.md
✓ project-docs/memory.md
```

Option B: root docs (--docs-dir .)

Repo already has `roadmap.md`, `memory.md` at the root? Point synlynk at them.

root docs

```
~/monorepo $ synlynk init --docs-dir .
✓ roadmap.md (migrated)
✓ memory.md (migrated)
```

Step 2 — Verify the shared state DB

synlynk status

```
$ synlynk status
State DB: ~/.synlynk/projects/a1b2c3d4/state.db · Worktrees sharing DB: 3
Git root: /Users/nikhil/monorepo · Docs dir: project-docs/
```

Step 3 — Run an initial source scan

synlynk scan --deep

```
$ synlynk scan --deep
Scanning 3,420 files (skipping node_modules, dist, build, .next, vendor...)
✓ 847 symbols across 9 languages · Wrote project-docs/source-map.md
✓ Cache stored at HEAD sha a1b2c3d — zero cost on next exec
```

MONOREPO CHECKLIST

1. One `synlynk init` at the git root — not per-package. 2. Use `--docs-dir .` if docs already exist at root; default otherwise. 3. Run `synlynk scan --deep` once after init. 4. All worktrees share one state DB — capability scores apply repo-wide.

Scan exclusions — always skipped

```
node_modules/ · dist/ · build/ · .next/ · out/ · vendor/
target/ · __pycache__/ · .venv/ · venv/ · .git/ · .synlynk/
```

MULTI-REPO TEAMS

Multiple repos. One workflow.

Each repository gets its own state database keyed to its git root. Synlynk agents stay scoped to the repo they're running in. The pattern: run `synlynk init` once in each repo, independently.

init each repo independently

```
# Repo 1 — API service
~/dev/api $ synlynk init --mode team --org acme --repo api
✓ State DB: ~/.synlynk/projects/f1a2b3c4/state.db
# Repo 2 — Frontend app
~/dev/app $ synlynk init --mode team --org acme --repo app
✓ State DB: ~/.synlynk/projects/9e8d7c6b/state.db
```

Linking repos to a shared GitHub Project

Use `--project-id` to fill the same GitHub Projects v2 node ID into agent instruction files across all repos. Track stories and PRs from different repos in one unified board.

shared project board

```
~/dev/api $ synlynk init --org acme --repo api --project-id PJ_kwDO1234
✓ CLAUDE.md · GEMINI.md · AGENTS.md — project ID baked in
~/dev/app $ synlynk init --org acme --repo app --project-id PJ_kwDO1234
✓ Same project board, separate state DB
```

Multi-repo workflow tips

Separate Costs per Repo

Each repo has its own `costs.md` and budget limits. Set a tighter limit on the frontend and a generous limit on the core API.

Scoped Source Scans

Each scan is scoped to that repo's git root. Agents in the API repo only see API source files — no frontend noise.

Shared Standards

Keep a shared `AI_INSTRUCTIONS.md` template. Reference it in each repo's init so every agent gets the same standards.

Cross-Repo at v0.10

Workspace federation — dispatching across repos from one session — is planned for v0.10.0. Until then, dispatch per-repo.

ONE INIT PER GIT ROOT

Never run `synlynk init` inside a git submodule — it creates a separate state DB for the submodule root, not the parent repo. Always run from the repo's own root directory.

SOURCE SCAN & DOCS_DIR

Source-aware agents. Docs wherever you need.

🔍 Synlynk Scan v0.7.0

The static scanner indexes your codebase and injects a `## Source Architecture` section into every agent dispatch. Cache keyed by git HEAD SHA — zero overhead on cache hit. Supports Python, JS, TS, Go, Rust, Ruby, Java, Kotlin, Shell.

scan commands

```
# Quick — injects ## Source Architecture into context
$ synlynk scan
✓ Scanned 284 files · 1,024 symbols · cache hot (HEAD: a1b2c3d)
# Deep — full symbol extraction + source-map.md
$ synlynk scan --deep
✓ Wrote 284 entries to source_symbols (state.db) + project-docs/source-map.md
# Status
$ synlynk scan --status
Cache age: 4 min · HEAD sha: a1b2c3d · 284 files · 1,024 symbols
```

📁 Docs_Dir Customization v0.9.1

Use `--docs-dir` at init time to point synlynk at any directory. The setting is persisted to `.synlynk/config.json` and respected by all subsequent commands.

set docs dir

```
# Docs at repo root
$ synlynk init --docs-dir .
# Docs in custom folder
$ synlynk init --docs-dir docs/agent
```

config.json

```
{
"project_docs_dir": "docs/agent",
"limit_usd": 10.00
}
```

⚡ **WHAT SYNLYNK AUTO-DISCOVERS DURING INIT**

Before writing any file, synlynk searches: **1)** the target dir, **2)** repo root, **3)** `project-docs/`, **4)** project-prefixed names like `rxcc_memory.md`. First match >200 bytes is migrated verbatim — your work, not a blank template.

V0.12.0 · MEASUREMENT & RELIABILITY

Trustworthy fleet. Measured costs.

v0.12.0 makes the agent fleet trustworthy: dispatched jobs finish their own git steps instead of leaving commits unfinished, story routing becomes a real 3-stage scoring engine (capability, quota headroom, cost tie-break) with a fleet batch scheduler to clear backlogs unattended, and every cost figure synlynk reports is now either structurally measured or visibly flagged as an estimate.

✅ Dispatch git-finalization

When a job transitions from running to terminal, synlynk stages, commits, pushes, and opens a PR if needed — once, idempotently. Agents no longer have to remember the last mile.

📊 Cost provenance

Every `cost_entries` row carries `cost_source` (`actual`, estimate tiers, or `legacy_unknown`). Manual sessions use `synlynk cost log`. Estimates are visibly flagged in status and Vizor.

🧭 3-stage routing + schedule

`_best_agent_for_story()` scores capability, hard-gates on quota headroom, then cost-tie-breaks. `synlynk schedule [--execute] [--max-stories N]` batch-assigns ready stories with in-batch headroom accounting.

```
schedule
$ synlynk schedule --execute --max-stories 5
✓ 5 ready stories · scored · dispatched
```

⬡ **WHY IT MATTERS**

Finished git steps mean dispatch output is merge-ready, not abandoned worktrees. Real routing clears backlogs without babysitting. Provenance means budget numbers are accountable — not mystery arithmetic.

V0.9.4 · CONTEXT, DISPATCH & RELAY

SQLite canon. Profiles. A real-time relay.

v0.9.4 makes task state authoritative in SQLite, gives every agent its own context budget, adds a publish/subscribe relay, and closes the verification gap with a new sentinel.

🗄️ SQLite Canon

The stories table is now the single source of truth for task state. `todo.md` is a generated view, regenerated by `_generate_todo_md()` on every change; existing checklists are absorbed back via `_import_todo_to_stories()`. Story IDs are deterministic, so the same story keeps its ID across machines and worktrees.

sqlite canon

```
$ synlynk story create --title "Add rate limiter"
✓ story-7f3a written to stories table · todo.md regenerated (view)
# edits to todo.md are imported back into SQLite on the next command
```

🧩 Agent Profiles

Each agent can carry `.agents/<agent>.json` setting `context_mode` (none/task/full) and `context_max_bytes`. Configure with `synlynk agent configure <name>`.

.agents/codex.json

```
{
"context_mode": "none",
"context_max_bytes": 2000
}
```

⚠ VERIFY_SKIP Sentinel

Fires when a job exits 0 but no test evidence is found in its output — catching "looks done, never verified" work before it ships.

⚠ VERIFY_SKIP
job-c9d4 exited 0 · no test output detected · review before merge.

📊 synlynk jobs + 📡 Relay

`synlynk jobs` reads job state directly from SQLite, with `--watch` for a live view, and every dispatch passes a pre-flight gate first. The new `synlynk relay start` launches an HTTP SSE broker on port **27472**; `synlynk relay broadcast` fans a message out to every subscriber.

jobs + relay

```
$ synlynk jobs --watch # live, SQLite-backed
● job-a3f2 claude running 2m
$ synlynk relay start --port 27472
✓ SSE broker on http://localhost:27472/events
$ synlynk relay broadcast "Standup in 5" --kind motd
✓ Delivered to 3 subscribers
```

⬡ WHY IT MATTERS

SQLite canon kills hand-maintained todos. Profiles cut per-agent token waste. The relay turns a set of isolated CLIs into a connected workgroup — and the pre-flight gate stops bad dispatches before they cost anything.

RELEASE HISTORY

Every release, in plain language.

v0.12.0	**Dispatched jobs now finish their own git steps** — commit, push, and PR happen automatically once work is verified complete. Story routing gets real capability + quota + cost scoring, plus a fleet batch scheduler. Every cost number is now measured or flagged as an estimate.
v0.11.0	**Agent Autonomy Bridge + live observatory.** Per-task permission grants, stalled-job handoff, interactive `doctor` fix wizard, `synlynk watch --live`, and full SOP blocks in every agent directive at init.
v0.10.0	**Developer Preview:** pipx install, `init --wizard` FTUE, `synlynk viz` browser dashboard, `launch` task picker, `migrate` to state.db, and `release` ship-cycle stub.
v0.9.4	**Tasks live in SQLite now** — todo.md is generated, never hand-maintained. Agents get per-profile context budgets. A new relay broadcast channel connects your workgroup in real time.
v0.9.3	**synlynk can run as a system service** — an always-on daemon with a job queue, priority chains, and crash-safe restarts. Set it once with `--install-service`.
v0.9.2	**Teams ship faster when onboarding is automatic.** `synlynk join` seeds a new member from git history. `synlynk decide` runs a multi-agent consensus panel in one command.
v0.9.1	**The install was broken for repos not in the source tree — fixed.** Init now migrates existing docs rather than overwriting them.
v0.9.0	**Dispatches used to inject 100KB of full context for a 5-line task.** Now they inject ~400 bytes of story-scoped context. 60% token reduction in practice.
v0.8.0	**The Support Engineer agent monitors your repo** for failing tests, flaky tests, stale deps, and open issues — filing GitHub issues and draft PRs automatically.
v0.7.0	**Every exec session includes a ## Source Architecture snapshot** from your actual codebase — agents understand your file tree without you explaining it.
v0.6.x	**Added model-aware routing.** Codex, Claude, and AGY each get capability scores — the best agent for a story gets picked automatically.
v0.5.0	**Introduced the capability engine:** 3D domain scoring (engg/org/industry), a SQLite ledger, and anti-gaming quality caps.
v0.4.x	**The dispatch system** — background jobs, the trio runner, agent discovery, init wizard. This is when synlynk became a workgroup tool.
v0.3.x	**Multi-agent from day one** — CLAUDE.md, GEMINI.md, AGENTS.md all generated from one init. Sentinel hardening.
v0.1–0.2.x	**The kernel:** context injection, cost tracking, flatline detection, budget enforcement.

📌 **VERSIONING**

Synlynk follows a layered roadmap: kernel → filesystem → dispatch → scheduler → relay. The CLI surface freezes at v1.0; everything before is additive and backwards-compatible within a minor series.

ALL COMMANDS

Everything at a glance.

Setup & Context

COMMAND	WHAT IT DOES
`init [--docs-dir]`	Bootstrap docs + agent files. Migrates existing docs. Accepts `--mode`, `--docs-dir`, `--org`, `--repo`, `--project-id`, `--force`, `--agents`.
`exec <agent>`	Run an agent interactively with full context injection. Captures cost on exit.
`scan [--deep] [--status]`	Index source files. `--deep` writes the symbol DB + source-map.md.
`checkpoint`	Snapshot docs + telemetry for a safe rollback point.
`status [--json]`	Dashboard: costs, alerts, jobs, team digest.
`upgrade · --version`	Apply latest release · print installed version.
`doctor`	Run health checks (TC-1–5) with interactive fix wizard. V0.10.0
`probe [--agent]`	Probe harness capability / version drift; record compatibility. V0.11.0
`repair · sync · exit`	Re-init from config · propagate artifacts · remove onboarding. Dry-run default. V0.9.8
`cost log`	Log a manual cost entry for native/unwrapped sessions. V0.12.0
`roles [--fix]`	Show agent role table; optionally write missing role fences. V0.10.0
`instructions status|diff|update|ack`	Manage synlynk-managed instruction sections across AI tools.
`viz [--serve|--generate|--open|--stop|--port]`	Local browser workspace dashboard (Vizor). V0.10.0
`release [--dry-run] [--version] [--minor]`	Cut a named release (VERSION, CHANGELOG, blog stub). V0.10.0

Dispatch, Jobs & Daemon

COMMAND	WHAT IT DOES
`dispatch <agent>`	Background job. Accepts `--task`, `--story`, `--context-mode none|task|full`. Pre-flight gated.
`jobs [--all] [--watch]`	List jobs from SQLite. `--watch` follows live.
`jobs handoff <job-id> [--to agent]`	Transfer a stalled job to another agent. V0.11.0
`logs --job <id> [--tail N]`	Tail a specific job's output log.
`shell · launch · run --trio`	REPL · interactive launcher · dispatch agents in parallel.
`schedule [--execute] [--max-stories N]`	Batch-assign ready stories (dry-run default). V0.12.0
`daemon start|stop|status`	Control the always-on service. V0.9.3
`daemon install-service`	Install as launchd / systemd service. Context server on :27471.

Relay, Team, Stories & Identity

COMMAND	WHAT IT DOES
`relay start [--port N]`	HTTP SSE broker (default :27472). V0.9.4
`relay broadcast <body>`	Publish to subscribers. `--kind motd|wellness|message|joke|custom`, `--relay-url`.
`agent configure <name>`	Edit `.agents/<name>.json` (context_mode, context_max_bytes). V0.9.4
`join · team status · decide`	Onboard from git history · team dashboard · multi-agent consensus.
`story create|list|show`	Manage tracked work items (SQLite canon). `--title`, `--tokens`, domain flags.
`goal create|list|link|status`	Business goals: create, list, link stories, completion rollup. V0.12.0

FEATURE MATRIX

What synlynk adds to raw AI CLIs.

Every feature below requires explicit setup without Synlynk — copy-paste context, manual cost tracking, custom scripts. Synlynk delivers them in a single `synlynk init`.

FEATURE	SINCE	WITHOUT SYNLYNK	WITH SYNLYNK
🧠 CONTEXT & MEMORY			
Project context injection	v0.1	Manual copy-paste each session	✓ Auto-injected every exec
Source architecture snapshot	v0.7	Describe files manually	✓ Git-cached, auto-injected
SQLite-canon task state	v0.9.4	× Hand-edited todo files	✓ SQLite primary, todo.md generated
Per-agent context profiles	v0.9.4	× Same context for all	✓ `.agents/<name>.json` budgets
Smart doc migration on init	v0.9.1	× Blank templates only	✓ Finds & migrates existing docs
💰 COST & SAFETY			
Per-session cost tracking	v0.1	Check billing dashboard manually	✓ Budget Pulse after every exec
Budget enforcement	v0.1	No guard — runaway spend possible	✓ Hard block at configured limit
Flatline / loop detection	v0.1	Discover after burning tokens	✓ FLATLINE alert at 3rd failure
Verification gap detection	v0.9.4	× None	✓ VERIFY_SKIP sentinel
Cost provenance	v0.12.0	× Hardcoded per-token estimates	✓ Structurally sourced or visibly flagged
⚡ DISPATCH & AUTONOMY			
Background agent dispatch	v0.4	nohup hacks, manual PID tracking	✓ First-class job store + logs
Parallel multi-agent trio	v0.4	3 separate shell windows	✓ `synlynk run --trio`
Scoped task context (60% less)	v0.9	Full context every dispatch	✓ Story-filtered domain slice
Always-on daemon + job queue	v0.9.3	× None	✓ launchd/systemd, priority chains
Dispatch git-finalization	v0.12.0	× Agent must remember to commit/push/PR	✓ synlynk finishes it automatically
👥 TEAMS, RELAY & IDENTITY			
Per-user devlogs	v0.2	One shared log or nothing	✓ `devlogs/username.md`
One-command onboarding	v0.9.2	Manual setup per teammate	✓ `synlynk join` + digest
Real-time relay broker	v0.9.4	× None	✓ HTTP SSE on :27472, broadcast
Capability routing + Ed25519	v0.5	Route by gut feel, no signing	✓ SQLite ledger, signed scores

💡 **25+ COMMANDS ACROSS 6 OS LAYERS**

kernel (init, flatline, budgets) · **filesystem** (costs, telemetry, scan, SQLite canon) · **IPC** (dispatch, profiles, identity) · **scheduler** (capability engine, daemon, priority chains) · **relay** (HTTP SSE broker, broadcast — shipped v0.9.4)

ROADMAP PREVIEW

What's coming next

Synlynk evolves from an agent wrapper into a full OS for multi-agent development.

v0.1–0.9.4 — **Kernel → Scan → Dispatch → Teams → Daemon → Relay** ✓ Shipped

472 tests · SQLite canon · agent profiles · always-on daemon · HTTP SSE relay · capability engine · Ed25519 signing

v0.10 — **Workspace Federation** Q4 2026

Cross-repo dispatch from one session · invisible state (SQLite replaces project-docs/ from git) · `synlynk migrate`

v0.11 — **Hosted Relay & Sync** Q1 2027

relay.synlynk.com · encrypted state sync · self-hosted VPS option · cross-machine subscriber federation

v1.0 — **Stable OS + Tokq Bridge** Q2 2027

Frozen CLI · pipx/Homebrew · NATS leaf node · Tokq capability marketplace ready

`github.com/nikhilsoman/synlynk`
Source · Issues · PRs

`synlynk.com`
Docs · Releases

SYNLYNK

THE OS FOR MULTI-AGENT DEVELOPMENT · V0.12.0