Hermes Agent 완벽 가이드

> Nous Research가 만든 자기개선형(self-improving) AI 에이전트의 모든 기능을 공식 문서와 GitHub 릴리스 노트만을 근거로 정리한 종합 안내서입니다. 본 문서의 모든 내용은 hermes-agent.nousresearch.com/docs와 github.com/NousResearch/hermes-agent의 공식 자료에 기반합니다.

목차

본 문서는 Hermes Agent 공식 사이드바의 구성을 그대로 따라 작성되었으며, 다음과 같은 흐름으로 구성됩니다. 먼저 Hermes Agent가 무엇이고 왜 만들어졌는지 큰 그림을 잡고, 그다음 핵심(Core), 자동화(Automation), 미디어 및 웹(Media & Web), 관리(Management), 고급(Advanced), 스킬(Skills), 메시징 플랫폼, 통합(Integrations), 마지막으로 GitHub 릴리스 노트의 주요 변경사항 순으로 살펴봅니다.

---

1. Hermes Agent란 무엇인가

Hermes Agent는 Nous Research에서 MIT 라이선스로 공개한, **자기개선형(self-improving) AI 에이전트**입니다. 단순한 대화형 AI를 넘어, 한 번 설치하면 사용자의 컴퓨터(또는 VPS, Termux를 통한 Android 기기 등)에서 백그라운드로 상주하면서 19개 이상의 메시징 플랫폼·터미널·웹 대시보드·IDE를 통해 접근할 수 있는 종합 자동화 인프라입니다. 사용자가 작업을 시키는 동안 에이전트는 작업하면서 배운 패턴을 **스킬(skill)**로 자동 저장하고, **메모리(memory)**에 사용자의 선호와 환경 정보를 누적해 다음 대화에서 자연스럽게 활용합니다.

Nous Research 공식 문서의 User Stories 섹션에 따르면 99건 이상의 실제 사용 사례가 GitHub, X, Reddit, Hacker News, YouTube, 블로그 등에서 수집되어 있으며, 그 분야는 개발 워크플로(16건), 개인 비서(16건), 통합(15건), 메타·생태계(10건), 비즈니스 운영(8건), 엔터프라이즈(6건), 콘텐츠 제작(5건), 연구(4건), 메시징(4건), 비용 최적화(4건), 트레이딩(3건), 창의 작업(3건), 프라이버시·자체 호스팅(3건), 마케팅(1건), 일반(1건)에 걸쳐 있습니다. 이는 Hermes Agent가 단일 용도가 아니라 사용자가 어떤 도메인에서 일하든 그 워크플로를 학습하여 적응하는 범용 인프라임을 보여줍니다.

Hermes Agent의 핵심 철학 세 가지

첫째, **자기개선(Self-Improvement)**입니다. 에이전트가 복잡한 작업을 5번 이상의 도구 호출로 해결한 뒤에는 그 워크플로를 `~/.hermes/skills/` 아래에 SKILL.md 파일로 저장하여 다음에 비슷한 요청이 들어왔을 때 즉시 재사용할 수 있도록 합니다. 또한 메모리 시스템(MEMORY.md, USER.md)을 통해 사용자의 선호, 프로젝트 구조, 환경 정보를 지속적으로 누적합니다.

둘째, **어디서든 접근 가능(Everywhere)**입니다. CLI, TUI, 19개 메시징 플랫폼, 웹 대시보드, IDE(VS Code/Zed/JetBrains), API 서버, Webhook 엔드포인트 등 사용자가 어디에 있든 동일한 에이전트와 동일한 메모리·스킬에 접근할 수 있습니다.

셋째, **실용적이고 비용 효율적(Practical & Cost-Effective)**입니다. Anup Karanjkar의 Medium 블로그에 따르면 OpenClaw 대비 약 90%의 토큰 비용 절감이 보고되었고, 하나의 ChatGPT Plus 구독 가격($20/월) 미만의 VPS만으로 24시간 가동되는 개인 AI 어시스턴트를 운영할 수 있습니다. 또한 Tinker-Atropos 기반의 RL 학습 파이프라인까지 내장하여 AI 연구·실험에도 활용 가능합니다.

> 출처: User Stories & Use Cases, Features Overview

---

2. 전체 기능 개요

공식 Features Overview 페이지는 Hermes Agent의 기능을 다섯 가지 큰 카테고리로 나눕니다. 이를 빠르게 훑은 뒤 이후 섹션에서 각각을 자세히 다루겠습니다.

Core(코어) 카테고리에는 도구와 도구셋(Tools & Toolsets), 스킬 시스템(Skills System), 영속 메모리(Persistent Memory), 컨텍스트 파일(Context Files), 컨텍스트 참조(Context References), 체크포인트(Checkpoints)가 포함됩니다. 이들은 에이전트가 무엇을 할 수 있고 무엇을 기억하는지를 정의하는 가장 기초적인 구성 요소입니다.

Automation(자동화) 카테고리에는 스케줄 작업(Cron), 서브에이전트 위임(Delegation), 코드 실행(Code Execution), 이벤트 훅(Hooks), 배치 처리(Batch Processing)가 포함됩니다. 에이전트가 사람의 개입 없이 스스로 일을 진행하고, 다른 에이전트를 호출하고, 외부 이벤트에 반응하는 능력을 다룹니다.

Media & Web(미디어 및 웹) 카테고리에는 음성 모드(Voice Mode), 브라우저 자동화(Browser), 비전 및 이미지 붙여넣기(Vision), 이미지 생성(Image Generation), 음성 및 TTS(Voice & TTS)가 포함됩니다. 텍스트를 넘어 음성·이미지·웹 페이지를 다루는 멀티모달 기능군입니다.

Integrations(통합) 카테고리에는 MCP 통합, 프로바이더 라우팅, 폴백 프로바이더, 크리덴셜 풀, 메모리 프로바이더, API 서버, IDE 통합(ACP), RL 학습이 포함됩니다. 외부 서비스와 LLM 제공자, 다른 에이전트와 어떻게 연결되는지를 정의합니다.

Customization(커스터마이징) 카테고리에는 인격 및 SOUL.md, 스킨 및 테마, 플러그인이 포함됩니다. 에이전트의 정체성, 시각적 표현, 확장성을 사용자가 직접 정의할 수 있게 합니다.

> 출처: Features Overview

---

3. Tool Gateway: Nous 구독 통합 도구

Tool Gateway는 유료 Nous Portal 구독자를 위한 통합 도구 게이트웨이입니다. 일반적으로 웹 검색·이미지 생성·TTS·브라우저 자동화를 사용하려면 각각 Firecrawl, FAL, OpenAI, Browser Use에 별도의 API 키를 등록해야 했지만, Tool Gateway를 활성화하면 Nous 구독 하나로 이 네 가지 도구를 모두 사용할 수 있습니다.

포함된 도구는 다음과 같습니다. **웹 검색 및 추출**(Firecrawl 기반)으로 웹페이지 내용을 검색하고 추출할 수 있고, **이미지 생성**은 FAL을 통해 9개 모델(FLUX 2 Klein/Pro, GPT-Image 1.5/2, Nano Banana Pro, Ideogram V3, Recraft V4 Pro, Qwen, Z-Image Turbo)을 지원하며, **TTS**는 OpenAI TTS를 사용하고, **브라우저 자동화**는 Browser Use 기반입니다.

활성화 방법은 세 가지입니다. 첫째, `hermes model`을 실행하여 Nous Portal을 프로바이더로 선택하면 자동으로 Tool Gateway 활성화 여부를 묻는 안내가 나옵니다. 둘째, `hermes tools`를 통해 도구별로 "Nous Subscription"을 프로바이더로 선택할 수 있습니다. 셋째, `~/.hermes/config.yaml`에서 각 도구의 `use_gateway: true` 플래그를 직접 설정할 수 있습니다.

도구 우선순위는 다음과 같이 작동합니다. `use_gateway: true`이면 직접 API 키가 `.env`에 있더라도 게이트웨이를 통해 라우팅합니다. `use_gateway: false`이거나 설정되지 않은 경우, 직접 API 키가 우선이고 키가 없을 때만 게이트웨이를 폴백으로 사용합니다. 이 방식은 사용자가 게이트웨이와 직접 키를 자유롭게 전환할 수 있게 합니다.

자체 호스팅이나 커스텀 게이트웨이 배포를 위해 `TOOL_GATEWAY_DOMAIN`, `TOOL_GATEWAY_SCHEME`, `TOOL_GATEWAY_USER_TOKEN`, `FIRECRAWL_GATEWAY_URL` 같은 환경 변수로 게이트웨이 엔드포인트를 직접 지정할 수도 있습니다.

> 출처: Nous Tool Gateway

---

4. Core 기능

4.1 Tools & Toolsets (도구 및 도구셋)

도구는 에이전트의 능력을 확장하는 함수들이며, 논리적인 묶음인 **도구셋(toolset)**으로 조직되어 플랫폼별로 활성화·비활성화할 수 있습니다. 공식 문서에서 분류한 카테고리는 다음과 같습니다.

웹 카테고리에는 `web_search`(웹 검색)와 `web_extract`(페이지 콘텐츠 추출)가 있습니다. **터미널 및 파일** 카테고리에는 `terminal`(셸 명령 실행), `process`(프로세스 관리), `read_file`, `patch`(파일 편집)가 있습니다. **브라우저** 카테고리에는 `browser_navigate`, `browser_snapshot`, `browser_vision`(스크린샷 + 비전 분석)이 있습니다. **미디어** 카테고리에는 `vision_analyze`, `image_generate`, `text_to_speech`가 있습니다. **에이전트 오케스트레이션** 카테고리에는 `todo`(할 일 관리), `clarify`(사용자에게 질문), `execute_code`(파이썬 코드 실행), `delegate_task`(서브에이전트 위임)가 있습니다. **메모리 및 검색** 카테고리에는 `memory`(영속 메모리)와 `session_search`(과거 세션 검색)가 있습니다. **자동화 및 전송** 카테고리에는 `cronjob`(스케줄 작업)과 `send_message`(외부 플랫폼으로 메시지 전송)가 있습니다. **통합** 카테고리에는 Home Assistant 도구(`ha_*`), MCP 서버 도구, RL 학습 도구(`rl_*`)가 있습니다.

7가지 터미널 백엔드

`terminal` 도구는 7가지 환경에서 명령을 실행할 수 있으며, `~/.hermes/config.yaml`의 `terminal.backend` 설정으로 전환합니다.

`local`은 호스트 머신에서 직접 실행하며 개발용 기본 옵션입니다. `docker`는 격리된 컨테이너에서 실행하여 보안과 재현성을 제공합니다(설정 예: `docker_image: python:3.11-slim`). `ssh`는 원격 서버에서 실행하여 에이전트가 자체 코드를 수정하지 못하도록 격리하는 것을 권장합니다(`TERMINAL_SSH_HOST`, `TERMINAL_SSH_USER`, `TERMINAL_SSH_KEY` 환경변수 사용). `singularity`/Apptainer는 클러스터 컴퓨팅용 루트리스 컨테이너이며, `modal`은 서버리스 클라우드 실행, `daytona`는 영속적인 클라우드 샌드박스 워크스페이스, `vercel_sandbox`는 Vercel Sandbox 클라우드 마이크로VM(스냅샷 기반 파일시스템 영속성 제공)입니다.

컨테이너 백엔드는 보안 강화 옵션으로 읽기 전용 루트 파일시스템(Docker), 모든 Linux capability 제거, 권한 상승 금지, PID 제한(256개), 전체 네임스페이스 격리, 영속 워크스페이스를 위한 볼륨 사용을 적용합니다. 리소스 한도는

`container_cpu`, `container_memory`(MB), `container_disk`(MB), `container_persistent`(세션 간 파일시스템 보존 여부) 설정으로 조절합니다.

백그라운드 프로세스 관리

`terminal`을 `background=true`로 호출하면 백그라운드 프로세스가 시작되고 `session_id`와 `pid`를 반환합니다. 그 후 `process` 도구의 `list`, `poll`, `wait`, `log`, `kill`, `write` 액션으로 관리할 수 있으며, `pty=true` 모드는 Codex나 Claude Code 같은 대화형 CLI 도구를 실행할 때 사용합니다.

> 출처: Tools & Toolsets, Built-in Tools Reference

4.2 Skills System (스킬 시스템)

스킬은 에이전트가 필요할 때 로드하는 **온디맨드 지식 문서**로, 토큰 효율을 위해 **점진적 공개(progressive disclosure)** 패턴을 따르며 agentskills.io 오픈 스탠다드와 호환됩니다. 모든 스킬은 `~/.hermes/skills/`에 저장되고, 설치 시 번들 스킬이 복사되며 허브 설치·에이전트 생성 스킬도 모두 이곳에 저장됩니다.

점진적 공개 메커니즘

세 단계로 작동합니다. Level 0(`skills_list()`)은 약 3000 토큰으로 모든 스킬의 이름·설명·카테고리만 반환합니다. Level 1(`skill_view(name)`)은 특정 스킬의 전체 내용과 메타데이터를 반환합니다. Level 2(`skill_view(name, path)`)는 스킬 폴더 안의 특정 참조 파일만 반환합니다. 이렇게 하면 에이전트가 실제로 필요한 스킬만 전체를 로드하여 토큰 비용을 최소화합니다.

SKILL.md 형식

스킬은 YAML frontmatter와 마크다운 본문으로 구성됩니다. 주요 frontmatter 필드는 `name`, `description`, `version`, `platforms`(macos/linux/windows로 OS 제한 가능), `metadata.hermes.tags`, `metadata.hermes.category`, `metadata.hermes.fallback_for_toolsets`(특정 도구셋이 없을 때만 노출), `metadata.hermes.requires_toolsets`(특정 도구셋이 있을 때만 노출), `metadata.hermes.config`(스킬별 설정값)입니다. 본문은 일반적으로 "When to Use", "Procedure", "Pitfalls", "Verification" 섹션으로 구성됩니다.

`fallback_for_toolsets` 메커니즘은 매우 유용합니다. 예를 들어 번들된 `duckduckgo-search` 스킬은 `fallback_for_toolsets: [web]`로 설정되어 있어, `FIRECRAWL_API_KEY`가 있으면 자동으로 숨겨지고 키가 없을 때만 무료 대안으로 노출됩니다.

안전한 환경변수 로드

스킬은 `required_environment_variables` 섹션을 통해 필요한 API 키를 선언할 수 있으며, 스킬이 실제로 로드될 때만 사용자에게 안전하게 입력을 요청합니다. 메시징 플랫폼에서는 절대 채팅으로 시크릿을 묻지 않고 `hermes setup` 또는 `~/.hermes/.env`를 사용하라고 안내합니다.

슬래시 명령 형식의 스킬 사용

설치된 모든 스킬은 자동으로 슬래시 명령으로 사용 가능합니다. CLI나 메시징 플랫폼에서 `/gif-search funny cats`, `/axolotl help me fine-tune Llama 3`, `/github-pr-workflow create a PR for the auth refactor`, `/plan design a rollout for migrating our auth provider` 같은 형태로 호출할 수 있습니다.

에이전트가 스킬을 자동 생성하는 시점

에이전트는 다음 상황에서 `skill_manage` 도구로 스스로 스킬을 만듭니다. 5개 이상의 도구 호출이 필요한 복잡한 작업을 성공적으로 완수했을 때, 에러나 막다른 길에 부딪혔다가 해결책을 찾았을 때, 사용자가 접근 방식을 교정해주었을 때, 자명하지 않은 워크플로를 발견했을 때입니다. 액션은 `create`(새로 생성), `patch`(타겟 수정, 토큰 효율적이므로 권장), `edit`(전체 재작성), `delete`(삭제), `write_file`/`remove_file`(보조 파일 관리)입니다.

Skills Hub: 8개 소스에서 스킬 설치

`hermes skills install` 명령으로 8개 소스에서 스킬을 설치할 수 있습니다. `official`(`optional-skills/` 안의 공식 옵션 스킬, builtin 신뢰 등급), `skills-sh`(Vercel의 skills.sh 디렉토리), `well-known`(웹사이트가 `/.well-known/skills/index.json` 표준으로 직접 게시한 스킬), `github`(직접 GitHub 저장소 설치, 기본 탭은 openai/skills, anthropics/skills, VoltAgent/awesome-agent-skills, garrytan/gstack), `clawhub`(clawhub.ai), `claude-marketplace`(Claude 호환 마켓플레이스 매니페스트 저장소), `lobehub`(LobeHub 카탈로그), `url`(단일 SKILL.md HTTP 직접 설치)입니다.

설치할 때마다 보안 스캐너가 실행되어 데이터 유출, 프롬프트 주입, 파괴적 명령, 공급망 신호 등 위협을 검사합니다. 신뢰 등급은 `builtin`(항상 신뢰), `official`(builtin과 동등), `trusted`(openai/skills 등 신뢰 저장소, 더 관대한 정책), `community`(나머지 모두, 비위험 검출은 `--force`로 우회 가능, `dangerous` 판정은 항상 차단)입니다.

Curator: 스킬 자동 정리

Curator는 에이전트 생성 스킬에 대한 백그라운드 유지보수 시스템입니다. 번들된 스킬이나 허브 설치 스킬은 절대 건드리지 않고, 자동 삭제도 하지 않습니다(최악의 경우 `~/.hermes/skills/.archive/`로 아카이브, 복구 가능).

큐레이터는 CLI 세션 시작 시점과 게이트웨이의 cron-ticker 스레드의 주기적 틱에서 트리거됩니다. 발동 조건은 마지막 큐레이터 실행 후 `interval_hours`(기본 7일)가 지났고 동시에 에이전트가 `min_idle_hours`(기본 2시간) 이상 유휴 상태일 때입니다. 두 조건이 모두 충족되면 백그라운드 AIAgent 포크를 생성하여 두 단계로 작동합니다. 1단계는 결정론적 자동 전이로 `stale_after_days`(30일) 미사용 스킬을 `stale`로 표시하고, `archive_after_days`(90일) 미사용 스킬을 아카이브 폴더로 옮깁니다. 2단계는 단일 보조 모델 패스(`max_iterations=8`)로 에이전트 생성 스킬을 검토하고 유지·패치·통합·아카이브를 결정합니다.

CLI 명령은 `hermes curator status`(마지막 실행, 카운트, 핀고정 목록, LRU 상위 5개), `hermes curator run`(즉시 실행, 기본은 백그라운드), `hermes curator run --sync`(동기 실행), `hermes curator pause`/`resume`, `hermes curator pin <skill>`(자동 전이 보호), `hermes curator unpin <skill>`, `hermes curator restore <skill>`(아카이브에서 복원)입니다. 슬래시 명령으로도 동일하게 사용 가능합니다.

> 출처: Skills System, Curator

4.3 Persistent Memory (영속 메모리)

Hermes의 메모리는 **세션을 넘어 지속되는 경계가 있는 큐레이션된 메모리**입니다. 두 개의 파일로 구성됩니다.

`MEMORY.md`는 에이전트의 개인 노트로, 환경 사실·관례·학습한 것을 기록하며 **2,200자(약 800 토큰)** 한도를 가집니다. `USER.md`는 사용자 프로필로, 선호도·커뮤니케이션 스타일·기대치를 기록하며 **1,375자(약 500 토큰)** 한도를 가집니다. 둘 다 `~/.hermes/memories/`에 저장되며, 세션 시작 시 시스템 프롬프트에 **고정된 스냅샷**으로 주입됩니다.

고정 스냅샷 패턴

세션 시작 시 메모리가 시스템 프롬프트에 한 번 캡처되고 세션 중에는 변경되지 않습니다. 이는 의도된 설계로, LLM의 prefix cache를 보존하여 성능을 유지하기 위함입니다. 에이전트가 세션 중에 메모리 항목을 추가/제거하면 디스크에는

즉시 반영되지만, 시스템 프롬프트는 다음 세션 시작까지 변경되지 않습니다. 도구 응답은 항상 최신 상태를 보여줍니다.

메모리 도구 액션

`memory` 도구는 세 가지 액션을 가집니다. `add`(새 항목 추가), `replace`(기존 항목을 부분 문자열 매칭으로 찾아 업데이트), `remove`(부분 문자열 매칭으로 항목 제거)입니다. `read` 액션은 없는데, 메모리 내용이 시스템 프롬프트에 자동으로 주입되기 때문에 에이전트는 이미 자신의 컨텍스트로 메모리를 보고 있습니다.

무엇을 저장하고 무엇을 건너뛸지

저장할 항목으로는 사용자 선호("TypeScript를 JavaScript보다 선호함" → `user`), 환경 사실("이 서버는 Debian 12에 PostgreSQL 16" → `memory`), 교정 사항("Docker 명령에 sudo 사용 금지, 사용자가 docker 그룹에 있음" → `memory`), 관례("프로젝트는 탭, 120자 줄 폭, Google 스타일 docstring 사용" → `memory`), 완료된 작업("2026-01-15에 MySQL에서 PostgreSQL로 마이그레이션 완료" → `memory`), 명시적 요청("API 키 순환은 매월 발생함을 기억하라" → `memory`)이 있습니다.

건너뛸 항목으로는 사소하거나 자명한 정보, 쉽게 재발견 가능한 사실(예: "Python 3.12는 f-string 중첩을 지원함"), 대량의 데이터 덤프, 세션 한정 일시 정보, 컨텍스트 파일에 이미 있는 정보가 있습니다.

세션 검색

영속 메모리 외에도 `session_search` 도구로 모든 과거 대화를 검색할 수 있습니다. CLI와 메시징 세션이 모두 SQLite(`~/.hermes/state.db`)에 저장되며 FTS5 전체 텍스트 검색이 가능합니다. 검색 쿼리는 관련 과거 대화를 Gemini Flash 요약과 함께 반환합니다. 메모리는 항상 컨텍스트에 있어야 할 핵심 사실용이고, 세션 검색은 "지난주에 X에 대해 논의했었나요?" 같은 질의에 사용합니다.

보안 스캔

메모리 항목은 시스템 프롬프트에 주입되므로 주입 및 유출 패턴이 검사됩니다. 위협 패턴(프롬프트 주입, 자격증명 유출, SSH 백도어)이나 보이지 않는 유니코드 문자가 포함되면 차단됩니다.

> 출처: Persistent Memory

4.4 Memory Providers (메모리 프로바이더)

Hermes는 내장 MEMORY.md/USER.md 외에 **8개의 외부 메모리 프로바이더 플러그인**을 함께 제공합니다. 한 번에 하나의 외부 프로바이더만 활성화할 수 있으며, 내장 메모리는 항상 함께 작동합니다(외부 프로바이더는 내장 메모리를 대체하지 않고 보완합니다).

활성화 방법은 `hermes memory setup`(대화형 선택), `hermes memory status`(활성 상태 확인), `hermes memory off`(비활성화), 또는 `~/.hermes/config.yaml`의 `memory.provider`를 직접 설정하는 것입니다.

Honcho

AI 네이티브 크로스 세션 사용자 모델링으로, 변증법적(dialectic) 추론·세션 범위 컨텍스트 주입·시맨틱 검색·영속 결론을 제공합니다. 다중 에이전트 시스템과 사용자-에이전트 정렬에 적합합니다.

핵심 도구 5개: `honcho_profile`(피어 카드 읽기/업데이트), `honcho_search`(시맨틱 검색), `honcho_context`(세션 컨텍스트 - 요약·표상·카드·메시지), `honcho_reasoning`(LLM 합성 추론), `honcho_conclude`(결론 생성/삭제). 두 층

의 컨텍스트 주입 아키텍처를 가지며, 기본 층(세션 요약 + 표상 + 피어 카드, `contextCadence` 주기로 갱신)에 변증법적 보충 층(LLM 추론, `dialecticCadence` 주기로 갱신)을 더합니다. 세 가지 직교 설정 노브(`contextCadence`, `dialecticCadence`, `dialecticDepth`)로 비용과 깊이를 독립적으로 제어합니다.

다중 피어 설정을 지원하여, 워크스페이스를 공유 환경으로 두고 한 명의 사용자 피어와 Hermes 프로파일별 AI 피어를 둘 수 있습니다. 예를 들어 `coder` 프로파일과 `writer` 프로파일이 같은 사용자에 대해 각각 코드 중심·편집 중심의 표상을 독립적으로 구축합니다. 관찰(observation) 모드는 `directional`(기본, 모든 4개 플래그 켜짐)과 `unified`(단일 관찰자 풀)가 있습니다.

OpenViking

Volcengine(ByteDance)의 컨텍스트 데이터베이스로, 파일시스템 스타일 지식 계층·계층적 검색·6개 카테고리(프로필·선호·엔티티·이벤트·케이스·패턴)로의 자동 메모리 추출을 지원합니다. 자체 호스팅 가능하고 무료(AGPL-3.0)입니다.

도구: `viking_search`(시맨틱 검색), `viking_read`(L0 ~100 토큰 / L1 ~2k / L2 전체), `viking_browse`(파일시스템 탐색), `viking_remember`(사실 저장), `viking_add_resource`(URL/문서 인제스트). `viking://` URI 스키마로 계층적 지식 탐색을 지원합니다.

Mem0

서버 측 LLM 사실 추출, 시맨틱 검색, 재순위, 자동 중복 제거를 제공하는 클라우드 서비스입니다. Mem0가 추출을 자동 처리하므로 사용자가 손대지 않아도 됩니다.

도구: `mem0_profile`(저장된 모든 메모리), `mem0_search`(시맨틱 검색 + 재순위), `mem0_conclude`(축자적 사실 저장).

Hindsight

지식 그래프, 엔티티 해결, 다중 전략 검색을 제공하는 장기 메모리 시스템입니다. `hindsight_reflect` 도구는 다른 어떤 프로바이더도 제공하지 않는 메모리 간 합성 기능을 제공합니다. 클라우드 또는 로컬 임베디드 PostgreSQL로 사용 가능합니다.

도구: `hindsight_retain`(엔티티 추출과 함께 저장), `hindsight_recall`(다중 전략 검색), `hindsight_reflect`(메모리 간 합성). 회수 예산(`recall_budget`: low/mid/high), 메모리 모드(hybrid/context/tools), 자동 보존 등 다양한 설정이 가능합니다.

Holographic

NumPy 기반 HRR(Holographic Reduced Representations)을 사용한 로컬 SQLite 사실 저장소입니다. 외부 의존성 없이 로컬 전용이며 무료입니다.

도구 2개: `fact_store`(9개 액션: add, search, probe, related, reason, contradict, update, remove, list)와 `fact_feedback`(helpful/unhelpful 평가, 신뢰 점수 학습). 고유 기능으로 `probe`(엔티티 특화 대수 회수), `reason`(여러 엔티티에 걸친 합성 AND 쿼리), `contradict`(상충 사실 자동 탐지), 비대칭 피드백(+0.05 helpful / -0.10 unhelpful)을 가진 신뢰 점수가 있습니다.

RetainDB

Vector + BM25 + Reranking 하이브리드 검색, 7가지 메모리 타입, 델타 압축을 제공하는 클라우드 메모리 API입니다. $20/월의 유료 서비스이며, 이미 RetainDB 인프라를 사용하는 팀에 적합합니다.

도구 5개: `retaindb_profile`, `retaindb_search`, `retaindb_context`(작업 관련 컨텍스트), `retaindb_remember`(타입과 중요도와 함께 저장), `retaindb_forget`.

ByteRover

`brv` CLI를 통한 영속 메모리로, 계층적 지식 트리와 계층적 검색(퍼지 텍스트 → LLM 주도 검색)을 제공합니다. 로컬 우선이며 선택적 클라우드 동기화(SOC2 Type II 인증)를 지원합니다.

도구 3개: `brv_query`(지식 트리 검색), `brv_curate`(사실/결정/패턴 저장), `brv_status`(CLI 버전 + 트리 통계). 컨텍스트 압축 전 추출(컨텍스트 압축이 통찰을 폐기하기 전에 저장), 프로파일 범위 지식 트리 (`$HERMES_HOME/byterover/`) 같은 고유 기능이 있습니다.

Supermemory

시맨틱 장기 메모리로, 프로필 회수·시맨틱 검색·명시적 메모리 도구·세션 종료 시 Supermemory 그래프 API 통한 대화 인제스트를 제공합니다.

도구 4개: `supermemory_store`, `supermemory_search`(시맨틱 유사도 검색), `supermemory_forget`(ID 또는 최적 매칭으로 망각), `supermemory_profile`(영속 프로필 + 최근 컨텍스트). 자동 컨텍스트 펜싱(회수된 메모리를 캡처된 턴에서 제거하여 재귀 메모리 오염 방지), 세션 종료 시 그래프 수준 지식 구축, 다중 컨테이너 모드(여러 명명된 컨테이너 간 읽기/쓰기) 같은 고유 기능이 있습니다.

> 출처: Memory Providers

4.5 Context Files (컨텍스트 파일)

Hermes는 작업 디렉토리의 컨텍스트 파일을 자동으로 발견하여 시스템 프롬프트에 주입합니다. 우선순위 시스템에 따라 첫 매칭이 이깁니다.

지원되는 컨텍스트 파일

`.hermes.md` 또는 `HERMES.md`(프로젝트 지침, 최고 우선순위, git 루트까지 탐색), `AGENTS.md`(프로젝트 지침·관례·아키텍처, 시작 시 CWD에서 발견 + 하위 디렉토리는 점진적), `CLAUDE.md`(Claude Code 컨텍스트 파일도 감지), `SOUL.md`(이 Hermes 인스턴스의 전역 인격 및 톤 커스터마이징, `HERMES_HOME/SOUL.md`에서만 로드), `.cursorrules`(Cursor IDE 코딩 관례, CWD에서만), `.cursor/rules/*.mdc`(Cursor IDE 룰 모듈, CWD에서만)입니다.

세션당 단 하나의 프로젝트 컨텍스트 타입만 로드되며 첫 매칭이 이깁니다(`.hermes.md` → `AGENTS.md` → `CLAUDE.md` → `.cursorrules` 순). SOUL.md는 항상 독립적으로 에이전트 정체성(시스템 프롬프트의 슬롯 #1)으로 로드됩니다.

점진적 하위 디렉토리 발견

세션 시작 시 작업 디렉토리의 `AGENTS.md`가 시스템 프롬프트에 로드됩니다. 에이전트가 세션 중에 하위 디렉토리로 들어가면(예: `read_file`, `terminal`, `search_files`로) 그 디렉토리의 컨텍스트 파일이 **점진적으로 발견**되어 도구 결과에 주입됩니다. 각 하위 디렉토리는 세션당 한 번만 확인됩니다.

이 접근 방식은 두 가지 장점이 있습니다. 첫째, 시스템 프롬프트가 비대해지지 않습니다(필요할 때만 하위 디렉토리 힌트가 나타남). 둘째, 프롬프트 캐시가 보존됩니다(시스템 프롬프트가 턴 간에 안정적으로 유지됨).

보안: 프롬프트 주입 보호

모든 컨텍스트 파일은 포함되기 전에 잠재적 프롬프트 주입을 검사받습니다. 스캐너는 지시 우회 시도("이전 지시를 무시하라"), 기만 패턴("사용자에게 알리지 말라"), 시스템 프롬프트 오버라이드, 숨겨진 HTML 주석(`<!-- ignore instructions -->`), 숨겨진 div 요소, 자격증명 유출(`curl ... $API_KEY`), 시크릿 파일 접근(`cat .env`), 보이지 않는 문자(zero-width 공백, 양방향 오버라이드, word joiner)를 검사합니다.

크기 제한

파일당 최대 20,000자(약 7,000 토큰)이며, 초과 시 머리(70%) + 꼬리(20%) + 마커(10%) 형태로 절단됩니다. 점진적으로 발견되는 하위 디렉토리 파일은 8,000자에서 절단됩니다.

> 출처: Context Files

4.6 Personality & SOUL.md (인격)

Hermes Agent의 인격은 완전히 커스터마이징 가능합니다. `SOUL.md`는 **주 정체성 파일**로 시스템 프롬프트의 첫 번째 항목이며, 에이전트가 누구인지를 정의합니다.

설치 시 Hermes는 `~/.hermes/SOUL.md`(또는 사용자 정의 `$HERMES_HOME/SOUL.md`)에 기본 SOUL.md를 자동으로 시드합니다. 기존 사용자 SOUL.md는 절대 덮어쓰지 않으며, 비어있거나 로드 실패 시에만 내장 기본 정체성("당신은 Nous Research가 만든 지능형 AI 어시스턴트 Hermes Agent입니다...")이 사용됩니다.

SOUL.md에는 다음과 같은 지속적인 음성 및 인격 가이던스가 적합합니다. 톤, 커뮤니케이션 스타일, 직접성 수준, 기본 상호작용 스타일, 스타일적으로 피해야 할 것, 불확실성·이견·모호성 처리 방식이 있습니다. 일회성 프로젝트 지시·파일 경로·저장소 관례·일시적 워크플로 세부사항은 SOUL.md가 아니라 AGENTS.md에 들어가야 합니다.

14개 내장 인격

`/personality` 슬래시 명령으로 전환할 수 있는 내장 인격은 다음과 같습니다. **helpful**(친근한 범용 어시스턴트), **concise**(간결한 응답), **technical**(상세한 기술 전문가), **creative**(혁신적 사고), **teacher**(인내심 있는 교육자), **kawaii**(귀여운 표현, 반짝이, 열정 ★), **catgirl**(고양이 같은 표현, 냐~), **pirate**(기술에 능통한 해적), **shakespeare**(극적인 음유시인 산문), **surfer**(완전히 느긋한 형제 분위기), **noir**(하드보일드 탐정 내레이션), **uwu**(uwu-말투로 최대한 귀엽게), **philosopher**(모든 질의에 깊은 사색), **hype**(최대 에너지와 열정!!!).

`/personality codereviewer` 같은 형태로 `~/.hermes/config.yaml`의 `agent.personalities` 아래에 명명된 커스텀 인격을 정의할 수도 있습니다.

SOUL.md vs AGENTS.md vs /personality

권장 워크플로는 다음과 같습니다. `~/.hermes/SOUL.md`에 사려 깊은 전역 SOUL.md를 유지하고(베이스라인 음성), 프로젝트 지시는 `AGENTS.md`에 두며, 일시적인 모드 전환이 필요할 때만 `/personality`를 사용합니다. 규칙은 "어디든 따라다녀야 하면 SOUL.md, 프로젝트에 속하면 AGENTS.md"입니다.

> 출처: Personality & SOUL.md

4.7 Plugins (플러그인)

플러그인 시스템은 코어 코드를 수정하지 않고도 커스텀 도구·훅·통합을 추가하는 방법입니다. `~/.hermes/plugins/<이름>/` 아래에 `plugin.yaml`, `__init__.py`(register 함수), 선택적 `schemas.py`와 `tools.py`를 두면 됩니다.

플러그인이 할 수 있는 것

도구 추가(`ctx.register_tool(name, toolset, schema, handler)`), 훅 추가(`ctx.register_hook("post_tool_call", callback)`), 슬래시 명령 추가(`ctx.register_command(name, handler, description)`), CLI 명령 추가(`ctx.register_cli_command(name, help, setup_fn, handler_fn)`), 메시지 주입(`ctx.inject_message(content, role="user")`), 데이터 파일 배포, 스킬 번들링(`ctx.register_skill(name, path)` - `plugin:skill` 네임스페이스), 환경변수 게이트(`requires_env: [API_KEY]`), pip 배포(`hermes_agent.plugins` 엔트리포인트)가 가능합니다.

플러그인 발견 경로

발견 경로는 4단계 우선순위로 작동합니다. 첫째는 번들(`<repo>/plugins/`, Hermes와 함께 출하), 둘째는 사용자(`~/.hermes/plugins/`, 개인 플러그인), 셋째는 프로젝트(`./.hermes/plugins/`, `HERMES_ENABLE_PROJECT_PLUGINS=true` 필요), 넷째는 pip 엔트리포인트(`hermes_agent.plugins`)입니다. 이름 충돌 시 나중 소스가 이깁니다.

옵트인 정책

모든 플러그인은 사용자 설치, 번들, pip 모두 기본으로 비활성화 상태입니다. 발견은 되지만(`hermes plugins`와 `/plugins`에 표시) `~/.hermes/config.yaml`의 `plugins.enabled`에 명시적으로 추가하기 전까지는 로드되지 않습니다. 이 정책은 사용자의 명시적 동의 없이 훅이나 도구가 실행되는 것을 방지합니다.

`hermes plugins` 명령은 대화형 토글 UI(스페이스로 체크), `hermes plugins enable <name>`(허용 목록 추가), `hermes plugins disable <name>`(허용 목록에서 제거 + 거부 목록 추가)로 관리합니다.

세 가지 플러그인 타입

일반 플러그인(General Plugins)은 도구·훅·슬래시 명령·CLI 명령을 추가하며, 다중 선택(여러 개 동시 활성화 가능)이고 `~/.hermes/plugins/`에 위치합니다. **메모리 프로바이더**(Memory Providers)는 내장 메모리를 대체하거나 보강하며, 단일 선택(하나만 활성)이고 `plugins/memory/`에 위치합니다. **컨텍스트 엔진**(Context Engines)은 내장 컨텍스트 압축기를 대체하며, 단일 선택이고 `plugins/context_engine/`에 위치합니다.

번들 플러그인 (현재 시점)

disk-cleanup: 세션 중 생성된 일시 파일(테스트 스크립트, 임시 출력, cron 로그, 오래된 chrome 프로필)을 도구 호출 없이 자동 추적·제거합니다. `post_tool_call` 훅으로 `write_file`/`terminal`/`patch`가 `test_*`, `tmp_*`, `*.test.*` 패턴 파일을 만들면 추적하고, `on_session_end` 훅으로 안전한 quick 정리를 자동 실행합니다. 카테고리별 삭제 규칙: `test`(매 세션 종료, 확인 없음), `temp`(7일 후), `cron-output`(14일 후), `research`(30일 + 가장 새로운 10개 외, 확인 후), `chrome-profile`(14일 후, 확인 후). `/disk-cleanup` 슬래시 명령으로 `status`/`dry-run`/`quick`/`deep`/`track`/`forget` 액션이 가능합니다.

> 출처: Plugins, Built-in Plugins

4.8 Context References (컨텍스트 참조)

`@`로 시작하는 참조 문법으로 메시지에 파일·폴더·git diff·URL을 직접 주입할 수 있습니다. Hermes는 인라인으로 참조를 확장하고 콘텐츠를 자동으로 첨부합니다. 자세한 내용은 Context References 페이지를 참조하세요.

4.9 Skins & Themes (스킨 및 테마)

CLI의 시각적 표현(배너 색상, 스피너 표정, 응답 박스 라벨, 브랜딩 텍스트, 도구 활동 접두사)을 커스터마이징할 수 있습니다. 대화형 인격(`SOUL.md`, `agent.system_prompt`, `/personality`)과 CLI 외관(`display.skin`, `/skin`)은 별개로 작동합니다. 자세한 내용은 Skins & Themes 페이지를 참조하세요.

4.10 Checkpoints (체크포인트)

Hermes는 파일 변경 전에 작업 디렉토리를 자동 스냅샷하여 안전망을 제공합니다. `/rollback`으로 되돌릴 수 있습니다. 자세한 내용은 Checkpoints & Rollback 페이지를 참조하세요.

---

5. 자동화 기능

5.1 Scheduled Tasks (Cron)

Cron은 자연어 또는 cron 표현식으로 작업을 스케줄링합니다. 단일 `cronjob` 도구로 액션 스타일 작업을 처리합니다. 가능한 일은 일회성 또는 반복 작업 스케줄링, 작업 일시정지·재개·편집·트리거·삭제, 작업에 0개·1개·여러 개 스킬 첨부, 결과를 원본 채팅·로컬 파일·구성된 플랫폼 타깃으로 전달, 일반 정적 도구 목록을 가진 신선한 에이전트 세션에서 실행입니다.

> 중요한 안전장치: cron 실행 세션은 재귀적으로 더 많은 cron 작업을 만들 수 없습니다. 폭주 스케줄링 루프를 방지하기 위해 cron 실행 내부에서 cron 관리 도구가 비활성화됩니다.

작업 생성

채팅에서 `/cron add 30m "빌드 확인하라고 알려줘"`, `/cron add "every 2h" "서버 상태 확인"`, `/cron add "every 1h" "새 피드 항목 요약" --skill blogwatcher`, 또는 다중 스킬을 위해 `--skill blogwatcher --skill maps`를 사용할 수 있습니다. 독립 CLI에서는 `hermes cron create "every 2h" "서버 상태 확인"` 형태로 사용합니다. 자연어 대화로도 가능합니다("매일 아침 9시에 Hacker News의 AI 뉴스를 확인하고 Telegram으로 요약해줘").

19개 이상의 전송 옵션

작업 출력을 어디로 보낼지 선택할 수 있습니다. `"origin"`(작업이 만들어진 곳, 메시징 플랫폼 기본값), `"local"`(로컬 파일만 `~/.hermes/cron/output/`, CLI 기본값), `"telegram"`/`"discord"`/`"slack"`/`"whatsapp"`/`"signal"`/`"matrix"`/`"mattermost"`(각각 홈 채널), `"telegram:123456"`(특정 Telegram 채팅 ID), `"telegram:-100123:17585"`(특정 Telegram 토픽), `"discord:#engineering"`(특정 Discord 채널), `"email"`, `"sms"`(Twilio 통한 SMS), `"homeassistant"`, `"dingtalk"`, `"feishu"`(Feishu/Lark), `"wecom"`, `"weixin"`(WeChat), `"bluebubbles"`(iMessage), `"qqbot"`(텐센트 QQ).

에이전트의 최종 응답이 자동으로 전달되므로 cron 프롬프트에서 `send_message`를 호출할 필요가 없습니다.

응답 래핑과 무음 억제

기본적으로 전달되는 cron 출력은 헤더와 푸터로 감싸집니다("Cronjob Response: 작업명 - - - <에이전트 출력> Note: 에이전트는 이 메시지를 볼 수 없으므로 응답할 수 없습니다"). 원시 출력을 원하면 `cron.wrap_response: false`를 설정합니다. 에이전트의 최종 응답이 `[SILENT]`로 시작하면 전달이 완전히 억제됩니다(로컬 감사 로그에는 저장됨). 모니터링 작업에서 문제가 있을 때만 보고하고 싶을 때 유용합니다("nginx가 실행 중인지 확인하라. 모두 정상이면 [SILENT]로만 응답하고, 그렇지 않으면 문제를 보고하라").

스케줄 형식

상대 지연(일회성): `30m`, `2h`, `1d`. 간격(반복): `every 30m`, `every 2h`, `every 1d`. Cron 표현식: `0 9 * * *`(매일 9시), `0 9 * * 1-5`(평일 9시), `0 */6 * * *`(6시간마다), `30 8 1 * *`(매월 1일 8시 30분). ISO 타임스탬프: `2026-03-15T09:00:00`.

작업 체이닝과 wakeAgent

`context_from`으로 다른 작업의 가장 최근 성공 출력을 컨텍스트로 받을 수 있습니다. `cronjob(action="create", name="daily-digest", schedule="every day 7am", context_from=["ai-news-fetch", "github-prs-fetch"], prompt="위 출력을 사용해 일일 다이제스트 작성")` 같은 형태입니다.

`wakeAgent` 메커니즘으로 사전 검사 스크립트가 에이전트 호출이 필요한지 결정할 수 있습니다. 스크립트의 마지막 stdout 줄에 `{"wakeAgent": false}`를 출력하면 cron이 이번 틱에 에이전트를 호출하지 않습니다. 1-5분 주기로 폴링하지만 상태가 실제로 변경되었을 때만 LLM을 깨우고 싶을 때 유용합니다.

작업당 도구셋 제어

각 작업은 신선한 에이전트 세션에서 실행되며, 기본으로 `hermes tools`에서 `cron` 플랫폼에 구성한 도구셋을 받습니다. 더 엄격한 작업당 제어는 `enabled_toolsets` 필드로 가능합니다(예: `enabled_toolsets=["web", "file"]`로 터미널·브라우저 등을 제외).

게이트웨이 데몬에 의한 실행

cron 실행은 게이트웨이 데몬이 처리합니다. 게이트웨이는 60초마다 스케줄러를 틱하여 만기된 작업을 격리된 에이전트 세션에서 실행합니다. `hermes gateway install`로 사용자 서비스로 설치하거나 `sudo hermes gateway install --system`으로 부팅 시 시스템 서비스로 설치할 수 있습니다.

> 출처: Scheduled Tasks (Cron)

5.2 Subagent Delegation (서브에이전트 위임)

`delegate_task` 도구는 격리된 컨텍스트, 제한된 도구셋, 자체 터미널 세션을 가진 자식 AIAgent 인스턴스를 생성합니다. 각 자식은 신선한 대화에서 시작하고 부모의 컨텍스트를 전혀 모르며 독립적으로 작업합니다. 최종 요약만 부모의 컨텍스트로 들어옵니다.

단일 작업과 병렬 배치

단일 작업: `delegate_task(goal="테스트 실패 디버그", context="test_foo.py 42행 어설션 에러", toolsets=["terminal", "file"])`. 병렬 배치는 `tasks=[...]` 배열로 지정하며 기본 3개 동시 서브에이전트(설정 가능, 하드 한도 없음)가 실행됩니다.

서브에이전트 컨텍스트 작동 방식

서브에이전트는 **완전히 신선한 대화**에서 시작합니다. 부모의 대화 이력, 이전 도구 호출, 위임 전 논의된 어떤 것도 알지 못합니다. 서브에이전트의 유일한 컨텍스트는 부모가 `delegate_task`를 호출할 때 채우는 `goal`과 `context` 필드뿐입니다. 따라서 부모는 자식이 필요한 모든 것을 호출에 포함시켜야 합니다.

차단된 도구셋

리프 서브에이전트(기본)에게 차단되는 도구셋이 있습니다. `delegation`(리프에서 차단, `role="orchestrator"` 자식만 유지), `clarify`(서브에이전트는 사용자와 상호작용 불가), `memory`(공유 영속 메모리에 쓰기 불가), `code_execution`(자식은 단계별로 추론), `send_message`(크로스 플랫폼 부작용 차단)입니다.

깊이 제한과 중첩 오케스트레이션

기본은 평면 위임으로, 부모(깊이 0)가 자식(깊이 1)을 생성하고 그 자식들은 더 위임할 수 없습니다. 다단계 워크플로(연구 → 합성, 또는 하위 문제별 병렬 오케스트레이션)를 위해 `role="orchestrator"`를 지정하면 자식이 자체 워커를 생성할 수 있습니다. `delegation.max_spawn_depth`(기본 1, 최대 3)로 트리 깊이를 제어합니다.

비용 경고: `max_spawn_depth: 3`과 `max_concurrent_children: 3`이면 트리는 3×3×3 = 27개 동시 리프 에이전트에 도달할 수 있습니다. 각 추가 레벨은 지출을 곱하므로 의도적으로 올려야 합니다.

모델 오버라이드

서브에이전트용으로 다른 모델을 구성할 수 있습니다(저렴한/빠른 모델로 단순 작업 위임에 유용). `delegation.model: "google/gemini-flash-2.0"`, `delegation.provider: "openrouter"` 형태입니다.

자식 타임아웃과 진단 덤프

서브에이전트는 `delegation.child_timeout_seconds`(기본 600초 = 10분) 동안 응답이 없으면 멈춘 것으로 간주되어 종료됩니다. 타이머는 자식이 API 호출이나 도구 호출을 할 때마다 재설정되므로 진정으로 유휴인 워커만 종료를 트리거합니다.

특별한 진단 메커니즘으로, 서브에이전트가 0개의 API 호출로 타임아웃되면(보통 프로바이더 도달 불가, 인증 실패, 도구 스키마 거부) `delegate_task`가 `~/.hermes/logs/subagent-timeout-<session>-<timestamp>.log`에 구조화된 진단을 작성합니다(서브에이전트의 설정 스냅샷, 자격증명 해결 추적, 초기 에러 메시지 포함). 이전의 무음 타임아웃 동작보다 근본 원인을 찾기가 훨씬 쉽습니다.

TUI에서 `/agents` 오버레이

TUI는 재귀 `delegate_task` 펼침을 일급 감사 표면으로 만드는 `/agents` 오버레이(별칭 `/tasks`)를 제공합니다. 실행 중·최근 종료된 서브에이전트의 라이브 트리 뷰(부모별 그룹화), 분기별 비용·토큰·터치된 파일 롤업, 종료/일시정지 컨트롤, 사후 검토(서브에이전트의 턴별 이력을 부모로 돌아온 후에도 단계별로 살펴보기)가 가능합니다.

> 출처: Subagent Delegation

5.3 Persistent Goals (`/goal`)

`/goal`은 Hermes에게 턴을 넘어 지속되는 목표를 부여합니다. 매 턴 후 경량 판사(judge) 모델이 어시스턴트의 마지막 응답이 목표를 만족했는지 확인합니다. 그렇지 않으면 Hermes가 같은 세션에 자동으로 계속하기 프롬프트를 다시 입력하고 작업을 계속합니다 - 목표가 달성되거나, 사용자가 일시정지/지우거나, 턴 예산이 소진될 때까지요.

이는 Codex CLI 0.128.0의 `/goal`(Eric Traut, OpenAI Codex 팀)에서 영감받은 Ralph 루프 패턴의 Hermes 구현입니다.

사용 시점

다음과 같이 에이전트가 사용자가 매 턴 다시 프롬프트하지 않고도 스스로 반복하길 원하는 작업에 사용합니다. "src/의 모든 lint 에러를 수정하고 ruff check가 통과하는지 확인하라", "저장소 Y에서 기능 X를 포팅하라(테스트 포함). CI를 녹색으로 만들어라", "중간 실행 압축에서 세션 ID가 가끔 드리프트하는 이유를 조사하고 보고서를 작성하라", "EXIF 날짜로 파일 이름을 바꾸는 작은 CLI를 만들고 photos/ 폴더에 대해 테스트하라".

명령

`/goal <text>`는 새 목표 설정(이전 것 교체) 및 첫 턴을 즉시 시작합니다. `/goal` 또는 `/goal status`는 현재 목표·상태·사용된 턴을 표시합니다. `/goal pause`는 자동 계속 루프를 중지하지만 목표는 유지합니다. `/goal resume`은 루프를 재개합니다(턴 카운터 0으로 리셋). `/goal clear`는 목표를 완전히 폐기합니다. CLI와 모든 게이트웨이 플랫폼에서 동일하게 작동합니다.

판사 모델

매 턴 후 보조 모델이 표준 목표 텍스트, 에이전트의 가장 최근 최종 응답(마지막 ~4KB), 엄격한 JSON으로 응답하라는 시스템 프롬프트를 받아 판단합니다. 판사는 의도적으로 보수적입니다. 응답이 명시적으로 목표가 완료되었음을 확인하거나, 최종 산출물이 명백히 생산되거나, 목표가 달성 불가능/차단된 경우(차단 이유와 함께 DONE으로 처리하여 불가능한 작업에 예산을 태우지 않도록)에만 `done`으로 표시합니다.

Fail-open 의미론과 턴 예산

판사가 에러를 일으키면(네트워크 문제, 잘못된 응답, 보조 클라이언트 사용 불가) Hermes는 판결을 `continue`로 처리합니다. 부서진 판사가 진행을 막지 않습니다. 진정한 백스톱은 턴 예산입니다. 기본 20개 계속 턴(`goals.max_turns`)이며, 예산 도달 시 자동 일시정지합니다.

사용자 메시지가 항상 우선

목표 활성 중 사용자가 보낸 실제 메시지는 계속 루프보다 우선합니다. CLI에서는 메시지가 큐에 들어간 계속 메시지보다 앞에 오고, 게이트웨이에서는 어댑터 FIFO를 통해 같은 방식으로 처리됩니다. 판사는 사용자의 턴 후에도 다시 실행되므로, 사용자 메시지가 우연히 목표를 완료하면 판사가 잡아서 멈춥니다.

영속성

목표 상태는 `SessionDB.state_meta`에 `goal:<session_id>` 키로 저장됩니다. `/resume`은 목표를 그대로(활성·일시정지·완료) 유지한 채 정확히 마지막 위치에서 재개합니다.

판사 모델 선택

판사는 `goal_judge` 보조 작업을 사용하며 기본은 메인 모델로 해결됩니다. 비용을 절감하려면 `auxiliary.goal_judge.provider: openrouter`, `auxiliary.goal_judge.model: google/gemini-3-flash-preview` 같은 오버라이드를 추가하세요. 판사 호출은 작아서(약 200 출력 토큰) 매 턴 한 번 실행되므로 저렴한 빠른 모델이 일반적으로 적합합니다.

> 출처: Persistent Goals

5.4 Code Execution (코드 실행)

`execute_code` 도구는 에이전트가 Hermes 도구를 프로그램적으로 호출하는 파이썬 스크립트를 작성하여 다단계 워크플로를 단일 LLM 턴으로 줄일 수 있게 합니다. 스크립트는 에이전트 호스트의 샌드박스 자식 프로세스에서 실행되며

Unix 도메인 소켓 RPC로 통신합니다.

작동 방식

에이전트는 `from hermes_tools import ...`로 시작하는 파이썬 스크립트를 작성합니다. Hermes는 RPC 함수가 있는 `hermes_tools.py` 스텁 모듈을 생성하고, Unix 도메인 소켓을 열어 RPC 리스너 스레드를 시작합니다. 스크립트가 자식 프로세스에서 실행되면서 도구 호출이 소켓을 통해 Hermes로 돌아옵니다. 스크립트의 `print()` 출력만 LLM에 반환되고, 중간 도구 결과는 컨텍스트 윈도우에 들어가지 않습니다.

샌드박스에서 사용 가능한 도구

`web_search`, `web_extract`, `read_file`, `write_file`, `search_files`, `patch`, `terminal`(전경 전용)입니다.

에이전트가 이를 사용하는 시점

처리 로직이 사이에 있는 3개 이상의 도구 호출, 대량 데이터 필터링이나 조건부 분기, 결과에 대한 루프가 있을 때 사용합니다. 핵심 이점은 중간 도구 결과가 컨텍스트 윈도우에 들어가지 않고 최종 `print()` 출력만 돌아온다는 것입니다 - 토큰 사용량이 극적으로 줄어듭니다.

리소스 한도

타임아웃 5분(300초, 스크립트는 SIGTERM 후 5초 유예 후 SIGKILL), stdout 50KB(초과 시 절단), stderr 10KB(0이 아닌 종료 시 디버깅용으로 출력에 포함), 도구 호출 50개당 실행. 모두 `code_execution.timeout`, `code_execution.max_tool_calls` 설정으로 조정 가능합니다.

보안 모델

자식 프로세스는 **최소 환경**으로 실행됩니다. API 키·토큰·자격증명은 완전히 제거됩니다. 스크립트는 RPC 채널을 통해서만 도구에 접근하므로 환경변수에서 시크릿을 읽을 수 없습니다. 이름에 `KEY`, `TOKEN`, `SECRET`, `PASSWORD`, `CREDENTIAL`, `PASSWD`, `AUTH`가 들어가는 환경변수는 제외되고, 안전한 시스템 변수(`PATH`, `HOME`, `LANG`, `SHELL`, `PYTHONPATH`, `VIRTUAL_ENV` 등)만 통과합니다.

플랫폼 지원

코드 실행은 Unix 도메인 소켓이 필요하므로 **Linux와 macOS에서만** 사용 가능합니다. Windows에서는 자동으로 비활성화되고 에이전트는 일반 순차 도구 호출로 폴백합니다.

> 출처: Code Execution

5.5 Event Hooks (이벤트 훅)

Hermes에는 라이프사이클 시점에서 커스텀 코드를 실행하는 세 가지 훅 시스템이 있습니다.

게이트웨이 훅(Gateway hooks)은 `~/.hermes/hooks/`의 `HOOK.yaml` + `handler.py`로 등록되며 게이트웨이에서만 작동합니다. 로깅, 알림, 웹훅에 사용됩니다. **플러그인 훅**(Plugin hooks)은 플러그인의 `ctx.register_hook()`으로 등록되며 CLI와 게이트웨이 모두에서 작동합니다. 도구 가로채기, 메트릭, 가드레일에 사용됩니다. **셸 훅**(Shell hooks)은 `~/.hermes/config.yaml`의 `hooks:` 블록에서 셸 스크립트를 가리키며 CLI와 게이트웨이 모두에서 작동합니다. 차단·자동 포맷팅·컨텍스트 주입을 위한 드롭인 스크립트에 사용됩니다.

세 시스템 모두 비차단으로, 어떤 훅의 에러도 잡혀 로깅되며 절대 에이전트를 충돌시키지 않습니다.

게이트웨이 훅 이벤트

`gateway:startup`(게이트웨이 프로세스 시작), `session:start`(새 메시징 세션 생성), `session:end`(세션 종료), `session:reset`(`/new` 또는 `/reset`), `agent:start`(에이전트가 메시지 처리 시작), `agent:step`(도구 호출 루프의 각 반복), `agent:end`(처리 완료), `command:*`(슬래시 명령 실행, 와일드카드 매칭).

핸들러는 `handle`이라는 이름이어야 하며, `event_type`(문자열)과 `context`(dict)를 받습니다. `async def`나 `def` 모두 가능합니다.

플러그인 훅 (CLI + Gateway)

가장 많이 사용되는 훅들입니다.

`pre_tool_call`은 모든 도구 실행 직전에 발생합니다(내장·플러그인 도구 모두). 콜백 시그니처: `def my_callback(tool_name: str, args: dict, task_id: str, **kwargs):`. **반환 값으로 도구 호출을 막을 수 있습니다**: `return {"action": "block", "message": "차단된 이유"}`. 다른 반환 값은 무시되어 기존 관찰자 콜백이 변경 없이 작동합니다. 사용 사례: 로깅, 감사 추적, 도구 호출 카운터, 위험한 작업 차단, 속도 제한.

`post_tool_call`은 모든 도구 실행 반환 후 발생합니다. 시그니처: `def my_callback(tool_name, args, result, task_id, duration_ms, **kwargs):`. 반환 값 무시. 사용 사례: 도구 결과 로깅, 메트릭 수집, 도구 성공/실패율 추적, 지연 대시보드.

`pre_llm_call`은 매 턴마다 한 번, 도구 호출 루프 시작 전에 발생합니다. **반환 값이 사용되는 유일한 훅**입니다 - 현재 턴의 사용자 메시지에 컨텍스트를 주입할 수 있습니다. `return {"context": "회상된 메모리:\n- 사용자는 Python을 좋아함\n- hermes-agent에서 작업 중"}` 또는 `return "회상된 컨텍스트:\n- ..."`. 컨텍스트는 항상 사용자 메시지에 추가되며 시스템 프롬프트에는 절대 추가되지 않습니다(프롬프트 캐시 보존). 모든 주입된 컨텍스트는 일시적이며 세션 데이터베이스에 저장되지 않습니다.

`post_llm_call`은 매 턴마다 한 번, 도구 호출 루프 완료 후 발생합니다(성공한 턴만). 외부 메모리 시스템에 대화 데이터 동기화, 응답 품질 메트릭 계산에 사용합니다.

`on_session_start`(새 세션의 첫 턴), **`on_session_end`**(매 `run_conversation()` 호출 끝), **`on_session_finalize`**(CLI/게이트웨이가 활성 세션을 해체할 때), **`on_session_reset`**(게이트웨이가 새 세션 키를 교체할 때) 같은 라이프사이클 훅도 있습니다.

`subagent_stop`은 `delegate_task` 자식이 종료될 때마다 발생합니다. 위임 활동 로깅, 자식 지속시간 누적, 위임 후 감사 기록에 사용됩니다.

`pre_gateway_dispatch`는 게이트웨이가 사용자 메시지를 받았을 때 인증·디스패치 전에 발생합니다. `{"action": "skip" | "rewrite" | "allow"}`를 반환하여 흐름에 영향을 줄 수 있습니다. 청취 전용 그룹 채팅, 인간 핸드오버, 정책 주도 라우팅에 사용됩니다.

`pre_approval_request`와 **`post_approval_response`**는 위험한 명령에 대한 승인 요청 전후에 발생합니다. 데스크톱 알림, 푸시 알림, 감사 로깅, Slack 웹훅에 사용됩니다.

BOOT.md 튜토리얼

커뮤니티에서 인기 있는 패턴은 `~/.hermes/BOOT.md`에 자연어 시작 지시 파일을 두고, 게이트웨이가 시작될 때마다 에이전트가 이를 한 번 실행하도록 하는 것입니다. "매 부팅 시 야간 cron 실패를 확인하고 Discord에 핑하라" 또는 "마지막 24시간 deploy.log를 요약하여 Slack #ops에 게시하라" 같은 사용 사례에 유용합니다.

`gateway:startup` 이벤트에 등록한 게이트웨이 훅을 만들고, `_resolve_gateway_model()`과 `_resolve_runtime_agent_kwargs()`로 게이트웨이의 현재 구성된 모델과 자격증명을 사용하는 일회성 에이전트를 생성합니다. 응답에 `[SILENT]`가 포함되면 메시지 전송을 옵트아웃할 수 있습니다.

셸 훅

YAML 설정만으로 모든 언어(Bash, Python, Go 바이너리 등)의 스크립트를 등록할 수 있습니다. 각 이벤트에서 Hermes는 매칭하는 모든 훅에 대해 서브프로세스를 생성하고, JSON 페이로드를 stdin으로 파이프하고, stdout을 JSON으로 다시 읽습니다.

stdin 페이로드 구조: `{"hook_event_name": "pre_tool_call", "tool_name": "terminal", "tool_input": {"command": "rm -rf /"}, "session_id": "sess_abc123", "cwd": "/home/user/project", "extra": {"task_id": "...", "tool_call_id": "..."}}`.

stdout 응답: `{"decision": "block", "reason": "Forbidden: rm -rf"}`(또는 Hermes 표준 `{"action": "block", "message": "..."}`). `pre_llm_call`을 위해 `{"context": "오늘은 금요일, 2026-04-17"}`. 빈 또는 비매칭 출력은 무음 무동작입니다.

동의 모델: 각 고유한 (이벤트, 명령) 쌍은 Hermes가 처음 볼 때 사용자에게 승인을 요청하고 결정을 `~/.hermes/shell-hooks-allowlist.json`에 영속화합니다. 비대화형 실행(게이트웨이, cron, CI)은 `--accept-hooks`, `HERMES_ACCEPT_HOOKS=1`, 또는 `hooks_auto_accept: true` 중 하나가 필요합니다.

`hermes hooks` CLI: `hermes hooks list`(설정된 훅 덤프), `hermes hooks test <event>`(합성 페이로드에 대해 매칭하는 모든 훅 발사), `hermes hooks revoke <command>`, `hermes hooks doctor`(실행 비트, 허용 목록 상태, mtime 드리프트, JSON 출력 유효성 확인).

> 출처: Event Hooks

5.6 Batch Processing (배치 처리)

배치 처리는 수백 또는 수천 개의 프롬프트에 걸쳐 Hermes 에이전트를 병렬로 실행하여 구조화된 트레젝토리 데이터를 생성합니다. 주로 **학습 데이터 생성**에 사용되며, 파인튜닝이나 평가에 사용할 수 있는 도구 사용 통계와 함께 ShareGPT 형식 트레젝토리를 생산합니다.

Quick Start

```
python batch_runner.py \
    --dataset_file=data/prompts.jsonl \
    --batch_size=10 \
    --run_name=my_first_run \
    --model=anthropic/claude-sonnet-4-20250514 \
    --num_workers=4
```

`--resume`으로 중단된 실행을 재개할 수 있습니다.

데이터셋 형식

JSONL 파일(한 줄에 하나의 JSON 객체)이며 각 항목은 `prompt` 필드를 가져야 합니다. 선택적으로 `image`/`docker_image`(Docker, Modal, Singularity 백엔드와 함께 작동하는 컨테이너 이미지)와 `cwd`(작업 디렉토리

오버라이드)를 가질 수 있습니다.

출력 형식

`data/<run_name>/`에 다음이 생성됩니다. `trajectories.jsonl`(결합된 최종 출력, 모든 배치 병합), `batch_0.jsonl`, `batch_1.jsonl`, ... (개별 배치 결과), `checkpoint.json`(재개 체크포인트), `statistics.json`(집계 도구 사용 통계).

각 트레젝토리는 ShareGPT 유사 형식(`from`/`value` 필드를 가진 `conversations` 필드)에 메타데이터, `completed`/`partial` 플래그, `api_calls` 카운트, `toolsets_used` 배열, 도구별 통계(`tool_stats`)와 에러 카운트(`tool_error_counts`)를 포함합니다.

도구셋 분포와 품질 필터링

각 프롬프트는 **분포(distribution)**에서 무작위로 도구셋을 받습니다. 다양한 도구 조합을 학습 데이터에 보장합니다. 자동 품질 필터링: 추론 없는 샘플(어시스턴트 턴 중 0개가 추론을 포함) 폐기, 손상된 항목 필터(환각 도구 이름 포함) 제거.

컨텐트 기반 재개

`--resume`은 단순히 인덱스를 매칭하지 않고 실제 텍스트 콘텐츠로 완료된 프롬프트를 매칭하므로, 데이터셋 순서가 변경되어도 복구가 가능합니다.

> 출처: Batch Processing

---

6. 미디어 및 웹

6.1 Voice Mode (음성 모드)

CLI와 메시징 플랫폼 전반에서 완전한 음성 상호작용을 지원합니다. 마이크로 에이전트와 대화하고, 말로 응답을 듣고, Discord 음성 채널에서 라이브 음성 대화를 할 수 있습니다.

CLI 음성 모드

`hermes`로 CLI를 시작하고 `/voice on`으로 음성 모드를 활성화합니다. 그 다음 흐름: **Ctrl+B**(설정 가능)를 누르면 비프음(880Hz)이 재생되고 녹음이 시작됩니다. 라이브 오디오 레벨 바(● [▁▂▃▅▇▇▅▂] ❯)가 입력을 보여줍니다. 3초 침묵 후 자동 정지되며, 두 번의 비프음(660Hz)이 녹음 종료를 확인합니다. 오디오는 Whisper로 전사되어 에이전트로 전송되고, TTS가 활성화된 경우 응답이 음성으로 재생됩니다. 녹음이 자동으로 다시 시작되어 키 누름 없이 다시 말할 수 있습니다.

명령: `/voice` 토글, `/voice on`/`/voice off`, `/voice tts`(TTS 출력 토글), `/voice status`.

침묵 감지 알고리즘

두 단계입니다. 1단계 음성 확인: RMS 임계값(200) 이상의 오디오를 0.3초 이상 대기하며 음절 사이의 짧은 떨어짐 허용. 2단계 종료 감지: 음성 확인 후 3.0초 연속 침묵 시 트리거. 15초 동안 음성이 전혀 감지되지 않으면 자동 정지. 두 임계값 모두 `config.yaml`에서 설정 가능하며 `voice.beep_enabled: false`로 비프음 비활성화도 가능합니다.

스트리밍 TTS

TTS가 활성화되면 에이전트는 텍스트를 생성하면서 **문장별로** 응답을 말합니다 - 전체 응답을 기다릴 필요가 없습니다. 텍스트 델타를 완전한 문장으로 버퍼링(최소 20자), 마크다운 형식과 `<think>` 블록 제거, 문장당 실시간 오디오 생성·재생합니다.

환각 필터

Whisper는 가끔 침묵이나 배경 소음에서 환각 텍스트("Thank you for watching", "Subscribe" 등)를 생성합니다. 에이전트는 다국어 26개 알려진 환각 구문과 반복 변형을 잡는 정규식 패턴으로 이를 필터링합니다.

게이트웨이 음성 응답 (Telegram & Discord)

Telegram과 Discord에서 모드: `off`(텍스트만), `voice_only`(`/voice on`, 음성 메시지를 보낼 때만 음성 응답), `all`(`/voice tts`, 모든 메시지에 음성 응답). 설정은 게이트웨이 재시작 간 영속됩니다.

플랫폼 전송: **Telegram**은 음성 버블(Opus/OGG)로 채팅에서 인라인 재생됩니다. ffmpeg가 MP3 → Opus를 변환합니다. **Discord**는 네이티브 음성 버블(Opus/OGG)로 사용자 음성 메시지처럼 인라인 재생되며, 음성 버블 API 실패 시 파일 첨부로 폴백됩니다.

Discord 음성 채널

가장 몰입적인 음성 기능입니다. 봇이 Discord 음성 채널에 들어가 사용자의 말을 듣고, 음성을 전사하고, 에이전트 파이프라인을 통해 처리하고, TTS로 응답을 다시 음성 채널에서 말합니다.

권한 통합: Connect, Speak, Use Voice Activity가 필요합니다. 권한 정수는 텍스트만 `274878286912`에서 텍스트+음성 `274881432640`으로 업데이트됩니다.

특권 게이트웨이 인텐트 셋: Presence Intent, Server Members Intent, Message Content Intent 모두 필요합니다. **Server Members Intent**가 특히 중요한데, 음성 SSRC 식별자를 Discord 사용자 ID에 매핑하는 데 필수입니다.

명령: `/voice join`(봇이 현재 음성 채널 참가), `/voice channel`(별칭), `/voice leave`(연결 해제), `/voice status`. `/voice join` 전에 음성 채널에 있어야 합니다.

작동 방식: 봇이 음성 채널 참가 후 각 사용자의 오디오 스트림을 독립적으로 수신, 침묵 감지(0.5초 음성 후 1.5초 침묵 시 처리 트리거), Whisper STT로 전사, 전체 에이전트 파이프라인 처리, TTS로 음성 채널에서 응답 말함. 텍스트 채널에 전사 표시(`[Voice] @user: 말한 내용`), 에이전트 응답이 채널에 텍스트로 전송됨과 동시에 음성 채널에서 말해짐.

에코 방지: 봇은 TTS 응답 재생 중 자동으로 오디오 리스너를 일시정지하여 자체 출력을 듣고 재처리하는 것을 방지합니다.

STT/TTS 프로바이더

STT 프로바이더는 우선순위 순으로: **Local**(`faster-whisper`, 무료, API 키 불필요, 모델 크기 tiny/base/small/medium/large-v3 선택 가능), **Groq**(`whisper-large-v3-turbo` 빠름, 무료 티어), **OpenAI**(`whisper-1` 또는 `gpt-4o-transcribe`, 유료). 자동 폴백: local > groq > openai.

TTS 프로바이더: **Edge TTS**(무료 기본, 322개 음성 74개 언어, MP3 출력), **ElevenLabs**(우수 품질, 유료, Opus 네이티브), **OpenAI TTS**(양호 품질, 유료, alloy/echo/fable/onyx/nova/shimmer 음성), **MiniMax**, **Mistral Voxtral**, **Google Gemini**, **xAI**, **NeuTTS**(로컬, 무료, espeak-ng 필요), **KittenTTS**, **Piper**(v0.12.0 추가됨). 커스텀 명령 프로바이더로 모든 로컬 TTS CLI를 사용할 수 있습니다.

> 출처: Voice Mode, Voice & TTS

6.2 Browser Automation (브라우저 자동화)

여러 백엔드 옵션과 함께 완전한 브라우저 자동화 도구셋을 제공합니다.

백엔드 옵션 6가지

Browserbase 클라우드 모드(browserbase.com)는 관리되는 클라우드 브라우저와 안티봇 도구를 제공합니다. 키: `BROWSERBASE_API_KEY`, `BROWSERBASE_PROJECT_ID`. **Browser Use 클라우드 모드**(browser-use.com)는 REST API를 통한 클라우드 브라우저 대안입니다. 키: `BROWSER_USE_API_KEY`. **Firecrawl 클라우드 모드**(firecrawl.dev)는 내장 스크래핑이 있는 클라우드 브라우저입니다. 키: `FIRECRAWL_API_KEY`. **Camofox 로컬 모드**(Camofox)는 Camoufox(C++ 핑거프린트 스푸핑이 있는 Firefox 포크)를 래핑한 자체 호스팅 Node.js 서버로, 클라우드 의존성 없는 로컬 안티탐지 브라우징을 제공합니다. **로컬 Chrome via CDP**(`/browser connect`)로 자체 실행 중인 Chrome 인스턴스에 Chrome DevTools Protocol을 통해 부착할 수 있습니다. **로컬 브라우저 모드**는 클라우드 자격증명을 설정하지 않고 `/browser connect`도 사용하지 않을 때 `agent-browser` CLI와 로컬 Chromium 설치로 작동합니다.

하이브리드 라우팅: 공개 URL은 클라우드, LAN/localhost는 로컬

클라우드 프로바이더가 구성된 경우 Hermes는 사설/루프백/LAN 주소(`localhost`, `127.0.0.1`, `192.168.x.x`, `10.x.x.x`, `172.16-31.x.x`, `*.local`, `*.lan`, `*.internal`, IPv6 루프백 `::1`, 링크 로컬 `169.254.x.x`)로 해결되는 URL에 대해 **로컬 Chromium 사이드카**를 자동 생성합니다. 공개 URL은 같은 대화에서 클라우드 프로바이더를 계속 사용합니다. 일반적인 "로컬에서 개발 중인데 Browserbase를 사용하고 있다" 워크플로를 해결합니다. 클라우드 프로바이더는 사설 URL을 절대 보지 못합니다.

페이지 표현: 접근성 트리

페이지는 **접근성 트리**(텍스트 기반 스냅샷)로 표현되어 LLM 에이전트에 이상적입니다. 상호작용 요소는 클릭과 입력에 사용되는 ref ID(`@e1`, `@e2` 같은)를 받습니다.

도구

`browser_navigate`(URL로 이동, 다른 모든 브라우저 도구 전 호출 필수, Browserbase 세션 초기화), `browser_snapshot`(현재 페이지 접근성 트리 텍스트 기반 스냅샷, `full=false` 컴팩트 뷰는 상호작용 요소만, `full=true`는 전체 콘텐츠, 8000자 초과 시 LLM 자동 요약), `browser_click(ref="@e5")`, `browser_type`(입력 필드에 텍스트 입력, 먼저 지우고 새 텍스트 입력), `browser_scroll`, `browser_press`(Enter, Tab, Escape, ArrowDown, ArrowUp 등), `browser_back`, `browser_get_images`(현재 페이지의 모든 이미지 URL과 alt 텍스트 나열), `browser_vision`(스크린샷 + 비전 AI 분석, 텍스트 스냅샷이 중요한 시각 정보를 포착하지 못할 때 - CAPTCHA, 복잡한 레이아웃, 시각적 검증 챌린지에 특히 유용), `browser_console`(콘솔 출력과 잡히지 않은 JS 예외), `browser_cdp`(원시 Chrome DevTools Protocol 패스스루, 다른 도구로 처리되지 않는 작업의 탈출구), `browser_dialog`(네이티브 JS 대화상자 응답).

세션 녹화

브라우저 세션을 WebM 비디오 파일로 자동 녹화: `browser.record_sessions: true`. 첫 `browser_navigate`에서 자동 시작되고 세션이 닫힐 때 `~/.hermes/browser_recordings/`에 저장됩니다. 로컬과 클라우드(Browserbase) 모드 모두 작동. 72시간 이상 된 녹화는 자동 정리됩니다.

Browserbase 스텔스 기능

기본 스텔스(항상 켜짐, 무작위 핑거프린트·뷰포트 무작위화·CAPTCHA 해결), 주거용 프록시(켜짐, 더 나은 접근성), 고급 스텔스(꺼짐, 커스텀 Chromium 빌드, Scale Plan 필요), Keep Alive(켜짐, 네트워크 문제 후 세션 재연결).

> 출처: Browser Automation

6.3 Vision & Image Paste (비전 및 이미지 붙여넣기)

Hermes Agent는 **멀티모달 비전**을 지원하여 클립보드의 이미지를 CLI에 직접 붙여넣고 에이전트에게 분석·설명·작업을 요청할 수 있습니다. 이미지는 base64 인코딩 콘텐츠 블록으로 모델에 전송되므로 모든 비전 가능 모델이 처리할 수 있습니다.

흐름: 이미지를 클립보드에 복사(스크린샷, 브라우저 이미지 등), 아래 방법 중 하나로 첨부, 질문 입력 후 Enter, 입력 위에 `[ 🖼 Image #1]` 배지로 이미지 표시, 제출 시 비전 콘텐츠 블록으로 모델에 전송. 보내기 전 여러 이미지 첨부 가능(각각 자체 배지), Ctrl+C로 모든 첨부 이미지 지우기. 이미지는 타임스탬프 파일명으로 `~/.hermes/images/`에 PNG로 저장됩니다.

붙여넣기 방법

`/paste` **명령**이 가장 안정적이며 어디서나 작동합니다. 클립보드 백엔드를 명시적으로 호출하므로 터미널 키 바인딩 가로채기를 걱정할 필요가 없습니다. **Ctrl+V / Cmd+V**(브래킷 붙여넣기): 클립보드에 이미지와 텍스트가 함께 있을 때만 작동합니다(일부 앱이 둘 다 클립보드에 둘 때). 클립보드에 이미지만 있으면 대부분의 터미널에서 동작하지 않습니다. **Alt+V**: 대부분의 터미널 에뮬레이터에서 ESC + 키로 통과합니다. VSCode 통합 터미널에서는 Alt+키 콤보를 가로채므로 작동하지 않습니다. **Ctrl+V (Linux 데스크톱 터미널만)**: GNOME Terminal, Konsole, Alacritty 등에서 Ctrl+V는 붙여넣기 단축키가 아니므로(Ctrl+Shift+V가 그것), Ctrl+V가 애플리케이션에 원시 바이트를 보내고 Hermes가 이를 클립보드 검사로 처리합니다.

플랫폼 호환성

macOS Terminal/iTerm2(가장 좋은 경험, `osascript` 항상 사용 가능), Linux X11 데스크톱(`xclip` 필요, `apt install xclip`), Linux Wayland 데스크톱(`wl-paste` 필요, `apt install wl-clipboard`), WSL2(Windows Terminal, `powershell.exe`를 통한 추가 설치 불필요), VSCode Terminal(로컬, Alt+V 미지원), VSCode Terminal(SSH, 클립보드 접근 불가), SSH 터미널(원격 클립보드 접근 불가).

SSH 우회 방법

SSH 세션에서는 클립보드 붙여넣기가 불가능합니다. 대안: 이미지 파일 업로드(scp 등으로 원격 서버에 업로드 후 경로로 참조), URL 사용(이미지가 온라인에 있으면 URL을 메시지에 붙여넣고 `vision_analyze`로 직접 분석), X11 포워딩(`ssh -X`로 X11 포워드, 큰 이미지엔 느림), 메시징 플랫폼 사용(Telegram/Discord/Slack/WhatsApp으로 이미지 전송, 클립보드/터미널 제한 영향 없음).

> 출처: Vision & Image Paste

6.4 Image Generation (이미지 생성)

FAL.ai의 **FLUX 2 Pro** 모델을 사용해 텍스트 프롬프트에서 이미지를 생성하고, **Clarity Upscaler**로 자동 2배 업스케일하여 품질을 향상시킵니다.

작동 방식

생성: 프롬프트가 FLUX 2 Pro 모델(`fal-ai/flux-2-pro`)로 전송됩니다. 업스케일: 생성된 이미지가 Clarity Upscaler(`fal-ai/clarity-upscaler`)로 자동 2배 업스케일됩니다. 전송: 업스케일된 이미지 URL이 반환됩니다. 업스케일이 어떤 이유로든 실패하면 원본 이미지가 폴백으로 반환됩니다.

매개변수

`prompt`(필수), `aspect_ratio`(`landscape`/`square`/`portrait`, 기본 `landscape`), `num_inference_steps`(1-100, 기본 50, 더 많을수록 고품질·느림), `guidance_scale`(0.1-20.0, 기본 4.5, 프롬프트 충실도), `num_images`(1-4, 기본 1), `output_format`(`png`/`jpeg`, 기본 png), `seed`(재현 가능한 결과용).

종횡비 매핑: `landscape` → `landscape_16_9`(배경화면, 배너, 장면), `square` → `square_hd`(프로필 사진, 소셜 미디어), `portrait` → `portrait_16_9`(캐릭터 아트, 폰 배경화면). 원시 FLUX 2 Pro 크기 프리셋(`square_hd`, `square`, `portrait_4_3`, `portrait_16_9`, `landscape_4_3`, `landscape_16_9`)을 직접 사용할 수도 있고, 최대 2048x2048 커스텀 크기도 지원됩니다.

자동 업스케일 설정

업스케일 팩터 2배, Creativity 0.35, Resemblance 0.6, Guidance Scale 4, Inference Steps 18, Positive Prompt(`"masterpiece, best quality, highres"` + 원래 프롬프트), Negative Prompt(`"(worst quality, low quality, normal quality:2)"`).

> 출처: Image Generation

6.5 Voice & TTS

3개의 TTS 프로바이더가 있습니다. **Edge TTS**(기본)는 양호한 품질에 무료이며 API 키가 필요 없습니다. **ElevenLabs**는 뛰어난 품질에 유료이며 `ELEVENLABS_API_KEY`가 필요합니다. **OpenAI TTS**는 양호한 품질에 유료이며 `VOICE_TOOLS_OPENAI_KEY`가 필요합니다.

플랫폼 전송

Telegram은 음성 버블(인라인 재생) Opus `.ogg` 형식으로, Discord는 오디오 파일 첨부 MP3로, WhatsApp은 오디오 파일 첨부 MP3로, CLI는 `~/.hermes/audio_cache/`에 MP3로 저장됩니다.

Telegram 음성 버블과 ffmpeg

Telegram 음성 버블은 Opus/OGG 오디오 형식이 필요합니다. OpenAI와 ElevenLabs는 Opus를 네이티브로 생성하므로 추가 설정이 필요 없습니다. Edge TTS(기본)는 MP3를 출력하므로 변환에 **ffmpeg**가 필요합니다(`apt install ffmpeg` 또는 `brew install ffmpeg`). ffmpeg가 없으면 Edge TTS 오디오는 일반 오디오 파일로 전송됩니다(재생은 되지만 음성 버블이 아닌 직사각형 플레이어로 표시).

STT 프로바이더

`local`(faster-whisper, 무료, 키 불필요, CPU 기본 또는 GPU, 모델 크기 tiny/base/small/medium/large-v3), `groq`(Groq Whisper API, 무료 티어, 빠름), `openai`(`whisper-1`, `gpt-4o-mini-transcribe`, `gpt-4o-transcribe` 지원).

폴백 동작: 로컬 faster-whisper 사용 불가 → 로컬 `whisper` CLI 시도 → 클라우드 프로바이더. Groq 키 없음 → 로컬 → OpenAI. OpenAI 키 없음 → 로컬 → Groq. 아무것도 없음 → 음성 메시지가 사용자에게 정확한 노트와 함께 통과.

> 출처: Voice & TTS

---

7. 관리 도구: Web Dashboard

웹 대시보드는 Hermes Agent 설치를 관리하기 위한 브라우저 기반 UI입니다. YAML 파일을 편집하거나 CLI 명령을 실행하는 대신 깔끔한 웹 인터페이스에서 설정 구성·API 키 관리·세션 모니터링을 할 수 있습니다.

`hermes dashboard` 명령으로 로컬 웹 서버를 시작하고 `http://127.0.0.1:9119`(기본 포트)를 자동으로 엽니다. 대시보드는 완전히 사용자 머신에서 실행되며, 데이터가 localhost를 떠나지 않습니다.

옵션

`--port`(기본 9119), `--host`(기본 127.0.0.1), `--no-open`(자동 브라우저 열기 비활성화), `--insecure`(localhost 외 호스트에 바인딩 허용 - **위험**, 네트워크에 API 키 노출), `--tui`(인브라우저 채팅 탭 노출).

페이지

Status(상태) 페이지는 라이브 개요를 보여줍니다. 에이전트 버전과 릴리스 날짜, 게이트웨이 상태(실행 중/정지, PID, 연결된 플랫폼과 그 상태), 활성 세션(최근 5분 내 활성 세션 카운트), 최근 세션(가장 최근 20개 세션의 모델·메시지 수·토큰 사용량·대화 미리보기). 5초마다 자동 새로고침합니다.

Chat 탭은 전체 Hermes TUI(`hermes --tui`로 얻는 것과 동일한 인터페이스)를 브라우저에 직접 임베드합니다. 슬래시 명령·모델 선택기·도구 호출 카드·마크다운 스트리밍·승인 프롬프트·스킨 테마 모두 동일하게 작동합니다. `/api/pty` WebSocket이 PTY 뒤에서 `hermes --tui`를 생성하고 ANSI 출력을 브라우저로 스트리밍하는 xterm.js WebGL 렌더러로 픽셀 단위 셀 레이아웃을 그립니다. **Sessions** 탭의 재생 아이콘(▶)으로 기존 세션을 재개할 수 있습니다.

Config 페이지는 `config.yaml`의 폼 기반 편집기입니다. 150개 이상의 모든 구성 필드가 `DEFAULT_CONFIG`에서 자동 발견되어 탭 카테고리(model, terminal, display, agent, delegation, memory, approvals 등)로 조직됩니다. 알려진 유효 값(터미널 백엔드, 스킨, 승인 모드 등)이 있는 필드는 드롭다운으로, 부울은 토글로, 나머지는 텍스트 입력으로 렌더링됩니다. 액션: 저장, 기본값으로 리셋, JSON으로 내보내기, JSON 가져오기.

API Keys 페이지는 API 키와 자격증명이 저장되는 `.env` 파일을 관리합니다. LLM 프로바이더(OpenRouter, Anthropic, OpenAI, DeepSeek 등), 도구 API 키(Browserbase, Firecrawl, Tavily, ElevenLabs 등), 메시징 플랫폼(Telegram, Discord, Slack 봇 토큰 등), 에이전트 설정(`API_SERVER_ENABLED` 같은 비밀 아닌 환경변수)으로 그룹화됩니다.

Sessions 페이지는 모든 에이전트 세션을 탐색·검사할 수 있게 합니다. 각 행은 세션 제목, 소스 플랫폼 아이콘, 모델 이름, 메시지 수, 도구 호출 수, 마지막 활성 이후 시간을 표시합니다. 라이브 세션은 펄스 배지로 표시됩니다. FTS5를 사용한 모든 메시지 콘텐츠 전체 텍스트 검색, 클릭으로 세션 확장 시 전체 메시지 이력 로드(역할별 색상 코딩, 마크다운 + 구문 강조 렌더링).

Logs 페이지는 에이전트, 게이트웨이, 에러 로그 파일을 필터링과 라이브 테일링으로 봅니다. 파일·레벨·컴포넌트·라인 수 필터, 자동 새로고침 토글, 심각도별 색상 코딩.

Analytics 페이지는 세션 이력에서 계산된 사용량 및 비용 분석을 제공합니다. 시간 기간 선택(7/30/90일), 요약 카드(총 토큰 입출력, 캐시 적중률, 총 추정/실제 비용, 일평균 세션 수), 일별 토큰 차트(스택 막대 그래프), 일별 분해 표, 모델별 분해.

Cron 페이지에서 cron 작업을 생성·관리합니다. 이름·프롬프트·cron 표현식·전송 타깃 입력, 작업 목록, 일시정지/재개, 즉시 트리거, 삭제.

Skills 페이지에서 스킬과 도구셋을 탐색·검색·토글합니다. 카테고리 필터(MLOps, MCP, Red Teaming, AI), 개별 스킬 활성화/비활성화 스위치.

6개 내장 테마

Hermes Teal(기본, 어두운 청록 + 크림), **Midnight**(짙은 블루-바이올렛, Inter + JetBrains Mono), **Ember**(따뜻한 진홍 + 청동, Spectral + IBM Plex Mono), **Mono**(그레이스케일, 컴팩트), **Cyberpunk**(검정 위 네온 그린, Share Tech Mono), **Rosé**(핑크 + 아이보리, Fraunces, 여유로운). 헤더 바에서 라이브 전환 가능, `config.yaml`의 `dashboard.theme`에 영속됩니다.

REST API

대시보드는 프론트엔드가 사용하는 REST API를 노출하여 자동화에 직접 호출할 수 있게 합니다. `/api/status`(에이전트 버전·게이트웨이 상태), `/api/sessions`(20개 최근 세션), `/api/config`(현재 config.yaml), `/api/config/defaults`, `/api/config/schema`, `/api/env`(환경변수 관리), `/api/sessions/{id}`, `/api/sessions/{id}/messages`, `/api/sessions/search`, `/api/logs`, `/api/analytics/usage`, `/api/cron/jobs`(생성/일시정지/재개/트리거/삭제), `/api/skills`(토글), `/api/tools/toolsets`.

> 출처: Web Dashboard

---

8. 고급 기능: RL Training

Hermes Agent에는 **Tinker-Atropos** 기반의 통합 RL(Reinforcement Learning) 학습 파이프라인이 포함되어 있습니다. LoRA 어댑터를 사용한 GRPO(Group Relative Policy Optimization)로 환경 특화 작업에 대해 언어 모델을 학습시킬 수 있으며, 모두 에이전트의 도구 인터페이스를 통해 오케스트레이션됩니다.

세 가지 컴포넌트

Atropos: 환경 상호작용 조율, 롤아웃 그룹 관리, 어드밴티지 계산을 담당하는 트레젝토리 API 서버. **Tinker**: 모델 가중치, LoRA 학습, 샘플링/추론, 옵티마이저 단계를 처리하는 학습 서비스. **Environments**: 작업·점수·보상 함수를 정의하는 파이썬 클래스(예: GSM8K 수학 문제).

요구사항

Python ≥ 3.11(Tinker 패키지 요구사항), Tinker 학습 서비스의 `TINKER_API_KEY`, Weights & Biases 메트릭 추적의 `WANDB_API_KEY`, `tinker-atropos` 서브모듈. 두 키 모두 있고 Python ≥ 3.11이면 `rl` 도구셋이 자동 활성화됩니다.

도구

`rl_list_environments`(사용 가능한 RL 환경 발견), `rl_select_environment`(환경 선택과 설정 로드), `rl_get_current_config`(설정 가능 및 잠긴 필드 보기), `rl_edit_config`(설정 가능한 학습 매개변수 수정), `rl_start_training`(학습 실행 시작 - 3개 프로세스 생성), `rl_check_status`(학습 진행과 WandB 메트릭 모니터링), `rl_stop_training`(실행 중인 학습 작업 정지), `rl_get_results`(최종 메트릭과 모델 가중치 경로 조회), `rl_list_runs`(모든 활성 및 완료된 실행 목록), `rl_test_inference`(OpenRouter 사용 빠른 추론 테스트).

작업 흐름

환경 발견(`rl_list_environments` AST 파싱으로 `BaseEnv` 상속 클래스 스캔), 선택과 설정(설정 가능 필드: group_size·batch_size·wandb_name 등 / 잠긴 필드: tokenizer_name·rollout_server_url·max_token_length·max_num_workers·total_steps·lora_rank·learning_rate 등), 학습 시작(YAML 설정 파일 생성, 고유 실행 ID 생성, 3개 프로세스 생성: Atropos API 서버 / Tinker 트레이너 / 환경, 5초/30초/90초 시차로 시작), 진행 모니터링(상태 확인은 30분마다 한 번 속도 제한), 정지 또는 결과 가져오기.

추론 테스트

전체 학습 실행에 시간을 들이기 전에 `rl_test_inference`로 환경이 올바르게 작동하는지 테스트할 수 있습니다. OpenRouter를 사용한 추론 및 점수 매기기 몇 단계 실행(Tinker API 키 불필요, `OPENROUTER_API_KEY`만 있으면 됨). 기본 설정: 3단계 × 16개 완성 = 모델당 48개 롤아웃, 견고성을 위한 3개 모델 테스트(qwen/qwen3-8b 작은, z-ai/glm-4.7-flash 중간, minimax/minimax-m2.7 큰), 총 ~144개 롤아웃.

> 출처: RL Training

---

9. 메시징 플랫폼

Hermes Agent는 **19개 이상의 메시징 플랫폼**을 단일 백그라운드 게이트웨이 프로세스로 연결합니다. 게이트웨이는 모든 구성된 플랫폼에 연결하고, 세션을 처리하고, cron 작업을 실행하고, 음성 메시지를 전달합니다.

플랫폼 비교 표 (요약)

각 플랫폼별 기능 지원: 음성(TTS 응답·음성 메시지 전사), 이미지(송수신), 파일(첨부), 스레드(스레드 대화), 반응(이모지), 타이핑(처리 중 표시), 스트리밍(메시지 진행 업데이트).

Telegram: 모두 지원(반응만 미지원). **Discord**: 모두 지원(반응 포함). **Slack**: 모두 지원. **WhatsApp**: 음성·스레드·반응 미지원, 나머지 지원. **Signal**: 비슷. **SMS**: 모두 미지원(텍스트만). **Email**: 이미지·파일·스레드 지원, 음성·반응·타이핑·스트리밍 미지원. **Home Assistant**: 모두 미지원(텍스트만). **Mattermost**: 반응만 미지원. **Matrix**: 모두 지원. **DingTalk**: 음성·스레드 미지원. **Feishu/Lark**: 모두 지원. **WeCom**: 스레드·반응 미지원. **WeCom Callback**: 모두 미지원. **Weixin**: 스레드·반응 미지원. **BlueBubbles**(iMessage): 음성·스레드·스트리밍 미지원. **QQ**: 스레드·반응·스트리밍 미지원. **Yuanbao**: 스레드·반응·스트리밍 미지원. **Microsoft Teams**: 음성·파일·반응·스트리밍 미지원.

게이트웨이 명령

`hermes gateway`(전경 실행), `hermes gateway setup`(메시징 플랫폼 대화형 구성), `hermes gateway install`(사용자 서비스로 설치 - Linux는 systemd, macOS는 launchd), `sudo hermes gateway install --system`(Linux 부팅 시 시스템 서비스), `hermes gateway start`/`stop`/`status`.

메시징 내 채팅 명령

20개 이상의 슬래시 명령이 모든 메시징 플랫폼에서 작동합니다. 주요 명령: `/new` 또는 `/reset`(새 대화), `/model [provider:model]`(모델 변경), `/personality [name]`(인격 설정), `/retry`(마지막 메시지 재시도), `/undo`(마지막 교환 제거), `/status`, `/stop`(실행 중 에이전트 정지), `/approve`/`/deny`(위험 명령 승인), `/sethome`(이 채팅을 홈 채널로 설정), `/compress`(수동 컨텍스트 압축), `/title [name]`(세션 제목 설정), `/resume [name]`(이전 명명된 세션 재개), `/usage`(토큰 사용량), `/insights [days]`(분석), `/reasoning [level|show|hide]`, `/voice`(음성 제어),

`/rollback [number]`(파일시스템 체크포인트), `/background <prompt>`(별도 백그라운드 세션에서 실행), `/reload-mcp`, `/update`, `/help`, `/<skill-name>`(설치된 모든 스킬 호출).

세션 관리

세션은 메시지 간 영속됩니다. 리셋 정책은 `daily`(특정 시간 매일, 기본 4:00 AM), `idle`(N분 비활성 후, 기본 1440분 = 24시간), `both`(둘 중 먼저 발생). `~/.hermes/gateway.json`에서 플랫폼별 오버라이드 가능.

보안: 기본 거부 + DM 페어링

기본적으로 게이트웨이는 허용 목록에 없거나 DM을 통해 페어링되지 않은 모든 사용자를 거부합니다. 이는 터미널 접근 권한을 가진 봇의 안전한 기본값입니다. `TELEGRAM_ALLOWED_USERS`, `DISCORD_ALLOWED_USERS`, `SIGNAL_ALLOWED_USERS` 등 플랫폼별 환경변수로 사용자 ID를 화이트리스트할 수 있습니다.

DM 페어링: 알 수 없는 사용자가 봇에 DM을 보내면 일회성 페어링 코드를 받습니다("페어링 코드: XKGH5N7P"). 관리자가 `hermes pairing approve telegram XKGH5N7P`로 승인합니다. 페어링 코드는 1시간 후 만료, 속도 제한, 암호학적 무작위성 사용.

에이전트 인터럽트와 busy_input_mode

에이전트가 작업 중일 때 메시지를 보내면 인터럽트됩니다. 진행 중인 터미널 명령은 즉시 종료(SIGTERM 후 1초 후 SIGKILL), 도구 호출이 취소되어 현재 실행 중인 것만 진행됩니다. 인터럽트 중 보낸 여러 메시지는 하나의 프롬프트로 결합됩니다.

`busy_input_mode` 설정: `interrupt`(기본, 즉시 인터럽트), `queue`(후속 메시지가 대기하여 현재 작업 완료 후 다음 턴으로 실행), `steer`(`/steer`를 통해 현재 실행에 주입, 다음 도구 호출 후 에이전트에 도착, 인터럽트나 새 턴 없음).

백그라운드 세션

`/background <prompt>`로 별도 백그라운드 세션에서 프롬프트를 실행할 수 있습니다. 메인 채팅은 완전히 인터랙티브 상태로 유지되고, 작업 완료 시 결과가 동일한 채팅으로 반환됩니다. 사용 사례: 서버 모니터링("/background 모든 서비스 헬스 체크하고 다운된 것 알려줘"), 긴 빌드("/background 스테이징 환경 빌드 및 배포"), 연구 작업("/background 경쟁사 가격 조사하고 표로 요약"), 파일 작업("/background 다운로드 폴더의 사진을 날짜별로 폴더에 정리").

`display.background_process_notifications` 설정으로 알림 모드 제어: `all`(실행 출력 업데이트와 최종 완료 메시지), `result`(최종 완료 메시지만), `error`(0이 아닌 종료 코드 시 최종 메시지만), `off`(프로세스 와처 메시지 없음).

19개 플랫폼 개별 설정 페이지

각 메시징 플랫폼은 자체 설정 가이드를 가집니다(공식 문서의 메시징 플랫폼 섹션 참조).

Telegram, Discord, Slack, WhatsApp, Signal, Email, SMS (Twilio), Home Assistant, Mattermost, Matrix, DingTalk, Feishu/Lark, WeCom, WeCom Callback, Weixin (WeChat), BlueBubbles (iMessage), QQ Bot, Yuanbao, Microsoft Teams, Open WebUI, Webhooks.

Webhooks

GitHub, GitLab, JIRA, Stripe 등 외부 서비스에서 이벤트를 받아 Hermes 에이전트 실행을 자동으로 트리거할 수 있습니다. 웹훅 어댑터는 POST 요청을 받고, HMAC 서명을 검증하고, 페이로드를 에이전트 프롬프트로 변환하고, 응답을 소

스 또는 다른 구성된 플랫폼으로 라우팅하는 HTTP 서버를 실행합니다.

라우트 속성: `events`(허용 이벤트 타입), `secret`(필수, HMAC 시크릿, 라우트가 없으면 전역 fallback), `prompt`(점 표기법 페이로드 접근이 있는 템플릿 - `{pull_request.title}`, `{__raw__}`로 전체 페이로드 덤프), `skills`(에이전트 실행에 로드할 스킬), `deliver`(응답 전송 위치: `github_comment`, `telegram`, `discord`, `slack` 등 19개 옵션), `deliver_extra`(추가 전송 설정), `deliver_only`(true이면 에이전트를 완전히 건너뛰고 렌더링된 prompt 템플릿을 그대로 메시지로 전송 - 0 LLM 토큰, 1초 미만 전송).

직접 전송 모드(Direct Delivery Mode)는 `deliver_only: true`로 활성화하며, 다음 사용 사례에 유용합니다: 외부 서비스 푸시(Supabase/Firebase 웹훅이 DB 변경에 발사 → Telegram 사용자에게 즉시 알림), 모니터링 알림(Datadog/Grafana 알림 웹훅 → Discord 채널로 푸시), 에이전트 간 핑(에이전트 A가 에이전트 B의 사용자에게 긴 작업이 완료되었음을 알림), 백그라운드 작업 완료(cron 작업 완료 → Slack에 결과 게시).

동적 구독: `hermes webhook subscribe github-issues --events "issues" --prompt "..." --deliver telegram --deliver-chat-id "-100..." --description "..."` 같은 CLI 명령으로 게이트웨이 재시작 없이 즉시 라이브 웹훅 구독을 만들 수 있습니다. 어댑터가 매 들어오는 요청에서 `~/.hermes/webhook_subscriptions.json`을 핫 리로드합니다(mtime 게이트, 무시할 만한 오버헤드).

보안: HMAC 서명 검증(GitHub의 `X-Hub-Signature-256`, GitLab의 `X-Gitlab-Token`, 일반 `X-Webhook-Signature`), 시크릿 필수(개발/테스트 전용으로 `"INSECURE_NO_AUTH"` 가능), 속도 제한(라우트당 분당 30 요청 기본), 멱등성(전송 ID 1시간 캐시), 본문 크기 제한(1MB 기본).

> 출처: Messaging Gateway, Webhooks

---

10. 통합(Integrations)

10.1 MCP (Model Context Protocol)

MCP는 Hermes Agent를 외부 도구 서버에 연결하여 에이전트가 Hermes 외부에 존재하는 도구(GitHub, 데이터베이스, 파일 시스템, 브라우저 스택, 내부 API 등)를 사용할 수 있게 합니다.

두 가지 MCP 서버 종류

Stdio 서버는 로컬 서브프로세스로 실행되며 stdin/stdout으로 통신합니다. 서버가 로컬에 설치된 경우, 로컬 리소스에 저지연 접근을 원하는 경우, MCP 서버 문서가 `command`, `args`, `env`를 보여주는 경우에 사용합니다.

```
mcp_servers:
  github:
    command: "npx"
    args: ["-y", "@modelcontextprotocol/server-github"]
    env:
      GITHUB_PERSONAL_ACCESS_TOKEN: "***"
```

HTTP 서버는 Hermes가 직접 연결하는 원격 엔드포인트입니다.

```
mcp_servers:
  remote_api:
```

```
    url: "https://mcp.example.com/mcp"
    headers:
      Authorization: "Bearer ***"
```

MCP 도구가 Hermes에 등록되는 방식

Hermes는 충돌을 피하기 위해 MCP 도구 이름에 접두사를 붙입니다: `mcp_<server_name>_<tool_name>`. 예: `filesystem` 서버의 `read_file`은 `mcp_filesystem_read_file`이 됩니다. `github` 서버의 `create-issue`는 `mcp_github_create_issue`가 됩니다.

서버별 필터링

개별 MCP 서버가 Hermes에 기여하는 도구를 제어할 수 있습니다. `enabled: false`(서버 완전 비활성화), `tools.include: [create_issue, list_issues]`(화이트리스트), `tools.exclude: [delete_customer]`(블랙리스트), `tools.prompts: false` / `tools.resources: false`(유틸리티 래퍼 비활성화). 우선순위 규칙: `include`와 `exclude`가 모두 있으면 `include`가 이깁니다.

동적 도구 발견

MCP 서버는 `notifications/tools/list_changed` 알림을 보내 런타임에 사용 가능한 도구가 변경되었음을 Hermes에 알릴 수 있습니다. Hermes가 이 알림을 받으면 자동으로 서버의 도구 목록을 다시 가져오고 레지스트리를 업데이트합니다 - 수동 `/reload-mcp` 불필요. 동적으로 능력이 변경되는 MCP 서버(새 데이터베이스 스키마 로드 시 도구 추가, 서비스 오프라인 시 도구 제거)에 유용합니다.

MCP 샘플링 지원

MCP 서버는 `sampling/createMessage` 프로토콜을 통해 Hermes에 LLM 추론을 요청할 수 있습니다. 이를 통해 MCP 서버가 자체 모델 접근 없이도 LLM 능력이 필요한 경우 Hermes에 텍스트 생성을 요청할 수 있습니다. 샘플링은 모든 MCP 서버에 대해 기본 활성화되어 있으며, 서버별로 `sampling` 키 아래 구성합니다(`enabled`, `model`, `max_tokens_cap`, `timeout`, `max_rpm` 슬라이딩 윈도우 속도 제한, `max_tool_rounds`, `allowed_models`).

Hermes를 MCP 서버로 실행하기

다른 MCP 호환 에이전트(Claude Code, Cursor, Codex 또는 모든 MCP 클라이언트)가 Hermes의 메시징 능력을 사용할 수 있도록 Hermes를 MCP 서버로 실행할 수 있습니다 - 대화 목록, 메시지 이력 읽기, 모든 연결된 플랫폼에 메시지 전송.

`hermes mcp serve`로 stdio MCP 서버를 시작합니다. 노출되는 10개 도구: `conversations_list`(활성 메시징 대화 목록, 플랫폼별 필터 또는 이름 검색), `conversation_get`(세션 키로 대화 상세 정보), `messages_read`(대화의 최근 메시지 이력 읽기), `attachments_fetch`(특정 메시지에서 비텍스트 첨부 - 이미지, 미디어 - 추출), `events_poll`(커서 위치 이후 새 대화 이벤트 폴링), `events_wait`(다음 이벤트가 도착할 때까지 롱폴/차단), `messages_send`(플랫폼을 통한 메시지 전송 - `telegram:123456`, `discord:#general`), `channels_list`(모든 플랫폼의 사용 가능한 메시징 타깃 목록), `permissions_list_open`(이 브리지 세션 동안 관찰된 보류 중 승인 요청), `permissions_respond`(보류 중 승인 요청 허용 또는 거부).

라이브 이벤트 브리지가 Hermes의 세션 데이터베이스를 폴링하여 새 메시지를 찾고 MCP 클라이언트에 거의 실시간 들어오는 대화 인식을 제공합니다. 이벤트 큐는 메모리 내이며 브리지 연결 시 시작됩니다.

> 출처: MCP (Model Context Protocol)

10.2 ACP (Agent Client Protocol) Editor Integration

Hermes Agent는 ACP 서버로 실행될 수 있어 ACP 호환 에디터가 stdio를 통해 Hermes와 대화하고 채팅 메시지·도구 활동·파일 diff·터미널 명령·승인 프롬프트·스트리밍된 사고/응답 청크를 렌더링할 수 있게 합니다.

Hermes가 ACP 모드에서 노출하는 것

에디터 워크플로용으로 큐레이션된 `hermes-acp` 도구셋과 함께 실행됩니다. 파일 도구(`read_file`, `write_file`, `patch`, `search_files`), 터미널 도구(`terminal`, `process`), 웹/브라우저 도구, 메모리·todo·세션 검색, 스킬, `execute_code`와 `delegate_task`, 비전. 일반적인 에디터 UX에 맞지 않는 메시징 전송과 cron 관리는 의도적으로 제외됩니다.

설치와 실행

`pip install -e '.[acp]'`로 ACP 추가 모듈을 설치하면 `hermes acp`, `hermes-acp`, `python -m acp_adapter` 명령이 활성화됩니다. 모두 Hermes를 ACP 모드로 시작합니다. Hermes는 stderr로 로깅하므로 stdout은 ACP JSON-RPC 트래픽을 위해 예약됩니다.

에디터 설정

VS Code: ACP 클라이언트 익스텐션 설치 후 저장소의 `acp_registry/` 디렉토리를 가리키도록 합니다. Zed: `~/.config/zed/settings.json`에 `agent_servers` 항목 추가. JetBrains: ACP 호환 플러그인 사용 후 `acp_registry` 경로 설정.

작업 디렉토리 동작

ACP 세션은 에디터의 cwd를 Hermes 작업 ID에 바인딩하므로 파일과 터미널 도구가 서버 프로세스 cwd가 아닌 에디터 워크스페이스에 상대적으로 실행됩니다.

승인

위험한 터미널 명령은 에디터로 다시 라우팅되어 승인 프롬프트로 표시될 수 있습니다. ACP 승인 옵션은 CLI 흐름보다 단순합니다: 한 번 허용, 항상 허용, 거부. 타임아웃이나 에러 시 승인 브리지는 요청을 거부합니다.

> 출처: ACP Editor Integration

10.3 API Server (OpenAI 호환)

API 서버는 Hermes Agent를 OpenAI 호환 HTTP 엔드포인트로 노출합니다. OpenAI 형식을 사용하는 모든 프론트엔드(Open WebUI, LobeChat, LibreChat, NextChat, ChatBox, AnythingLLM, Jan, HF Chat-UI, big-AGI, OpenAI Python SDK 등 수백 개)가 Hermes Agent에 연결하여 백엔드로 사용할 수 있습니다.

활성화와 시작

`~/.hermes/.env`에 `API_SERVER_ENABLED=true`, `API_SERVER_KEY=...` 추가, `hermes gateway` 시작. 기본적으로 `http://127.0.0.1:8642`에서 수신합니다.

엔드포인트

`POST /v1/chat/completions`는 표준 OpenAI Chat Completions 형식이며 무상태(전체 대화가 각 요청에 `messages` 배열로 포함). 인라인 이미지 입력 지원(`content`로 `text`와 `image_url` 부분 배열). `"stream": true`로 SSE 스트리밍 가능 - 표준 `chat.completion.chunk` 이벤트와 도구 시작 가시성을 위한 Hermes 커스텀 `hermes.tool.progress` 이벤트.

`POST /v1/responses`는 OpenAI Responses API 형식이며 `previous_response_id`를 통한 서버 측 대화 상태 지원 - 서버가 도구 호출과 결과 포함 전체 대화 이력을 저장하므로 클라이언트가 관리하지 않아도 다중 턴 컨텍스트가 보존됩니다. `conversation: "my-project"` 같은 명명된 대화로 응답 ID를 추적하지 않고도 자동 체이닝 가능합니다.

`GET /v1/models`는 에이전트를 사용 가능한 모델로 나열합니다. 광고된 모델 이름은 기본적으로 프로파일 이름(또는 기본 프로파일은 `hermes-agent`)입니다.

`GET /v1/capabilities`는 외부 UI·오케스트레이터·플러그인 브리지를 위해 API 서버의 안정적 표면을 머신 읽기 가능 형태로 반환합니다.

`GET /health`(`/v1/health`도 가능)와 **`GET /health/detailed`**(활성 세션, 실행 중 에이전트, 리소스 사용 보고)도 있습니다.

Runs API

`/v1/chat/completions`와 `/v1/responses` 외에 서버는 클라이언트가 자체 스트리밍 관리 대신 진행 이벤트를 구독하길 원하는 장기 세션을 위한 **runs API**를 노출합니다. **`POST /v1/runs`**(새 에이전트 실행 생성, `run_id` 반환), **`GET /v1/runs/{run_id}`**(현재 실행 상태 폴링), **`GET /v1/runs/{run_id}/events`**(실행의 도구 호출 진행, 토큰 델타, 라이프사이클 이벤트의 SSE 스트림), **`POST /v1/runs/{run_id}/stop`**.

Jobs API

배경 스케줄 작업을 위한 가벼운 jobs CRUD 표면. **`GET /api/jobs`**, **`POST /api/jobs`**(`hermes cron`과 동일한 모양 - 프롬프트, 스케줄, 스킬, 프로바이더 오버라이드, 전송 타깃), **`GET/PATCH/DELETE /api/jobs/{job_id}`**, **`POST /api/jobs/{job_id}/pause/resume/run`**.

인증과 보안

Bearer 토큰 인증(`Authorization: Bearer ***`). API 서버는 Hermes Agent의 도구셋 전체(터미널 명령 포함)에 대한 접근을 제공하므로 비루프백 주소(`0.0.0.0` 같은)에 바인딩할 때 `API_SERVER_KEY`가 **필수**입니다. CORS는 명시적 허용 목록(`API_SERVER_CORS_ORIGINS`)으로 제한합니다.

다중 사용자 설정 with Profiles

여러 사용자에게 자체 격리된 Hermes 인스턴스(별도 설정·메모리·스킬)를 제공하려면 프로파일을 사용합니다. `hermes profile create alice`/`bob` 후 각 프로파일의 API 서버를 다른 포트에 구성합니다. Open WebUI에서 각각을 별도 연결로 추가하면 모델 드롭다운이 `alice`와 `bob`을 별개의 모델로 보여주며 각각이 완전히 격리된 Hermes 인스턴스로 백업됩니다.

Proxy Mode

API 서버는 게이트웨이 프록시 모드의 백엔드로도 작동합니다. 다른 Hermes 게이트웨이 인스턴스가 `GATEWAY_PROXY_URL`을 이 API 서버에 가리키도록 설정되면 자체 에이전트를 실행하는 대신 모든 메시지를 여기로 전

달합니다. 분할 배포(예: Matrix E2EE를 처리하는 Docker 컨테이너가 호스트 측 에이전트로 릴레이)를 가능하게 합니다.

> 출처: API Server

10.4 Provider Routing (OpenRouter 한정)

OpenRouter를 LLM 프로바이더로 사용할 때 Hermes Agent는 **프로바이더 라우팅**을 지원하여 어떤 기저 AI 프로바이더가 요청을 처리하고 어떻게 우선순위를 매길지 세밀하게 제어할 수 있습니다.

옵션

`sort: "price"`(저렴한 프로바이더 먼저), `"throughput"`(가장 빠른 토큰/초), `"latency"`(첫 토큰까지 가장 짧은 시간). `only`(화이트리스트, 이 프로바이더만 사용). `ignore`(블랙리스트, 절대 사용 안 함). `order`(명시적 우선순위 순서, 나열되지 않은 프로바이더는 폴백). `require_parameters: true`(요청의 모든 매개변수를 지원하는 프로바이더만 라우팅). `data_collection: "deny"`(프로바이더가 학습용으로 프롬프트 사용 차단).

실용적 예시

비용 최적화: `sort: "price"`. 속도 최적화: `sort: "latency"`. 처리량 최적화: `sort: "throughput"`. 특정 프로바이더 잠금: `only: ["Anthropic"]`. 특정 프로바이더 회피: `ignore: ["Together", "Lepton"], data_collection: "deny"`. 폴백 있는 선호 순서: `order: ["Anthropic", "Google"], require_parameters: true`.

> 출처: Provider Routing

10.5 Fallback Providers (폴백 프로바이더)

Hermes Agent에는 프로바이더가 문제를 만났을 때 세션을 계속 실행시키는 세 가지 복원력 계층이 있습니다.

1. **자격증명 풀**(아래 섹션): 같은 프로바이더의 여러 API 키를 회전(첫 번째 시도)
2. **주 모델 폴백**: 메인 모델 실패 시 자동으로 다른 provider:model 쌍으로 전환
3. **보조 작업 폴백**: 비전·압축·웹 추출 등 부가 작업을 위한 독립 프로바이더 해결

자격증명 풀은 같은 프로바이더 회전을 처리하고, 폴백 프로바이더는 크로스 프로바이더 페일오버입니다. 풀이 먼저 시도되며 모든 풀 키가 소진되면 폴백 프로바이더가 활성화됩니다.

주 모델 폴백 설정

```
fallback_model:
  provider: openrouter
  model: anthropic/claude-sonnet-4
```

`provider`와 `model` 모두 필수입니다. 둘 중 하나라도 없으면 폴백이 비활성화됩니다.

25개 이상의 지원 프로바이더

AI Gateway, OpenRouter, Nous Portal, OpenAI Codex, GitHub Copilot, GitHub Copilot ACP, Anthropic, z.ai/GLM, Kimi/Moonshot, MiniMax, MiniMax (China), DeepSeek, NVIDIA NIM, Ollama Cloud, Google Gemini

(OAuth), Google AI Studio, xAI (Grok), AWS Bedrock, Qwen Portal (OAuth), OpenCode Zen, OpenCode Go, Kilo Code, Xiaomi MiMo, Arcee AI, Alibaba/DashScope, Hugging Face, 커스텀 엔드포인트.

폴백 트리거 조건

기본 모델이 다음으로 실패할 때 폴백이 자동으로 활성화됩니다. 속도 제한(HTTP 429)은 재시도 시도 소진 후, 서버 에러(HTTP 500/502/503)도 재시도 시도 소진 후, 인증 실패(HTTP 401/403)는 즉시(재시도 의미 없음), Not Found(HTTP 404)는 즉시, 잘못된 응답은 API가 반복적으로 잘못된 형식이나 빈 응답 반환 시.

활성화되면 Hermes는 폴백 프로바이더의 자격증명을 해결하고, 새 API 클라이언트를 빌드하고, 모델·프로바이더·클라이언트를 즉시 교체하고, 재시도 카운터를 리셋한 후 대화를 계속합니다. 전환은 매끄럽습니다 - 대화 이력·도구 호출·컨텍스트가 보존됩니다.

턴 단위 (세션 단위 아님)

폴백은 **턴 범위**입니다. 각 새 사용자 메시지는 기본 모델이 복원된 상태로 시작합니다. 기본이 턴 중에 실패하면 폴백이 그 턴에만 활성화됩니다. 다음 메시지에서 Hermes는 다시 기본을 시도합니다. 단일 턴 내에서 폴백은 최대 한 번 활성화됩니다 - 폴백도 실패하면 일반 에러 처리(재시도 후 에러 메시지)가 인계됩니다.

보조 작업 자동 감지 체인

Hermes는 비전·웹 추출·압축·세션 검색·스킬 허브·MCP·메모리 플러시·승인 분류·제목 생성 같은 부가 작업에 별도의 가벼운 모델을 사용합니다. 각 작업의 프로바이더가 `"auto"`(기본)로 설정되면 Hermes는 작동할 때까지 순서대로 프로바이더를 시도합니다.

텍스트 작업(압축·웹 추출 등): OpenRouter → Nous Portal → 커스텀 엔드포인트 → Codex OAuth → API 키 프로바이더(z.ai, Kimi, MiniMax, Xiaomi MiMo, Hugging Face, Anthropic) → 포기.

비전 작업: 메인 프로바이더(비전 가능한 경우) → OpenRouter → Nous Portal → Codex OAuth → Anthropic → 커스텀 엔드포인트 → 포기.

해결된 프로바이더가 호출 시 실패하면 Hermes는 내부 재시도도 가집니다: 프로바이더가 OpenRouter가 아니고 명시적 `base_url`이 설정되지 않은 경우 OpenRouter를 마지막 수단 폴백으로 시도합니다.

> 출처: Fallback Providers

10.6 Credential Pools (자격증명 풀)

자격증명 풀은 같은 프로바이더에 대해 여러 API 키 또는 OAuth 토큰을 등록할 수 있게 합니다. 한 키가 속도 제한이나 청구 할당량에 도달하면 Hermes는 자동으로 다음 건강한 키로 회전하여 프로바이더를 전환하지 않고도 세션을 살아있게 유지합니다.

흐름

요청 → 풀에서 키 선택(round_robin / least_used / fill_first / random) → 프로바이더로 전송. 429 속도 제한? → 같은 키 한 번 재시도(일시적 깜박임) → 두 번째 429 → 다음 풀 키로 회전 → 모든 키 소진 → fallback_model(다른 프로바이더). 402 청구 에러? → 즉시 다음 풀 키로 회전(24시간 쿨다운). 401 인증 만료? → 토큰 새로고침 시도(OAuth) → 새로고침 실패 → 다음 풀 키로 회전. 성공 → 정상 계속.

빠른 시작

`hermes auth add openrouter --api-key sk-or-v1-...`(API 키 추가), `hermes auth add anthropic --type api-key --api-key sk-ant-...`, `hermes auth add anthropic --type oauth`(브라우저 OAuth 로그인). `hermes auth list`로 풀 확인. 출력에서 ←는 현재 선택된 자격증명을 표시합니다.

4가지 회전 전략

`fill_first`(기본, 첫 건강한 키를 소진까지 사용 후 다음으로), `round_robin`(키들을 균등하게 순환, 각 선택 후 회전), `least_used`(가장 낮은 요청 수의 키를 항상 선택), `random`(건강한 키 중 무작위 선택).

에러 복구

429 속도 제한 시 같은 키 한 번 재시도(일시적), 두 번째 연속 429에 다음 키로 회전, 1시간 쿨다운. 402 청구/할당량 시 즉시 다음 키로 회전, 24시간 쿨다운. 401 인증 만료 시 OAuth 토큰 새로고침 먼저 시도, 새로고침 실패 시에만 회전. `has_retried_429` 플래그는 모든 성공한 API 호출에서 리셋되므로 단일 일시적 429가 회전을 트리거하지 않습니다.

위임과 서브에이전트 공유

에이전트가 `delegate_task`를 통해 서브에이전트를 생성할 때 부모의 자격증명 풀이 자동으로 자식과 공유됩니다. 같은 프로바이더면 자식이 부모의 전체 풀을 받아 속도 제한 시 키 회전이 가능하고, 다른 프로바이더면 자식이 그 프로바이더의 자체 풀을 로드하고(구성된 경우), 풀이 구성되지 않은 경우 자식은 상속된 단일 API 키로 폴백합니다. 작업당 자격증명 임대로 자식들이 동시에 키를 회전할 때 서로 충돌하지 않도록 보장합니다.

자동 발견

Hermes는 시작 시 여러 소스에서 자격증명을 자동 발견하여 풀에 시드합니다. 환경변수(`OPENROUTER_API_KEY`, `ANTHROPIC_API_KEY`), OAuth 토큰(`auth.json`의 Codex 디바이스 코드, Nous 디바이스 코드), Claude Code 자격증명(`~/.claude/.credentials.json`), Hermes PKCE OAuth(`~/.hermes/auth.json`), 커스텀 엔드포인트 설정(`config.yaml`의 `model.api_key`). 자동 시드된 항목은 풀 로드 시마다 업데이트되며, 환경변수를 제거하면 풀 항목이 자동으로 정리됩니다. 수동 항목(`hermes auth add`로 추가)은 절대 자동 정리되지 않습니다.

> 출처: Credential Pools

---

11. 번들 스킬 카탈로그

Hermes는 설치 시 `~/.hermes/skills/`에 복사되는 큰 내장 스킬 라이브러리와 함께 출하됩니다. 약 90개의 번들 스킬이 다음 카테고리로 조직되어 있습니다.

apple

`apple-notes`(memo CLI를 통한 Apple Notes 관리), `apple-reminders`(remindctl을 통한 Apple Reminders), `findmy`(macOS의 FindMy.app을 통한 Apple 디바이스/AirTags 추적), `imessage`(macOS의 imsg CLI를 통한 iMessage/SMS 송수신).

autonomous-ai-agents

`claude-code`(Claude Code CLI에 코딩 위임), `codex`(OpenAI Codex CLI에 코딩 위임), `hermes-agent`(Hermes Agent 구성·확장·기여), `opencode`(OpenCode CLI에 코딩 위임).

creative

`architecture-diagram`(어두운 테마의 SVG 아키텍처/클라우드/인프라 다이어그램을 HTML로), `ascii-art`(pyfiglet, cowsay, boxes, image-to-ascii), `ascii-video`(비디오/오디오를 컬러 ASCII MP4/GIF로 변환), `baoyu-comic`(지식 만화 - 교육용·전기·튜토리얼), `baoyu-infographic`(인포그래픽 - 21개 레이아웃 × 21개 스타일), `claude-design`(일회성 HTML 아티팩트 디자인 - 랜딩, 덱, 프로토타입), `comfyui`(ComfyUI로 이미지·비디오·오디오 생성), `ideation`(창의적 제약을 통한 프로젝트 아이디어 생성), `design-md`(Google의 DESIGN.md 토큰 사양 파일 작성/검증/내보내기), `excalidraw`(손으로 그린 Excalidraw JSON 다이어그램), `humanizer`(텍스트를 인간화 - AI-isms 제거, 진짜 음성 추가), `manim-video`(Manim CE 애니메이션 - 3Blue1Brown 수학/알고리즘 비디오), `p5js`(p5.js 스케치 - 생성 아트, 셰이더, 인터랙티브, 3D), `pixel-art`(시대 팔레트가 있는 픽셀 아트 - NES, Game Boy, PICO-8), `popular-web-designs`(54개 실제 디자인 시스템 - Stripe, Linear, Vercel - HTML/CSS로), `pretext`(@chenglou/pretext로 창의적 브라우저 데모 - DOM 없는 텍스트 레이아웃), `sketch`(일회성 HTML 목업 - 비교용 2-3 디자인 변형), `songwriting-and-ai-music`(작곡 기술과 Suno AI 음악 프롬프트), `touchdesigner-mcp`(twozero MCP를 통한 실행 중인 TouchDesigner 인스턴스 제어, 36개 네이티브 도구).

data-science

`jupyter-live-kernel`(라이브 Jupyter 커널을 통한 반복 Python).

devops

`kanban-orchestrator`(분해 플레이북 + 전문가 명단 관례 + Kanban을 통한 작업 라우팅 오케스트레이터 프로파일을 위한 안티유혹 규칙), `kanban-worker`(Hermes Kanban 워커의 함정·예시·엣지 케이스), `webhook-subscriptions`(이벤트 주도 에이전트 실행).

dogfood

`dogfood`(웹 앱의 탐색적 QA - 버그·증거·보고서 찾기).

email

`himalaya`(Himalaya CLI - 터미널에서 IMAP/SMTP 이메일).

gaming

`minecraft-modpack-server`(모드된 Minecraft 서버 호스팅 - CurseForge, Modrinth), `pokemon-player`(헤드리스 에뮬레이터 + RAM 읽기를 통한 Pokemon 플레이).

github

`codebase-inspection`(pygount으로 코드베이스 검사 - LOC, 언어, 비율), `github-auth`(GitHub 인증 설정 - HTTPS 토큰, SSH 키, gh CLI 로그인), `github-code-review`(PR 리뷰 - 차이, gh 또는 REST를 통한 인라인 코멘트), `github-issues`(gh 또는 REST를 통한 GitHub 이슈 생성·트리아지·라벨·할당), `github-pr-workflow`(GitHub PR 라이프사이클 - 브랜치, 커밋, 열기, CI, 머지), `github-repo-management`(저장소 클론/생성/포크, 원격, 릴리스 관리).

mcp

`native-mcp`(MCP 클라이언트 - 서버 연결, 도구 등록 stdio/HTTP).

media

gif-search(curl + jq를 통한 Tenor에서 GIF 검색/다운로드), heartmula(HeartMuLa - 가사 + 태그에서 Suno 유사 노래 생성), songsee(CLI를 통한 오디오 스펙트로그램/특징 mel, chroma, MFCC), spotify(Spotify - 재생, 검색, 큐, 플레이리스트와 디바이스 관리), youtube-content(YouTube 전사를 요약·스레드·블로그로).

mlops

audiocraft-audio-generation(AudioCraft - MusicGen 텍스트 투 음악, AudioGen 텍스트 투 사운드), axolotl(Axolotl - YAML LLM 파인튜닝 LoRA, DPO, GRPO), dspy(DSPy - 선언적 LM 프로그램, 자동 최적화 프롬프트, RAG), huggingface-hub(HuggingFace hf CLI - 모델·데이터셋 검색/다운로드/업로드), llama-cpp(llama.cpp 로컬 GGUF 추론 + HF Hub 모델 발견), evaluating-llms-harness(lm-eval-harness - LLM 벤치마크 MMLU, GSM8K 등), obliteratus(OBLITERATUS - LLM 거부 폐지 차이수단), outlines(Outlines - 구조화된 JSON/regex/Pydantic LLM 생성), segment-anything-model(SAM - 점·박스·마스크를 통한 제로샷 이미지 분할), fine-tuning-with-trl(TRL - SFT, DPO, PPO, GRPO, LLM RLHF용 보상 모델링), unsloth(Unsloth - 2-5배 빠른 LoRA/QLoRA 파인튜닝, 적은 VRAM), serving-llms-vllm(vLLM - 고처리량 LLM 서빙, OpenAI API, 양자화), weights-and-biases(W&B - ML 실험 로그, 스윕, 모델 레지스트리, 대시보드).

note-taking

obsidian(Obsidian 볼트의 노트 읽기·검색·생성).

productivity

airtable(curl을 통한 Airtable REST API - 레코드 CRUD, 필터, 업서트), google-workspace(gws CLI 또는 Python을 통한 Gmail, Calendar, Drive, Docs, Sheets), linear(Linear - GraphQL + curl을 통한 이슈·프로젝트·팀 관리), maps(OpenStreetMap/OSRM을 통한 지오코드, POI, 경로, 시간대), nano-pdf(nano-pdf CLI를 통한 PDF 텍스트/오타/제목 편집 - NL 프롬프트), notion(curl을 통한 Notion API - 페이지, 데이터베이스, 블록, 검색), ocr-and-documents(PDF/스캔에서 텍스트 추출 - pymupdf, marker-pdf), powerpoint(.pptx 덱·슬라이드·노트·템플릿 생성·읽기·편집).

red-teaming

godmode(LLM 탈옥 - Parseltongue, GODMODE, ULTRAPLINIAN).

research

arxiv(키워드, 저자, 카테고리 또는 ID로 arXiv 논문 검색), blogwatcher(blogwatcher-cli 도구를 통한 블로그와 RSS/Atom 피드 모니터링), llm-wiki(Karpathy의 LLM Wiki - 상호 연결된 마크다운 KB 빌드/쿼리), polymarket(Polymarket 쿼리 - 마켓, 가격, 오더북, 이력), research-paper-writing(NeurIPS/ICML/ICLR용 ML 논문 작성 - 디자인→제출).

smart-home

openhue(OpenHue CLI를 통한 Philips Hue 조명·장면·룸 제어).

social-media

xurl(xurl CLI를 통한 X/Twitter - 게시, 검색, DM, 미디어, v2 API).

software-development

`debugging-hermes-tui-commands`(Hermes TUI 슬래시 명령 디버그 - Python, 게이트웨이, Ink UI), `hermes-agent-skill-authoring`(저장소 내 SKILL.md 작성 - frontmatter, 검증기, 구조), `node-inspect-debugger`(--inspect + Chrome DevTools Protocol CLI를 통한 Node.js 디버그), `plan`(계획 모드 - .hermes/plans/에 마크다운 계획 작성, 실행 안 함), `python-debugpy`(Python 디버그 - pdb REPL + debugpy 원격 DAP), `requesting-code-review`(사전 커밋 리뷰 - 보안 스캔, 품질 게이트, 자동 수정), `spike`(빌드 전 아이디어 검증을 위한 일회성 실험), `subagent-driven-development`(delegate_task 서브에이전트를 통한 계획 실행 - 2단계 리뷰), `systematic-debugging`(4단계 근본 원인 디버깅 - 수정 전 버그 이해), `test-driven-development`(TDD - RED-GREEN-REFACTOR 강제, 코드 전 테스트), `writing-plans`(구현 계획 작성 - 작은 작업, 경로, 코드).

yuanbao

`yuanbao`(Yuanbao 그룹 - @멘션 사용자, 정보/멤버 쿼리).

이 외에도 약 60개의 추가 옵션 스킬이 Optional Skills Catalog에서 사용 가능하며 `hermes skills install official/...`로 설치할 수 있습니다.

> 출처: Bundled Skills Catalog

---

12. 버전별 주요 변경사항

다음은 GitHub Release Notes에 기재된 v0.9.0(2026-04-13)부터 v0.12.0(2026-04-30)까지의 주요 변경사항을 시간순으로 정리한 것입니다.

v0.12.0 — "Curator Release" (2026년 4월 30일)

이 릴리스의 가장 주목할 만한 추가는 **자율 큐레이터(Autonomous Curator)**입니다. 큐레이터는 에이전트가 만든 스킬을 백그라운드에서 자동으로 정리하여 시간이 지남에 따라 스킬 카탈로그가 좁은 거의 중복 항목들로 오염되는 것을 방지합니다(7일 주기 기본).

자기개선 루프 업그레이드와 함께 ComfyUI(이미지·비디오·오디오 생성)와 TouchDesigner-MCP(36개 네이티브 도구로 TouchDesigner 제어) 스킬이 번들로 추가되었습니다.

4개의 새 프로바이더가 추가되었습니다. **GMI Cloud**, **Azure AI Foundry**, **MiniMax OAuth**, **Tencent Tokenhub**입니다. **LM Studio**가 일급 시민이 되어 로컬 모델 사용이 더 매끄러워졌습니다.

메시징 플랫폼 측에서는 두 개의 큰 추가가 있었습니다. **Microsoft Teams**가 19번째 메시징 플랫폼으로 플러그인을 통해 추가되었고, **Yuanbao**(중국 텐센트의 메시징 앱)가 18번째 플랫폼으로 추가되었습니다.

Spotify 네이티브 통합과 함께 음성 명령으로 음악을 제어하는 번들 스킬이 추가되었으며, **Google Meet** 플러그인으로 Meet 통화 자동 전사가 가능해졌습니다.

CLI 측에서는 `hermes -z` 일회성 모드가 추가되어 스크립트나 일회성 작업을 실행하기 쉬워졌고, 웹 대시보드에 **Models** 탭이 추가되어 모델 카탈로그를 직접 탐색할 수 있게 되었습니다. 원격 모델 카탈로그 매니페스트로 새 모델이 자동으로 발견되며, 네이티브 멀티모달 이미지 라우팅이 지원되어 비전 가능 모델로 자동 전환됩니다.

TUI에는 **LaTeX 렌더링**이 추가되어 수학 콘텐츠가 제대로 표시되며, TUI 콜드 스타트가 약 57% 단축되었습니다. **Piper** 로컬 TTS가 새 옵션으로 추가되었고, **Vercel Sandbox** 백엔드가 클라우드 마이크로VM 실행을 지원합니다(스냅샷 기반 파일시스템 영속성).

보안 측면에서 시크릿 리덱션이 기본 비활성화로 변경되었고(작업 흐름 방해 감소), Kanban이 #16098에서 되돌려져 디자인이 재작업 중입니다.

> 출처: v0.12.0 Release Notes

v0.11.0 (2026년 4월 23일)

이 릴리스의 가장 큰 변화는 **Ink 기반 TUI 재작성**입니다. React 기반 Ink 프레임워크로 TUI를 완전히 재작성하여 더 매끄러운 렌더링·더 나은 성능·정교한 UI 컴포넌트(다이얼로그·메뉴·진행바)를 가능하게 했습니다.

Transport ABC + AWS Bedrock: 추상 베이스 클래스 패턴이 도입되어 새 LLM 프로바이더 추가가 표준화되었으며, AWS Bedrock이 첫 번째 ABC 구현으로 추가되었습니다.

`/steer` **중간 실행 너지**: 에이전트가 실행 중일 때 인터럽트 없이 다음 도구 호출 후 주입될 메시지를 보낼 수 있게 되었습니다. busy_input_mode의 새 옵션이기도 합니다.

셸 훅(Shell Hooks): Python 플러그인 작성 없이 YAML 설정만으로 셸 스크립트를 라이프사이클 이벤트에 연결할 수 있게 되었습니다.

웹훅 직접 전송 모드: `deliver_only: true`로 LLM을 호출하지 않고 메시지를 직접 전송하여 0 LLM 토큰·1초 미만 전송이 가능해졌습니다.

스마트 위임: 오케스트레이터 역할이 도입되어 다단계 워크플로(연구 → 합성, 또는 하위 문제별 병렬 오케스트레이션)가 가능해졌습니다.

보조 모델 설정 UI: 비전·압축·웹 추출 등 부가 작업의 모델을 대시보드에서 시각적으로 설정할 수 있게 되었습니다.

> 출처: v0.11.0 Release Tag

v0.10.0 — "Tool Gateway Release" (2026년 4월 16일)

이 릴리스의 핵심은 **Nous Tool Gateway**입니다. 유료 Nous Portal 구독자가 Firecrawl(웹 검색·추출)·FAL(이미지 생성)·OpenAI TTS·Browser Use(브라우저 자동화) 네 가지 핵심 도구를 별도 API 키 없이 Nous 구독 하나로 사용할 수 있게 되었습니다.

> 출처: v0.10.0 Release Tag

v0.9.0 — "Everywhere Release" (2026년 4월 13일)

이 릴리스는 Hermes Agent를 진정으로 어디서나 사용할 수 있게 만들었습니다.

Termux/Android 지원: Termux + Termux API를 통해 Android 폰에서 Hermes를 실행하여 SMS·센서·온디바이스 소셜 게시 능력을 잠금 해제했습니다. Greg Isenberg와 Imran Muthuvappa의 Startup Ideas 팟캐스트에서 언급된 것처럼, 저렴한 Android 폰만으로도 24시간 가동되는 개인 AI 에이전트를 운영할 수 있게 되었습니다.

iMessage(BlueBubbles를 통한)와 **WeChat** 메시징 플랫폼이 추가되었습니다.

Fast Mode (`/fast`): 빠른 응답이 필요할 때 추론을 건너뛰는 모드가 추가되었습니다.

백그라운드 프로세스 모니터링(`watch_patterns`): 백그라운드 프로세스의 출력을 패턴 매칭으로 모니터링하여 특정 출력이 나타나면 알리거나 행동할 수 있게 되었습니다.

웹 대시보드 출시: 브라우저 기반 관리 인터페이스의 첫 공개 릴리스입니다.

xAI와 **Xiaomi MiMo** 네이티브 프로바이더가 추가되었습니다.

플러그인 가능 컨텍스트 엔진: 내장 컨텍스트 압축기를 사용자가 자체 구현으로 대체할 수 있게 되어 RAG 통합·맞춤 압축 전략이 가능해졌습니다.

> 출처: v0.9.0 Release Notes

---

13. 참고 자료

공식 문서

Hermes Agent 공식 문서 - 전체 문서 진입점입니다. 본 가이드의 모든 기능 설명은 이 사이트의 공식 페이지를 출처로 합니다.

GitHub 저장소 - 소스 코드, 이슈, 커뮤니티 토론, 릴리스 노트 모두 여기에 있습니다. MIT License로 공개되어 있어 누구나 자유롭게 사용·수정·배포할 수 있습니다.

릴리스 노트 직접 링크

v0.12.0 Release (Curator), v0.11.0 Release Tag, v0.10.0 Release Tag (Tool Gateway), v0.9.0 Release (Everywhere).

커뮤니티 채널

Nous Research Discord - 실시간 도움과 토론, GitHub Discussions - 질문과 아이디어 공유, Skills Hub (agentskills.io) - 커뮤니티 스킬 검색과 공유.

카테고리별 문서 페이지

시작하기: Quickstart, Installation, Updating & Uninstalling, Learning Path, Nix & NixOS Setup.

Hermes 사용: CLI Interface, TUI, Configuration, Sessions, Profiles, Security, Checkpoints & Rollback, Git Worktrees, Docker.

가이드 및 튜토리얼: Tips & Best Practices, Tutorial: Daily Briefing Bot, Tutorial: Team Telegram Assistant, Using Hermes as a Python Library, Use MCP with Hermes, Use SOUL.md with Hermes, Use Voice Mode with Hermes, Build a Hermes Plugin, GitHub PR Review Agent.

개발자 가이드: Architecture, Contributing, Adding Tools, Creating Skills, ACP Internals, Provider Runtime Resolution, Tools Runtime, Memory Provider Plugin.

참조(Reference): CLI Commands, Slash Commands, Profile Commands, Environment Variables, Built-in Tools Reference, Toolsets Reference, MCP Config Reference, Model Catalog, Bundled Skills Catalog, Optional Skills Catalog, FAQ & Troubleshooting.

마치며

Hermes Agent는 단순한 AI 도구가 아니라, **시간이 지날수록 사용자에게 맞춰 성장하는 인프라**입니다. 처음 설치했을 때는 평범한 LLM 어시스턴트처럼 보일 수 있지만, 며칠·몇 주를 사용하면서 에이전트는 사용자의 코드베이스 구조·선호하는 워크플로·자주 사용하는 명령어를 학습하고, 5번 이상의 도구 호출이 필요한 복잡한 작업은 자동으로 재사용 가능한 스킬로 변환되어 다음에는 한 번에 처리됩니다.

본 가이드에서 살펴본 것처럼 Hermes Agent의 진정한 가치는 단일 기능에 있지 않습니다. 19개 메시징 플랫폼 통합·8개 외부 메모리 프로바이더·14개 내장 인격·약 90개의 번들 스킬·25개 이상의 LLM 프로바이더·7개 터미널 백엔드·3가지 훅 시스템·자기개선 큐레이터·통합 RL 학습 파이프라인 - 이 모든 것이 하나의 일관된 시스템으로 작동하면서 사용자의 컴퓨팅 환경 위에 자율적인 AI Assistant를 구축합니다.

User Stories에서 "한 번에 12개의 Hermes 인스턴스를 병렬로 실행하여 Hermes Agent를 만들고 있다. 이제 GitHub 저장소 상위 100위 안에 들었다"고 말한 Teknium처럼, 또는 "WhatsApp에서 가족 3명이 한 번 설정으로 모두 다른 사용 사례에 활용한다. ChatGPT 구독 한 개($200)면 충분하다. 사전 능동적 행동이 있는 메시징 봇이 새로운 세계를 열어주었다"고 말하는 것처럼, Hermes Agent의 사용 패턴은 사용자마다 다릅니다.