ThunderGate v1.2.3 Operator Manual

ThunderGate

> *Sovereign AI runtime. Your agent. Your hardware. Your data.*

ThunderGate is a personal AI runtime that hosts, manages, and continuously improves an AI agent. Unlike cloud AI services, ThunderGate runs on hardware you control, maintains persistent memory across sessions, and learns from every interaction — building a genuine model of who you are and how you work over time.

This README is the full manual: what ThunderGate is, how to install and set it up, how to migrate an existing agent, and how to use every corner of the runtime — written for both the human operator and the agent running inside it.

Table of Contents

What ThunderGate Is

A ThunderGate **agent** is a persistent AI identity that lives on your machine. It has a name, a behavioral contract, long-term memory, a credential vault, and a set of tools it can use to act in the world (run commands, read/write files, browse the web, send messages, schedule tasks). It runs as a supervised background service, talks to you over the channels you configure (Slack, Telegram, etc.), and every night distills what happened into durable memory.

You can run **one** agent or a **federation** of agents that monitor and assist each other over a private mesh.

Install

ThunderGate ships as a **pre-built runtime** (the `dist/` build is committed — no compile step) installed by a one-command script. It clones the repo to `~/.thundergate`, wires the `thundergate` CLI onto your PATH, and seeds

a sterile `config.json` (an existing config is never overwritten).

Prerequisites: Node.js 18+ and `git` — and the installer **auto-installs both when missing** (Node via `nvm`, no sudo; `git` via your system package manager). The native modules install as **prebuilt binaries** (no C compiler needed on common Linux/macOS). The only thing you must supply yourself is an **Anthropic API key**, entered later at `thundergate setup`. See Requirements & Feature Dependencies for the full breakdown and what each optional feature needs.

```
# One-command install (interactive runtime)
curl -fsSL https://thunderai.us/install.sh | bash

# Install AND register the supervised service (launchd on macOS / systemd on Linux)
curl -fsSL https://thunderai.us/install.sh | bash -s -- --service
```

Supported platforms: Linux, macOS, and Windows via WSL2.

> Note: the runtime is distributed via the installer above (git-based), not via `npm install`. The installer keeps `dist/` current so there is no build step on the target machine.

Requirements & Feature Dependencies

ThunderGate is built so a **single command on a clean box** gets you a working agent. The installer auto-provisions everything it can; this table is the honest map of what's needed for what — so you (or your agent) know up front, instead of finding out when a tool silently fails.

Core — required to run at all

Need	How you get it	Notes
Node.js 18+ (20 LTS recommended)	**Auto-installed** by the installer via `nvm` if missing/old	User-space, no sudo
git	**Auto-installed** via your package manager if missing	Used to fetch + update the runtime
Anthropic API key	**You provide it** at `thundergate setup`	The brain. Nothing runs without it — it's a credential, so there's nothing to "download"
glibc Linux / macOS / WSL2	Already present	Native modules ship **prebuilt binaries** (`better-sqlite3`, `@node-rs/argon2`) — a C compiler is needed *only* as a fallback on musl/Alpine or exotic arch, and the installer auto-installs the toolchain if that fallback triggers

Auto-provisioned — no action from you

Capability	What happens
Local vector memory (semantic search, episodic recall)	The `Xenova/all-MiniLM-L6-v2` embedding model (~23 MB) downloads automatically the first time memory is indexed (during `setup`). Needs internet once, then runs fully local — no API calls
Encrypted vault + SQLite storage	Installed as prebuilt native binaries — no compiler

Optional features → what each one needs

Like any app, some capabilities need extra software on the machine. None of these block the core agent — without them, that one feature is simply unavailable and everything else works.

Capability	Needs	Without it
Browser automation (ThunderBrowser)	Electron build + Chromium + (headless Linux) Xvfb and shared libs. Opt in with the installer flag **`--with-browser`**	Browser tools are unavailable; the rest of the agent runs normally. *(Full headless-Linux browser support is still being ported.)*
Voice input (speech-to-text)	`python3` + Whisper	Voice transcription is off
Voice output (text-to-speech)	A platform TTS (`say` on macOS, `espeak` on Linux)	Spoken replies are off
Slack / Telegram / etc.	The channel's token, entered at `thundergate setup`	That channel stays inactive; others still work
Supervised background service	`launchd` (macOS) or `systemd` (Linux) — add the `--service` flag at install	Run the agent manually with `thundergate start`

> **Install flags:** `--service` registers the supervised background service; `--with-browser` installs the browser stack. Combine them: `curl -fsSL https://thunderai.us/install.sh | bash -s -- --service --with-browser`.

Setup (new agent)

Create a brand-new agent identity from scratch:

```
thundergate setup
```

Interactive setup walks you through: - **Identity** — the agent's name and operator (you). This seeds the layered identity (`identity.json`, `contract.md`, `SOUL.md`). - **AI keys** — Anthropic / OpenAI / xAI / Google keys. Keys are written **vault-first** (into the encrypted agent vault), never left in plaintext config when avoidable. - **Channels** — Slack / Telegram / etc. tokens and the operator DM target. - **Capability discovery** — detects and offers to install helpful local tools.

After setup, `thundergate start` brings the agent online.

Migrate (existing agent)

Bring an existing agent (e.g. from OpenClaw) into ThunderGate, preserving its identity and history:

```
thundergate migrate
```

Migration: - Detects and slugifies the agent ID, creates `~/.thundergate/agents/<id>/`. - Runs an LLM-backed **contract distillation** (one call) to produce the behavioral contract — so it requires a working LLM key (resolved **vault-first**, so a vault-only key works). - Reconciles identity, memory, and crons; writes a post-migration checklist. - Is **re-runnable** — a prior partial migration can be cleanly overwritten rather than blocking a retry.

Running the Runtime

```
thundergate start      # start the agent (supervised; auto-restart on crash)
thundergate stop       # stop it
thundergate restart    # restart (re-arm the launchd/systemd supervisor)
thundergate status     # health: channels, peers, memory, deps
thundergate doctor     # deeper diagnostics (DB, vault, identity, embeddings)
thundergate update     # pull the latest build
thundergate cost [period]   # token/cost accounting
thundergate watch-cli  # live tail of agent activity
```

The runtime is a Node.js process kept alive by `launchd` (macOS) or `systemd` (Linux). A crash respawns automatically; `restart` re-arms the supervisor.

Core Capabilities

1. Persistent Identity

Every ThunderGate agent has a structured identity that persists across sessions — who they are, what they've learned, how they operate. Conversations don't start from scratch. The agent remembers.

Identity is layered: - **L0 — identity.json** — always loaded, structured entity store (name, behavioral patterns, active promises, significant decisions) - **L1 — contract.md** — behavioral rules distilled from lived experience - **L2 — working.json** — live session state, refreshed every 25 turns - **L3 — episodic index** — past conversations, vector-searchable on demand

2. Unified Memory Index

ThunderGate maintains a single searchable corpus across all memory sources — conversations, daily logs, project documents, design briefs, and commit messages — searched with **local vector embeddings** (no API call, no cloud dependency) in under a second.

Ask *"what did we decide about X last month?"* and the agent searches its entire history — not a loaded context window, but a semantic search across everything it has ever worked on. The agent can also **list what it knows** (a live table of contents by source type) and page through the recency-ranked long tail.

3. Ambient Intelligence

The Ambient Intelligence layer runs silently on every conversation: - Detects topic shifts and triggers a context refresh before the conversation drifts - Captures load-bearing decisions in real time to a persistent decision log - **Pre-stages relevant memories before you ask for them** (anticipatory injection) - Builds behavioral patterns that improve over time — corrections, affirmations, preferences

This is the agent's subconscious: always present, never intrusive.

4. Two-Mode Inference

ThunderGate operates in **Cloud Mode** (Anthropic Claude, OpenAI, xAI Grok) or **Local Inference Mode** (any Ollama-compatible model on local hardware).

The master switch is human-controllable, agent-controllable, and includes automatic failover — if local inference fails, the runtime falls back to cloud and latches until you deliberately switch back. Switching is hot —

no restart required.

5. Sovereign Vault

Credentials, API keys, and sensitive data live in an **AES-256-GCM** encrypted vault (Argon2id KDF) with biometric authorization (Touch ID on macOS). The agent can use vault-stored secrets to authenticate to external services without those secrets ever appearing in conversation context or logs.

Access is purpose-bound: the agent requests capability for a specific action, receives a scoped credential, and the access expires. Hash-chained audit receipts log every vault interaction. There are two vaults: an **agent vault** (operational keys the agent uses freely) and a **human vault** (PII, gated to unlock-on-read).

6. Tool Execution

ThunderGate agents can execute shell commands, read/write files, make API calls, schedule cron jobs, and browse the web — all through a unified tool dispatch layer with security gates.

Key safety mechanisms: - **Exec validation** — malformed or dangerous commands (unbounded recursive scans, trailing operators) are caught before execution - **Irreversible-action gate** — actions matching destructive patterns surface a confirmation before executing - **ThunderBrowser** — agent-controlled browser automation with DOM-grounded actions, shadow-DOM piercing, and a confirmation gate on form submissions

7. Multi-Agent Federation

Multiple ThunderGate agents communicate via a peer mesh (currently Tailscale-based). Agents can exchange messages directly, monitor each other’s health and restart peers that go down, share context about shared projects, and operate as a coordinated team with defined roles.

8. Self-Healing Runtime

The runtime monitors itself and agents watch each other. On trouble: the runtime attempts recovery before alerting you, peer agents can diagnose and restart each other, and detailed provenance logging captures every significant event for post-incident review.

9. Nightly Distillation (The Scribe)

Every night, the Scribe runs automatically: - Distills the day’s conversations into MEMORY.md (long-term memory) - Indexes new content into the unified memory corpus + trims stale vector entries - Archives the decision log and generates a fresh ambient-context feed for the next session - **Stages** durable “graduations” to long-term memory for your review rather than writing them silently — you approve or reject them with `scribe_review` (a quality gate on what becomes permanent)

The agent wakes up oriented — aware of current projects, recent activity, and standing behavioral patterns.

CLI Command Reference

Every command is `thundergate <command>`. Grouped by area:

Lifecycle & health | Command | What it does | |—|—| | `start` / `stop` / `restart` | Run / halt / restart the supervised agent | | `status` | Channels, peers, memory, dependency health | | `doctor` | Deep diagnostics (DB,

vault, identity, embeddings) | | `update` | Pull the latest build | | `cost [period]` | Token & cost accounting | | `watch-cli` | Live activity tail | | `worldstate` | Dump the agent's current WorldState |

Identity & onboarding | Command | What it does | |—|—| | `setup` | Create a new agent | | `migrate` | Migrate an existing agent | | `onboard` | Guided first-run walkthrough |

Memory | Command | What it does | |—|—| | `memory list` / `memory show <key>` | Inspect curated memory | | `untrain remove <key>` | Remove a learned pattern | | `frame current` / `recent` / `transitions` | Inspect the agent's "frame" (working focus) state | | `promises list` / `close <id>` | Track / close commitments the agent made | | `log` | Write/inspect the decision log |

Scheduling | Command | What it does | |—|—| | `cron list` / `add <when> <channel> <message…>` / `remove <id>` | Manage scheduled prompts/tasks |

Vault — human vault (PII, gated) | Command | What it does | |—|—| | `vault status` / `unlock` / `lock` | Vault state | | `vault add <category> <label>` / `list` | Store / list secrets | | `vault access <label>` / `grant` / `receipts` | Gated read, capability grant, audit receipts | | `vault providers` / `key-setup` / `recover …` | Key provider config, setup, recovery |

Vault-A — agent vault (operational keys) | Command | What it does | |—|—| | `vault-a status` / `unlock` / `list` | Vault state / contents | | `vault-a add <name> <value>` / `use <name>` | Store / select a key | | `vault-a key-setup` / `key-migrate` | Boot auto-unlock setup; migrate config keys into the vault |

Browser & misc | Command | What it does | |—|—| | `browser start` / `stop` / `status` / `restart` / `state` | ThunderBrowser lifecycle | | `github read <repo> <filepath>` | Read a file from GitHub | | `transcribe <audio>` | Transcribe an audio file | | `test-request` | Diagnostic inference round-trip | | `gate` | Operator approval gate controls |

The Agent's Toolbox

Inside a conversation the agent acts by emitting tool tags (`<tool:name .../>`). The runtime dispatches them through security gates and returns results. The toolbox:

- **System** — `exec`, `exec_ps` (shell, validated + irreversible-gated), `read_file`, `list_dir`, `write_file`
- **Memory** — `memory_search` (semantic search + `sources=` listing), `memory_expand` (load a pointer's full exchange), `scribe_review` (approve/reject staged graduations), `self_inspect`, `self_inspect_learning`
- **Comms** — `send_message`, `send_sms`, `email_read`, `email_reply`, `slack_upload_file`, `slack_react`, `session_context`
- **Calendar** — `calendar_read`, `calendar_google`, `calendar_setup`
- **Vault** — `vault_status/unlock/list/read/add/delete/edit`, `agent_secret_set/delete/ref`, `human_secret_ref`, `vault_divulge` (password-gated)
- **Browser (ThunderBrowser)** — `browser_open`, `browser_navigate`, `browser_act`, `browser_read`, `browser_read_dom`, `browser_watch`, `browser_screenshot`, `browser_fill`, `browser_audit_log`
- **Scheduling & tasks** — `cron_add`, `cron_list`, `cron_remove`, `taskflow`, `skill_manage`
- **Knowledge & search** — `search` (web), `weather`, `flight_status`, `youtube_stats`, `ask_llm` (call another model), `gog`
- **Self-management** — `deep_mode` (enter focused work mode), `knob_list` (adjustable runtime knobs), `config_write` (whitelisted config), `tuning_decision`, `learn_analyze`, `learning_feedback`, `check_agent_status` (peer liveness), `touch_id_authorize`, `passkey_vault`

Configuration

Configuration lives in `~/.thundergate/config.json` (seeded from `config.template.json`; **this file holds secrets — never commit it**). Top-level sections:

Section	Holds
`runtime`	`agentId`, `operatorName`, `model`, `maxTokens`, `temperature`, `mentionGated`
`user`	Operator profile
`fleet`	Multi-agent topology + Slack channel targets
`channels`	Per-channel tokens & settings (Slack, Telegram, …)
`browser`	ThunderBrowser config
`deepMode`	Deep-mode thresholds
`location`	Operator location/timezone
`proactive`	Proactive-messaging intensity & cooldowns
`memory`	Semantic retrieval, composite scoring, preemptive injection, briefs dir
`learning`	Ambient-intelligence / learning-loop knobs
`ghost`	Ghost-mode (shadow testing)

Many runtime knobs are self-adjustable by the agent (within bounds) and inspectable via `knob_list` / the `config_write` whitelist — the agent can tune its own behavior without an operator edit, but security-critical settings are hardcoded and never config-writable.

Channels

- **Slack** — full two-way integration (bot + DM)
- **Telegram** — bot integration
- **Discord** — bot integration
- **Email** — SMTP/IMAP
- **WhatsApp** — via wacli bridge
- **ThunderCommo** — native iOS app (in development)

For the Agent

If you are the agent running inside ThunderGate, this is how your world works:

- **Your identity** is loaded at boot from `identity.json` (L0) + `contract.md` (L1). Honor the contract — it's distilled from your lived corrections, not imposed from outside.
- **Your memory is bigger than your context window.** The slice you're shown is relevant, not total. When you need history — a past decision, an old conversation, a principle, *why* a commit was made — use `memory_search` rather than assuming it's already in front of you. Use `memory_search sources="true"` to see what you have at all.
- **Anticipated context** may be pre-staged into your prompt (`[ANTICIPATED CONTEXT …]`) — use it if pertinent, ignore it if not.
- **At session start**, if memory items are staged for graduation, you'll be nudged to run `scribe_review`. Show them to the operator and approve/reject **per their decision** — this is their curation gate; don't auto-approve.
- **Acting in the world**: emit tool tags. Destructive/irreversible actions surface a confirmation — don't try to bypass the gate. Credentials come from the vault via handles; never paste raw secrets into a channel.

- **Deep work**: enter `deep_mode` for multi-step tasks so the surface layer can field interrupts while you stay heads-down.
- **Going quiet vs. speaking**: the Ambient Intelligence layer advises on delivery; the operator's curation and corrections are the ground truth you adapt to.

Architecture

ThunderGate v1.2.1 completed a full modular partition of the runtime — what was an ~11,000-line monolith is now a slim composition root delegating to focused modules; v1.2.2 builds the unified memory index, anticipatory injection, and the Scribe quality gate on top.

Module	Role
TurnProcessor	Every message in, every response out
ToolRouter	Tool dispatch, security gating
ContextAssembler	Prompt assembly, session context
ScribeManager	Memory maintenance, nightly distillation
DeepModeManager	Focused work mode, task tracking
AmbientIntelligence	Background learning, anticipatory memory
InferenceRouter	Cloud/local model switching
Vault	Encrypted credential management

Security Model

- All credentials stored encrypted at rest (AES-256-GCM + Argon2id KDF)
- Biometric authorization for sensitive vault access; two vaults (agent / human-PII)
- Outbound redaction — agent responses are scanned for credential patterns before delivery
- Exec security gate — destructive operations require explicit confirmation
- Peer authentication — inter-agent messages are authenticated
- Audit log — all significant actions logged with hash-chained provenance

---

Patents

ThunderGate incorporates novel architectures covered by pending patent applications filed by Michael Joseph Lovell. Key innovations include:

- Sovereign agent identity with unbounded vector-searchable memory (Patent #16)
- Two-mode AI gateway architecture with seamless cloud/local inference failover (Patent #9)
- Bidirectional behavioral learning system with symmetric correction and affirmation (Patent #14)
- Layered multi-signal anticipatory memory injection (Patent #17)
- Unified vector memory index across heterogeneous sources (Patent #18)

Patent details available under NDA. Contact the team for licensing and partnership discussions.

ThunderGate is built by Michael Lovell and his AI team.
Boost and Bolt LLC / Thrust N Thunder