COMMAND REFERENCE

SYNLYNK

Every command. Every flag. Every scenario.

v0.12.0 · Complete CLI Reference

```
$ synlynk --help

setup init · exec · checkpoint · status · scan
dispatch jobs · logs · shell · launch · run
relay start · broadcast
team join · team status · decide
daemon start · stop · status · install-service
```

Python 3.8+ | stdlib only | 472 tests | MIT

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

SETUP & CONTEXT

Bootstrap, run, and inspect your project.

`synlynk init` Bootstrap docs, agent files, and the state DB. Migrates existing docs.

FLAG	EFFECT
`--mode single|team`	Single user or team mode (per-user devlogs + pull guard).
`--docs-dir <path>`	Where docs live. Use `.` for repo root. Default `project-docs`.
`--force`	Overwrite existing synlynk-managed files instead of skipping.

`synlynk exec <agent>` Run an agent interactively with full context injection.

Prepends memory + roadmap + todos + source scan, spawns the agent, then records cost and runs sentinel checks on exit. No flags — the agent name is positional (claude / agy / codex).

`synlynk scan` Index source files into a Source Architecture snapshot.

FLAG	EFFECT
`--deep`	Full symbol extraction; writes source-map.md + populates state.db.
`--status`	Show cache age, HEAD sha, file and symbol counts.

`checkpoint`

Snapshot current docs + telemetry state for a safe rollback point.

`status [--json]`

Dashboard: costs, alerts, running jobs, team digest. `--json` for tooling.

`upgrade`

Pull and apply the latest synlynk release from the remote repo.

`--version`

Print the installed synlynk version and exit.

Scenario — bootstrap a repo and run a first session

setup flow

```
~/api $ synlynk init --mode team --docs-dir .
✓ Migrated roadmap.md · memory.md · 4 agents discovered (claude · agy · codex · grok)
~/api $ synlynk scan --deep
✓ 284 files · 1,024 symbols · source-map.md written
~/api $ synlynk status
💰 $0.00 / $10.00 · 0 alerts · 0 jobs · 3 members
~/api $ synlynk exec claude
claude › Context loaded. Next roadmap item is...
```

DISPATCH & JOBS

Fire agents.
Track every job.

`synlynk dispatch <agent>` Run an agent as a background job; returns a job ID.

FLAG	EFFECT
`--task "..."`	The prompt / work item to hand the agent.
`--story <id>`	Link to a tracked story; injects only the story-scoped context slice.
`--context-mode none|task|full`	How much context to inject: nothing, story-scoped, or the full snapshot.

`jobs`

`--all`	Include completed and failed jobs.
`--watch`	Live-follow status (SQLite-backed).

`logs`

`--job <id>`	Which job's log to show.
`--tail N`	Limit to the last N lines.

`shell`

Open an interactive synlynk REPL with context preloaded.

`launch`

Open the agent launcher to pick an agent and task interactively.

`run --trio`

Dispatch one task to all three agents in parallel; one job ID each.

Scenario — dispatch two agents, then watch and tail

dispatch flow

```
$ synlynk dispatch claude --story story-abc123 --context-mode task --task "Add tests"
✓ Pre-flight gate passed · ~420 bytes context · job-a3f2
$ synlynk dispatch codex --context-mode none --task "Rename util fn"
✓ Dispatched job-b7e1 (codex) self-contained, no context
$ synlynk jobs --watch
● job-a3f2 claude running ● job-b7e1 codex done
$ synlynk logs --job job-a3f2 --tail 3
→ 12/12 tests passing
```

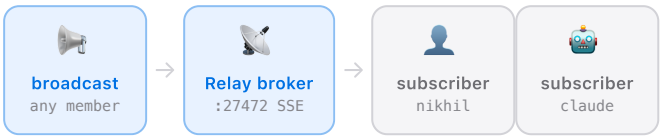
⚠ **PRE-FLIGHT DISPATCH GATE** V0.9.4

Every dispatch is checked before launch — agent installed, budget available, context within the profile's `context_max_bytes`. Failing the gate blocks the job before any tokens are spent.

RELAY V0.9.4

A real-time channel for your workgroup.

The relay is a lightweight HTTP Server-Sent-Events broker. One member starts it; everyone else subscribes over HTTP. Broadcasts fan out to every connected agent and human in the workgroup — a publish/subscribe bus with no external service.



`synlynk relay start` Start the SSE broker. Subscribers connect over HTTP.

FLAG	EFFECT
`--port N`	Port to bind. Default `27472`.

`synlynk relay broadcast <body>` Publish a message to every subscriber.

FLAG	EFFECT
`<body>`	Positional — the message text to broadcast.
`--kind motd|wellness|message|joke|custom`	Message classification subscribers can filter on.
`--relay-url URL`	Target a remote relay instead of localhost.

Scenario — start the relay, broadcast a wellness check

relay flow

```
~/api $ synlynk relay start --port 27472
✓ Relay broker listening on http://localhost:27472/events
Subscribers: 2 (nikhil · claude-daemon)
# from another shell or teammate
$ synlynk relay broadcast "Take a 5-min break — long run ahead" --kind wellness
✓ Delivered to 2 subscribers · kind=wellness
# target a remote relay
$ synlynk relay broadcast "Deploy at 4pm" --kind motd --relay-url https://relay.acme.dev
✓ Delivered to remote relay
```

PAIR IT WITH THE DAEMON

Run the relay alongside `synlynk daemon` so an always-on subscriber can react to broadcasts — enqueue jobs on an MOTD, pause on a wellness signal.

STORIES & CAPABILITY

Track work.
Route to the best agent.

Stories are the unit of work — stored in the SQLite stories table and rendered into `todo.md`. Scores rate how well each agent handles a domain; the capability engine uses them to pick agents automatically.

`synlynk story create` Create a tracked work item with a deterministic ID.

FLAG	EFFECT
`--title "..."`	Story title.
`--tokens N`	Estimated token budget for the story.
`--engg / --org / --industry`	3D domain scores used by capability routing.

`story list`
List all stories with status and IDs.

`story show <id>`
Full detail for one story: tasks, domain scores, dispatch history.

`score list`
Aggregated capability scores per agent per domain.

`score attest`
Ed25519-sign a score so it is tamper-evident.

`synlynk identity init` Create the Ed25519 signing key at ~/.synlynk/identity.key.
Prints the public key for Tokq export. Required before `score attest` can sign.

Scenario — create a story, dispatch to the best agent

story flow

```
$ synlynk story create --title "Refactor auth middleware" --engg 9 --tokens 8000
✓ Created story-7f3a · written to stories table · todo.md regenerated
$ synlynk score list
claude engg 4.7 · codex engg 4.9 · agy engg 4.2
$ synlynk dispatch codex --story story-7f3a --task "Refactor"
✓ Best engg score → codex selected · job-c9d4 running
$ synlynk score attest --story story-7f3a --agent codex --quality 5
✓ Score signed (Ed25519) · attestation recorded
```

🎯 **ANTI-GAMING QUALITY CAPS**

The capability engine caps how fast a score can rise from repeated self-attestation, so an agent can't inflate its own routing weight. Signed attestations from real outcomes carry the most.

AGENT & PROFILE V0.9.4

Per-agent context budgets.

Each agent can carry a profile at `.agents/<agent>.json` that controls how much context it receives. A code-completion agent rarely needs the full roadmap — give it a tight budget and skip context entirely for self-contained tasks.

`synlynk agent configure <name>` Create or edit an agent profile interactively.

Profile schema — .agents/<name>.json

.agents/codex.json

```
{
"context_mode": "none", # none | task | full
"context_max_bytes": 2000 # hard ceiling on injected context
}
```

FIELD	MEANING
context_mode	`none` = inject nothing; `task` = story-scoped slice; `full` = entire snapshot. A `--context-mode` flag on dispatch overrides the profile for one run.
context_max_bytes	Upper bound on injected context. The pre-flight gate truncates or blocks if exceeded.

Scenario — make codex self-contained, claude full

profile flow

```
$ synlynk agent configure codex
context_mode? [none/task/full] › none
context_max_bytes? › 0
✓ Wrote .agents/codex.json — codex now gets zero context
$ synlynk agent configure claude
context_mode? › full
✓ Wrote .agents/claude.json — claude gets the full snapshot
$ synlynk dispatch codex --task "Rename helper()"
✓ No context injected (profile: none) · job-d1e2
```

🧩 **WHY PER-AGENT PROFILES**

Self-contained refactors don't need a 100KB project snapshot. Profiles let each agent pay only for the context it actually uses — measurable token savings across a workgroup.

DAEMON V0.9.3

Always-on synlynk. A job queue that survives.

The daemon runs synlynk as a background service with a persistent job queue, priority chains, and crash-safe restarts. It also serves an HTTP context endpoint on `localhost:27471` so other tools can fetch the live project snapshot.

daemon start

Start the daemon in the foreground (or detached if installed as a service).

daemon stop

Stop the running daemon and flush the queue to disk.

daemon status

Show PID, queue depth, context server port, and uptime.

daemon install-service

Install as a launchd (macOS) or systemd (Linux) service — auto-start on login/boot.

Scenario — install as a launchd service

daemon flow

```
$ synlynk daemon install-service
✓ Wrote ~/Library/LaunchAgents/dev.synlynk.daemon.plist
✓ launchctl loaded · starts on login · KeepAlive=true
$ synlynk daemon status
PID 48213 · up 12s · queue: 0 jobs · context server :27471
$ curl localhost:27471/context
{ "memory": "...", "roadmap": "...", "stories": 12 }
$ synlynk dispatch claude --task "..." # enqueued, not blocking
✓ Queued behind 0 jobs · daemon will run it
```

⏱ **PRIORITY CHAINS**

Jobs enqueued through the daemon run in priority order, and a chain can fan a story out to dependent follow-up jobs. A crash mid-queue restarts cleanly — state persists in `state.db`.

⚠ **ONE DAEMON PER MACHINE**

The context server binds a fixed port. Run a single daemon per host; it serves all repos by reading each one's `state.db` on request.

TEAM V0.9.2

Onboard, sync, and decide together.

`synlynk join` Onboard the current user into a team repo.

FLAG	EFFECT
`--user <name>`	Override the identity (defaults to git config user.name).
`--mode single|team`	Confirm the repo mode when joining.

Seeds `devlogs/<user>.md` from git history, regenerates the agent instruction files, and prints a team digest.

`synlynk team status` Show members, last-active times, open stories, and active jobs.

`synlynk decide` Run a multi-agent consensus panel on a question.

FLAG	EFFECT
`--topic "..."`	The decision question put to the panel.
`--panel claude,agy,codex,grok`	Which agents sit on the panel (defaults to all functional agents).

Scenario — onboard a teammate, run a consensus decision

team flow

```
~/api $ synlynk join --user priya --mode team
✓ Seeded devlogs/priya.md from 38 commits · agent files regenerated
Digest: nikhil (relay broker) · arjun (capability routing)
~/api $ synlynk team status
3 members · 12 open stories · 2 active jobs
~/api $ synlynk decide --topic "Postgres or SQLite for the ledger?" --panel claude,agy,codex,grok
claude › SQLite — zero ops
agy › SQLite
codex › SQLite
grok › SQLite
✓ Consensus: SQLite (4/4) · logged to memory.md [@priya]
```

PULL-BEFORE-WRITE GUARD

In team mode synlynk auto-pulls from origin before writing any shared doc, preventing divergence across concurrent sessions and worktrees. Decisions carry `[@username]` attribution.

SENTINEL & INIT FLAGS

Manage alerts. Master init.

`synlynk sentinel list` Show active alerts with severity, code, message, timestamp.

`synlynk sentinel clear` Clear alerts in bulk.

FLAG	EFFECT
`--severity CRITICAL|WARN`	Clear all alerts at this severity.
`--code <CODE>`	Clear alerts of one code (e.g. `VERIFY_SKIP`, `FLATLINE`).

Sentinel codes you'll see

FLATLINE
Same command failed 3× in a row.

⚠ SUCCESS_LOOP
Same command succeeded 5× in <10 min.

QUOTA_EXHAUSTED
Output matched a quota-exhaustion phrase.

⚠ VERIFY_SKIP `V0.9.4`
Job exited 0 but no test evidence in output.

Full init flags

FLAG	EFFECT
`--force`	Overwrite synlynk-managed files instead of skipping existing ones.
`--agents claude,agy,codex,grok`	Restrict discovery to a specific set of agents.
`--mode single|team`	Single user, or team mode (per-user devlogs + pull guard).
`--org <name>`	Org slug baked into agent instruction files.
`--repo <name>`	Repo slug baked into agent instruction files.
`--project-id <id>`	GitHub Projects v2 node ID for a shared board across repos.
`--docs-dir <path>`	Where docs live. `.` for repo root; default `project-docs`.

Scenario — clear a false-positive VERIFY_SKIP alert

sentinel flow

```
$ synlynk sentinel list
[WARN] VERIFY_SKIP job-c9d4 exited 0, no test output — 2026-06-24 11:14
# the job ran a docs-only change — no tests expected
$ synlynk sentinel clear --code VERIFY_SKIP
✓ Cleared 1 VERIFY_SKIP alert
```