Hermes Agent Mastery

Complete Course Guide

Learn to Install, Configure, Deploy & Monetize Hermes Agent

by Nous Research

Published: July 2026

HERMES AGENT MASTERY - COMPLETE COURSE

Learn to Install, Configure, Deploy & Monetize Hermes Agent

by Nous Research

Published: July 2026 | Version: 1.0

What is Hermes Agent & Why It Matters

Back to Course Dashboard What is Hermes Agent & Why It Matters Module 1 of 10 ■ Full Walkthrough Video Your browser doesn't support video playback. All Modules Next: Installation & First Run **Course:** Hermes Agent — From Zero to Autonomous Agent **Module:** 1 of 8 **Reading time:** ~15 minutes **Difficulty:** Beginner — no prior AI agent experience needed TL;DR **What it is** An **open-source AI agent framework by Nous Research** that runs in your terminal, connects to 20+ LLM providers, and gets smarter over time through a built-in learning loop. **Not a chatbot** Unlike ChatGPT or Claude, Hermes has **persistent memory**, a **skills system** that self-improves, **tool calling** (file system, terminal, web, APIs), and can run **autonomously on a schedule** from a $5 VPS. **Key superpowers** Provider-agnostic, self-improving skills, persistent memory, multi-platform (Telegram, Discord, Slack, Signal + 15+ more), cron jobs, subagent delegation, MCP server support. **Who it's for** Developers, sysadmins, power users, researchers, home automators — anyone who wants an AI that does work, not just answers questions. **License** MIT — free forever, self-hosted, no data leaves your infra unless you choose. The Problem: Chatbots Answer Questions. Hermes Gets Things Done. Let's be honest — you've used ChatGPT, Claude, or Gemini. You've been impressed. You've asked it to write code, explain concepts, draft emails. And then… you copy-paste the output into some other tool, run the command yourself, save the file manually, switch to Slack to share it. That's the **chatbot pattern**: LLM in a browser tab, disconnected from your actual workflow. Hermes Agent breaks that pattern completely. Instead of a web chat window, it lives inside your **terminal** — connected to your file system, your shell, your browser, your APIs, your messaging apps. It doesn't just *tell you* the command to run — it **runs it**. It doesn't just *write* a script — it **executes it**, **tests it**, and **fixes it** if it fails. And then it remembers what it learned for next time. What Hermes Agent Actually Is Hermes Agent is an **open-source, autonomous AI agent framework** built by [Nous Research](https://nousresearch.com) — the same lab that trained the Hermes, Nomos, and Psyche language models. It's not a product you subscribe to; it's software you **self-host** on your own machine (or a $5 VPS, or a GPU cluster, or serverless infrastructure). Here's what makes it fundamentally different from a web chatbot: 1. It Runs in Your Terminal ChatGPT / Claude Hermes Agent **Interface** Web browser / mobile app Terminal CLI, TUI, or messaging apps **Execution** Tells you what to do Does it for you **Your data** On their servers On your machine **Tools** Browsing (limited) 70+ built-in tools: shell, files, web, browser, git, APIs **Memory** Session-only (or thin persistent) Persistent, curated, cross-session **Autonomy** Responds when prompted Runs on cron, spawns subagents, acts proactively When you launch Hermes, you get a `$` prompt in your terminal — just like a shell. But instead of running commands directly, you're talking to an AI that **has access** to your shell, file system, and a growing toolkit. Type a request, and Hermes: 1. **Thinks** about what needs to happen 2. **Calls tools** — writes files, runs terminal commands, searches the web, reads your codebase 3. **Iterates** — if the command fails, it reads the error and fixes it 4. **Delivers** the result 2. Provider-Agnostic: No Lock-In This is one of the most liberating features. Hermes works with **20+ LLM providers** out of the box, and you can switch between them with a single command: hermes model # Interactive provider/model selector hermes model openrouter:anthropic/claude-opus-4 # Or do it in one line Supported providers include: - **OpenRouter** — 200+ models, one API key (best starting point) - **Nous Portal** — Nous Research's own inference endpoint - **OpenAI** — GPT-4o, o3, o4-mini - **Anthropic** — Claude Opus 4, Sonnet 4 - **xAI** — Grok - **Google Gemini** — Gemini 2.5 Pro - **Hugging Face** — open models - **NovitaAI**, **NVIDIA NIM**,

MiniMax, **Kimi/Moonshot**, **z.ai/GLM**, **Xiaomi MiMo** - **Local endpoints** — Ollama, vLLM, LM Studio, or any OpenAI-compatible API Switch models mid-conversation with `/model`. Use different models for different tasks. Set up fallback chains so if one provider goes down, another takes over automatically. **No lock-in, no vendor risk.** 3. The Skills System: It Learns and Improves This is Hermes' secret weapon. Skills are **on-demand knowledge documents** that the agent loads when needed — like procedural memory for an AI. **How it works:** - Hermes has a library of skills in `~/.hermes/skills/` - Each skill is a markdown file with instructions for a specific task - Skills use **progressive disclosure** — the agent only loads skill content when it needs it, saving tokens - Built-in skills cover software development, research, creative work, system admin, and more - **Hermes can create new skills autonomously** after completing complex tasks - **Skills self-improve** — Hermes refines them during use Real-world examples of built-in skills: Skill What it does `plan` Creates implementation plans before coding `github-pr-workflow` Manages the entire PR lifecycle `research-paper-writing` Writes academic papers with templates `p5js` Creates generative art `manim-video` Produces animated explainer videos `gif-search` Searches and sends GIFs `axolotl` Fine-tunes LLMs with Axolotl `blog-watcher` Monitors RSS feeds and summarizes Skills are compatible with the agentskills.io open standard, meaning the community can share, modify, and port skills across compatible agent frameworks. 4. Persistent Memory That Actually Works Most chatbots forget everything the moment you close the tab. Hermes has a **bounded, curated memory system** that persists across sessions: - **MEMORY.md** — General knowledge about your preferences, environment, and recurring patterns - **USER.md** — A deepening model of who you are: your projects, your style, your goals - **Session search** — FTS5 full-text search across all past conversations, with LLM summarization - **Memory nudges** — Hermes periodically prompts itself to persist important information - **External providers** — Plug in Honcho, Mem0, Supermemory, or other backends for advanced memory After a few sessions, Hermes knows your project structure, your preferred tools, your coding style, and your communication preferences — without you having to re-explain everything every time. 5. Multi-Platform Gateway: Talk to It From Anywhere Hermes isn't tied to your terminal. A single gateway process connects it to **20+ platforms** simultaneously: - **Telegram** — Chat with Hermes from your phone - **Discord** — AI agent in your server - **Slack** — Workspace assistant - **WhatsApp** — Personal AI on your phone - **Signal** — Encrypted messaging - **Matrix** — Decentralized chat - **Mattermost** — Self-hosted team chat - **Email / SMS** — Old school, still supported - **DingTalk, Feishu, WeCom, Weixin, QQ Bot, Yuanbao, BlueBubbles** - **Home Assistant** — Control your smart home - **Microsoft Teams, Google Chat** Start a task from Telegram, check progress on Discord, get results delivered to Slack. The gateway handles **cross-platform continuity** — Hermes knows who you are regardless of which app you're using. 6. Scheduled Automation (Cron) Tell Hermes to do something on a schedule, and it will — even when you're not online: # In chat: /cron add "every 6h" "Check server health and alert if anything's wrong" /cron add "daily at 9am" "Summarize new items from my RSS feeds" --skill blogwatcher # From CLI: hermes cron create "every 1h" "Back up project files" --skill backup Cron jobs can: - Run one-shot or recurring - Attach zero, one, or multiple skills - Deliver results to any configured platform - Run in **no-agent mode** (pure script execution, zero LLM cost) 7. Subagent Delegation: Parallel Work Need to research three topics simultaneously? Hermes can **spawn isolated child agents** that work in parallel: delegate_task(tasks=[{"goal": "Research Python async patterns", "toolsets": ["web"]}, {"goal": "Find the best CI/CD tools for 2026", "toolsets": ["web"]}, {"goal": "Draft a database migration plan", "toolsets": ["terminal", "file"]}]) Each subagent gets a fresh context, restricted tools, and its own terminal. Only the summary enters the parent's context — saving tokens while maximizing throughput. Up to 3 concurrent subagents by default (configurable with no hard ceiling). 8. MCP Server Support Hermes connects to **Model Context Protocol (MCP) servers** — the emerging standard for AI tool integration. This means you can plug in: - **GitHub MCP** — Manage repos, issues, PRs - **Database MCP** — Query Postgres, SQLite, MySQL - **Filesystem MCP** — More granular file access - **Any custom MCP server** you build Per-server tool filtering lets you control exactly what Hermes can access. Use Cases: What People Actually Build With Hermes ■■ Software Development The most common use case. Hermes becomes your **AI pair programmer in the terminal**: - **Code generation** — "Create a FastAPI service with

auth, database models, and tests" - **Debugging** — "The CI is failing on this test, figure out why" → runs it, reads logs, fixes it - **Code review** — "Review this PR branch for security issues and performance problems" - **Refactoring** — "Rename all instances of `old_api` to `new_api` across the codebase" - **DevOps** — "Deploy the staging environment and run the smoke tests" ■ Research - "Research the latest papers on multi-agent RL and give me a summary with citations" - "Compare the architecture of llama.cpp vs vLLM" - "Track this open-source project's issues and PR activity over the past week" ■■ System Administration - "Check disk usage across all servers, find anything over 80%, and suggest cleanup" - "Rotate my SSL certificates" - "Monitor the logs for error rates and alert me on Telegram if they spike" ✍■ Content Creation - With the `manim-video` skill: "Create an animated explainer video explaining how transformers work" - With the `p5js` skill: "Generate a procedural art piece with flowing color gradients" - With the `research-paper-writing` skill: "Draft an ACL-style paper on our findings" ■ Home Automation Through the Home Assistant integration: - "Turn off all lights when nobody's home" - "What's the energy usage this month compared to last?" - "If the temperature drops below 60°F, turn on the heating" ■ Trading Bots - "Run this trading strategy on a schedule and notify me on Telegram if it triggers" - "Analyze these market conditions and write a report" Real Examples From the Community **"I have Hermes running on a $5 DigitalOcean droplet. It monitors my production servers, auto-deploys via GitHub Actions when tests pass, and sends me daily summaries on Telegram. I haven't SSH'd into that box in months."** — Senior DevOps Engineer **"I built a full-stack Next.js app in an afternoon. Hermes scaffolded it, set up the database, wrote the API routes, and deployed it. I just reviewed the code and merged."** — Indie Hacker **"My Hermes instance manages a Kanban board across my team. It auto-assigns tasks, checks progress daily, and escalates blocked items. It's basically a free project manager that never sleeps."** — Startup Founder **"I use Hermes as a personal research assistant. It scours ArXiv, reads papers, cross-references citations, and writes literature reviews. What used to take me two weeks now takes two hours."** — PhD Researcher Why Open Source Matters Hermes is **MIT-licensed open source**. This isn't just a checkbox — it has real implications: 1. **Your data stays yours** — No data ever leaves your infrastructure unless you explicitly connect a third-party API. Everything runs locally. 2. **No rate limits, no tiers** — There's no "Pro" plan that unlocks tool calling or memory. Every feature is available to every user. 3. **You can inspect every line** — The entire codebase is on GitHub. You can audit security, understand behavior, and contribute fixes. 4. **No vendor lock-in** — If Nous Research disappears tomorrow, your Hermes keeps running. It's your software. 5. **Community-driven improvement** — 17,000+ tests, hundreds of contributors, active Discord community. The pace of development is staggering — new features ship weekly. 6. **The skills ecosystem** — Anyone can create, share, and install skills. The [Skills Hub](https://agentskills.io) is a growing marketplace of community-contributed capabilities. The Hermes Ecosystem Resource URL Purpose **GitHub** github.com/NousResearch/hermes-agent Source code, issues, releases **Documentation** hermes-agent.nousresearch.com/docs Full docs, guides, references **Discord** discord.gg/NousResearch Community, support, show-and-tell **Skills Hub** agentskills.io Browse and share community skills **LLMs.txt** [docs/llms.txt](https://hermes-agent.nousresearch.com/docs/llms.txt) Machine-readable doc index for AI coding tools The community on Discord is exceptionally active — developers share setups, troubleshoot issues, contribute skills, and showcase what they've built. It's the best place to see what Hermes can really do. Key Takeaways - **Hermes Agent is an autonomous AI agent** that lives in your terminal, not a web browser. It can execute commands, write files, search the web, and interact with APIs — not just answer questions. - **It's provider-agnostic** — works with 20+ LLM providers including OpenRouter, OpenAI, Anthropic, and local models. No lock-in. - **The skills system is a closed learning loop** — Hermes creates, improves, and reuses skills autonomously, getting smarter the more you use it. - **Persistent memory** means it remembers you across sessions — your preferences, projects, and past conversations. - **Multi-platform gateway** lets you talk to Hermes from Telegram, Discord, Slack, WhatsApp, Signal, and 15+ other platforms — all from a single process. - **Cron jobs, subagent delegation, and MCP support** make it a true automation platform, not just a

chat interface. - **It's open source (MIT)** — self-hosted, no data leaves your machine, no vendor lock-in, no paid tiers. - **A vibrant ecosystem** surrounds it: GitHub, Discord community, Skills Hub, and comprehensive documentation. Quiz Questions Test your understanding: 1. **What makes Hermes Agent fundamentally different from ChatGPT or Claude?** - a) It has a better web interface - b) It's a terminal-native agent with tool calling, persistent memory, and autonomous execution - c) It only works with Nous Research models - d) It requires a GPU to run 2. **What does "provider-agnostic" mean in the context of Hermes Agent?** - a) It only uses one provider - b) It can switch between 20+ LLM providers with no code changes - c) It doesn't need any API keys - d) It only works with open-source models 3. **How does the skills system keep Hermes from getting too expensive in token usage?** - a) It caps the number of skills at 5 - b) It uses progressive disclosure — only loads skill content when needed - c) It requires a subscription - d) Skills are always loaded at startup 4. **Which of these is NOT a supported integration for the multi-platform gateway?** - a) Telegram - b) Discord - c) Snapchat - d) Home Assistant 5. **What license is Hermes Agent released under?** - a) GPLv3 - b) Apache 2.0 - c) MIT - d) Proprietary **Answers:** 1-b, 2-b, 3-b, 4-c, 5-c *Next Module: [Module 2: Installation & First Run](./module-02-installation.md)* Dashboard Module 2

Installation & First Run

← Back to Course Dashboard Installation & First Run Module 2 of 10 ■ Full Walkthrough Video ■ Venice.ai Get Private API Key → No data stored. Stake DM token for $1/day API credits. ■■ Hostinger VPS $12/mo VPS Deal → Run Hermes 24/7. Referral code applied. ■ Linux / macOS / WSL2 curl -fsSL https://raw.githubusercontent.com/NousResearch/hermes-agent/main/scripts/install.sh | bash ← Click to copy ■ Windows (PowerShell) iex (irm https://raw.githubusercontent.com/NousResearch/hermes-agent/main/scripts/install.ps1) ■■ Early Beta — details in the module ■ Android (Termux) Same Linux one-liner — auto-detected ■ Installer detects Termux automatically ← Module 1 All Modules Next: Configuration Deep Dive → **Course:** Hermes Agent — From Zero to Autonomous Agent **Module:** 2 of 8 **Reading time:** ~20 minutes **Difficulty:** Beginner — no prior Hermes experience needed, basic terminal familiarity helpful TL;DR **One-liner install** `curl -fsSL https://raw.githubusercontent.com/NousResearch/hermes-agent/main/scripts/install.sh \ bash` **Prerequisites** Git + a Linux/macOS/WSL2 system. The installer handles Python, Node.js, ripgrep, and ffmpeg automatically. **After install** `source ~/.bashrc && hermes` **First run** Setup wizard walks you through choosing a provider and getting API keys **Time to first chat** Under 2 minutes **Docker alternative** `docker run -it ghcr.io/nousresearch/hermes-agent` **Got stuck?** `hermes doctor` diagnoses everything Before You Start Supported Platforms Platform Status Notes **Linux** (x86_64, arm64) ■ Production-ready Ubuntu, Debian, Fedora, Arch, openSUSE all tested **macOS** (Intel, Apple Silicon) ■ Production-ready Intel and M-series both supported **WSL2** (Windows) ■ Production-ready Use Ubuntu from Microsoft Store, then install as Linux **Windows Native** (PowerShell) ■■ Early Beta Works but fewer battle-tested paths **Termux** (Android) ■ Supported Auto-detected by installer **NixOS** ■ Supported Dedicated flake and NixOS module Prerequisites **Absolute minimum:** - **Git** — run `git --version` to check. If missing: - **Ubuntu/Debian:** `sudo apt install git` - **macOS:** `xcode-select --install` or `brew install git` - **Fedora:** `sudo dnf install git` - **Windows WSL:** `sudo apt install git` (inside WSL) **The installer handles everything else automatically** — you don't need to pre-install Python, Node.js, ripgrep, or ffmpeg. It detects what's missing and provisions it. **Optional but nice to have:** - A terminal emulator that supports true color (kitty, iTerm2, Windows Terminal, GNOME Terminal) - `curl` (most systems have it; install with `sudo apt install curl` if missing) Getting an API Key (Do This Now) You'll need an API key from an LLM provider. For beginners, **OpenRouter** is the best starting point — one API key gives access to 200+ models including GPT-4o, Claude, Gemini, and open-source models. 1. Go to openrouter.ai/keys 2. Sign up (free tier available with initial credits) 3. Create an API key 4. Copy it somewhere safe — you'll paste it into the setup wizard **Pro tip:** OpenRouter's free tier gives you enough credits to explore for weeks. If you run out, top up with as little as $5 to access everything. Method 1: One-Line Installer (Recommended) This is the fastest and most battle-tested path. Works on **Linux, macOS,

WSL2, and Termux**. Open your terminal and run: curl -fsSL https://raw.githubusercontent.com/NousResearch/hermes-agent/main/scripts/install.sh | bash **What you'll see:** ■ Hermes Agent Setup → Checking for uv... ✓ uv found (0.6.x) → Creating virtual environment... → Installing hermes-agent and dependencies... → Installing Node.js 22... → Installing system tools (ripgrep, ffmpeg)... → Installing Playwright browsers for web automation... → Symlinking hermes command to ~/.local/bin/hermes... ✓ Installation complete! Run 'source ~/.bashrc' then 'hermes' to start chatting! **Time estimate:** 30-90 seconds on a decent internet connection. What the Installer Actually Does Let's demystify that one-liner. The install script: 1. **Detects your platform** — Linux, macOS, Termux, or other POSIX 2. **Installs `uv`** — a lightning-fast Python package manager (replaces pip) 3. **Clones the repo** to `~/.hermes/hermes-agent/` 4. **Creates a Python virtual environment** inside the repo (Python 3.11) 5. **Installs all dependencies** — Hermes core, browser automation, voice tools 6. **Installs Node.js v22** — needed for browser automation and the WhatsApp bridge 7. **Installs system tools** — `ripgrep` (fast file search) and `ffmpeg` (audio format conversion) 8. **Installs Playwright browsers** — headless Chromium for web automation 9. **Symlinks the `hermes` command** to `~/.local/bin/hermes` 10. **Updates your shell config** — adds `~/.local/bin` to PATH in `.bashrc` or `.zshrc` **Install layout after completion:** What Where Hermes code `~/.hermes/hermes-agent/` Virtual environment `~/.hermes/hermes-agent/venv/` `hermes` command `~/.local/bin/hermes` (symlink to venv) Your config & data `~/.hermes/` Skills `~/.hermes/skills/` Session history `~/.hermes/sessions/` Logs `~/.hermes/logs/` After Installation Reload your shell so the `hermes` command is found: source ~/.bashrc # if you use bash # OR source ~/.zshrc # if you use zsh Verify the installation: hermes --version # Expected output (version number will differ): # Hermes Agent 0.x.x Method 2: Docker Install Prefer containers? Hermes has an official Docker image. Good for isolated environments, CI/CD pipelines, or if you don't want the installer touching your system. # Pull and run docker run -it --rm \ -v ~/.hermes:/root/.hermes \ -e OPENROUTER_API_KEY=your_key_here \ ghcr.io/nousresearch/hermes-agent **Breaking this down:** - `-it` — interactive terminal mode - `--rm` — remove container when done - `-v ~/.hermes:/root/.hermes` — persist your config and sessions on the host - `-e OPENROUTER_API_KEY=...` — pass your API key as an environment variable For a persistent setup (recommended), create a data directory first: mkdir -p ~/.hermes-docker docker run -it --rm \ -v ~/.hermes-docker:/root/.hermes \ -e OPENROUTER_API_KEY=your_key_here \ ghcr.io/nousresearch/hermes-agent **Note:** The Docker image is built from the `main` branch. For a pinned version, use a specific tag like `ghcr.io/nousresearch/hermes-agent:v0.x.x`. Method 3: Manual Git Clone Install For developers, contributors, or anyone who wants full control: # 1. Clone the repo git clone https://github.com/NousResearch/hermes-agent.git cd hermes-agent # 2. Run the setup script ./setup-hermes.sh The `setup-hermes.sh` script does the same work as the one-liner installer — installs `uv`, creates a venv, installs dependencies, and symlinks the `hermes` command. The difference is you have the repo on disk, so you can: - Switch branches (`git checkout develop`) - Make local changes - Run the test suite (`scripts/run_tests.sh`) - Stay on a specific version Windows-Specific Setup Option A: WSL2 (Easiest, Most Stable) 1. **Install WSL2** — Open PowerShell as Administrator and run: ```powershell wsl --install ``` This installs Ubuntu by default. Restart when prompted. 2. **Launch Ubuntu** from the Start Menu. On first boot, set up a Linux username and password. 3. **Update packages:** ```bash sudo apt update && sudo apt upgrade -y ``` 4. **Install Git:** ```bash sudo apt install git curl -y ``` 5. **Run the one-liner installer:** ```bash curl -fsSL https://raw.githubusercontent.com/NousResearch/hermes-agent/main/scripts/install.sh | bash ``` 6. **Reload and launch:** ```bash source ~/.bashrc hermes ``` Option B: Native Windows (Early Beta) Open **PowerShell** (not CMD) and run: iex (irm https://raw.githubusercontent.com/NousResearch/hermes-agent/main/scripts/install.ps1) **What the PowerShell installer does:** 1. Installs `uv` for Python package management 2. Clones the repo to `%LOCALAPPDATA%\hermes\hermes-agent` 3. Creates a virtual environment 4. Installs a **portable Git Bash** (PortableGit) — no admin rights needed, completely isolated from any system Git install 5. Installs Node.js 22, ripgrep, ffmpeg 6. Adds `hermes` to your User PATH **After install:** Open a **new PowerShell window** (PATH updates need a fresh session) and run `hermes`. **Heads up:** The browser-based dashboard chat pane requires WSL2 on Windows. Everything else — CLI, gateway, cron, browser tool, MCP servers — runs natively.

First Launch: The Setup Wizard This is where the magic happens. Run: hermes The first time you launch, you'll be greeted by an interactive **setup wizard**. It looks something like this: ■■■ ■ ■ Hermes Agent Setup Wizard ■ ■ Let's get you started! ■ ■■■ I'll help you configure: • Your LLM provider and model • Your API credentials • Tool preferences • Platform integrations (optional) Let's begin! → Which provider would you like to use? 1) OpenRouter (recommended — 200+ models, one API key) 2) OpenAI (GPT-4o, o1, o3-mini) 3) Anthropic (Claude Opus 4, Sonnet 4) 4) Nous Portal (Nous Research endpoint) 5) Local / Other (Ollama, vLLM, custom endpoint) > 1 Step-by-Step Wizard Walkthrough **Step 1: Choose a provider.** Pick **OpenRouter** (option 1) for the easiest start. It gives you access to every major model through a single API key. **Step 2: Enter your API key.** Paste the key you created earlier. It gets saved to `~/.hermes/.env`, not your shell history. **Step 3: Choose a model.** OpenRouter offers 200+ models. Good defaults: Model Best for Cost `openrouter:openai/gpt-4o` General purpose, writing, reasoning Low `openrouter:anthropic/claude-sonnet-4` Coding, complex tasks Medium `openrouter:google/gemini-2.5-pro` Long context, research Low `openrouter:meta-llama/llama-4-maverick` Open model, fast Very low `openrouter:qwen/qwen-2.5-72b-instruct` Strong open model Very low For beginners, **`openrouter:openai/gpt-4o`** is a solid default — fast, capable, and affordable. **Step 4: Configure tools.** The wizard asks which tools you want enabled: - **Terminal** (run shell commands) — **Recommended: Yes** - **File system** (read/write files) — **Recommended: Yes** - **Web search** — **Recommended: Yes** - **Browser automation** — Optional, can enable later - **Vision** — Optional, for image analysis **Step 5: Messaging platforms (optional).** You can set up Telegram, Discord, or Slack now, or skip and configure later with `hermes gateway setup`. When the wizard finishes, you'll see: ✓ Setup complete! Your configuration has been saved. You can change any of these settings later with: hermes model — Change provider or model hermes tools — Enable/disable tools hermes config — View/edit all settings hermes gateway — Set up messaging platforms Ready to chat! Type your message below or /help for commands. Your First Chat After the wizard, you're dropped into the Hermes CLI. You'll see a `>` prompt. Try these: > Hello! What can you do? Hermes will respond with a friendly introduction and a list of capabilities. You'll see the **KawaiiSpinner** — an animated face (`■■■`, `■■■■■`, etc.) that shows the agent is thinking. Asking Questions > What time is it right now? This will trigger Hermes to use its **date/time tool** — you'll see the tool call in the output: ■■■ Thinking... ■ Calling tool: get_current_date The current date and time is Saturday, May 23, 2026, 10:15:42 AM UTC. Seeing Tool Calls in Action This is where Hermes separates from vanilla chatbots. Try: > Check my disk usage You'll see Hermes: 1. **Decide** what tool to use 2. **Call** the terminal tool with `df -h` 3. **Read** the output 4. **Format** a readable response Example output: ■■■■■ Working... ■ Running: df -h ■ Disk Usage Summary: Filesystem Size Used Avail Use% /dev/sda1 50G 23G 27G 46% tmpfs 2.0G 1.2M 2.0G 1% Your root partition is 46% full — plenty of room. Try more: > Create a new directory called test-project and a file called notes.txt > Search the web for "latest AI agent frameworks 2026" > What's in my current directory? > Write a Python script that prints the Fibonacci sequence Each command will show tool calls in real-time — you can watch Hermes think, act, and iterate. Understanding the Interface ■■■ Thinking... ← The spinner shows the agent is processing ■ Calling tool: run_terminal ← Every tool call is shown ■ Running: python3 script.py ← The actual command being executed ✓ Command completed (exit: 0) ← Status of the tool call Here's the result... ← The agent's natural language response Key interface elements: - **`>` prompt** — Type your message here - **Slash commands** — Type `/` to see available commands (`/help`, `/new`, `/model`, `/skills`, etc.) - **Tool call output** — Every tool the agent uses is shown with `■` prefix - **Tab completion** — Press Tab to autocomplete slash commands - **`Ctrl+C`** — Interrupt the current action - **`Ctrl+L`** — Clear the screen - **Up/Down arrows** — Navigate conversation history - **`/new` or `/reset`** — Start a fresh conversation Troubleshooting: `hermes doctor` If something goes wrong, run: hermes doctor This runs a comprehensive diagnostic check: ■ Hermes Doctor System: ✓ OS: Linux 6.8.0-110-generic (x86_64) ✓ Shell: bash 5.2.21 Installation: ✓ Hermes path: /root/.local/bin/hermes ✓ Hermes home: /root/.hermes ✓ Git: available ✓ Install method: git installer Python:

✓ Python 3.11.9 (via uv) ✓ Virtual environment: active Dependencies: ✓ uv: 0.6.x ✓ Node.js: v22.x ✓ ripgrep: 14.x ✓ ffmpeg: 7.x ✓ Playwright browsers: installed Configuration: ✓ Config file: /root/.hermes/config.yaml ✓ API keys: OPENROUTER_API_KEY configured ✓ Provider: openrouter ✓ Model: openai/gpt-4o ✗ Telegram: not configured ✗ Discord: not configured ✓ 2 checks passed, 2 optional checks skipped. `hermes doctor` checks for: - Missing dependencies - Invalid configuration - Unset API keys - Broken symlinks - Permission issues - Python version compatibility Each issue comes with a suggested fix command. Common Issues & Fixes **`hermes: command not found`** - **Fix:** Run `source ~/.bashrc` (or `source ~/.zshrc`), then try again. - **If it persists:** Your `~/.local/bin` may not be in PATH. Add this to your shell config: ```bash export PATH="$HOME/.local/bin:$PATH" ``` **Setup wizard doesn't start — blank screen or hangs** - **Fix:** Your terminal might not be fully interactive. Try: ```bash hermes setup ``` This launches the wizard directly. - **If that also hangs:** Try the non-interactive config path: ```bash hermes config set provider openrouter hermes config set OPENROUTER_API_KEY your_key_here hermes config set model openrouter:openai/gpt-4o hermes chat ``` **`ModuleNotFoundError: No module named 'dotenv'`** - **Fix:** You're invoking the repo source file directly with system Python instead of the venv launcher. Make sure you're running `~/.local/bin/hermes` (or just `hermes` after `source ~/.bashrc`), not `~/hermes-agent/hermes`. **`API key not set`** - **Fix:** Run `hermes model` to configure your provider, or: ```bash hermes config set OPENROUTER_API_KEY your_key_here ``` **`Permission denied` when running `hermes`** - **Fix:** The installer needs write access to `~/.local/bin/`. If you installed as root, the binary might be in `/usr/local/bin/`. Try: ```bash sudo ln -s ~/.hermes/hermes-agent/venv/bin/hermes /usr/local/bin/hermes ``` **Slow first launch after install** - **Normal:** The first launch triggers Playwright browser download (~100MB) if the installer skipped it. Subsequent launches are instant. **`curl: command not found` during install** - **Fix:** Install curl first: ```bash sudo apt install curl -y # Debian/Ubuntu sudo dnf install curl -y # Fedora ``` **Git not found during install** - **Fix:** ```bash sudo apt install git -y # Debian/Ubuntu sudo dnf install git -y # Fedora brew install git # macOS ``` **Docker: `ghcr.io/nousresearch/hermes-agent: not found`** - **Fix:** First pull the image: ```bash docker pull ghcr.io/nousresearch/hermes-agent ``` Then run. If pulling fails, you may need to authenticate with GitHub Container Registry: ```bash echo $GITHUB_TOKEN | docker login ghcr.io -u your-username --password-stdin ``` **WSL: `hermes` command works in Ubuntu but not in PowerShell** - **Expected.** WSL is a separate Linux environment. `hermes` only works inside the WSL terminal window. Use the native Windows installer if you need PowerShell access. Next Steps After First Chat You're up and running. Here's what to try next: 1. **`hermes model`** — Experiment with different models (Claude for coding, Gemini for long context, open models for speed) 2. **`hermes tools`** — Browse available tools and enable/disable them per platform 3. **`/skills`** — See what skills are installed and available 4. **`hermes gateway setup`** — Connect Telegram or Discord so you can chat from your phone 5. **`hermes cron`** — Schedule your first automated task 6. **`/help`** — Full list of slash commands Key Takeaways - **One command** is all it takes: `curl -fsSL https://raw.githubusercontent.com/NousResearch/hermes-agent/main/scripts/install.sh | bash` - **The installer handles everything** — Git is the only prerequisite; Python, Node.js, ripgrep, and ffmpeg are auto-installed - **Docker alternative** available for containerized deployments - **WSL2** is the most stable Windows path; native Windows PowerShell install is in early beta - **First launch triggers the setup wizard** — choose OpenRouter for the simplest start - **Watch tool calls happen live** — Hermes shows every command it runs and tool it uses - **`hermes doctor`** diagnoses any issues with clear, actionable fixes - **From install to first chat is under 2 minutes** on a decent internet connection Quiz Questions 1. **What is the only prerequisite for the one-line installer on Linux/macOS?** - a) Python 3.11 - b) Git - c) Docker - d) Node.js 2. **Where does the installer put the `hermes` command?** - a) `/usr/bin/hermes` - b) `~/hermes-agent/hermes` - c) `~/.local/bin/hermes` (symlink to the venv) - d) `/opt/hermes/hermes` 3. **What provider is recommended for beginners during the setup wizard?** - a) OpenAI - b) Anthropic - c) OpenRouter - d) Nous Portal 4. **What command do you run if you encounter issues after installation?** - a) `hermes check` - b) `hermes diagnose` - c) `hermes doctor` - d) `hermes status` 5. **Which of these is NOT a step you should take after installation?** - a) `source ~/.bashrc` - b) `hermes` to launch the setup wizard - c) Installing Python manually before running the installer - d) Pasting your API key when prompted **Answers:** 1-b, 2-c, 3-c, 4-c, 5-c *Next Module: Module 3 —

Configuration & Customization (coming soon)* ← Module 1 Dashboard Module 3 →

Configuration Deep Dive

Back to Course Dashboard Configuration Deep Dive Module 3 of 10 ■ Full Walkthrough Video Your browser doesn't support video playback. Installation & First Run All Modules Next: Daily Usage & Power Featu **Duration:** ~45 minutes reading **Prerequisites:** Module 1 (Installation) & Module 2 (Quickstart) **Goal:** Master every configuration mechanism in Hermes Agent — from API keys to credential pools to security profiles. 3.1 Where Config Lives Hermes Agent keeps all configuration in two files under `~/.hermes/` (or wherever `$HERMES_HOME` points): File Purpose Security Level `~/.hermes/config.yaml` Behavioral settings, model preferences, tool config, display options Safe to share; no secrets `~/.hermes/.env` API keys and sensitive credentials **Never share** — treat like SSH keys This split means you can version-control your `config.yaml` or share it between machines without accidentally leaking API keys. The Config Hierarchy Hermes loads settings in this order (last wins): 1. **Hardcoded defaults** — built into the CLI 2. **`~/.hermes/config.yaml`** — your personal overrides 3. **`./cli-config.yaml`** — project-level config, useful for team repos 4. **Environment variables** — override everything at runtime (e.g., `HERMES_YOLO_MODE=1`) Environment variables are documented at hermes-agent.nousresearch.com/docs/reference/environment-variables. Editing Your Config You have two options: # Quick method — opens in your $EDITOR hermes config edit # Programmatic method — set individual values hermes config set model.default "openrouter/anthropic/claude-sonnet-4-20250514" hermes config set agent.max_turns 120 # View current config (merged from all sources) hermes config Manual YAML editing (`vim ~/.hermes/config.yaml`) gives you the most control and is perfectly safe — the parser is lenient and validates on load. 3.2 Model Provider Setup Hermes supports every major LLM provider through a unified plugin system. Each provider is a separate plugin under `plugins/model-providers/`. Provider Comparison Table Provider Plugin Name Auth Method Env Var(s) API Docs **OpenRouter** `openrouter` API key `OPENROUTER_API_KEY` openrouter.ai/keys **Anthropic** `anthropic` API key, OAuth `ANTHROPIC_API_KEY`, `ANTHROPIC_TOKEN` console.anthropic.com **OpenAI (Chat)** (built-in) API key `OPENAI_API_KEY` platform.openai.com/api-keys **OpenAI Codex** `openai-codex` OAuth (Plus/Pro) (none — OAuth flow) Requires ChatGPT Plus/Pro **DeepSeek** `deepseek` API key `DEEPSEEK_API_KEY` platform.deepseek.com/api_keys **Google Gemini** `gemini` API key (AI Studio) `GOOGLE_API_KEY`, `GEMINI_API_KEY` aistudio.google.com/apikey **Nous (Portal)** `nous` OAuth or API key `NOUS_API_KEY` portal.nousresearch.com **xAI** `xai` API key `XAI_API_KEY` console.x.ai **Z.AI (GLM)** `zai` API key `ZHIPUAI_API_KEY` open.bigmodel.cn **AWS Bedrock** `bedrock` IAM credentials `AWS_ACCESS_KEY_ID` + `AWS_SECRET_ACCESS_KEY` aws.amazon.com/bedrock **Hugging Face** `huggingface` API key `HUGGINGFACE_API_KEY` huggingface.co/settings/tokens **NovitaAI** `novita` API key `NOVITA_API_KEY` novita.ai **NVIDIA NIM** `nvidia` API key `NVIDIA_API_KEY` build.nvidia.com **Ollama (Cloud)** `ollama-cloud` API key `OLLAMA_API_KEY` ollama.cloud **Alibaba** `alibaba` API key `ALIBABA_API_KEY` bailian.console.aliyun.com **Kimi/Moonshot** `kimi-coding` API key `MOONSHOT_API_KEY` platform.moonshot.cn **MiniMax** `minimax` API key `MINIMAX_API_KEY` platform.minimaxi.com **Xiaomi MiMo** `xiaomi` API key `MIMO_API_KEY` platform.xiaomimimo.com **Stepfun** `stepfun` API key `STEPFUN_API_KEY` stepfun.com **Arcee** `arcee` API key `ARCEE_API_KEY` arcee.ai **Vercel AI Gateway** `ai-gateway` API key `AI_GATEWAY_API_KEY` vercel.com/ai-gateway **Custom (any)** `custom` API key `OPENAI_API_KEY`

+ set `base_url` Your own endpoint Quick Setup: OpenRouter (Recommended Starting Point) OpenRouter is the easiest way to start — 200+ models with a single API key. 1. Go to openrouter.ai/keys and create a key 2. Add it to `~/.hermes/.env`: ```bash OPENROUTER_API_KEY=sk-or-v1-your-key-here ``` 3. Run `hermes model` and select OpenRouter → any model you like Setting Up Other Providers **Anthropic / Claude:** # In ~/.hermes/.env: ANTHROPIC_API_KEY=sk-ant-your-key-here Then `hermes model` → pick Claude Sonnet 4 or Haiku 3.5. **OpenAI:** # In ~/.hermes/.env: OPENAI_API_KEY=sk-proj-your-key-here Then `hermes model` → pick GPT-4o or o3-mini. **DeepSeek:** # In ~/.hermes/.env: DEEPSEEK_API_KEY=sk-your-key-here Then `hermes model` → pick `deepseek/deepseek-chat-v3-0324`. **Google Gemini:** # In ~/.hermes/.env: GOOGLE_API_KEY=AIza-your-key-here Then `hermes model` → pick `gemini/gemini-3-flash-preview`. **Local Models (via Custom Provider):** # In ~/.hermes/.env (if auth needed): OPENAI_API_KEY=lm-studio # Or set OPENAI_API_KEY=not-needed for unauthenticated endpoints # In ~/.hermes/config.yaml: model: default: "local-model-name" provider: "custom" base_url: "http://localhost:1234/v1" # LM Studio, Ollama, vLLM, etc. The `hermes model` Command The fastest way to configure a provider is the interactive model picker: hermes model This presents a paginated, searchable list of all models from all configured providers. It detects which API keys you have set and only shows providers with valid credentials. You can also set the model programmatically: hermes model "openrouter/anthropic/claude-sonnet-4-20250514" hermes model "gpt-4o" --provider openai 3.3 Credential Pools A credential pool lets you register **multiple API keys for the same provider** for automatic failover or load balancing. This is essential for production use where rate limits or quota exhaustion can interrupt work. Pool Strategies Strategy Behavior Best For `fill_first` Use first key until exhausted, then rotate Simple redundancy `round_robin` Cycle through keys evenly Load balancing multiple keys `random` Pick a random key each time Spreading usage across plans `least_used` Track API call counts per key, use the least-loaded Maximizing remaining quota Managing Credential Pools # Interactive pool management hermes auth # Add a second OpenRouter key hermes auth pool add openrouter --key "sk-or-v1-backup-key" hermes auth pool strategy openrouter round_robin # List all credentials hermes auth pool list # Remove a pool entry hermes auth pool remove openrouter --index 1 The credential pool is persisted in the encrypted auth store (`~/.hermes/auth.json`), not in plaintext `.env` files. When a request to a provider fails with a 401, 429, or 402 status, Hermes automatically marks that credential as exhausted and rotates to the next one in the pool. 3.4 Toolsets: What They Are and How to Configure A **toolset** is a named group of tools that you can enable or disable as a unit. Hermes ships with ~40+ tools organized into ~30 named toolsets. Core Toolsets Toolset Description Default? `web` Web search (`web_search`, `web_extract`) ■ Always `terminal` Shell command execution (`terminal`, `process`) ■ Always `file` File read/write/patch/search ■ Always `browser` Full web browser automation ■ Off by default `vision` Image analysis ■ Always `image_gen` Image generation ■ Off by default `code_execution` Run Python scripts that call tools via RPC ■ Always `skills` Create, view, and manage skills ■ Always `memory` Persistent cross-session memory ■ Always `session_search` Search past conversations ■ Always `delegation` Spawn subagents for parallel work ■ Always `cronjob` Schedule recurring tasks ■ Off by default (CLI only) `clarify` Ask you clarifying questions ■ Always `todo` Task planning and tracking ■ Always `tts` Text-to-speech output ■ Off by default `moa` Mixture of Agents (advanced reasoning) ■ Off by default `homeassistant` Smart home control ■ Requires HASS_TOKEN `kanban` Multi-agent coordination board ■ Off by default `search` Web search only (no scraping) ■ Off by default `messaging` Cross-platform message sending ■ Always (gated on gateway) `computer_use` macOS desktop control ■ Requires cua-driver `video` Video analysis ■ Off by default `video_gen` Video generation ■ Off by default Enabling and Disabling Toolsets # Interactive tool management (recommended) hermes tools # Or via CLI commands hermes tools enable browser hermes tools disable code_execution # List all toolsets and their status hermes tools list The `hermes tools` command opens an interactive TUI where you can toggle each toolset on or off, see which tools belong to each set, and preview which models support all your active tools. How Toolsets Affect Conversation Each toolset adds its tools to the LLM's function-calling schema. More tools = larger prompt = more tokens consumed. Best practice: - **Start conservative** — use the defaults - **Enable toolsets reactively** — when you need browser

automation, enable `browser` - **Disable what you don't need** — `hermes tools disable moa` saves ~300 tokens/turn - **Create composite toolsets** — toolsets can include other toolsets (e.g., `debugging` includes `web`, `file`, and `terminal`) Platform-Specific Toolsets Each platform has its own toolset that inherits from the core tools: - `hermes-cli` — full interactive CLI - `hermes-telegram` — Telegram bot (terminal has safety checks) - `hermes-discord` — Discord bot (+ Discord-specific tools) - `hermes-slack` — Slack bot - `hermes-whatsapp` — WhatsApp bot - `hermes-cron` — Cron scheduler agent You configure each independently. Browser automation might be fine in the CLI but disabled on Telegram. Custom Toolsets You can define custom toolset compositions in `config.yaml`: toolsets: custom: my-workflow: description: "Tools I need for daily dev work" tools: - web_search - terminal - read_file - write_file includes: - vision 3.5 Profiles Profiles let you run **multiple independent Hermes instances** on the same machine, each with its own config, API keys, memory, skills, sessions, and gateway. When to Use Profiles - **Work/Personal separation** — different models, different API keys, different skills - **Multi-tenant servers** — each user gets their own profile - **Kanban workers** — specialized profiles for different task categories - **Testing** — a "default" profile for daily use, an "experimental" profile for new models Profile Architecture Profiles live under `~/.hermes/profiles/ /`. Each profile is a complete `HERMES_HOME`: ~/.hermes/profiles/coder/ ■■■ config.yaml # Independent config ■■■ .env # Independent API keys ■■■ sessions/ # Separate conversation history ■■■ skills/ # Different skills ■■■ memories/ # Separate memory ■■■ logs/ # Separate logs ■■■ cron/ # Separate scheduled jobs ■■■ home/ # Isolated subprocess HOME (git config, ssh, etc.) Profile Commands # Create a new profile hermes profile create coder # Creates ~/.hermes/profiles/coder/ with bundled skills seeded # Create with cloned config from the default profile hermes profile create work --clone --clone-all # List all profiles hermes profile list # Switch to a profile hermes profile use coder # This writes "coder" to ~/.hermes/active_profile # All future `hermes` commands use the coder profile # Use a profile for a single command hermes -p coder chat -q "What's my schedule?" # Profiles also get shell aliases: # After `hermes profile create coder`, you can run: coder chat # Delete a profile (irreversible — removes all data!) hermes profile delete coder Profile Isolation - **Gateway** — each profile can run its own gateway on different ports - **Sessions** — completely separate — `hermes sessions list` only shows the active profile's sessions - **Memory** — curated memory is per-profile - **Subprocesses** — the `home/` directory isolates git, ssh, and npm configs per profile 3.6 Memory Backends Hermes has three memory backends with different tradeoffs. Built-in Memory (Default) - **Storage:** `~/.hermes/memories/MEMORY.md` + `USER.md` - **Mechanism:** The agent periodically writes curated knowledge to Markdown files. On session start, these files are loaded as system context. - **Pros:** Zero setup, works offline, fully transparent (readable Markdown) - **Cons:** Simple key-value, no semantic search or deduplication **Config:** memory: provider: "default" auto_save: true # Auto-persist after conversations auto_save_threshold: 5 # Turns before prompting for memory Honcho (Dialectic Memory) - **Storage:** Honcho server (local or cloud) - **Mechanism:** Maintains a dialectic user model — the agent updates its understanding of you with each conversation - **Pros:** Sophisticated user modeling, web dashboard, good for chatbot use cases - **Cons:** Requires Honcho server running **Setup:** `hermes honcho start` or configure Honcho cloud. Config in `config.yaml`: memory: provider: "honcho" honcho: base_url: "http://localhost:8000" token: "your-honcho-token" Mem0 - **Storage:** Mem0 Platform API - **Mechanism:** Automatic semantic deduplication and relevance scoring - **Pros:** Smart deduplication, good recall for large knowledge bases - **Cons:** Paid API, internet connection required **Setup:** # In ~/.hermes/.env: MEM0_API_KEY=m0-your-key Or via `~/.hermes/mem0.json`: { "api_key": "m0-your-key", "api_type": "platform", "user_id": "my-user" } 3.7 TTS & STT Setup for Voice Mode Voice mode lets you speak to Hermes and hear spoken responses. Text-to-Speech (TTS) — Hermes Talks Provider Quality Cost Config Key **Edge TTS** Good (free) Free `edge` (default) **OpenAI TTS** Excellent Paid (per char) `openai` **ElevenLabs** Excellent Paid (per char) `elevenlabs` **xAI TTS** Good Paid `xai` **Gemini TTS** Good Free tier `gemini` **Piper TTS** Decent Free (local) `piper` **KittenTTS** Good Free (local) `kittentts` # In config.yaml: tts: provider: edge # Default: free, no API key needed voice: en-US-JennyNeural # Voice name (provider-specific) For OpenAI TTS, set `OPENAI_API_KEY` or `VOICE_TOOLS_OPENAI_KEY` in `.env`. Speech-to-Text (STT) — Hermes Listens Provider Quality Cost Setup **Faster-Whisper** Good Free (local) `pip install faster-whisper`

Groq Whisper Excellent Free tier Set `GROQ_API_KEY` **OpenAI Whisper** Excellent Paid Set `OPENAI_API_KEY` # In config.yaml: stt: provider: faster-whisper # Or: groq, openai model: base.en # Model size (faster-whisper only) language: en # Language hint **Activating voice mode:** # Inside a chat session /voice on # Or start with voice hermes --voice 3.8 Security & Approval Modes Hermes has a layered security system to prevent accidental dangerous operations. Approval Modes Controls whether the agent must ask for permission before running dangerous commands. # In config.yaml: security: approval_mode: smart # Default: "smart" — let the LLM decide Mode Behavior When to Use `manual` Always prompt for dangerous commands Maximum safety, shared systems `smart` Auxiliary LLM auto-approves low-risk commands; prompts for risky ones Best balance (default) `off` Never prompt for approval Trusted environments, automation What Counts as "Dangerous" The approval system checks `terminal` tool commands against a pattern database. Examples that trigger approval: - `rm -rf`, `dd`, `mkfs`, `fdisk` — destructive filesystem operations - `curl ... | bash` — remote code execution - `sudo`, `su`, `chmod 777` — privilege escalation - `kill -9`, `pkill` — process termination - `pip install`, `npm install` — package installation (can be configured per-package) YOLO Mode Sometimes you want zero friction. YOLO mode bypasses **all** approval prompts: # Inside a session /yolo # Toggle ON /yolo # Toggle OFF # Via env var HERMES_YOLO_MODE=1 hermes chat When YOLO is active, the status bar shows a red ■ YOLO indicator. Allowlisting Commands You can permanently allow specific commands in `config.yaml`: security: approval_allowlist: - "apt update" - "pip install requests" - "git status" # Or via pattern approval_allowlist_patterns: - "curl https://.*" Secret Redaction API keys and sensitive values in `.env` are automatically redacted from logs and conversation history. The `_secret_redact` utility strips patterns matching known API key formats from all agent outputs. Subagent Auto-Approval For automated workflows (cron jobs, batch runs), subagents can be configured to auto-approve dangerous commands: delegation: auto_approve: true # Subagents bypass approval prompts 3.9 Full Config Reference Top-Level Keys Key Type Default Description `model` dict/string `""` Model configuration (see below) `provider` string `"auto"` Legacy root-level provider fallback `base_url` string `""` Legacy root-level base URL fallback `agent` dict See below Agent behavior `display` dict See below CLI/UI display preferences `terminal` dict See below Terminal backend configuration `browser` dict See below Browser automation `compression` dict See below Context compression `delegation` dict See below Subagent settings `code_execution` dict See below Sandboxed code execution `auxiliary` dict See below Auxiliary model routing `clarify` dict See below Clarifying questions `security` dict See below Approval & security `memory` dict See below Memory backend `tts` dict See below Text-to-speech `stt` dict See below Speech-to-text `voice` dict See below Voice mode settings `toolsets` dict `{}` Custom toolsets `credential_pool_strategies` dict `{}` Per-provider pool strategy `network` dict See below Network preferences `onboarding` dict `{}` First-run hints `model` Section model: default: "openrouter/anthropic/claude-sonnet-4-20250514" provider: "openrouter" # Provider plugin name base_url: "" # Custom endpoint override `agent` Section agent: max_turns: 90 # Max tool-calling iterations verbose: false # Show all tool I/O system_prompt: "" # Custom system prompt override prefill_messages_file: "" # File with initial messages reasoning_effort: "" # none/minimal/low/medium/high/xhigh service_tier: "" # OpenAI Priority tier personalities: # Built-in personalities helpful: "You are a helpful, friendly AI assistant." concise: "You are a concise assistant..." # ... (10+ built-in personalities) `display` Section display: compact: false # Minimal output mode resume_display: full # Show full or compact on session resume show_reasoning: false # Display <thinking> blocks streaming: true # Stream tokens in real-time busy_input_mode: interrupt # interrupt/queue/steer persistent_output: true # Persist output after new input persistent_output_max_lines: 200 skin: default # Theming skin `terminal` Section terminal: env_type: local # local/docker/ssh/modal/daytona/singularity cwd: "." # Working directory timeout: 60 # Per-command timeout (seconds) lifetime_seconds: 300 # Session max lifetime docker_image: "nikolaik/python-nodejs:python3.11-nodejs20" docker_volumes: [] # host:container mounts docker_mount_cwd_to_workspace: false `security` Section security: approval_mode: smart # manual/smart/off approval_allowlist: [] approval_allowlist_patterns: [] `compression` Section compression: enabled: true # Auto-compress near context limit threshold: 0.50 # Compress at 50% of context capacity `network` Section network: force_ipv4: false # Prefer IPv4 (fixes IPv6 timeout issues) 3.10 Recommended Config for Beginners

Here's a complete, safe starting config that works immediately after installation: # ~/.hermes/config.yaml — Recommended Beginner Config model: # Start with OpenRouter — one key for 200+ models default: "openrouter/anthropic/claude-sonnet-4-20250514" provider: "openrouter" agent: max_turns: 90 verbose: false reasoning_effort: "" # Let the model decide terminal: env_type: local cwd: "." # Your current directory timeout: 120 # 2 minutes per command security: approval_mode: smart # Safe default — LLM decides when to ask # Add safe commands to the allowlist: approval_allowlist: - "ls" - "cat" - "pwd" - "pip install" - "npm install" - "git status" - "git diff" display: streaming: true show_reasoning: false skin: default compression: enabled: true threshold: 0.50 memory: provider: "default" # Built-in — no setup needed tts: provider: edge # Free, no API key needed # Don't forget to add your API keys to ~/.hermes/.env: # OPENROUTER_API_KEY=sk-or-v1-... **Setup checklist:** 1. ■ Copy this config to `~/.hermes/config.yaml` 2. ■ Add API key to `~/.hermes/.env` 3. ■ Run `hermes model` to verify your provider 4. ■ Run `hermes tools` and disable any toolsets you don't need 5. ■ Run `hermes` and start chatting! Key Takeaways - **Two config files:** `config.yaml` for settings, `.env` for secrets - **25+ providers** supported through a unified plugin architecture - **Credential pools** provide automatic failover between multiple API keys for the same provider - **Toolsets** group tools logically — enable only what you need to save tokens - **Profiles** run completely isolated Hermes instances from one machine - **Memory backends** range from simple Markdown files to semantic Mem0 - **Security modes** let you balance friction vs. safety — from `manual` to `yolo` - **Config hierarchy** means environment variables always win for emergency overrides **Next:** Module 4 — Daily Usage & Power Features. Learn the slash commands, session management, cron jobs, delegation, checkpoints, and more. Module 2 Dashboard Module 4

Daily Usage & Power Features

Back to Course Dashboard Daily Usage & Power Features Module 4 of 10 ■ Full Walkthrough Video Your browser doesn't support video playback. Configuration Deep Dive All Modules Next: Skills System - The Kille **Duration:** ~50 minutes reading **Prerequisites:** Module 3 (Configuration) **Goal:** Master Hermes Agent as a daily driver — from interactive chat to cron jobs, delegation, and multi-agent orchestration. 4.1 Interactive Chat vs. Single Query Mode Hermes Agent has two main interaction modes. Interactive Chat (Default) hermes This launches the full terminal UI with: - A Rich-formatted banner showing the active model, provider, and profile - A persistent text input area at the bottom (prompt_toolkit) - Split-pane output with streaming responses - A status bar showing model, tokens, and session ID - Full tab-completion for slash commands Inside interactive mode, you type normally and Hit `Enter` to send. The agent can chain multiple tool calls — searching the web, reading files, writing code, running terminals — all in a single turn. Single Query Mode hermes chat -q "List all files in /root" # Or equivalently: hermes -q "List all files in /root" This sends one query, waits for the response, prints it, then exits. Use this for: - **Shell scripts:** `result=$(hermes chat -q "What's the weather?")` - **Cron jobs:** `hermes -p daily cron run daily-report` - **Batch processing:** Pipe queries from a file list or CI pipeline The single-query mode sets `HERMES_YOLO_MODE=1` internally so dangerous commands auto-approve — ideal for automation. # Named session (persists for follow-ups) hermes chat -q "Write a Python script" --continue my-session Key Flags Flag Effect `-q "..."` Single query mode `-p ` Use a specific profile `-w` Worktree mode (isolated git worktree) `--continue ` Resume a named session `--resume` Resume the last session `--ignore-user-config` Skip `config.yaml`, use defaults `--list-tools` Print all tool schemas and exit `--voice` Start with voice input enabled 4.2 Slash Commands Master List Inside an interactive session, everything starts with `/`. Here's every command organized by category. Session Management Command Aliases What It Does `/new [name]` `/reset` Start a completely fresh session `/retry` — Resend your last message to the agent `/undo` — Remove the last exchange (you + agent) `/title [name]` — Name the current session for resumption `/compress [topic]` — Manually trigger context compression `/save` — Dump the current conversation to a file `/history` — Show the full conversation so far `/resume [name]` — Jump back into a named session `/sessions` — Browse all past sessions `/branch [name]` `/fork` Fork the current conversation `/snap create` — Snapshot config/state (exportable) `/rollback [N]` — Restore filesystem to checkpoint N `/stop` — Kill all agent background processes `/agents` `/tasks` List active subagents and tasks `/background ` `/bg`, `/btw` Run a prompt in a background

thread `/queue ` `/q` Queue input for next turn `/steer ` — Inject a message after the next tool call `/goal [text/pause/resume/clear/status]` — Persistent cross-turn objective `/subgoal [text/remove N/clear]` — Extra criteria on the active goal `/status` — Session info (model, tokens, uptime) `/profile` — Show active profile name and path Configuration Command Aliases What It Does `/model [name]` `/provider` Switch provider or model mid-session `/personality [name]` — Set a predefined personality `/verbose` — Cycle tool progress verbosity `/yolo` — Toggle YOLO mode on/off `/reasoning [level]` — Set reasoning effort `/voice [on/off/tts]` — Toggle voice mode `/fast [normal/fast]` — Toggle priority processing (OpenAI/Anthropic) `/skin [name]` — Change the CLI theme `/indicator [style]` — Change the busy indicator animation Tools & Skills Command Aliases What It Does `/tools [list/enable/disable]` — Manage tools (CLI only) `/toolsets` — List available toolsets `/skills [search/inspect/install]` — Browse and manage skills `/bundles` — List skill bundles `/cron [subcommand]` — Manage scheduled tasks `/curator [run/pause/pin]` — Background skill maintenance `/kanban [subcommand]` — Multi-agent coordination board Information & Help Command What It Does `/help` Show all commands `/usage` Show token usage and cost `/insights [--days N]` Analyze recent usage patterns `/limits` Check per-provider rate limits `/whoami` Show your access level (admin/user) 4.3 Switching Models Mid-Session with `/model` One of Hermes's most powerful features: you can change models **without losing your conversation context**. # Inside a session, switch models mid-conversation: /model "gemini/gemini-3-flash-preview" # Switch provider entirely: /model --provider anthropic /model --provider openai # Set the new model as default (persists to config.yaml): /model "openrouter/anthropic/claude-sonnet-4" --global **Use cases:** - Start brainstorming with a cheap/fast model, then `/model` to a reasoning model for deep analysis - Hit a rate limit? `/model` to a different provider - Need vision? Switch to Gemini for image analysis, then back The full conversation history is preserved across model switches — only the inference backend changes. All active toolsets remain unchanged. 4.4 Context Compression Long conversations consume tokens and degrade quality. Hermes has two compression strategies. Automatic Compression (Default) # In config.yaml: compression: enabled: true threshold: 0.50 # Compress at 50% of the model's context limit When the conversation approaches the model's context window, Hermes automatically summarizes earlier turns into a compact form. This happens transparently — you just keep chatting. Manual Compression # Trigger compression immediately /compress # Compress around a specific topic /compress "focus on the database schema decisions" The `/compress` command: 1. Takes the conversation so far 2. Asks an auxiliary LLM to summarize it 3. Replaces the full history with the summary 4. Keeps the most recent N turns intact (so you don't lose the current thread) The `/usage` command shows your current context usage: /usage # Output: # Tokens: 12,847 / 200,000 (6.4%) # Cost this session: $0.04 4.5 Subagent Delegation The `delegate_task` tool spawns **isolated child agents** with their own context, tools, and terminal sessions. The parent blocks until all children complete and only sees their summary results — never intermediate tool calls. How Delegation Works You ■■> Hermes (parent) ■■> delegate_task("Write unit tests for parser.py") ■ ■■■■■■■■■■■■■■■■■■ ▼ ▼ Child Agent 1 Child Agent 2 (writes tests) (verifies they pass) ■ ■ ■■■■■■■■■■■■■■■■■■ ▼ Summary back to parent Configuring Delegation # In config.yaml: delegation: max_iterations: 45 # Max turns per child model: "" # Child model (empty = inherit parent) provider: "" # Child provider (empty = inherit parent) auto_approve: true # Auto-approve dangerous commands in children Using Delegate in Conversation You: Implement a REST API for the todo app. Delegate the database layer to one subagent and the route handlers to another. Hermes: I'll split this into parallel workstreams. ■ Delegating: Database schema & models... ■ Delegating: Route handlers & middleware... ■ Database layer complete (child: 8 turns) - Created models.py with SQLAlchemy schema - Added migration script ■ Route handlers complete (child: 12 turns) - Created routes.py with CRUD endpoints - Added input validation Both branches finished. Here's the full implementation... Batch Delegation The delegate tool also supports batch mode for processing multiple independent tasks in parallel — perfect for code review, file analysis, or bulk operations. # Behind the scenes, delegation uses ThreadPoolExecutor # with configurable max_workers (default: half your CPU cores) 4.6 Session Management Every conversation is a **session** — persisted to SQLite with full-text search. Naming and Resuming Sessions # Start a named session hermes chat --continue "code-review-session" # Inside the session, give it a title /title "Implementing the payment module" # Later, resume it hermes --continue

"Implementing the payment module" # Or via the session browser hermes sessions list hermes --continue <session-id> Session Persistence Sessions are stored in `~/.hermes/state.db` (SQLite with FTS5). Each session records: - Full message history - Model and provider used - Token usage and cost - Timestamps - Title (if set) Session Search The `/sessions` command opens an interactive browser: /sessions # Lists all sessions with: title, model, date, token count # Type a number to resume, or filter by date/model The `session_search` tool inside conversations lets the agent search past sessions to recall decisions, code patterns, or facts from weeks ago. Branching /branch "try-with-postgres" # Creates a fork of the current conversation # You now have two diverging threads from the same starting point 4.7 Cron Jobs Hermes has a built-in cron scheduler that runs inside the gateway. Jobs execute as agent conversations with full tool access. Creating Cron Jobs # Inside a session, ask the agent: "Create a daily summary cron job that runs at 8 AM" # Or use the cron tool directly: /cron create # Interactive prompt: # Name: daily-report # Schedule: 0 8 * * * (cron expression) # Prompt: "Summarize yesterday's activity in /root/logs" # Platform: cli (or telegram, discord, etc.) Cron Commands # List all jobs /cron list # Pause/resume /cron pause daily-report /cron resume daily-report # Run immediately (one-shot) /cron run daily-report # Edit a job's prompt or schedule /cron edit daily-report # Remove a job /cron remove daily-report Cron Schedule Format Standard cron expressions with 5 fields: ■■■■■■■■■■ minute (0-59) ■ ■■■■■■■■■■ hour (0-23) ■ ■ ■■■■■■■■■ day of month (1-31) ■ ■ ■ ■■■■■■■■ month (1-12) ■ ■ ■ ■ ■■■■■■ day of week (0-7, 0/7 = Sunday) ■ ■ ■ ■ ■ 0 8 * * 1-5 # Weekdays at 8 AM Cron Safety Features - **Prompt injection scanning** — every assembled job prompt is scanned for injection before execution - **Auto-approval** — cron agents run with auto-approved dangerous commands (set `delegation.auto_approve: false` to disable) - **Tool filtering** — cron uses the `hermes-cron` toolset, which you configure independently of the CLI toolset - **Output delivery** — cron results are sent to the configured platform (Telegram, email, Slack, etc.) Cron with Platforms hermes cron create \ --name "weekly-audit" \ --schedule "0 9 * * 1" \ --prompt "Audit /var/log for errors in the last week and summarize findings" \ --platform telegram # Deliver results to Telegram 4.8 Webhooks The webhook platform lets external systems trigger Hermes agents via HTTP POST requests. Setting Up Webhook Endpoints # Start the gateway with webhook support hermes gateway start # The webhook listener runs on the gateway port # Default: http://localhost:8290/webhook Webhook Payload Format POST /webhook HTTP/1.1 Content-Type: application/json { "message": "A new deployment is ready for review", "platform": "webhook", "metadata": { "source": "github-actions", "repo": "myorg/myapp", "commit": "abc123" } } Use Cases - **CI/CD pipelines** — trigger agent to review deployment output - **Monitoring alerts** — agent diagnoses and responds to system alerts - **GitHub webhooks** — agent reviews PRs, comments on issues - **IoT events** — agent processes sensor data and takes action # Example: GitHub Actions workflow curl -X POST https://hermes.myserver.com/webhook \ -H "Content-Type: application/json" \ -d '{"message": "Deploy to prod completed. Verify health endpoints.", "platform": "webhook"}' 4.9 Goal Mode Goals persist across conversation turns, giving the agent a standing directive it keeps working toward. Setting a Goal /goal "Refactor the database layer to use async SQLAlchemy" # Add sub-goals /subgoal "Add connection pooling" /subgoal "Rewrite all CRUD operations as async" /subgoal "Update tests" How Goals Work When a goal is active: 1. The goal text is appended to the system prompt on every turn 2. The agent autonomously works toward the goal across multiple user messages 3. Completing a sub-goal triggers the agent to move to the next one 4. Interleaving unrelated questions doesn't reset progress Managing Goals /goal status # See current goal + sub-goal progress /goal pause # Temporarily suspend goal pursuit /goal resume # Re-activate the goal /goal clear # Remove the current goal completely Goals are not persisted across sessions — they're session-scoped. 4.10 Worktree Mode The `-w` flag creates an **isolated git worktree** for the current session, letting you run multiple Hermes instances in the same repository without conflicts. # Start a session in worktree mode hermes -w # Creates: ~/my-repo/.worktrees/hermes-<uuid>/ # The session's CWD is inside this worktree Why Worktrees Matter Without worktrees, two Hermes sessions modifying the same files race against each other. Worktrees give each session: - Its own working directory (a detached HEAD copy of the repo) - No interference between parallel sessions - Clean isolation — changes in one worktree don't affect others - The original repository remains untouched until you merge Worktree Lifecycle # Create a worktree session hermes -w # Inside the session, you

can: # - Edit files # - Run tests # - Create branches # All isolated from other sessions # On exit, the worktree is cleaned up automatically # Uncommitted changes are lost (by design — commit before exiting!) Multi-Agent Worktree Workflows # Terminal 1 hermes -w -q "Refactor the auth module" # Terminal 2 (same repo, different worktree) hermes -w -q "Write tests for the auth module" # Terminal 3 (gateway, providing chat access) hermes gateway start # Both Telegram and CLI sessions get their own worktrees 4.11 Checkpoints & Rollback Checkpoints let you rewind the **filesystem state** to a previous point — like undo for your whole project. Creating Checkpoints Checkpoints are created automatically by the `terminal` and `file` tools. Every file write, patch, or terminal command that modifies files records a snapshot. # List all checkpoints for this session /rollback # Shows: # [0] Just now - Initial state # [1] 2 min ago - After creating routes.py # [2] 30 sec ago - After modifying database.py Restoring a Checkpoint # Restore to checkpoint 1 /rollback 1 # Files are reverted. Cannot be undone (but you can restore forward too). Snapshot System (Portable) # Create a named snapshot (exportable) /snap create "pre-refactor" # Saved to ~/.hermes/checkpoints/<name>/ # Restore from a snapshot /snap restore "pre-refactor" # List all snapshots /snap ls # Prune old snapshots /snap prune Snapshots capture: - Files in the working directory (configurable scope) - Active conversation context - Agent memory state This is useful for: - **Experiment before big refactors** — explore freely, revert instantly - **Teaching/demos** — create snapshots at different stages - **Reproducible debugging** — save the exact state before a bug occurs 4.12 Multi-Agent Coordination with tmux and Kanban For complex workflows, Hermes supports running multiple agents in parallel with structured coordination. tmux Integration Hermes can be used inside tmux sessions for persistent, detachable agent instances: # Create a tmux session for each agent tmux new-session -s dev-agent 'hermes -p developer' tmux new-window -t dev-agent -n test-agent 'hermes -p tester' tmux new-window -t dev-agent -n deploy-agent 'hermes -p deploy' # Detach and reattach later tmux detach tmux attach -t dev-agent Kanban Board The kanban system provides structured multi-agent coordination with a shared task board: # Initialize the kanban board (first time) /kanban init # Create a task on the board /kanban create "Refactor payment processor" --assignee coder # List all tasks /kanban list # Shows: ID, Title, Assignee, Status, Priority # Link tasks to show dependencies /kanban link 5 --depends-on 3 # Complete a task /kanban complete 5 --result "...summary..." Kanban Worker Profiles When an agent is spawned as a kanban worker (via the kanban dispatcher in the gateway), it automatically receives tools to: - Show assigned tasks (`kanban_show`) - Mark tasks complete with structured handoffs (`kanban_complete`) - Block for human input (`kanban_block`) - Heartbeat during long operations (`kanban_heartbeat`) - Comment on task threads (`kanban_comment`) Multi-Profile Orchestration # ~/.hermes/config.yaml in the kanban orchestrator profile: kanban: dispatch_in_gateway: true workers: - name: coder description: "Full-stack development and code writing" - name: reviewer description: "Code review, security audit, best practices" - name: sysadmin description: "Server management, deployment, monitoring" The kanban decomposer automatically breaks incoming tasks into subtasks and routes them to the most appropriate worker profile based on their descriptions. 4.13 Power User Tips 1. Chain Multiple Commands # Use single-query mode in pipelines curl -s https://api.github.com/repos/user/repo/issues \ | hermes chat -q "Summarize these GitHub issues and prioritize them" 2. Background Processing # Inside a session, do work without blocking /background "Download the dataset and preprocess it" # Keep chatting while the background task runs # The agent interrupts when done 3. Session Context Files # Create ~/.hermes/context-files/ for persistent project context # The agent loads these into system prompt automatically 4. Custom Persona Files (SOUL.md) # ~/.hermes/SOUL.md defines your agent's core personality # It's loaded into every session across all profiles 5. Keyboard Shortcuts in CLI Shortcut Action `Ctrl+C` Interrupt current agent task `Ctrl+L` Clear screen `Ctrl+R` Reverse search command history `Tab` Autocomplete slash commands `Shift+Enter` New line (multi-line input) `Ctrl+Enter` Send multi-line input `Up/Down` Navigate command history 6. Daily Workflow Example # Morning: start the gateway for mobile access hermes gateway start # Check in from CLI hermes chat -q "Summarize what I was working on yesterday" # Start a focused session hermes --continue "payment-refactor" # During session: /model "openrouter/deepseek/deepseek-chat-v3-0324" # Switch to cheaper model /verbose 2 # Show intermediate tool calls /goal "Refactor the payment module to use Stripe" # After session: schedule daily report hermes cron create

\ --name "daily-ticket-summary" \ --schedule "0 17 * * 5" \ --prompt "Summarize this week's GitHub activity" \ --platform telegram 4.14 Quick Reference: Common Task Cheatsheet Task Command Start chatting `hermes` Ask one question `hermes chat -q "question"` Use a specific profile `hermes -p work` Resume last session `hermes --resume` Switch model in-session `/model "provider/model"` Check usage `/usage` Compress context `/compress` Set a goal `/goal "objective"` Schedule a task `hermes cron create` (or `/cron create` in-session) Enable browser `hermes tools enable browser` Toggle YOLO `/yolo` Rollback file changes `/rollback ` Fork conversation `/branch "exploration-name"` See all sessions `hermes sessions list` Send webhook `curl -X POST /webhook -d '{"message":"..."}'` Create a profile `hermes profile create ` Manage credentials `hermes auth` Key Takeaways - **Two modes**: Interactive (`hermes`) for deep work, single-query (`hermes -q`) for automation - **Slash commands** are your interface to session management, model switching, and tools - **Context compression** keeps long conversations efficient — both automatic and manual (`/compress`) - **Subagent delegation** parallelizes work without contaminating parent context - **Session management** gives you named, searchable, resumable conversations - **Cron jobs** let the agent work unattended on schedules - **Webhooks** bridge external systems into agent conversations - **Goal mode** gives the agent persistent cross-turn objectives - **Worktrees** isolate parallel agent instances in the same git repo - **Checkpoints** provide filesystem undo for safe experimentation - **Kanban + tmux** enable full multi-agent orchestration for complex workflows - **The `/model` command** is arguably Hermes's killer feature — switch inference backends mid-conversation without losing context **Next Steps:** Explore the Skills System (Module 5) to create procedural knowledge that makes your agent smarter with every interaction. Module 3 Dashboard Module 5

Skills System - The Killer Feature

Back to Course Dashboard Skills System - The Killer Feature Module 5 of 10 ■ Full Walkthrough Video Your browser doesn't support video playback. Daily Usage & Power Featu All Modules Next: Training Your Agent **Duration:** ~45 minutes **Prerequisites:** Module 4 (Tools & Capabilities) **Skill Level:** Intermediate 5.1 Introduction Most AI assistants are amnesiacs. You teach them something one day, and the next day they've forgotten it entirely. You show them exactly how to deploy your app, and tomorrow you have to explain it all over again. This is the fundamental limitation of the "chatbot" paradigm — every conversation starts from zero. Hermes Agent breaks this cycle with **Skills**. Skills are the single most important feature of Hermes Agent. They transform the agent from a transactional assistant (ask → answer → forget) into a **compound-learning system** that gets provably better over time. Every time you teach Hermes something, that knowledge persists, compounds, and makes future interactions faster, more accurate, and more autonomous. By the end of this module, you'll understand: - What skills are and why they're revolutionary - The anatomy of a skill file (frontmatter + body) - How to create, load, and manage skills - The skill lifecycle (creation → curation → archival) - The Skills Hub ecosystem (browse, install, publish) - Best practices and anti-patterns - How 50+ skills make you a super-agent user 5.2 What Are Skills? A **skill** is a reusable, versioned procedure stored as a Markdown file with YAML frontmatter. Think of it as a "recipe" the agent can follow to accomplish a specific task. --- name: deploy-to-nginx description: Deploy a static site to an Nginx server on this machine version: 1.2.0 tags: [deployment, nginx, devops, web] author: ops-team --- # Deploy to Nginx 1. Verify the site source directory exists at `/var/www/{site_name}` 2. Check that Nginx is running: `systemctl status nginx` 3. Copy the site configuration: ``` cp /etc/nginx/sites-available/{site_name} /etc/nginx/sites-enabled/ ``` 4. Test the configuration: `nginx -t` 5. Reload Nginx: `systemctl reload nginx` 6. Verify the site responds: `curl -s -o /dev/null -w "%{http_code}" http://localhost/{site_name}` ## Pitfalls - If `nginx -t` fails, check for syntax errors in the config file - Don't use `restart` — use `reload` to avoid downtime - Ensure the document root directory exists before reloading ## Verification - Site returns HTTP 200 - Nginx status shows "active (running)" - New config appears in `nginx -T` output That's it. A skill is just a Markdown file — simple enough to edit by hand, structured enough for the agent to parse and execute reliably. What Makes Skills Different from Prompts? Aspect Regular Prompt Skill **Persistence** Ephemeral (one conversation) Permanent (saved to disk) **Versioning** None Explicit version field **Structure** Freeform text Numbered steps + frontmatter **Reusability** Must re-explain Load by name

Curation None Auto-archived when stale **Sharing** Not possible Skills Hub registry 5.3 Why Skills Matter Skills solve three fundamental problems in AI-human interaction: Problem 1: The Repetition Tax Every time you ask the agent to do something non-trivial, you pay a "repetition tax" — explaining context, specifying commands, correcting mistakes. With skills, you pay that tax once. The next time, the agent already knows. **Without skills:** "No, use `systemctl reload` not `restart`... remember Nginx has a test flag... check the config syntax first..." **With skills:** "Deploy the site." The agent loads the skill and executes it perfectly. Problem 2: Tribal Knowledge Loss When you have workflows that only you know — deployment procedures, database backup commands, CI/CD conventions — that knowledge lives in your head (or scattered across docs). Skills make it explicit, versioned, and transferable. New team members get instant access to your operational knowledge. Problem 3: Inconsistent Execution Humans and AI alike make mistakes on complex multi-step procedures. A skill ensures the same steps execute in the same order every time, with the same verification checks. The Compound Effect Here's the key insight: **skills compound.** Each skill you create makes the agent more capable. After 10 skills, you have a capable assistant. After 50 skills, you have an agent that knows your entire stack — deployment, testing, debugging, monitoring, database management, code review, everything. You stop explaining and start **delegating**. 5.4 Skill Anatomy Every skill file has two parts: **frontmatter** and **body**. Frontmatter (YAML) The frontmatter is a YAML block between `---` delimiters at the very top of the file. It contains metadata that helps the agent find, load, and manage the skill. --- name: deploy-to-nginx # Unique identifier, lowercase with hyphens description: > # Description used for matching queries to skills Deploy a static site to an Nginx server on this machine. Handles config validation, reload, and verification. version: 1.2.0 # Semantic versioning: major.minor.patch tags: # Tags for discoverability and grouping - deployment - nginx - devops - web author: ops-team # Optional: who created/maintains this depends_on: [] # Optional: other skills this one requires os: [linux] # Optional: OS constraints --- **Required fields:** `name`, `description`, `version` **Strongly recommended:** `tags` **Key rules:** - `name` must be unique across all your skills - `version` follows semver — bump `patch` for fixes, `minor` for new features, `major` for breaking changes - `description` should be detailed; the agent uses semantic matching to find the right skill Body (Markdown Instructions) The body is a Markdown document with numbered steps, commands, pitfalls, and verification steps. **Structure conventions:** # {Skill Name} {Context paragraph — what this skill does and when to use it} 1. {Step 1 — what to do} ``` {exact command or code} ``` 2. {Step 2 — what to do next} 3. ... ## Pitfalls - {Common mistake and how to avoid it} - {Another pitfall} - {Edge case to watch for} ## Verification - {How to confirm the skill executed successfully} - {Another verification check} Numbered Steps Steps should be: - **Actionable:** "Copy the configuration file" not "Understand the configuration" - **Deterministic:** The same inputs should produce the same outputs - **Verifiable:** Each step should produce output you can check - **Exact:** Include exact commands in code blocks Pitfalls Section This is where you capture **experience** — the mistakes you've made and the edge cases you've encountered. A skill without pitfalls is a recipe that only works in the happy path. Good pitfalls: - Warnings about common errors - Environment-specific gotchas - Ordering dependencies ("do X before Y") - Failure modes and recovery steps Verification Section Every skill should end with verification steps that answer "how do I know this worked?" These are explicit checks the agent can run to confirm success. Good verification steps: - HTTP status checks (`curl -s -o /dev/null -w "%{http_code}"`) - Service status checks (`systemctl is-active`) - File existence checks (`test -f /path/to/file`) - Output consistency checks (`diff expected.txt actual.txt`) 5.5 Creating a Skill Skills are created using the `skill_manage` tool. The `skill_manage` Tool skill_manage(action="create", name="skill-name", path="/path/to/skill.md") You can also create skills by asking the agent directly: "Save this as a skill called 'deploy-to-nginx': first, check the source directory exists..." The agent will: 1. Generate the skill file with proper frontmatter 2. Save it to the skills directory 3. Register it in the skill index 4. Confirm creation Updating Skills skill_manage(action="update", name="skill-name", path="/path/to/updated-skill.md") Or just ask: "Update the deploy-to-nginx skill — add a step to check disk space first." Deleting Skills skill_manage(action="remove", name="skill-name") 5.6 Loading a Skill Skills are loaded on demand, either explicitly or implicitly. Explicit Loading **In Hermes Chat (Telegram):** /skill deploy-to-nginx /skill deploy-to-nginx site_name=myapp The `/skill` command loads the skill and executes it. You can pass variables as `key=value` pairs. **From the CLI:** hermes

-s deploy-to-nginx Implicit Loading (Auto-Matching) The agent automatically matches your query to relevant skills. If you say "deploy the site," the agent finds the `deploy-to-nginx` skill and asks if you'd like to use it. **User:** "Deploy the new landing page to production." > **Agent:** "I have a skill for that: `deploy-to-nginx`. Shall I use it? (I'll deploy from `/var/www/landing-page`)" Skill Variables Skills can have variable placeholders using `{variable_name}` syntax. The agent fills these in from context or prompts you for missing values. 1. Check that the database {db_name} is running 2. Run migration: `migrate -d {db_name}` When loading: `/skill run-db-migration db_name=production` 5.7 Complete Example Skill Here's a complete, real-world skill you can create and use immediately: --- name: diagnose-disk-space description: > Diagnose disk space issues on a Linux server. Checks overall usage, largest directories, and identifies old log files that can be rotated. version: 1.0.0 tags: - system - disk - diagnostics - linux author: hermes-agent os: [linux] --- # Diagnose Disk Space Use this when the system reports low disk space or when deployments fail due to insufficient storage. ## Prerequisites - Root or sudo access ## Steps 1. Check overall disk usage: ``` df -h ``` Look for partitions above 80% usage. 2. Find the largest directories in `/`: ``` du -sh /* | sort -hr | head -20 ``` 3. For the most-used partition (from step 1), drill into specific directories: ``` du -sh /var/log/* | sort -hr | head -10 du -sh /var/lib/* | sort -hr | head -10 ``` 4. Check for old log files (>30 days): ``` find /var/log -name "*.log" -type f -mtime +30 -exec ls -lh {} \; | head -20 ``` 5. Suggest cleanup actions: - Rotate logs: `logrotate -f /etc/logrotate.conf` - Clear apt cache: `apt-get clean` - Remove old kernels: `apt-get autoremove --purge` - Truncate specific large files (if safe) - Add disk usage alerting if this is recurring ## Pitfalls - Don't delete files without confirming with the user first - `/tmp` may be mounted as `tmpfs` (RAM) — freeing space there won't help with disk usage - Be careful with `find -delete` — always preview first - Some systems use LVM — check with `lvdisplay` if `df` is confusing - Docker containers can consume `overlay2` storage in `/var/lib/docker` ## Verification - At least 10% free space on critical partitions - No partitions above 90% after cleanup - System reports OK for `systemctl status` - `df -h` shows improvement from initial state 5.8 The Skill Lifecycle Skills aren't static — they evolve through a lifecycle managed by the Hermes Agent curation system. Phase 1: Creation You create a skill (or the agent creates one on your behalf). The skill is saved to the active skills directory and indexed. Phase 2: Active Use The skill is available for immediate loading. The agent tracks: - **Usage frequency:** How often each skill is loaded - **Success rate:** Whether verification steps pass - **Error rate:** How often the skill produces errors - **Last used:** When the skill was last invoked Phase 3: Staleness Detection The **Curator** (a built-in agent subsystem) periodically reviews all skills. A skill is flagged as "stale" if: - It hasn't been used in 30+ days - Its verification steps consistently fail - Its commands reference tools or paths that no longer exist - A newer version of the same workflow exists Phase 4: Archival Stale skills are auto-archived — moved to an archive directory and removed from the active index. They're not deleted; you can restore them if needed. skills/ ■■■ active/ ■ ■■■ deploy-to-nginx.md ■ ■■■ diagnose-disk-space.md ■■■ archived/ ■■■ 2025-11-01_deploy-to-apache.md ■■■ 2025-10-15_old-backup-script.md Phase 5: Restoration If you need an archived skill: skill_manage(action="restore", name="deploy-to-apache") Or ask the agent: "Restore the old Apache deploy skill." 5.9 The Skills Hub The **Skills Hub** is a central registry where you can browse, install, and publish skills. Browsing Check what's available: /skillhub browse /skillhub search "database backup" /skillhub search --tag devops Installing from the Registry hermes skill install deploy-to-nginx hermes skill install --from https://skills.example.com/my-skills.json Publishing Your Skills Share your skills with the community: hermes skill publish diagnose-disk-space Published skills are: - Reviewed for quality and safety - Indexed by tags for discoverability - Version-tracked so users get updates - Rated by the community Private Registries Organizations can run their own Skills Hub registries for internal workflows: hermes skill registry add internal https://skills.internal.company.com hermes skill install db-backup --registry internal 5.10 Best Practices DO: Write Good Triggers The `description` field is your skill's "trigger" — it's what the agent uses to match queries. Write descriptions that capture the words someone would naturally use. ■ Bad: "Handles deployment procedures" ■ Good: "Deploy a static site or web application to an Nginx server on Ubuntu/Debian" DO: Use Numbered Steps Numbered steps are deterministic. The agent executes them in order. Bullet lists are for context, not execution. DO: Include Exact Commands Don't describe what to do — show the exact command: ■ Bad: "Check the Nginx

configuration" ■ Good: "Run `nginx -t`" DO: Write a Useful Pitfalls Section This is where the skill earns its keep. Every mistake you've made while running this procedure should be a pitfall. DO: Always Verify Every skill should end with verification steps that prove success. "It worked because I checked" is the skill's warranty. DO: Version Your Skills Bump versions when you make changes. This lets you (and the curator) track evolution. DO: Tag Thoughtfully Tags make skills discoverable. Use a consistent tagging convention. DON'T: Make a Skill for One-Off Tasks If you'll never do this again, don't make a skill. Skills are for procedures you repeat. DON'T: Make Skills for Trivial Things "Open the browser" is not a skill. Use your judgment — if it takes longer to create the skill than to do the task once, skip it (unless you'll do it many times). DON'T: Hardcode Secrets Never put passwords, API keys, or tokens in a skill file. Use environment variables or the agent's secret store instead. # DO THIS 1. Connect: `psql -h {db_host} -U {db_user} -d {db_name}` The agent will prompt for connection details. # NOT THIS 1. Connect: `psql -h localhost -U admin -d myapp` Password: s3cret! 5.11 When NOT to Make a Skill Skills are powerful, but they're not the answer to everything. Here's when you should **not** create a skill: Scenario Why Not What Instead One-time migration You'll never do it again Just ask the agent "Hello World" example Trivial, no repetition Skip it Dangerous operation No safe verification Document as a runbook, not a skill Deeply contextual Depends on too many variables Break into smaller skills Rapidly changing Outdated in a week Wait for stability Personal preference "I like tabs not spaces" Save as a user profile preference **The litmus test:** Ask yourself "Will I need to do this again in the next 30 days?" If no, probably don't skill it. If yes, invest the 5 minutes. 5.12 Skills as Your Competitive Advantage Here's the truth: **Hermes Agent is only as good as your library of skills.** A new user with zero skills has a chatbot. A user with 10 skills has a capable assistant. A user with 50+ skills has a super-agent. The 50-Skill Milestone At 50 skills, something changes. The agent starts to: - **Predict what you need** — before you ask - **Chain skills together** — one task triggers a sequence of related skills - **Offer proactive improvements** — "I notice you deploy frequently; I could set up automatic verification" - **Become autonomous** — you say "handle the deployment" and it manages everything What 50 Skills Looks Like Category Skills **DevOps (10)** Deploy to Nginx, deploy to Docker, deploy to K8s, rollback, DB backup, log analysis, monitor health, scale services, SSL renewal, CI/CD setup **Development (10)** Code review checklist, run tests, lint code, create PR, review PR, debug crash, profile performance, optimize query, generate types, stub API **System Admin (8)** Diagnose disk, check memory, find process, configure firewall, set up user, audit security, update system, manage cron **Data (6)** ETL pipeline, backup DB, restore DB, analyze logs, generate report, data validation **Writing (5)** Draft changelog, write release notes, create RFC, review docs, format markdown **Research (4)** Search docs, find packages, compare solutions, summarize findings **Quality (4)** Run linter, check coverage, validate schema, audit dependencies **Automation (3)** Schedule task, watch directory, auto-cleanup The Winning Strategy 1. **Start small** — skill your most painful repeated task today 2. **Skill by correction** — whenever you correct the agent, ask: "Should this be a skill?" 3. **Review monthly** — look at archived skills, update stale ones, delete useless ones 4. **Publish and consume** — share skills internally, install from the hub 5. **Aim for 50** — it's the tipping point where the agent becomes truly autonomous 5.13 Summary Skills are Hermes Agent's killer feature. They transform the agent from a stateless chat interface into a persistent, learning system that gets better every time you use it. **Key takeaways:** - Skills are Markdown files with YAML frontmatter (name, description, version, tags) - They contain numbered steps, exact commands, pitfalls, and verification checks - Create with `skill_manage(action="create")`, load with `/skill` or `hermes -s` - The Curator auto-archives stale skills; the Skills Hub lets you share them - Best skills are repeatable, verifiable, and versioned - 50+ skills = a super-agent that knows your entire stack **In the next module,** we'll cover how to train your agent — memory management, corrections, user profiles, and the powerful correction-to-skills pipeline. 5.14 Exercises 1. **Create your first skill:** Pick a task you do weekly and write a skill for it. Create it with the `skill_manage` tool. 2. **Load a skill:** Use `/skill` to execute your new skill. 3. **Review a skill:** Open an existing skill file, check its frontmatter is complete, ensure it has a pitfalls and verification section. 4. **Publish (optional):** If your skill is general-purpose, publish it to the Skills Hub. 5.15 Further Reading - [Hermes Agent Skills Documentation](https://hermes-agent.nousresearch.com/docs/skills) - [Skill Creation Best Practices Guide](https://hermes-agent.nousresearch.com/docs/skills/best-practices) - [Skills

Hub API Reference](https://hermes-agent.nousresearch.com/docs/skills/hub) - [Curator Configuration](https://hermes-agent.nousresearch.com/docs/skills/curator) Module 4 Dashboard Module 6

Training Your Agent

Back to Course Dashboard Training Your Agent Module 6 of 10 ■ Full Walkthrough Video Your browser doesn't support video playback. Skills System - The Kille All Modules Next: Multi-Platform Gateway **Duration:** ~35 minutes **Prerequisites:** Module 5 (Skills System) **Skill Level:** Intermediate 6.1 Introduction In Module 5, you learned how skills make the agent permanently capable. But skills are just one piece of the training puzzle. A truly effective agent knows not just *what to do*, but *how you like things done* — your preferences, your conventions, your environment, your communication style. This module covers the complete training system: - **Memory:** How the agent stores and retrieves information about you - **Corrections:** How every mistake becomes a learning opportunity - **User profiles:** Who you are, what you prefer, how you work - **Session search:** Finding past work and context - **The compound effect:** How 50+ sessions of training produce an agent that knows your entire stack After this module, you'll be able to train Hermes to be *your* agent — not just a generic AI assistant. 6.2 How Memory Works Hermes Agent has a structured memory system with two main targets: Memory Targets Target Purpose Example `memory` General facts, preferences, and knowledge "Prefers tabs over spaces" `user_profile` Identity, roles, and communication style "Senior DevOps engineer, Python expert" The `memory` Target The `memory` target stores general facts the agent should remember across sessions. **What goes in memory:** - Environment configuration (OS, tools installed, paths) - Project conventions (naming, structure, workflow) - Personal preferences (editor, language, formatting) - Lessons learned (mistakes, edge cases, workarounds) - Common credentials or connection info (without secrets) **How it works:** - Memory is persistent across sessions - The agent queries memory when it needs context it doesn't have - Memories can be tagged for better retrieval - New memories are added explicitly or by correction The `user_profile` Target The `user_profile` target is a structured profile of *you* — your identity, expertise, and communication preferences. **What goes in user_profile:** - Your name and role - Technical expertise level - Communication style (verbose/concise, formal/casual) - Preferred tools and technologies - Timezone and working hours - Language and locale preferences 6.3 The Memory Tool Memory is managed through the `memory` tool with three actions: `add`, `replace`, and `remove`. Adding Memory memory(add, "The user prefers two-space indent in Python files") memory(add, "Project root is at /home/user/projects/myapp") memory(add, "Deployments target staging.example.com") You can also add memories directly through conversation: **User:** "Remember that I use tabs for Go files." > **Agent:** "Got it. I've memorized: User prefers tabs for Go files." The agent extracts the key information and stores it properly. Replacing Memory When a fact changes: memory(replace, "The user prefers tabs for Go files", "The user now prefers spaces for Go files") Removing Memory When a fact is no longer relevant: memory(remove, "The old project root was at /home/user/projects/legacy") Batch Memory Operations For larger updates: memory(add, ["Database host is db.internal.example.com", "Database port is 5432", "Production database is named 'myapp_prod'"]) 6.4 What to Save in Memory Effective memory management is about knowing what's worth remembering. Here's a guide: ■ Save These Category Examples **Preferences** "Prefers dark mode terminals", "Uses neovim not vim" **Environment facts** "Running on Ubuntu 24.04", "Python 3.12 is default" **Project conventions** "Uses pnpm not npm", "Tests use pytest" **Paths and locations** "Code at /home/user/code", "Logs at /var/log/myapp" **Credentials context** "SSH key is at ~/.ssh/id_ed25519" (not the key itself!) **Lessons learned** "Port 8080 requires sudo", "Docker needs --platform linux/amd64 on this machine" **Workflow preferences** "Always run tests before committing", "Use feature branches" **Access patterns** "Usually deploys on Tuesdays", "Check staging before production" ■ Don't Save These Category Why Not **Secrets/passwords** Memory is stored on disk — treat it as unencrypted **Task progress** "I'm 60% through the migration" — stale immediately **Completed work** "Deployed v2.3 yesterday" — irrelevant tomorrow **Temporary state** "The build failed with error X" — belongs in the current session **Obvious facts** "The sky is blue" — wastes memory space **Emotional state** "User was frustrated" — not useful for future sessions The Litmus Test Before saving something to memory, ask: 1. **Will this be true next month?** — If no,

it's temporary state, not memory. 2. **Will I need this again?** — If no, it's one-off context. 3. **Is this a preference or a fact?** — Preferences and stable facts are good memory. 4. **Can I look this up faster than storing it?** — Some things are better documented externally. 6.5 How Corrections Make the Agent Smarter Every interaction with the agent is a training opportunity. When the agent gets something wrong, your correction is the most valuable data it can receive. The Correction Mechanism When you correct the agent: User: "Deploy to staging, not production." Agent: "You're right. I've corrected my understanding. Memory updated: Deployments target staging first. Would you like me to save this as a skill?" The agent: 1. **Acknowledges** the correction 2. **Updates** its memory with the correct information 3. **Offers** to create a skill if the correction represents a repeatable workflow Types of Corrections Correction Type Example Agent Response **Factual** "The database is PostgreSQL, not MySQL" Updates memory **Preferential** "I prefer full sentences in responses" Updates user_profile **Procedural** "Always run lint before commit" Updates memory + offers skill **Environmental** "The API is on port 3000, not 8080" Updates memory **Safety** "Never delete files without confirmation" Updates memory as rule The One-Correction Rule A well-trained agent should never make the same mistake twice. If you correct the agent about something, and it makes the same error in a future session, that's a sign the memory didn't stick — escalate it to a skill. 6.6 Teaching Your Agent Your Preferred Workflows The most powerful training happens when you teach the agent your workflows step by step. The Walk-Through Method The first time you do a complex task, walk the agent through it: **User:** "Let me show you how I set up a new project. First, I clone the template repo..." > **Agent:** "I'm following. Clone from which repo?" > **User:** "git@github.com:myorg/project-template.git" > **Agent:** "Noted. What's next?" > **User:** "Then I run `pnpm install` and configure the `.env` file..." > **Agent:** "I see the pattern. After this walk-through, would you like me to save this as a skill called 'setup-new-project'?" The Shadow Mode For workflows you're experienced with, let the agent shadow you: **User:** "Watch what I do for deploying to production. I'll narrate." > (User performs the steps) > **Agent:** "I observed your deployment workflow. It involved: 1. Run tests (`pnpm test`) 2. Build (`pnpm build`) 3. Copy to server (`rsync -avz dist/ user@server:/var/www/`) 4. Reload nginx (`ssh user@server 'sudo systemctl reload nginx'`) > Shall I save this as a skill called 'deploy-production'?" The Correction Pipeline Every correction should be treated as a potential pipeline input: Correction → Memory Update → Skill Offer → (optional) Skill Creation **Real example:** **User:** "No, don't use `cp` — use `rsync` with compression." > **Agent:** "Corrected. Memory updated: Use `rsync -avz` instead of `cp` for deployments." > **User:** "Yes, save that as a skill." > *(Agent creates `deploy-with-rsync` skill)* This pipeline is the single most powerful training technique in Hermes Agent. 6.7 User Profiles The `user_profile` target stores structured information about who you are and how you like to interact. Creating Your Profile Start with the basics: memory(user_profile, "Senior DevOps engineer at Acme Corp") memory(user_profile, "Expert in: Python, Go, Kubernetes, Terraform") memory(user_profile, "Prefers concise, action-oriented responses") memory(user_profile, "Timezone: US/Eastern") memory(user_profile, "Communication style: direct, minimal pleasantries") memory(user_profile, "Preferred code editor: Neovim") memory(user_profile, "Shell: zsh with oh-my-zsh") Profile Sections A well-built profile covers: **Identity** - Your name, role, team, organization - Your technical role (developer, SRE, data scientist, etc.) **Expertise** - Languages you know well (so the agent can tailor code examples) - Frameworks and tools you use daily - Areas where you want more guidance vs. areas where you're expert **Communication** - Verbosity preference (one-liners vs. detailed explanations) - Formality level (casual vs. formal) - Response structure preference (bullet points, paragraphs, steps) - Whether you want citations, confidence levels, or options **Environment** - Operating system and version - Default shell and terminal - Package manager preferences - Editor/IDE configuration **Workflow Preferences** - Testing before deployment (always/never/ask) - Staging/production workflow - Git workflow (feature branches, rebase, squash) - Code review expectations Evolving Your Profile Your profile isn't static. As your preferences change: memory(user_profile_replace, "Prefers concise responses", "Now prefers detailed explanations with examples") Or let the agent observe: **Agent:** "I notice you've been asking for more detailed explanations lately. Would you like me to update your profile from 'concise' to 'detailed'?" 6.8 Memory Curation Memory accumulates. Left unchecked, it becomes cluttered with outdated facts, superseded preferences, and irrelevant context. Reviewing Memory List all stored memories:

/memory list /memory list --tag database /memory list --since 2025-01-01 Pruning Stale Memories **User:** "Check my memory for anything outdated." > The agent reviews each memory against current reality and suggests pruning. Automated Curation The agent can auto-curate memories: - **Staleness detection:** Memories older than 90 days are flagged - **Contradiction detection:** If two memories conflict, the agent asks for clarification - **Deduplication:** Similar memories are merged - **Low-value flagging:** Very generic memories are suggested for removal Memory Health Check Run a periodic memory audit: /memory health-check The agent produces a report: Memory Health Report ========== Total memories: 47 Recently added (30d): 12 Stale (90d+): 3 Conflicts: 0 Duplicates: 1 Recommendations: - Remove: "Used to use Apt" (superseded by "Uses Nix") - Remove: "Project path was /tmp/test" (temporary) - Merge: 2 memory entries about Go formatting preferences 6.9 Session Search Sometimes the answer isn't in memory — it's in a previous conversation. Session search lets you find past context. Searching Sessions /session search "database migration plan" /session search --from 2025-03-01 --to 2025-03-15 /session search --tag deployment Session Context Loading When you find a relevant session, you can load its context: /session load last-deployment-conversation This brings the agent up to speed on what was discussed and decided. Session Tags Tag important sessions for easy retrieval: /session tag "production-incident-2025-05-15" Smart Session Linking The agent automatically links sessions to memory and skills: **User:** "Remember that issue we debugged last week?" > **Agent:** "I found the session from May 16 about the database connection timeout. Key findings: - Pool size was too small (5 connections) - Fix: Increased to 25 connections in the config - Root cause: Connection leak in the worker pool > I've saved this as a memory entry. Shall I make a diagnostic skill for it?" 6.10 The Compound Effect The headline claim of Hermes Agent is that it gets better over time. Here's what that actually means in practice. Session 1: First Contact The agent knows nothing about you. Everything is manual. You describe your environment, preferences, and needs from scratch. The agent asks clarifying questions. **Experience:** Like talking to a competent stranger. Session 10: Familiarity The agent knows your basic preferences, your common tools, your environment. It stops asking "what OS?" and starts remembering previous conversations. **Experience:** Like a new coworker on week two. Session 25: Competence The agent has a solid library of skills (10-15), a populated memory, and a user profile. It anticipates your needs and suggests skills unprompted. Corrections are rare and specific. **Experience:** Like a junior team member who's been on the project for a month. Session 50: The Tipping Point The agent crosses into **autonomous competence**. It has 30-50 skills covering your entire workflow. It: - Loads the right skill without being asked - Chains multiple skills for complex tasks - Remembers project-specific conventions - Suggests improvements to your workflows - Corrects itself before you can - Handles entire workflows with a single command **Experience:** Like a senior engineer who's worked with you for years. Session 100+: Mastery The agent has your entire stack memorized. It knows every tool, every config file, every deployment target, every team convention, every edge case you've ever encountered. It's not just an assistant — it's a force multiplier. **Experience:** Telepathy. You think about what needs doing, and the agent has already started. What Drives the Compound Effect Session Range Key Actions Result **1-10** Create user profile, add core memories, skill 3-5 repeated tasks Basic competence **11-25** Skill all repeated workflows, curate memories, develop correction reflex Reliable execution **26-50** Refine skills with pitfalls, publish useful skills, build profile depth Autonomous operation **51-100** Skill complex multi-step processes, chain skills, teach edge cases Mastery 6.11 Practical Training Workflow Here's a step-by-step workflow for training your agent in any new domain: Step 1: Prime the Memory Before working in a new domain, add relevant context: memory(add, "Working on project AcmeWeb, a Django app at /home/user/acmeweb") memory(add, "Using Python 3.12, PostgreSQL 16, Redis 7") memory(add, "Deploy target: acmeweb.staging.company.com via systemd") Step 2: Walk Through the First Task Do the task once with the agent observing: User: "Let me show you how to deploy the staging environment..." (Walks through the deployment) Agent: "I see the pattern. Save as skill?" Step 3: Create Skills from Repeatable Tasks For every step you'll do again, create a skill: - Skill: `deploy-acmeweb-staging` - Skill: `rollback-acmeweb-staging` - Skill: `check-acmeweb-health` Step 4: Correct Liberally Every mistake is a teaching moment: User: "No, use the staging config, not production." Agent: "Corrected. Saving to memory: Always use staging config first." Step 5: Review and Curate At the end of the

session: Memory check: Are there any new preferences or facts to save? Session tag: Tag this session for future reference. 6.12 Common Training Mistakes Mistake 1: Saving Too Much Don't save everything. Focus on **reusable facts**, not ephemeral context. ■ `memory(add, "The build failed twice before succeeding")` — ephemeral, not useful ■ `memory(add, "Build sometimes fails on the first attempt due to cache issues")` — reusable lesson Mistake 2: Saving Secrets Never save passwords, API keys, or tokens in memory. Memory files are readable on disk. Mistake 3: Not Correcting The most common mistake is letting small errors slide. Every uncorrected error is a future repeated error. Mistake 4: Skipping the Skill Prompt When the agent asks "Should I save this as a skill?" — say yes if it's repeatable. The agent's judgment about what's skill-worthy is conservative; it's better to have an extra skill than to lose a useful procedure. Mistake 5: Neglecting Curation Old memories create noise. Prune monthly. A cluttered memory is almost as bad as no memory. 6.13 Summary Training your Hermes Agent is an investment that compounds with every session. **Key takeaways:** - Memory stores stable facts (preferences, environment, conventions) via `add`, `replace`, `remove` - The `user_profile` target captures who you are and how you communicate - Corrections are training data — every mistake is an opportunity to improve - The correction → skills pipeline turns one-time fixes into permanent capabilities - Session search lets you retrieve past context and decisions - After 50+ sessions, the agent transitions from helpful assistant to autonomous partner - Regular curation keeps memory clean and relevant **In the next module,** we'll cover advanced agent patterns — chaining skills, building autonomous workflows, and deploying the agent as a service. 6.14 Exercises 1. **Build your profile:** Use `memory(user_profile, ...)` to create a complete user profile with identity, expertise, and communication preferences. 2. **Memory audit:** Run `/memory list` and identify three memories that are outdated — remove or update them. 3. **Correction practice:** Use the agent for a task you know well. When it gets something wrong, correct it explicitly. Verify the correction stuck by asking about it later. 4. **Skill pipeline:** Pick a workflow you do manually. Walk the agent through it, then save it as a skill. Then run the skill to verify it works. 5. **Session search:** Tag an important session, then search for it by tag. Practice loading session context. 6.15 Further Reading - [Memory Management Guide](https://hermes-agent.nousresearch.com/docs/memory) - [User Profiles Documentation](https://hermes-agent.nousresearch.com/docs/user-profiles) - [Session Management API](https://hermes-agent.nousresearch.com/docs/sessions) - [Training Best Practices](https://hermes-agent.nousresearch.com/docs/training) Module 5 Dashboard Module 7

Multi-Platform Gateway

Back to Course Dashboard Multi-Platform Gateway Module 7 of 10 ■ Full Walkthrough Video Your browser doesn't support video playback. Training Your Agent All Modules Next: VPS Deployment **Course:** Hermes Agent — From Zero to Autonomous Agent **Module:** 7 of 10 **Reading time:** ~20 minutes **Difficulty:** Intermediate TL;DR **What it is** A single daemon process that connects Hermes to Telegram, Discord, Slack, WhatsApp, Signal, and 15+ other platforms — all from one agent instance **Key idea** Same AI agent, different chat interfaces. Your tools, memory, skills, and cron jobs work identically across every platform **Setup** `hermes gateway setup` — interactive wizard, or manual `.env` config **Service** `hermes gateway install` — installs as systemd (Linux) or launchd (macOS) **Security** Allowlists, DM pairing, admin/user tiers, command approval for dangerous operations **Cron** Scheduled jobs deliver to your home channel on any platform What Is the Gateway? The Hermes Gateway is a **background daemon** that runs your agent and connects it to messaging platforms. Think of it as the bridge between your LLM-powered agent and the apps you use every day. Without the gateway, you run Hermes in your terminal — useful, but tied to your desktop. With the gateway: - You message your agent from **Telegram on your phone** while commuting - Your **Discord server** has a shared AI assistant your whole team can DM - **Cron jobs** (scheduled tasks) run 24/7 and deliver results to your chat - Your agent is always online, always ready, whether you're at your desk or not The architecture is clean: Telegram ■■ Discord ■■ Slack ■■ WhatsApp ■■ Signal ■■ ■■■■■■■■■■■■■■■■■■■■■■■■ Email ■■■■ platform ■→■ Hermes Gateway ■■■→ AIAgent (your LLM) SMS ■■ adapters ■ (single daemon) ■■■→ Tools (shell, files, web) Matrix ■■ ■ Sessions, cron, ■■■→ Skills Home ■■ ■ security, voice ■■■→

Memory Assistant■■ ■■■■■■■■■■■■■■■■■■■■■■■■ One gateway process. Many platform adapters. Same agent, tools, skills, and memory everywhere. Supported Platforms Hermes ships with built-in adapters for the following platforms. Each adapter receives messages, routes them through a per-chat session store, and dispatches them to the agent for processing. Platform Voice Images Files Threads Typing Streaming ---------- :-----: :------: :-----: :-------: :------: :---------: **Telegram** ■ ■ ■ ■ ■ ■ **Discord** ■ ■ ■ ■ ■ ■ **Slack** ■ ■ ■ ■ ■ ■ **WhatsApp** — ■ ■ — ■ ■ **Signal** — ■ ■ — ■ ■ **SMS (Twilio)** — — — — — — **Email** — ■ ■ ■ — — **Matrix** ■ ■ ■ ■ ■ ■ **Mattermost** ■ ■ ■ ■ ■ ■ **Home Assistant** — — — — — — **DingTalk** — ■ ■ — ■ ■ **Feishu / Lark** ■ ■ ■ ■ ■ ■ **WeCom (WeChat Work)** ■ ■ ■ — ■ ■ **Weixin** ■ ■ ■ — ■ ■ **BlueBubbles (iMessage)** — ■ ■ — ■ — **QQ** ■ ■ ■ — ■ — **Yuanbao** ■ ■ ■ — ■ ■ **Microsoft Teams** — ■ — ■ ■ — **LINE** — ■ ■ — ■ — **Google Chat** — ■ ■ ■ ■ — **API Server** (OpenAI-compatible) — — — — — — **Webhooks** — — — — — — **Voice** = TTS audio replies and/or voice message transcription. **Images** = send/receive images. **Files** = send/receive attachments. **Threads** = threaded conversations. **Typing** = typing indicator while processing. **Streaming** = progressive message updates. Platforms without built-in adapters can be added via the plugin system. See `ADDING_A_PLATFORM.md` in the gateway source. How the Gateway Works Architecture Detail Each platform has its own **adapter** — a Python module that handles the platform-specific protocol (Telegram's polling/webhook API, Discord's WebSocket gateway, Signal's CLI bridge, etc.). When a message arrives: 1. **Authorization check** — Is this user allowed? (Allowlist, DM pairing, or admin check) 2. **Session lookup** — Does this chat have an existing conversation context? 3. **Platform-specific handling** — Mention checks, free-response channels, thread isolation 4. **Agent dispatch** — The message is sent to the AIAgent with the correct session context 5. **Response delivery** — The agent's response is streamed back to the platform with typing indicators Same Tools Everywhere Every platform gets the same toolset — just with a different toolset name for logging: Platform Toolset Name Capabilities Telegram `hermes-telegram` Full tools including terminal Discord `hermes-discord` Full tools including terminal Slack `hermes-slack` Full tools including terminal WhatsApp `hermes-whatsapp` Full tools including terminal SMS `hermes-sms` Full tools including terminal Email `hermes-email` Full tools including terminal Home Assistant `hermes-homeassistant` Full tools + HA device control This means the **same agent** that runs shell commands in your terminal can do it from Telegram, Discord, or any other platform. Session Management Sessions persist across messages until they reset. The agent remembers your conversation context. Reset policies (configurable): - **Daily** — Reset at a specific hour (default: 4:00 AM) - **Idle** — Reset after N minutes of inactivity (default: 1440 min) - **Both** — Whichever triggers first (default) - **None** — Never auto-reset (controlled by compression only) Configure per-platform overrides in `~/.hermes/gateway.json`: { "reset_by_platform": { "telegram": { "mode": "idle", "idle_minutes": 240 }, "discord": { "mode": "idle", "idle_minutes": 60 } } } Telegram Setup This is the most common gateway setup. Let's walk through it step by step. Step 1: Create a Bot via BotFather 1. Open Telegram and search for **@BotFather**, or visit t.me/BotFather 2. Send `/newbot` 3. Choose a **display name** (e.g., "My Hermes Agent") 4. Choose a **username** that ends in `bot` (e.g., `my_hermes_bot`) 5. BotFather replies with your **API token**: 123456789:ABCdefGHIjklMNOpqrSTUvwxYZ **Keep this token secret.** Anyone with it can control your bot. If it leaks, use `/revoke` in BotFather immediately. Step 2: Optional Customization Message BotFather and use: Command Purpose `/setdescription` What the bot can do — shown before a user chats `/setabouttext` Short text on the bot's profile `/setuserpic` Upload an avatar `/setcommands` Define the command menu A useful starting set for `/setcommands`: help - Show help information new - Start a new conversation sethome - Set this chat as the home channel Step 3: Privacy Mode (Critical for Groups) Telegram bots have **privacy mode enabled by default**. This is the #1 source of confusion. **With privacy mode ON**, your bot can only see: - Messages starting with `/` - Replies to its own messages - Service messages (joins, leaves, pins) **With privacy mode OFF**, the bot sees every message in the group. To disable it: 1. Message **@BotFather** 2. Send `/mybots` → select your bot 3. Go to **Bot Settings → Group Privacy → Turn off** **Important:** You must remove and re-add the bot to any group after changing this setting. Telegram caches the privacy state when the bot joins. Step 4: Find Your User ID Your Telegram user ID is a number like `123456789`.

Message [@userinfobot](https://t.me/userinfobot) to get it instantly. Step 5: Configure Hermes **Option A: Interactive Setup (Recommended)** hermes gateway setup Select **Telegram** when prompted. The wizard asks for your bot token and allowed user IDs, then writes the configuration. **Option B: Manual Configuration** Add to `~/.hermes/.env`: TELEGRAM_BOT_TOKEN=123456789:ABCdefGHIjklMNOpqrSTUvwxYZ TELEGRAM_ALLOWED_USERS=123456789 For multiple users, comma-separate the IDs: TELEGRAM_ALLOWED_USERS=123456789,987654321 Step 6: Start the Gateway hermes gateway The bot comes online within seconds. Send it a message on Telegram to verify. Discord Setup Step 1: Create a Discord Application 1. Go to the [Discord Developer Portal](https://discord.com/developers/applications) 2. Click **New Application** → enter a name → **Create** 3. Note your **Application ID** — you'll need it for the invite URL Step 2: Create the Bot 1. In the left sidebar, click **Bot** 2. Set **Public Bot** to **ON** (recommended) 3. Set a custom avatar if desired Step 3: Enable Privileged Gateway Intents **This is the most critical step.** Scroll down to **Privileged Gateway Intents** and enable: Intent Required? Why -------- :---------: ----- **Presence Intent** Optional See online/offline status **Server Members Intent** **Required** Resolve usernames for allowed users **Message Content Intent** **Required** Read the text content of messages **Without Message Content Intent enabled, your bot receives messages but the text content is empty.** It's the #1 reason Discord bots appear online but never respond. Step 4: Get the Bot Token Still on the **Bot** page, click **Reset Token**. Copy it immediately — it's only shown once. Step 5: Generate the Invite URL Use the **Installation** tab (if Public Bot is ON) or create a manual URL: https://discord.com/oauth2/authorize?client_id=YOUR_APP_ID&scope=bot+applications.commands&permissions=274878286912 Replace `YOUR_APP_ID` with the ID from Step 1. Minimum permissions: View Channels, Send Messages, Embed Links, Attach Files, Read Message History. Step 6: Invite to Your Server Open the invite URL → select your server → **Authorize**. The bot appears in your server's member list. Step 7: Find Your User ID 1. Open Discord → **Settings** → **Advanced** → enable **Developer Mode** 2. Right-click your username → **Copy User ID** Step 8: Configure Hermes **Interactive:** hermes gateway setup **Manual — add to `~/.hermes/.env`:** DISCORD_BOT_TOKEN=your-bot-token DISCORD_ALLOWED_USERS=284102345871466496 Step 9: Start the Gateway hermes gateway The bot comes online. Send it a DM or `@mention` it in a channel. Gateway as a Systemd Service For 24/7 operation, install the gateway as a background service. Installation hermes gateway install On Linux, this creates a systemd user service at `~/.config/systemd/user/hermes-gateway.service`. For a boot-time system service (recommended for VPS): sudo hermes gateway install --system Service Management hermes gateway start # Start the service hermes gateway stop # Stop the service hermes gateway restart # Restart the service hermes gateway status # Check if it's running Enable Lingering (Linux) Keeps user services running after you log out: sudo loginctl enable-linger $USER View Logs # Systemd journal (user service) journalctl --user -u hermes-gateway -f # Systemd journal (system service) journalctl -u hermes-gateway -f # Direct log file tail -f ~/.hermes/logs/gateway.log macOS (launchd) hermes gateway install # Install as launchd agent hermes gateway start # Start hermes gateway stop # Stop hermes gateway status # Check status tail -f ~/.hermes/logs/gateway.log # View logs The plist lives at `~/Library/LaunchAgents/ai.hermes.gateway.plist` and captures your PATH at install time. If you add new tools later, re-run `hermes gateway install` to update it. Gateway Status Check what's running: hermes gateway status This shows: - Whether the service is active - How long it's been running - Which platforms are connected - Last log entries From within any connected chat, use: /status — Show session info /whoami — Show your access tier and allowed commands /platforms — Show all active platform adapters The `/platform` command lets you inspect and steer individual adapters without restarting: /platform list — Show all adapters and their state /platform pause <name> — Stop dispatching new messages to one adapter /platform resume <name> — Re-enable a paused adapter Gateway Logs Logs are written to `~/.hermes/logs/gateway.log`. This is your primary debugging resource. Important log locations: What Where Gateway log `~/.hermes/logs/gateway.log` Agent log `~/.hermes/logs/agent.log` Errors `~/.hermes/logs/errors.log` Systemd journal (user) `journalctl --user -u hermes-gateway -f` Systemd journal (system) `journalctl -u hermes-gateway -f` Each adapter logs its name in brackets, making it easy to filter: [telegram] Connected to Telegram (polling mode) [discord] Connected to Discord [telegram] Message from

@username: /new Security Allowlists By default, the gateway denies all users who are not explicitly allowed. This is the safe default for a bot with terminal access. # Per-platform TELEGRAM_ALLOWED_USERS=123456789,987654321 DISCORD_ALLOWED_USERS=284102345871466496 SIGNAL_ALLOWED_USERS=+15551234567 # Global (applies to all platforms) GATEWAY_ALLOWED_USERS=123456789 # Or explicitly allow everyone (NOT recommended if the bot has terminal access) GATEWAY_ALLOW_ALL_USERS=true DM Pairing Instead of manually configuring user IDs, unknown users receive a one-time pairing code when they DM the bot: # User sees: "Pairing code: XKGH5N7P" # You approve them with: hermes pairing approve telegram XKGH5N7P # Other commands: hermes pairing list # View pending + approved users hermes pairing revoke telegram 123456789 # Remove access Pairing codes expire after 1 hour and use cryptographic randomness. Admin vs Regular Users Every allowed user falls into one of two tiers per scope: - **Admin** — full access to all slash commands and capabilities - **Regular user** — can chat normally but only run explicitly enabled slash commands Configure in `~/.hermes/config.yaml`: gateway: platforms: discord: extra: allow_from: ["111", "222", "333"] allow_admin_from: ["111"] user_allowed_commands: [status, model] Use `/whoami` from any platform to see your current tier and allowed commands. Command Approval When the agent wants to run a dangerous command (e.g., `rm -rf /`), the gateway sends an **approve/deny** button (Telegram) or prompts you to use `/approve` / `/deny`. This prevents the agent from executing destructive operations without your confirmation. Home Channel Concept The **home channel** is the default delivery destination for your platform. When a cron job specifies `deliver: telegram` without a specific chat ID, messages are sent to the home channel. Setting a Home Channel The easiest way — in any chat, use: /sethome The gateway confirms with something like: ■ This chat is now the home channel for Telegram. Manual Configuration In `~/.hermes/.env`: TELEGRAM_HOME_CHANNEL=-1001234567890 TELEGRAM_HOME_CHANNEL_NAME="My Notes" DISCORD_HOME_CHANNEL=987654321098765432 DISCORD_HOME_CHANNEL_NAME="Team Chat" Group chat IDs are negative numbers on Telegram. Cron Deliveries When Hermes executes a scheduled task (cron job), the result needs to go somewhere. The gateway handles delivery seamlessly. How It Works 1. Cron job fires at the scheduled time 2. Agent executes the task in a fresh session 3. Result is delivered to the configured destination Delivery Options When creating a cron job, specify where results go: # Deliver to the home channel hermes cron create "0 8 * * *" "Check server status" --deliver telegram # Deliver to Discord home channel hermes cron create "0 8 * * *" "Check server status" --deliver discord # Save to local files hermes cron create "0 8 * * *" "Check server status" --deliver local From a chat session, just tell the agent naturally: Every morning at 9am, check the server status and send me a summary on Telegram. Hermes creates the cron job and configures delivery automatically. Running the Gateway 24/7 on a VPS For a permanently online agent, run the gateway on a VPS: 1. Provision a VPS ($4-6/month — see Module 8) 2. Install Hermes with the standard one-liner 3. Configure your API keys and platform tokens in `.env` 4. Install the gateway as a system service: sudo hermes gateway install --system sudo hermes gateway start --system sudo loginctl enable-linger $USER 5. Verify with: sudo hermes gateway status --system The gateway restarts automatically at boot, runs your cron jobs, and delivers results to your home channel. Troubleshooting Gateway Crash Loop **Symptom:** The gateway starts and immediately crashes. `systemctl` shows `failed` status. **Diagnosis:** journalctl --user -u hermes-gateway -n 50 --no-pager **Common causes:** - **Missing API keys** — Your `.env` file doesn't have a required API key for your LLM provider. Check that the provider's API key is set. - **Invalid bot token** — The Telegram or Discord token is wrong. Verify it in `.env`. - **Platform auth failure** — Check logs for `authentication failed` messages. **Fix:** 1. Verify `.env` has all required keys 2. Run `hermes gateway` in foreground to see real-time errors 3. If a platform token is invalid, regenerate it and update `.env` 4. Restart: `hermes gateway restart` Discord Silent (Bot Online but Not Responding) **Symptom:** The bot shows as online/active in your server but never responds to messages. **Almost always caused by missing Message Content Intent.** **Check:** 1. Go to [Discord Developer Portal](https://discord.com/developers/applications) 2. Select your application → **Bot** → **Privileged Gateway Intents** 3. Is **Message Content Intent** toggled ON? **Fix:** 1. Toggle **Message Content Intent** ON 2. Click **Save Changes** 3. Restart the gateway: `hermes gateway restart` **Still not

working?** Check: - Is your user ID in `DISCORD_ALLOWED_USERS`? - Are you mentioning the bot? (In channels, you need `@mention` unless free-response channels are configured) - Check the gateway log for authorization denials Telegram Not Responding **Symptom:** The bot is online but doesn't reply to messages. **Check:** tail -f ~/.hermes/logs/gateway.log **Common causes:** - **Wrong bot token** — Verify in `.env` - **User not authorized** — Is your user ID in `TELEGRAM_ALLOWED_USERS`? - **Privacy mode blocking** — If the bot is in a group, check privacy mode - **Rate limiting** — Telegram rate-limits bots that send too many requests. Wait a few minutes. **Fix:** 1. Send a message and watch the logs in real time 2. If you see `Not authorized`, add your user ID to the allowlist 3. If you see nothing at all, the bot token is likely wrong 4. Restart the gateway Circuit Breaker Paused an Adapter **Symptom:** A platform that was working suddenly stops. Log shows "paused-by-breaker". **Cause:** The adapter encountered repeated failures (network blips, rate-limit replies, websocket disconnects) and the circuit breaker tripped. **Fix:** # From another platform or CLI /platform resume <name> The breaker does **not** auto-resume — this prevents the gateway from thrashing reconnects during a sustained outage. Logs Show "Failed" but Gateway Seems Fine The gateway uses **circuit breakers** per adapter. Repeated retryable failures cause the breaker to trip. Check: 1. `~/.hermes/logs/gateway.log` — search for `circuit breaker`, `paused`, or `disabled` 2. `/platform list` output — shows current state and last reason 3. The platform's status page (Telegram API status, Discord status, etc.) Once upstream is healthy, use `/platform resume ` to re-arm the adapter. Summary - The **gateway** is a single daemon connecting Hermes to 20+ messaging platforms - Each platform has its own **adapter** — but all share the same agent, tools, memory, and skills - **Telegram** setup: BotFather → get token → `hermes gateway setup` → done - **Discord** setup: Developer Portal → create bot → **enable Message Content Intent** → invite → configure - Install as a **systemd service** for 24/7 operation: `hermes gateway install` - **Security** is multi-layered: allowlists, DM pairing, admin/user tiers, command approval - The **home channel** (`/sethome`) is where cron deliveries land - **Troubleshooting** starts with the gateway log: `~/.hermes/logs/gateway.log` In the next module, we'll deploy the gateway to a VPS for always-on, production-grade operation. Module 6 Dashboard Module 8

VPS Deployment

Back to Course Dashboard VPS Deployment Module 8 of 10 ■ Full Walkthrough Video Your browser doesn't support video playback. Multi-Platform Gateway All Modules Next: Monetization Strategies **Course:** Hermes Agent — From Zero to Autonomous Agent **Module:** 8 of 10 **Reading time:** ~20 minutes **Difficulty:** Intermediate TL;DR **Why a VPS** 24/7 uptime, gateway always online, cron jobs always run, you don't need your laptop turned on **Cost** $4–$6/month — less than a streaming subscription **Specs** 1 CPU, 1GB RAM, 25GB SSD — plenty for Hermes **Setup** 30 minutes, follow the exact commands below **Key steps** SSH in → create user → secure SSH → install Hermes → configure gateway → done Why a VPS? Running Hermes on your laptop works great for development and testing. But for a **production setup** — a bot that your team uses, cron jobs that need to fire at 3 AM, or an agent that monitors your infrastructure — you need something that stays on. A **Virtual Private Server (VPS)** is a small cloud server that runs 24/7. For the price of a coffee subscription, you get: - **Always-on gateway** — your Telegram/Discord bot never goes offline - **Always-on cron** — scheduled tasks fire at 3 AM whether your laptop is open or not - **No dependency on your machine** — you can shut down, travel, or switch computers - **Stable internet** — data centers have redundant networking, unlike home connections - **Room to grow** — add more users, more skills, more cron jobs without slowdown VPS Providers These are the most cost-effective providers for running Hermes. All work with the standard install. Provider Cheapest Plan Notes ---------- :----------: ------- **Hetzner** ~$4/month (€3.79) Best value. 1 CPU, 2GB RAM, 20GB SSD. Excellent network. **DigitalOcean** ~$6/month 1 CPU, 1GB RAM, 25GB SSD. Easy to use. Good documentation. **Vultr** ~$2.50/month Cheapest option. 1 CPU, 0.5GB RAM, 10GB SSD. Tight on RAM. **Linode (Akamai)** ~$5/month 1 CPU, 1GB RAM, 25GB SSD. Solid and reliable. **Recommendation:** Start with **Hetzner** ($4/month) or **DigitalOcean** ($6/month). The extra $2 gets you more RAM and a simpler control panel. Recommended Specs Component Minimum Recommended ---------- :-------: :----------: CPU 1 core

1 core RAM 512 MB 1 GB Storage 10 GB SSD 25 GB SSD OS Ubuntu 22.04 / 24.04 LTS Ubuntu 24.04 LTS 1 GB RAM is the sweet spot. Hermes itself is lightweight — the LLM calls happen on the provider's servers. The RAM mostly goes to caching sessions and running cron job agents. Initial Server Setup Let's walk through hardening a fresh Ubuntu VPS. These are standard security practices. Step 1: SSH into the Server ssh root@<your-server-ip> You should see the Ubuntu welcome message. The first time you log in, you may be prompted to change the root password. Step 2: Update the System apt update && apt upgrade -y This installs the latest security patches and package updates. Step 3: Create a Non-Root User Operating as `root` full-time is dangerous — a single mistake (or a compromised script) can destroy your system. Create a regular user with sudo privileges: adduser hermes You'll be prompted for a password and some optional info (Full Name, etc.). Then add the user to the sudo group: usermod -aG sudo hermes Step 4: Copy Your SSH Key Before logging out of root, copy your public SSH key to the new user so you can log in without a password: mkdir -p /home/hermes/.ssh cp ~/.ssh/authorized_keys /home/hermes/.ssh/ chown -R hermes:hermes /home/hermes/.ssh chmod 700 /home/hermes/.ssh chmod 600 /home/hermes/.ssh/authorized_keys Step 5: Test the New User Open a **new terminal** window (don't close the root session yet) and test: ssh hermes@<your-server-ip> If you can log in without a password prompt, everything worked. If not, check the `.ssh` permissions before continuing. Step 6: Disable Root Login Now edit the SSH configuration to block direct root access: sudo nano /etc/ssh/sshd_config Find these lines and change them: PermitRootLogin no PasswordAuthentication no - `PermitRootLogin no` — blocks direct root SSH access - `PasswordAuthentication no` — requires SSH key authentication (no password logins) Then restart SSH: sudo systemctl restart sshd **Before closing your current root session**, open a third terminal and verify you can still log in as `hermes`. If SSH breaks, you can fix it via the root session you kept open. Step 7: Set Up the Firewall (UFW) Ubuntu's Uncomplicated Firewall is simple and effective: sudo ufw default deny incoming sudo ufw default allow outgoing sudo ufw allow ssh sudo ufw allow 22/tcp If you plan to use Telegram webhook mode (inbound from Telegram), also allow that port: sudo ufw allow 8443/tcp Enable the firewall: sudo ufw --force enable sudo ufw status You should see: Status: active To Action From -- ------ ---- 22/tcp ALLOW Anywhere 22/tcp (v6) ALLOW Anywhere (v6) Step 8: Install Fail2Ban Fail2Ban protects against brute-force SSH attacks by temporarily banning IPs that fail too many login attempts: sudo apt install fail2ban -y sudo systemctl enable fail2ban sudo systemctl start fail2ban That's it — it works out of the box with Ubuntu's default configuration. Installing Hermes on the VPS Now that the server is secure, install Hermes. The One-Liner Install Log in as `hermes` and run: curl -fsSL https://hermes-agent.nousresearch.com/install.sh | bash The installer detects your OS, installs dependencies, and sets up Hermes. It will ask if you want to run the setup wizard — say yes. Setup Wizard hermes setup The wizard asks for: 1. **LLM Provider** — Choose your provider (OpenAI, Anthropic, OpenRouter, etc.) 2. **API Key** — Paste your API key 3. **Model** — Choose a default model 4. **Gateway setup** — Configure Telegram/Discord/etc. (optional now, can do later) Setting Up .env If you skip the wizard or need to edit later, the `.env` file lives at `~/.hermes/.env`: nano ~/.hermes/.env Your `.env` should contain at minimum: # LLM Provider (pick one) OPENAI_API_KEY=sk-... # or ANTHROPIC_API_KEY=sk-ant-... # or OPENROUTER_API_KEY=sk-or-... # Telegram (if using) TELEGRAM_BOT_TOKEN=123456789:ABCdef... TELEGRAM_ALLOWED_USERS=123456789 # Discord (if using) DISCORD_BOT_TOKEN=your-discord-token DISCORD_ALLOWED_USERS=284102345871466496 Verifying the Install hermes --version hermes doctor `hermes doctor` runs a comprehensive health check — provider connectivity, configuration validity, system resources. Gateway as a Systemd Service For a production VPS, install the gateway as a **system service** that starts at boot. Install as System Service sudo hermes gateway install --system This creates: - A systemd service file at `/etc/systemd/system/hermes-gateway.service` - Service runs as your `hermes` user - Starts automatically at boot - Auto-restarts on failure Enable Lingering sudo loginctl enable-linger hermes This ensures user services keep running after you log out. Start the Gateway sudo hermes gateway start --system Check Status sudo hermes gateway status --system Expected output shows `active (running)` and the service uptime. View Logs sudo journalctl -u hermes-gateway -f Stop and Restart sudo hermes gateway stop --system sudo hermes gateway restart --system SSH Key Management Generating a Key Pair If you don't already have an SSH key on your local machine: # On your local machine ssh-keygen -t ed25519

-C "hermes-vps" Copying the Key to the VPS ssh-copy-id hermes@<your-server-ip> Or manually: cat ~/.ssh/id_ed25519.pub | ssh hermes@<your-server-ip> "mkdir -p ~/.ssh && cat >> ~/.ssh/authorized_keys" Using an SSH Config Entry Create `~/.ssh/config` on your local machine: Host hermes-vps HostName <your-server-ip> User hermes IdentityFile ~/.ssh/id_ed25519 Then connect with: ssh hermes-vps Monitoring `hermes doctor` Run a health check anytime: hermes doctor This checks: - Provider connectivity (can it reach your LLM API?) - Gateway status (is the daemon running?) - Memory usage - Disk space - Network connectivity Health Check Cron Job Set up a cron job that checks the server's health and reports to your Telegram: /check_server_health Hermes will inspect CPU, RAM, disk, and gateway status, then optionally set up a recurring monitor. Or manually: hermes cron create "0 * * * *" "Check the server's CPU, RAM, and disk usage. Check if the gateway is running. If anything is abnormal, include a warning. Send to telegram." System Monitoring Standard Linux tools: # Real-time resource usage htop # Disk usage df -h # Memory free -h # Process list ps aux | grep hermes # Gateway logs sudo journalctl -u hermes-gateway -n 50 --no-pager Backups Your Hermes data is located in `~/.hermes/`. Back this up regularly. What to Back Up Item Path Why **Config** `~/.hermes/.env` API keys and platform tokens **Config** `~/.hermes/config.yaml` Settings and preferences **Skills** `~/.hermes/skills/` Your custom skills **Session data** `~/.hermes/session_store/` Conversation history (optional) **Memory** `~/.hermes/memory/` Persistent memory **Cron state** `~/.hermes/cron/` Cron job definitions **Gateway config** `~/.hermes/gateway.json` Platform-specific gateway settings Simple Backup Script Save this as `~/backup-hermes.sh` on the VPS: #!/bin/bash BACKUP_DIR="/home/hermes/backups" DATE=$(date +%Y-%m-%d) mkdir -p "$BACKUP_DIR" tar -czf "$BACKUP_DIR/hermes-config-$DATE.tar.gz" \ -C /home/hermes .hermes/.env \ -C /home/hermes .hermes/config.yaml \ -C /home/hermes .hermes/skills \ -C /home/hermes .hermes/gateway.json # Optional: include session data tar -czf "$BACKUP_DIR/hermes-full-$DATE.tar.gz" \ -C /home/hermes .hermes # Keep only last 30 days find "$BACKUP_DIR" -name "hermes-*.tar.gz" -mtime +30 -delete echo "Backup complete: $BACKUP_DIR/hermes-config-$DATE.tar.gz" Make it executable and run it: chmod +x ~/backup-hermes.sh ~/backup-hermes.sh Download Backups to Your Local Machine scp hermes-vps:~/backups/hermes-config-2025-01-01.tar.gz . Automate with a Cron Job crontab -e Add: 0 3 * * * /home/hermes/backup-hermes.sh This runs the backup daily at 3 AM. Updating Hermes Standard Update hermes update This pulls the latest version, applies any migrations, and asks if you want to restart the gateway. Update with Gateway Restart hermes update sudo hermes gateway restart --system Checking the Current Version hermes --version Cost Breakdown Here's a realistic monthly cost for running Hermes on a VPS: Item Cost Notes ------ :----: ------- **VPS** (Hetzner CX22) ~$4.00 1 CPU, 2GB RAM, 20GB SSD **LLM API** (OpenAI GPT-4o mini) ~$5–20 Depends on usage. Cheap for light use. **LLM API** (OpenAI GPT-4o) ~$20–100 Heavy use with large contexts. **Telegram** Free No cost for bots **Discord** Free No cost for bots **Domain** (optional) ~$1/month For webhook mode. Annual billing. **Total (light use)** **~$9–25/month** VPS + cheap model **Total (heavy use)** **~$25–105/month** VPS + premium model **Ways to reduce LLM costs:** - Use **OpenRouter** to dynamically route to cheaper models - Use **local models** via Ollama if you have GPU access - Set session reset policies to prevent context bloat - Use smaller models for routine cron jobs Complete Setup Cheat Sheet Run these commands in order on a fresh Ubuntu VPS: # === Initial server setup === ssh root@<server-ip> apt update && apt upgrade -y adduser hermes usermod -aG sudo hermes # Copy SSH key mkdir -p /home/hermes/.ssh cp ~/.ssh/authorized_keys /home/hermes/.ssh/ chown -R hermes:hermes /home/hermes/.ssh chmod 700 /home/hermes/.ssh chmod 600 /home/hermes/.ssh/authorized_keys # Disable root login (exit root session first, verify hermes user works) sudo sed -i 's/^PermitRootLogin.*/PermitRootLogin no/' /etc/ssh/sshd_config sudo sed -i 's/^#PasswordAuthentication.*/PasswordAuthentication no/' /etc/ssh/sshd_config sudo sed -i 's/^PasswordAuthentication.*/PasswordAuthentication no/' /etc/ssh/sshd_config sudo systemctl restart sshd # Firewall sudo ufw default deny incoming sudo ufw default allow outgoing sudo ufw allow ssh sudo ufw --force enable # Fail2Ban sudo apt install fail2ban -y sudo systemctl enable fail2ban sudo systemctl start fail2ban # === Install Hermes === # Log out and log back in as hermes curl -fsSL https://hermes-agent.nousresearch.com/install.sh | bash hermes setup # === Gateway setup === sudo hermes

gateway install --system sudo loginctl enable-linger hermes sudo hermes gateway start --system sudo hermes gateway status --system Summary - A **VPS** is essential for 24/7 operation — $4–6/month gets you plenty - **Harden the server** first: non-root user, SSH key only, firewall, Fail2Ban - **Install Hermes** with the standard one-liner — it works the same on a VPS - **Install the gateway as a system service** for boot-time start and auto-restart - **Monitor** with `hermes doctor` and a health check cron job - **Back up** `~/.hermes/` regularly — especially `.env` and `config.yaml` - **Update** with `hermes update` when new versions are available The total cost is typically **$5–10/month** for the VPS itself — less than a streaming subscription, and you get a personal AI agent that works for you around the clock. In the next module, we'll explore monetization strategies — how to turn your Hermes setup into a source of income. Module 7 Dashboard Module 9

Monetization Strategies

Back to Course Dashboard Monetization Strategies Module 9 of 10 ■ Full Walkthrough Video Your browser doesn't support video playback. VPS Deployment All Modules Next: Advanced Use Cases **Course:** Hermes Agent — From Zero to Autonomous Agent **Module:** 9 of 10 **Reading time:** ~25 minutes **Difficulty:** Beginner — business strategy, not technical TL;DR Strategy Effort Revenue Potential Timeline ---------- :------: :----------: :--------: Agent as a Service Medium $20–50/mo per user 2–4 weeks Local Business Automation High $500–5,000/setup 4–8 weeks Custom Skill Development Low $100–500/skill 1–2 weeks Affiliate Marketing Low $50–500/mo passive 4–12 weeks Sell This Course Medium $500–5,000/mo 4–8 weeks White-Label Hermes High $2K–5K setup + $100–200/mo 6–12 weeks SaaS Product Very High $10–50/mo per seat 12–24 weeks Introduction Hermes Agent isn't just a powerful productivity tool — it's a **business platform**. Because it's open-source (MIT license), infinitely customizable, and runs on cheap infrastructure, you can build income streams around it that would be impossible with closed platforms like ChatGPT or Claude. In this module, we'll cover seven concrete monetization strategies, from easiest to most ambitious. Each includes: - What you need to build - How to find customers - Revenue projections - Real-world examples Strategy 1: Sell Access to Your Agent as a Service **Difficulty:** Medium | **Revenue:** $20–50/mo per user | **Setup time:** 2–4 weeks This is the simplest model: set up Hermes on a VPS, connect it to Telegram or Discord, and charge people for access to your shared AI assistant. What You Offer A team or individual gets a Telegram/Discord bot that: - Answers questions with any LLM model - Runs code and shell commands - Searches the web - Edits files and manages projects - Remembers past conversations - Runs scheduled tasks Technical Setup 1. **VPS** — DigitalOcean ($6/mo) or Hetzner ($4/mo) 2. **Hermes** — Standard install + gateway 3. **Platform** — Telegram bot (easiest) or Discord bot 4. **Users** — Add them to `TELEGRAM_ALLOWED_USERS` or `DISCORD_ALLOWED_USERS` For a team: TELEGRAM_ALLOWED_USERS=111111,222222,333333,444444 Or use **DM pairing** so users self-register and you approve them: hermes pairing approve telegram XKGH5N7P Pricing Models Model Price Example ------- :-----: --------- Individual $20/mo One person, unlimited messages Team (up to 5) $50/mo Small dev team, shared bot Team (up to 20) $100/mo Department or startup Enterprise Custom Custom skills, dedicated VPS Finding Customers - **Developer communities** — Hacker News, Reddit (r/programming, r/SideProject), Dev.to - **Twitter/X** — Post about what your agent does. Show screenshots of results. - **Indie hacker forums** — IndieHackers, ProductHunt - **Freelance platforms** — Upwork, Fiverr, Contra - **Professional networks** — LinkedIn, Slack communities, Discord servers Cost Analysis Item Cost ------ :----: VPS $5–10/mo LLM API $0.50–2 per active user per month Total per 10 users ~$10–30/mo in costs Revenue at $20/user $200/mo **Gross profit** **$170–190/mo** The magic is that LLM costs are sub-linear — one API key serves all users, and many queries are cheap (cached responses, small models for simple questions). Example Pitch "I've built an AI assistant for developers. It lives in Telegram, remembers what you've worked on, can run code, search the web, and automate repetitive tasks. $25/month per person. Try it free for a week." Strategy 2: Automate Tasks for Local Businesses **Difficulty:** High | **Revenue:** $500–5,000/setup + $100–500/mo retainer | **Setup time:** 4–8 weeks Local businesses have repetitive digital tasks they pay humans to do. Hermes can do many of them automatically. What You Offer A concierge AI agent that handles: - **Customer inquiries** — Answer FAQs via

website widget or WhatsApp - **Review management** — Monitor Google/Yelp reviews, draft responses, send reports - **Social media** — Create and schedule posts, monitor engagement - **Inventory alerts** — Check supplier prices, notify when stock is low - **Appointment management** — Send reminders, reschedule, follow up - **Invoice follow-up** — Detect overdue invoices, draft payment reminders - **Competitor monitoring** — Weekly reports on competitors' prices and offers Technical Setup Each business gets a dedicated Hermes instance on a VPS: 1. **VPS** — $6/mo DigitalOcean droplet 2. **WhatsApp bot** — Hermes WhatsApp integration for customer communication 3. **Email gateway** — For receiving/sending automated emails 4. **Custom skills** — Business-specific automation logic 5. **Cron jobs** — Scheduled reports and monitoring Pricing Service Setup Fee Monthly Retainer --------- :---------: :----------------: Basic (1-2 automations) $500 $100/mo Standard (3-5 automations) $1,500 $250/mo Premium (full suite) $5,000 $500/mo Finding Customers - **Walk into local businesses** — Restaurants, auto shops, salons, dental clinics, real estate agents - **Cold email** — Find small businesses without modern systems - **Chamber of Commerce** — Join your local chamber and offer member discounts - **Google Maps** — Find businesses with outdated websites/no online presence - **Referral program** — Offer existing clients $100 referral bonus Example A **dental clinic** pays $2,000 setup + $300/mo for: - Automated appointment reminders via SMS (saves 15h/week of front desk time) - Weekly social media posts with dental tips (generated by Hermes) - Monthly competitor analysis of nearby clinics' pricing - Automated review responses on Google Maps - Daily report of no-shows and reschedules **The clinic saves** ~$1,200/mo in labor costs. **You earn** $300/mo recurring. Win-win. Strategy 3: Build and Sell Custom Skills **Difficulty:** Low | **Revenue:** $100–500/skill | **Setup time:** 1–2 weeks Hermes skills are modular, sharable, and easy to create. If you can identify a repeatable workflow, package it as a skill and sell it. What You Offer A ready-to-use Hermes skill that handles a specific task: - **Code review skill** — Consistent PR review with team-specific guidelines - **SEO audit skill** — Crawls a URL, generates actionable SEO recommendations - **Legal document summarizer** — Extracts key clauses from contracts - **Competitive intelligence skill** — Monitors competitors and generates reports - **Crypto portfolio tracker** — Daily summaries of positions and news - **Real estate lead qualifier** — Analyzes property listings and scores leads - **Recipe generator / meal planner** — With dietary restriction handling - **Resume screener** — Analyzes resumes against job descriptions Technical Setup Skills are just markdown files. Create `~/.hermes/skills/your-skill/SKILL.md`: --- name: seo-audit description: Crawls a URL and generates actionable SEO recommendations --- # SEO Audit Guidelines When asked to audit a URL... 1. Fetch the page content 2. Check: title tag, meta description, H1-H3 structure, image alt text 3. Check: page speed (CLS, LCP, FID metrics via Lighthouse) 4. Check: keyword density, internal/external links 5. Generate a report with: score out of 100, top 3 issues, quick wins Package it for distribution: tar -czf seo-audit-skill.tar.gz -C ~/.hermes/skills seo-audit/ Pricing Skill Complexity Price :----------------: :-----: Simple (10–30 lines) $100–200 Medium (30–100 lines) $200–350 Complex (100+ lines, integrations) $350–500 Marketing Channels - **Hermes community** — GitHub Discussions, Discord, Reddit - **Skill marketplaces** — Gumroad, Sellfy, Lemonsqueezy - **Developer tool directories** — Awesome lists, GitHub topics - **Blog posts** — "How I automated [X] with Hermes" → link to your skill - **YouTube tutorials** — Walk through the problem → show the skill → sell Example You build a **blog content automation skill** that: 1. Scrapes trending topics in a niche 2. Generates an article outline 3. Writes the first draft 4. Creates social media snippets 5. Generates an SEO meta description Price: **$299**. Sell to bloggers, content marketers, and agency owners. Even if you only sell 10 copies, that's **$2,990** for a weekend of work. Strategy 4: Affiliate Marketing **Difficulty:** Low | **Revenue:** $50–500/mo passive | **Setup time:** 4–12 weeks This is the lowest-effort option. Create content about Hermes Agent, include affiliate links to VPS providers and LLM APIs, and earn commissions. What You Promote Service Commission Why It Fits --------- :----------: ---------- **DigitalOcean** $25–100 per referral Every VPS guide mentions them **Hetzner** ~€10 per referral Best value VPS for Hermes **Vultr** $10–50 per referral Another VPS option **OpenRouter** 5–10% recurring LLM API aggregator **Firecrawl** Commission varies Web search API for Hermes **OpenAI / Anthropic** None directly, but referral programs exist Mention in course content **Domain registrars** $1–5 per referral For webhook setups Content Ideas - **Blog posts** — "How to Set Up Your Own AI Agent in 10 Minutes" - **YouTube videos** — "I Built a Free AI Assistant (ChatGPT

Alternative)" - **Twitter threads** — "5 Things My AI Agent Does for Me Every Day" - **GitHub repos** — Curated list of Hermes skills and configurations - **Courses** — This course! Include affiliate links for VPS and API services - **Reddit posts** — Answer questions about self-hosted AI Expected Revenue Traffic Conversion Monthly Revenue :-------: :----------: :---------------: 1,000 visitors/mo 1% ~$50–100 10,000 visitors/mo 1% ~$500–1,000 100,000 visitors/mo 1% ~$5,000–10,000 Strategy 5: Create and Sell THIS Course **Difficulty:** Medium | **Revenue:** $500–5,000/mo | **Setup time:** 4–8 weeks You're reading the result — Hermes is a deep product with a growing user base, and there's no comprehensive course. Fill that gap. What You Offer A structured learning path: - **Beginner track** — What is Hermes, installation, basic usage (free or low-cost) - **Intermediate track** — Skills, training, gateway, VPS deployment (paid) - **Advanced track** — Plugin development, multi-agent systems, custom integrations (premium) Platforms Platform Pros Cons ---------- :----: :----: **Gumroad** Easy setup, handles payments Limited course features **Teachable** Full course platform Monthly fee **Udemy** Built-in audience 50%+ revenue share **Your own site** Full control You do all marketing **Lemonsqueezy** Developer-friendly, global tax compliance No course features Pricing Models Model Price Example ------- :-----: --------- One-time purchase $50–200 Full course access forever Subscription $10–30/mo Access to all modules + updates Tiered Free + $49 + $149 Starter / Pro / Enterprise Bundled $197 Course + custom skills pack Marketing - **GitHub** — Publish the course outline as a repo. Collect emails. - **Blog** — Publish one module for free. Gate the rest. - **YouTube** — Video version. Monetize with ads + course sales. - **Communities** — Share in Hermes Discord, Hacker News, Reddit - **Affiliate program** — Give 30% commissions to promoters Strategy 6: White-Label Hermes for Clients **Difficulty:** High | **Revenue:** $2K–5K setup + $100–200/mo maintenance | **Setup time:** 6–12 weeks Businesses want AI assistants but don't want to manage servers, API keys, or open-source software. White-label Hermes — brand it as their AI assistant and manage everything for them. What You Offer A branded AI assistant for their company: - **Their logo, their name** — Bot profile, welcome messages - **Custom knowledge base** — Their documentation, FAQ, SOPs - **Custom tools** — Integration with their CRM, support desk, database - **Shared team access** — Per-user authorization - **SLA** — Guaranteed uptime, response time - **Monthly reports** — Usage stats, common questions, cost analysis Technical Setup 1. **Dedicated VPS** per client (or isolated instances) 2. **Custom skill** per client containing their knowledge 3. **Branded Telegram/Discord bot** with custom name and avatar Pricing Tier Setup Monthly What's Included :----: :-----: :-------: ---------- Basic $2,000 $100/mo Telegram bot, basic knowledge, email support Standard $3,500 $150/mo Discord + Telegram, custom skills, priority support Premium $5,000 $200/mo Full platform support, custom integrations, SLA Finding Clients - **Target B2B** — Agencies, law firms, real estate brokerages, e-commerce stores - **Offering to other agencies** — "I can add AI assistant capabilities to your service offerings" - **LinkedIn outreach** — Connect with CTOs, Heads of Engineering, Operations Directors - **Referral from Strategy 2 clients** — They tell other business owners Example A **real estate agency** pays $3,500 setup + $150/mo for: - Branded WhatsApp bot for client inquiries - Automated property listing summaries - Lead qualification (scores leads based on budget/preferences) - Appointment scheduling and reminders - Monthly market analysis report **Their alternative:** Hire a full-time assistant ($40K+/yr). Hermes costs them $2,300/yr. Strategy 7: SaaS Product **Difficulty:** Very High | **Revenue:** $10–50/mo per seat | **Setup time:** 12–24 weeks Build a multi-tenant SaaS platform where users sign up, connect their Telegram/Discord, and get a personal AI agent. What You Offer A fully managed, multi-tenant Hermes service: - **Self-service signup** — Users create accounts, no sales call needed - **Free trial** — 7 days with limited messages - **Tiered plans** — Based on messages, users, features - **Multi-platform** — Users choose Telegram, Discord, Slack, or WhatsApp - **Admin dashboard** — Usage stats, billing, team management Technical Architecture Each user or team gets an isolated Hermes agent: User signs up ■→ Gateway creates a new Hermes profile → Profile has its own sessions, skills, cron jobs → All traffic routed through a shared gateway with tenant isolation → Billing metered by API calls / messages For true isolation, use Docker per tenant: version: '3' services: hermes-tenant-1: image: hermes-agent environment: - HERMES_HOME=/data/tenant-1 - OPENAI_API_KEY=tenant-1-key volumes: - /data/tenant-1:/data Pricing Model Plan Price Limits :----: :-----: -------- Free $0 50 messages/mo, 1 user Starter $10/mo 500 messages/mo, 3

users Pro $25/mo 2,000 messages/mo, 10 users Team $50/mo 10,000 messages/mo, unlimited users Enterprise Custom Custom limits, dedicated infra Estimated Costs per 100 Users Item Cost Notes ------ :----: ------- VPS (4 instances) $60/mo Hetzner or DO LLM API $50–200/mo Varies by usage Domain + email $20/mo Payment processing 3% + $0.30/transaction Stripe **Total** **~$130–280/mo** At 100 users, even at $10/mo: **$1,000/mo revenue** → **~$2–3k/mo at $25 avg.** Building the SaaS 1. **Start as a service** — Manually onboard first 10 customers (Strategy 1) 2. **Automate** — Build a simple signup flow that provisions new instances 3. **Scale** — Add billing, dashboard, self-service Real Revenue Projections Here's what each strategy can realistically generate at different stages: Strategy Month 1–3 Month 4–6 Month 7–12 Year 2+ ---------- :---------: :---------: :----------: :--------: Agent as a Service $200–500 $500–2,000 $1,000–5,000 $2,000–15,000 Local Business Automation $500–2,000 $1,000–5,000 $3,000–10,000 $5,000–25,000 Custom Skills $0–500 $500–2,000 $1,000–5,000 $2,000–10,000 Affiliate Marketing $0–100 $100–500 $200–1,000 $500–5,000 Sell This Course $200–1,000 $1,000–3,000 $2,000–10,000 $3,000–20,000 White-Label $0–5,000 $2,000–10,000 $5,000–25,000 $10,000–50,000 SaaS Product $0 $500–2,000 $2,000–10,000 $10,000–100,000+ Reality Check - **Most of these require active marketing.** Building the product is half the work — selling it is the other half. - **Start with one strategy.** Don't try all seven. Pick the one that fits your skills and audience. - **Combo strategies work best.** Sell Agent as a Service → upsell white-label. Create a course → promote VPS affiliate links. - **Customer support takes time.** Every additional user means more questions, bugs, and requests. - **LLM costs scale with usage.** Monitor them. Set rate limits. Use cheaper models for routine tasks. What Works Best for Different Backgrounds Your Background Best Strategy Developer Agent as a Service, Custom Skills, SaaS Freelancer / Consultant Local Business Automation, White-Label Content Creator Affiliate Marketing, Sell This Course Agency Owner White-Label, Local Business Automation Technical Marketer Affiliate Marketing, Sell This Course Getting Started Today Don't overthink it. Here's your 7-day launch plan: **Day 1:** Choose one strategy from this module **Day 2:** Set up Hermes on a $6 VPS (use Module 8) **Day 3:** Configure Telegram/Discord (use Module 7) **Day 4:** Build your first paying offering **Day 5:** Tell 10 people about it (friends, forums, social media) **Day 6:** Get feedback, adjust pricing **Day 7:** Launch Your first customer is the hardest. After that, it compounds. Summary - **Agent as a Service** — Simplest entry. $20–50/mo per user. Start here. - **Local Business Automation** — Highest per-client revenue. $500–5,000 setup. - **Custom Skills** — Low effort, good for developers. $100–500/skill. - **Affiliate Marketing** — Passive, low effort. Best as a supplement. - **Sell This Course** — Position yourself as an expert. $50–200/course. - **White-Label** — High ticket, high effort. Best for agencies. - **SaaS** — Highest ceiling, hardest execution. Build toward this. The common thread: **Hermes is open-source, customizable, and cheap to run.** That combination is rare and valuable. People will pay for access, automation, expertise, and convenience built on top of it. In the next module, we'll explore advanced use cases — trading bots, content automation, code review agents, research assistants, and more. Module 8 Dashboard Module 10

Advanced Use Cases

Back to Course Dashboard Advanced Use Cases Module 10 of 10 ■ Full Walkthrough Video Your browser doesn't support video playback. Monetization Strategies All Modules **Course:** Hermes Agent — From Zero to Autonomous Agent **Module:** 10 of 10 **Reading time:** ~25 minutes **Difficulty:** Advanced TL;DR Use Case What It Does Key Tools **Trading Bot** Automated crypto/stock monitoring and execution Cron jobs, API calls, terminal **Content Automation** AI script → voiceover → video → upload pipeline Skills, terminal, API calls **Code Review Bot** Auto-review PRs via webhooks or cron Webhooks, terminal, cron **Customer Support Agent** Automated support across Telegram/Discord Gateway, allowlists, skills **Research Assistant** Daily digests, paper summaries, trend analysis Cron, skills, web search **Personal Assistant** Email triage, calendar, todo management API server, skills, cron **Multi-Agent Systems** Delegated task execution with sub-agents Delegation, `/background` **Home Automation** Voice-controlled smart home via Home Assistant Home Assistant adapter **CI/CD Integration** Automated deployment pipeline assistant Webhooks, terminal **Data Pipeline** ETL, reporting, visualization automation Cron, skills, Python execution Use Case 1: Trading Bot Build

an automated trading bot that monitors markets, analyzes trends, and executes trades on your behalf. Setup You'll need API access to a trading platform. Hermes supports any exchange with a REST API via the terminal tool. **Bybit example:** # Add API keys to .env BYBIT_API_KEY=your_api_key BYBIT_API_SECRET=your_api_secret Create a cron job for market monitoring: / Monitor BTC price every 30 minutes. If price drops >3% in 24h, check RSI and volume. If RSI < 30 and volume > 2x average, alert me on Telegram with a buy recommendation. **Alpaca (stocks) example:** # Add API keys to .env ALPACA_API_KEY=your_api_key ALPACA_SECRET_KEY=your_api_secret Create a cron job: Every morning at 9:30 AM EST, check my Alpaca portfolio. Show: current positions, P&L, top gainers/losers. If any position is down >5%, recommend whether to hold or sell. Expected Results Task Output Price monitoring Alerts with technical analysis Portfolio summary Daily P&L report sent to Telegram Backtesting Historical performance analysis Execution Paper trading (always test before going live!) Safety Considerations - **Never give the agent full trading permissions** — Use read-only or whitelisted API keys - **Paper trade first** — Run in simulation mode for at least a month - **Set position limits** — Hard-code maximum trade sizes - **Human-in-the-loop** — Use command approval for actual trades - **Monitor costs** — Frequent API calls can add up Use Case 2: Content Automation Build an end-to-end content pipeline: Hermes writes a script, generates a voiceover, creates visuals, renders a video, and uploads it to YouTube. The Pipeline Cron job fires → Agent writes script → TTS generates voiceover → Python/moviepy renders video → Upload via YouTube API → Post link to Telegram/Discord Setup Create a content skill at `~/.hermes/skills/content-creator/SKILL.md`: --- name: content-creator description: Creates AI-generated video content from topic prompts --- # Content Creator Guidelines When creating a video from a topic: 1. Research the topic via web search 2. Write a 2-minute script (300 words, conversational tone) 3. Generate TTS audio using the configured voice 4. Find or generate relevant visuals 5. Use moviepy to compose the video with captions 6. Save to ~/content/output/ Cron Job Create a daily content job: Every morning at 6 AM, create a 2-minute explainer video about a trending AI news story from the past 24 hours. Use a professional, accessible tone. Save the video to ~/content/output/ and post the link to Telegram. Required Tools Tool Purpose Installation **ffmpeg** Audio/video processing `sudo apt install ffmpeg` **moviepy** Python video editing `pip install moviepy` **TTS** Voice generation Built into Hermes via configured provider **OpenAI/Anthropic** Script writing API key in `.env` Example Workflow 1. Cron fires at 6 AM 2. Agent searches for top AI news 3. Writes a script with hook, body, and CTA 4. Generates voiceover via TTS 5. Creates captions/subtitles 6. Renders video with background music 7. Saves to output directory 8. Posts link to Telegram Use Case 3: Code Review Bot Automatically review pull requests in your GitHub repositories. Two approaches: Approach A: Cron-Based (Simple) Poll for open PRs on a schedule. See Module 7's GitHub PR review guide for the full setup. # Cron job: every 2 hours, check for new PRs and review them hermes cron create "0 */2 * * *" \ "Check for new open PRs in myorg/backend-api.\n\n\ For each PR opened or updated in the last 4 hours:\n\ 1. Fetch the diff\n\ 2. Review for bugs, security issues, code quality\n\ 3. Format findings with file:line, severity, and fix suggestion\n\n\ If no new PRs, say: No new PRs to review." \ --skill code-review \ --deliver telegram Approach B: Webhook-Based (Real-time) Set up a webhook endpoint that GitHub pushes events to when PRs are opened or updated. 1. Configure the webhook adapter in `~/.hermes/gateway.json`: { "platforms": { "webhook": { "enabled": true, "extra": { "port": 9000, "endpoints": { "/github-pr": { "description": "GitHub PR review webhook", "allowed_senders": ["github.com"], "script": "review_pr" } } } } } } 2. Create a review script that Hermes executes when a webhook fires 3. Configure GitHub to send webhook events to `http://your-server:9000/github-pr` **Pros:** No polling delay. Reviews happen seconds after a PR is opened. **Cons:** Requires a public endpoint (port forwarding, or deploy on a cloud host). Expected Results - Every PR gets a review within minutes (webhook) or hours (cron) - Reviews include specific line numbers and code snippets - Team receives summary in Telegram/Discord - Catch bugs before they reach production Use Case 4: Customer Support Agent Set up a Telegram or Discord bot that handles customer support for your product or service. Setup 1. Create a skill with your product's knowledge base: ~/.hermes/skills/support/SKILL.md Include: - Product documentation and FAQs - Known issues and workarounds - Escalation criteria - Brand voice and tone guidelines 2. Configure the bot with appropriate authorization: # Allow all customers to message the bot GATEWAY_ALLOW_ALL_USERS=true #

But restrict dangerous commands # In config.yaml: gateway: platforms: telegram: extra: allow_admin_from: ["your-user-id"] user_allowed_commands: [help, status] Features Feature Implementation FAQ answering Skill with product knowledge Ticket creation Agent writes to your support system's API Order lookup API integration via terminal tool Escalation `/escalate` command starts a support ticket Status checks Cron job checks service health Example # User messages: "My order hasn't arrived in 2 weeks" # Hermes checks the order system, finds the tracking info: # "I can see order #12345 is with USPS (tracking: 1Z999AA10123456784). # Last scan was in Memphis distribution center. Would you like me to # contact USPS support? Or I can escalate this to our team." Use Case 5: Research Assistant An automated researcher that monitors topics, summarizes papers, and delivers digests. Daily Digest # Every morning at 7 AM hermes cron create "0 7 * * *" \ "Create a research digest:\n\n\ 1. Search arxiv for new papers about: large language models, AI agents\n\ 2. Search PubMed for: AI in healthcare\n\ 3. Search tech news for: AI industry developments\n\ 4. For each source, summarize the top 3 items\n\ 5. Format as a clean digest with sections and links\n\n\ Focus on papers and news from the past 24 hours only." \ --deliver telegram Paper Summary on Demand # Send this to your bot: "Summarize this paper: [arxiv link]. Focus on: methodology, results, limitations. Compare it to the previous work in this area. End with: should I read the full paper? (yes/no + 1 sentence reason)" Configurable Research Briefings Create multiple cron jobs for different topics: # AI research (weekdays only) "Cron: 0 8 * * 1-5 — AI digest" # Crypto (daily) "Cron: 0 9 * * * — Crypto news and market analysis" # Industry competitors (weekly) "Cron: 0 10 * * 1 — Weekly competitor analysis" Use Case 6: Personal Assistant Use Hermes as a full personal assistant for email triage, calendar management, and task tracking. Email Triage Via the **API server** or **webhook** adapter, Hermes can receive and process emails: 1. Gateway receives email 2. Agent reads it and determines: important?, requires action?, spam? 3. Responds with: summary, priority, suggested action 4. Delivers digest to Telegram Setup via email adapter: # In .env EMAIL_ENABLED=true EMAIL_ADDRESS=assistant@yourdomain.com EMAIL_PASSWORD=your_app_password EMAIL_ALLOWED_SENDERS=trusted@example.com Calendar & Todo Hermes can manage a todo list in a skill file: # Todo skill — stores tasks in ~/todo.md When I say "remind me to X": 1. Add to ~/todo.md with date 2. Confirm with "Added: X" 3. Create a cron job if there's a time: "Remind me at [time]: X" Daily check-in: Every morning at 8 AM, read my todo list, check my calendar, and give me a summary of what's scheduled and what's due today. Use Case 7: Multi-Agent Systems Hermes supports **delegation** — spawning sub-agents that work on tasks in parallel. How Delegation Works Within any session, you can tell Hermes to delegate tasks to sub-agents. Each sub-agent gets an isolated session and runs independently. You: "Research three topics and combine the results" Main agent: ■■■ → Sub-agent 1: "Research topic A" ■■■ → Sub-agent 2: "Research topic B" ■■■ → Sub-agent 3: "Research topic C" ← Collects all results ← Combines into one response Use Cases Task Structure Market research Each competitor gets its own sub-agent Code analysis Each file/repo gets its own sub-agent Writing Outline → draft → review in parallel Data processing Each data source gets its own sub-agent Via `/background` The `/background` command spawns a separate agent instance: /background Check all servers in the cluster and report any that are down The background agent runs independently — you can keep chatting while it works. Results arrive in the same chat when complete. Multi-Agent Kanban Board Hermes ships with a **kanban plugin** for multi-agent orchestration: hermes-plugin-kanban-install This provides a visual board where you assign tasks to agents, track progress, and manage complex workflows. Use Case 8: Home Automation via Home Assistant Hermes integrates with **Home Assistant** to give you voice-controlled smart home management through Telegram or Discord. Setup Configure the Home Assistant platform: # In .env HOMEASSISTANT_ENABLED=true HOMEASSISTANT_BASE_URL=http://192.168.1.100:8123 HOMEASSISTANT_ACCESS_TOKEN=your_long_lived_token The Home Assistant adapter adds special tools: - `ha_list_entities` — List all Home Assistant entities - `ha_get_state` — Get the state of an entity - `ha_call_service` — Call a Home Assistant service (turn on/off, set temp, etc.) - `ha_list_services` — List available services Examples "Turn off all lights in the living room" "Set the thermostat to 72 degrees" "What's the temperature outside?" "Lock the front door and arm the alarm" "Good night" → Agent runs a scene: lights off, doors locked, alarm on Automations with Cron Every day at sunset: close the garage door if it's open Every

morning at 7 AM: turn on the coffee maker and set lights to 50% When I leave home (detected by phone): turn off all lights and AC Use Case 9: CI/CD Integration Integrate Hermes into your CI/CD pipeline for automated deployments, monitoring, and rollback decisions. Webhook-Triggered Deployments When a CI/CD pipeline (GitHub Actions, GitLab CI, Jenkins) completes: 1. Pipeline sends webhook to Hermes 2. Hermes checks the build output, test results, and deployment logs 3. If tests pass and metrics are healthy → approve deployment 4. If tests fail or metrics degrade → notify the team with details Monitoring Cron Every 5 minutes, check the production server health: - Is the app responding on port 443? - Is the database connection pool under 80%? - Is the error rate below 0.1%? - Is memory usage under 80%? If any check fails, alert the team in Telegram with the details. Automated Rollback When a deployment causes issues: If error rate spikes above 5% within 10 minutes of a deployment: 1. Run: kubectl rollout undo deployment/app 2. Verify the previous version is serving traffic 3. Notify the team: "Rolled back deployment v1.2.3 due to error rate spike" Use Case 10: Data Pipeline (ETL, Reporting, Visualization) Use Hermes as an automated data processing engine that extracts, transforms, and visualizes data on a schedule. ETL Pipeline Cron job fires: 1. Extract: Download CSV from API or database 2. Transform: Clean data, compute metrics, aggregate 3. Load: Write results to a database or generate a report Delivery: - Send summary to Telegram - Save full report to local files - Generate charts and send them as images Example: Sales Dashboard hermes cron create "0 7 * * 1" \ "Generate a weekly sales report:\n\n\ 1. Run: python3 ~/scripts/extract_sales_data.py\n\ 2. Calculate: total revenue, top products, YoY growth\n\ 3. Create a bar chart comparing this week vs last week\n\ 4. Create a pie chart of revenue by product category\n\ 5. Save charts to ~/weekly-reports/[date]/\n\ 6. Write a summary with key numbers and insights\n\ 7. Send the summary + charts to Telegram" \ --deliver telegram Visualization Tools Tool Purpose Install **matplotlib** Python charts `pip install matplotlib` **pandas** Data manipulation `pip install pandas` **seaborn** Statistical charts `pip install seaborn` **plotly** Interactive charts `pip install plotly` **SQLite** Local database Built into Python Example: KPI Dashboard A cron job that runs every hour: 1. Queries your database for key metrics 2. Checks them against thresholds 3. If any metric is outside acceptable range: - Creates a chart showing the trend - Sends an alert to Telegram with the chart - Logs the incident for later review Combining Use Cases The real power of Hermes is combining multiple use cases into a single agent. Example: Full-Tech-Stack Automation A single Hermes instance running on a $6 VPS: - **Trading bot** (monitors crypto markets, alerts on opportunities) - **Code reviewer** (reviews team PRs every 2 hours) - **CI/CD monitor** (watches deployments, handles rollbacks) - **Home automation** (controls lights and thermostat via Telegram) - **Daily briefing** (news, weather, server health every morning) - **Research assistant** (weekly paper digest on AI topics) - **Personal assistant** (todo list, reminders, email triage) All from one gateway. All delivering to your Telegram. The Limit Is Your Imagination Hermes is a platform, not just a tool. The 10 use cases in this module are starting points — you can combine, extend, and customize them infinitely. Some ideas to explore on your own: - **Bot network** — Multiple Hermes instances, each specialized, communicating via webhooks - **Customer onboarding** — Automated Telegram bot that walks new users through your product - **Compliance monitoring** — Watch logs for PII leaks, security policy violations - **Content moderation** — Auto-flag inappropriate content in your community - **Language learning partner** — Chat with Hermes in your target language with corrections - **Virtual D&D DM** — Generative dungeon master for your party - **Personal tutor** — Analyzes your learning history and generates practice problems - **Automated pentesting** — Scheduled security scans with report generation Course Wrap-Up You've reached the end of the Hermes Agent course. Let's recap what you've learned: Module Topic :------: ------- 1 What Hermes is and why it matters 2 Installing Hermes on any platform 3 Configuration, providers, models 4 Basic usage, tools, terminal workflows 5 Building and training custom skills 6 Teaching and improving skills over time 7 Multi-platform messaging gateway 8 Production VPS deployment 9 Monetization strategies 10 Advanced use cases Next Steps 1. **Deploy** — Set up Hermes on a VPS (Module 8) 2. **Connect** — Wire up Telegram or Discord (Module 7) 3. **Build** — Create your first custom skill (Module 5) 4. **Automate** — Schedule your first cron job (Module 6) 5. **Monetize** — Pick a strategy and start (Module 9) 6. **Explore** — Try the advanced use cases that interest you Hermes is MIT-licensed, open-source, and maintained by Nous Research. The community is active on GitHub and Discord. Join us, share what you build, and help shape the

future of autonomous AI agents. Go build something amazing. Module 9 Dashboard