



Getting Started with HT

THE HAUTE OPERATING SYSTEM CLI

v0.3.6 – HAUTE Platform 2026



Table of Contents

Your journey through the HAUTE ecosystem

01	What is HT?	The unified enterprise CLI
02	Installation	Pre-built binaries or build from source
03	Authentication	Microsoft 365, JIRA, ServiceNow, Neo4j
04	Email, Calendar & Chat	Microsoft 365 productivity suite
05	Unified Message Archive	Local search, transcripts, digests
06	Task Management	Beads, JIRA, and ServiceNow
07	Semantic Memory	Neo4j knowledge graph (L2)
08	RIVER Workflows	State machine project management
09	MCP Server	Web search & docs for AI assistants
10	Trust Tiers	T0-T4 authorization levels
11	Configuration	config.yaml deep dive
12	Quick Reference	Command cheat sheet

Prerequisites

Rust 1.85+ (edition 2024) • macOS 12+ / Linux x64 / Windows 10+ • Microsoft 365 account • Optional: Neo4j 5.x, Chrome/Chromium, Atlassian instance



What is HT?

One CLI to rule them all

HT is the HAUTE Operating System CLI — a monolithic Rust binary that provides a unified command-line interface for the entire HAUTE platform. Think of it as your single entry point to email, calendar, chat, tasks, knowledge bases, semantic memory, and workflows across Microsoft 365, Atlassian, ServiceNow, and HAUTE-native systems.

Why HT?

Unified Access

One binary, 31 command groups, 215 MCP tools. No more switching between portals.

AI-Native

Built-in MCP server exposes web search and library docs (Brave, Exa, Tavily, Context7) to Claude Code, Cursor, and AI agents.

Trust-Gated

T0-T4 trust tiers ensure safe automation with ceremony-based promotion.

Architecture at a Glance

HT is organized as a Rust workspace with specialized crates:

Core Crates

- `ht-core` — Config, errors, output formatting
- `ht-auth` — OAuth, device code flow
- `ht-graph` — Microsoft Graph API client
- `ht-db` — Shared SQLite utilities

Feature Crates

- `ht-email`, `ht-chat` — M365 comms
- `ht-uma` — Unified Message Archive
- `ht-search` — Web search & AI (7 providers)
- `ht-jira`, `ht-confluence` — Atlassian
- `ht-neo4j` — Semantic memory
- `ht-river`, `ht-health` — Workflows & ops
- `ht-vault` — Obsidian vault (L3)
- `ht-spec` — Spec validation & scoring
- `ht-analyze` — Embedding analysis

Supported Integrations

Platform	Capabilities	Auth Method
Microsoft 365	Email, Calendar, Teams, SharePoint, To Do, Contacts, Org	Device Code / PKCE
Atlassian	JIRA (issues, sprints, boards, create, custom fields, change history), Confluence (pages, attachments, comments, search)	API Token
ServiceNow	Incidents, Changes, Problems, RITM, SCTASK, Releases, Requests, CMDB, KB articles, attachments, journal entries, approvals	OAuth PKCE
Neo4j	Knowledge graph, semantic search, RAG	Bolt (user/pass)
OpenAI	GPT-4.1 chat, o4-mini reasoning, Images 2.0, Whisper transcription	API Key
Chrome/CDP	Browser automation, screenshots, DOM interaction	Local (no auth)



Option A: Build from Source

```
# Clone and build (working trunk lives on the ht_MCP fork)
git clone https://github.com/vituity-eng/ht_MCP.git
cd ht_MCP
cargo install --path crates/cli

# Verify
ht version
ht --help
```

Option B: Pre-built Binary (GitHub Releases)

Download the latest release from github.com/vituity-eng/ht_MCP/releases . The macOS ARM binary is signed by Developer ID Application: DONATO JOSE CABAL (8HHPJW7TUF) and notarized by Apple.

Platform	Archive
macOS (Apple Silicon, signed + notarized)	ht-aarch64-apple-darwin.tar.gz
macOS (Intel)	ht-x86_64-apple-darwin.tar.gz
Linux (ARM64, musl)	ht-aarch64-unknown-linux-musl.tar.gz
Linux (x64, musl)	ht-x86_64-unknown-linux-musl.tar.gz
Windows (x64, mingw)	ht-x86_64-pc-windows-gnu.zip

```
# macOS/Linux: extract and move to PATH
tar xzf ht-aarch64-apple-darwin.tar.gz
mv ht ~/.local/bin/

# Or use gh CLI
gh release download v0.3.6 --repo vituity-eng/ht_MCP --pattern "*darwin*"
```

What Gets Created

Path	Purpose
~/.haute/config.yaml	Main configuration file
~/.haute/tokens/	OAuth token cache (auto-refreshed)
~/.haute/daemon.db	Daemon scheduler database
~/.haute/cbac_events.db	Command timing & analytics
~/.beads/	Beads issue tracker (optional)

First Run

```
# Initialize config (auto-detects MCP creds + Obsidian vault)
ht config init

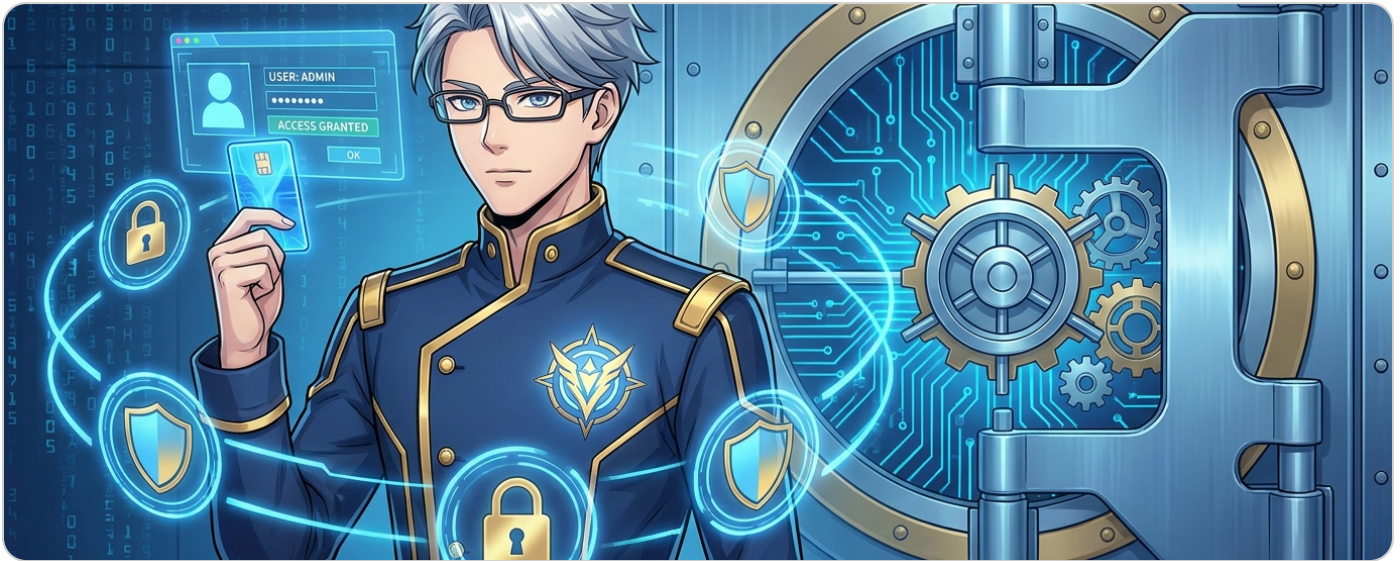
# Authenticate to Microsoft 365
ht auth login

# Run diagnostics (checks Graph, JIRA, SNOW, Neo4j, vault)
ht doctor

# See all available commands
ht --help
```

System Requirements

Rust 1.85+ (edition 2024) • macOS 12+, Linux x64, or Windows 10+ • ~50MB disk space for the binary



Most HT commands require authentication. HT supports multiple auth providers, each with its own flow. Tokens are cached locally and auto-refresh transparently. **Vituity defaults are embedded** — just run the login command.

Microsoft 365 (Device Code Flow)

```
# Step 1: Initiate login (Vituity tenant/client are pre-configured)
ht auth login

# CLI outputs a code and URL:
#   Enter code ABC123 at https://microsoft.com/devicelogin

# Step 2: Open URL in browser, enter the code
# Step 3: CLI auto-detects success and caches token

# Verify status
ht auth status
```

Non-Vituity Tenants

Override with `auth.tenant_id` and `auth.client_id` in `config.yaml`, or set `HT_AUTH_TENANT_ID` / `HT_AUTH_CLIENT_ID` env vars. Values come from your Azure AD App Registration.

All Auth Providers

Provider	Login Command	Flow
Microsoft Graph	<code>ht auth login</code>	Device Code (browser)
ServiceNow	<code>ht auth login --provider servicenow</code>	OAuth PKCE (browser, no client_secret needed)
JIRA / Confluence	API token in config.yaml	Basic Auth (token)
Neo4j	Credentials in config.yaml	Bolt protocol

Token Management

```
# Check all auth status
ht auth status

# Logout from specific provider
ht auth logout --provider graph

# Logout from everything
ht auth logout
```

Security Note

Tokens are stored in `~/.haute/tokens/` with filesystem permissions. Never commit this directory. The `.gitignore` already excludes it.

Troubleshooting Auth

Run `ht doctor` to check token freshness and connectivity to Graph, JIRA, Confluence, ServiceNow, Neo4j, and your Obsidian vault. If tokens expire, HT will auto-refresh. If refresh fails, simply re-run `ht auth login`.

04

Email, Calendar & Chat

Your Microsoft 365 productivity suite, from the terminal



Email

```
# List inbox
ht email list --unread

# Search emails
ht email search "budget Q4"

# Draft and send
ht email draft \
  --subject "Q4 Report" \
  --body "Attached." \
  --to alice@corp.com
ht email send DRAFT_ID

# Reply to a thread
ht email reply \
  --email_id MSG_ID \
  --body "Looks good!" \
  --reply_all

# Forward with attachment
ht email forward \
  --message_id MSG_ID \
  --to bob@corp.com
```

Calendar

```
# Today's schedule
ht cal today

# This week
ht cal range \
  --from 2026-04-19 \
  --to 2026-04-26

# Create a meeting
ht cal create \
  --title "Sprint Review" \
  --start 2026-04-25T14:00 \
  --end 2026-04-25T15:00

# Find available slots
ht cal available \
  --start 2026-04-25 \
  --end 2026-04-26 \
  --emails alice@corp.com

# Respond to invite
ht cal respond EVT_ID \
  --action accept
```

Teams Chat

```
# Send a direct message
ht chat send --message "Sprint starts Monday" --recipient alice@corp.com

# Send to group chat with mention and image
ht chat send --message "Check this out" --chat_id CHAT_ID \
  --image /path/to/screenshot.png --mention bob@corp.com

# Check presence
ht chat presence --user alice@corp.com
```



Unified Message Archive

Local search across email, chat, and meeting transcripts

UMA syncs your email, Teams chat messages, and meeting transcripts into a local SQLite database with FTS5 full-text search. Everything stays on your machine — search is instant, works offline, and includes BM25 ranking.

Syncing Data

EMAIL

Delta sync via Graph API. Tracks read state, importance, conversations.

```
ht uma sync
ht uma sync --folder sentitems
```

TEAMS

Chat messages, members, inline transcripts from chat events.

```
ht uma sync-teams
ht uma sync-teams \
  --max-chats 200
```

TRANSCRIPTS

Walks calendar events to find all meetings with transcripts.

```
ht uma sync-transcripts
# First run: 13 months
# After: incremental
```

Searching

```
# Full-text search across email + chat + transcripts (BM25 ranked)
ht uma search "budget approval" --from alice@corp.com --after 2026-01-01

# Search only transcripts
ht uma search "deployment timeline" --limit 5
```

Transcript Workflow

```
# Sync transcripts from calendar (discovers meetings the chat sync misses)
ht uma sync-transcripts

# AI-summarize unsummarized transcripts
ht uma summarize-transcripts --limit 20 --model haiku

# Backfill a specific meeting transcript
ht uma fetch-transcript --topic "Sprint Review"
```

Incremental Sync

Email uses Graph delta queries — only new/changed messages are fetched. Transcript sync saves a cursor and only scans calendar events since the last run (with a 3-day overlap buffer). Both are safe to run frequently via `ht daemon` .

HT provides a unified `ht tasks` command that works across three providers: **Beads** (HAUTE-native, default), **JIRA**, and **ServiceNow**. Select the provider with a flag.

Provider Selection

DEFAULT Beads

HAUTE-native issue tracker.
Local-first, git-synced.

```
ht tasks list
ht tasks show beads-001
ht tasks mine
```

FLAG JIRA

Full JQL, sprints, create, transitions,
comments, custom fields.

```
ht tasks --jira list
ht tasks --jira sprint
ht tasks --jira create "Fix bug"
```

FLAG ServiceNow

Incidents, changes, problems, KB
articles.

```
ht tasks --snow list
ht tasks --snow show INCO
ht tasks --snow mine
```

Write Operations (Require Trust Tier)

```
# Create a JIRA ticket (requires T2)
ht tasks --jira create "Fix login timeout" \
  --project PROD \
  --task-type Bug \
  --priority P2 \
  --custom-fields '{"customfield_13024":{"id":"10996"}}'

# Add a comment
ht tasks --jira --tier 2 comment PROD-1234 "Root cause identified"

# Transition a ticket (requires T3)
ht tasks --jira --tier 3 transition PROD-1234 31
```

Knowledge Base

```
# Search Confluence (full-text content search via CQL)
ht knowledge search "deployment runbook" --confluence

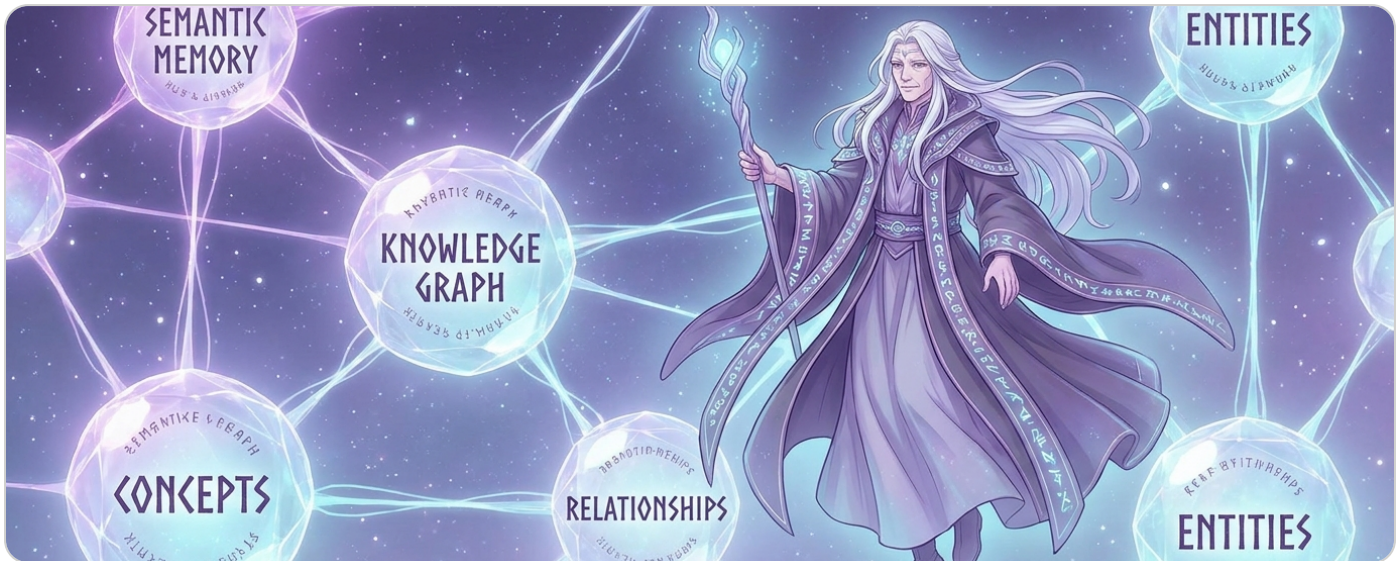
# Read an article
ht knowledge read ARTICLE_ID --confluence

# Search SharePoint
ht knowledge search "security policy" --sp
```

07

Semantic Memory

Your persistent knowledge graph (L2)



HT's semantic memory layer uses **Neo4j** as a knowledge graph, storing entities, observations, and relationships. It supports full-text search with temporal decay, graph traversal, and RAG (retrieval-augmented generation).

Finding Knowledge

```
# Search by name
ht mem find "OAuth 2.0"

# Full-text search
ht mem search "authentication" \
  --label Concept \
  --limit 5

# RAG query (vector + graph)
ht mem rag "auth flow"

# Explore neighbors
ht mem neighbors "Alice" \
  --depth 2
```

Storing Knowledge

```
# Create entity
ht mem create \
  --name "OAuth 2.0" \
  --label Concept

# Add observation
ht mem observe "OAuth 2.0" \
  "Device code flow for M365" \
  --upsert \
  --cite source=docs

# Create relationship
ht mem relate \
  "Alice" MANAGES "Project X"
```

Advanced: Raw Cypher

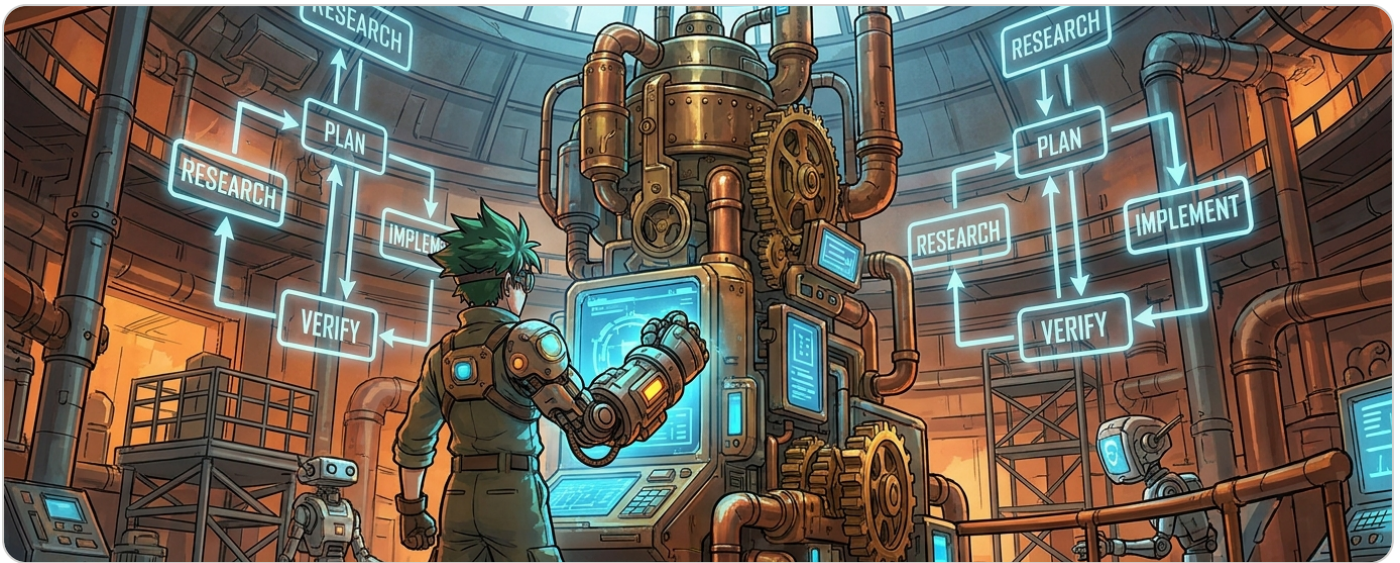
```
# Read-only query
ht mem cypher "MATCH (p:Person)-[:MANAGES]→(prj) RETURN p.name, prj.name LIMIT 10"

# Write query (requires --write flag)
ht mem cypher "MERGE (n:Concept {name: 'RBAC'})" --write

# Graph statistics
ht mem graph --stats
```

Temporal Decay

Search results are scored with temporal decay — recent observations rank higher. Use `--no_decay` to disable, or `--since 7d` to filter by time window.



RIVER 2 is HT's unified workflow engine. It models projects as state machines that flow through well-defined stages, with configurable tier, blast radius, and intent.

State Flow



Creating Work

```

# Create an epic
ht river create \
  --title "Build auth system" \
  --tier T3 \
  --blast_radius for_company \
  --intent MAKE
# Returns: R-0001

# Create child PBI
ht river create \
  --title "Design schema" \
  --parent 0001 \
  --tier T2 \
  --intent LEARN
  
```

Moving Forward

```

# Check status
ht river status 0001

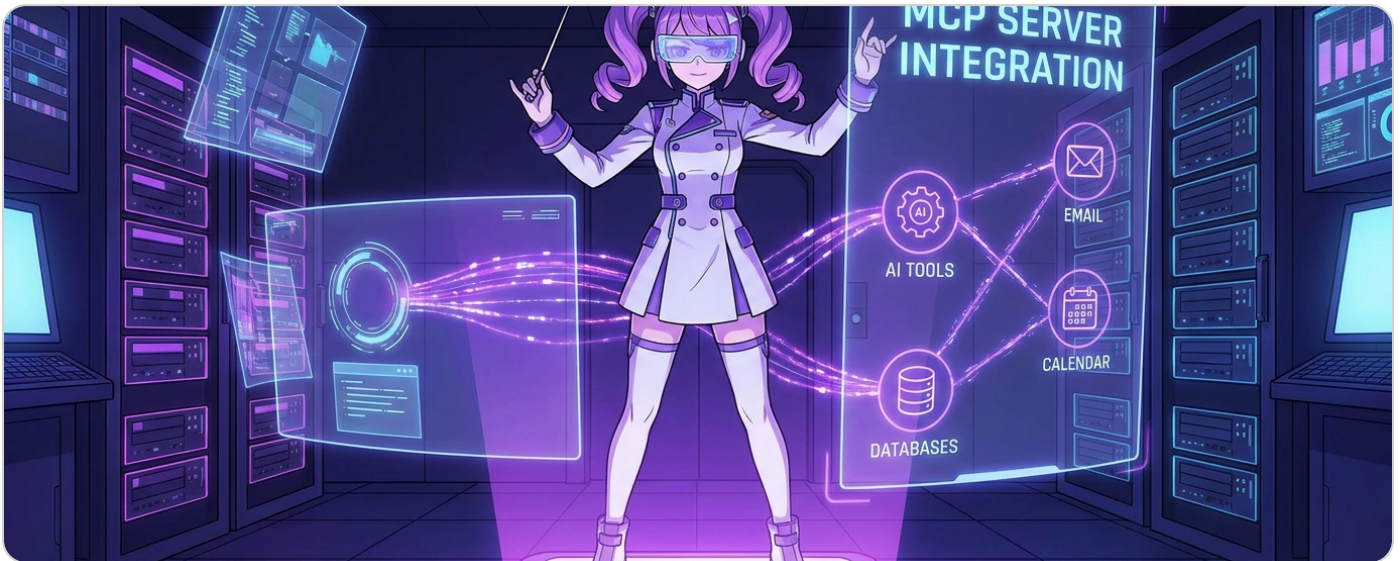
# Advance state
ht river advance 0001 \
  --reason "Review passed"

# Ship it! (fast-forward)
ht river ship 0001

# Wiggum (loop back)
ht river wiggum 0001 \
  --reason "Need more research"
  
```

Dimensions

Dimension	Values	Purpose
Tier	T1–T5	Rigor level (T1=yolo, T5=full ceremony)
Blast Radius	for_me, for_team, for_company	Impact scope
Intent	MAKE, LEARN, FIX	Work type classification
Dispatch	awl, tmux, manual	How work is dispatched
Complexity	C0–C3	Documentation & investment level



HT includes an embedded **Model Context Protocol** server that exposes **215 tools over stdio** — the entire enterprise surface (Microsoft 365, JIRA, Confluence, ServiceNow, SharePoint, Teams), plus AI search providers, semantic memory, UMA, browser automation, and more. AI clients (Claude Code, Cursor, etc.) call them directly via MCP — no shell-out required.

Setup

Add to your `.mcp.json` or client configuration:

```
{
  "mcpServers": {
    "ht": {
      "command": "ht",
      "args": ["mcp", "serve"]
    }
  }
}
```

Tool Surfaces (215 total)

MICROSOFT 365

Outlook (email + calendar), Teams chat & channels, SharePoint, OneNote, To Do, Contacts. Verbose-mode flags surface IDs, attendees, attachments, recurrence, and reactions.

ATLASSIAN

JIRA: `jira_get_issue`, sprint/board tools, custom fields, change history, attachments, JQL search. Confluence: pages, attachments, search, page children.

SERVICENOW

36+ `servicenow_*` tools: incidents, changes, problems, RITM variables, SCTASK, releases, approvals, journal entries, attachments, CMDB, KB.

SEARCH & AI

Brave web/local, Exa neural search + contents, Tavily search + extract, Context7 library docs, plus Perplexity, Gemini, OpenAI providers.

MEMORY & KNOWLEDGE

Neo4j semantic memory (find, search, observe, relate, RAG), UMA full-text search across email + Teams + transcripts + JIRA + SNOW.

AUTOMATION

Browser automation via CDP (screenshots, click, type, eval), RIVER workflow engine, daemon scheduler, vault read/write, health and trust controls.

How It Works

When you run `ht mcp serve`, HT starts a stdio-based MCP server. Your AI client (Claude Code, Cursor, etc.) communicates with it via the MCP protocol, gaining direct access to all 215 tools. Run `ht mcp list` to see the registered tool catalog. Tool descriptions include parameter types, defaults, and verbose-mode flags so clients can discover capabilities at runtime.

Example: AI-Driven Triage

```
// Claude Code can now chain enterprise tools natively:
// "Show me INC1029547 and any related JIRA tickets"
// → calls servicenow_get_record_history("INC1029547")
// → reads jira_link custom field, calls jira_get_issue(...)
// → calls jira_get_change_history(...) for the audit trail
// → returns a unified timeline across both systems
```



Trust Tiers

T0–T4 authorization for safe automation



The trust tier system controls what operations are allowed, from read-only browsing to fully autonomous workflows. Higher tiers require ceremony-based promotion.

Tier	Name	Capabilities
T0	Read-Only Chatbot	SharePoint read, calendar read, ticket read, knowledge search
T1	Read + PII	T0 + email read (full), contacts, Teams read (advanced)
T2	Write Reversible	T1 + email draft, ticket create, calendar create, Teams send
T3	Write Irreversible	T2 + email send, ticket close, calendar cancel, approvals
T4	Autonomous	T3 + autonomous workflows, plugin install, elevated ops

Promotion Ceremony

```
# Request tier promotion
ht trust promote 2

# Admin lists pending requests
ht trust list

# Admin approves (HMAC-SHA256 ceremony)
ht trust approve REQUEST_UUID

# Check your current tier
ht trust status
```

Using Tiers in Commands

```
# Default is T0 (read-only) – no flag needed
ht tasks list

# Write operations require explicit tier
ht tasks --jira --tier 2 create --summary "Fix bug" --project PROD

# Irreversible operations need T3
ht tasks --jira --tier 3 transition PROD-1234 31
```

Insufficient Tier Error

If you see `insufficient trust tier for operation 'email:send'`, you need to promote with `ht trust promote 3` and have an admin approve the request.



Configuration

The ~/.haute/config.yaml deep dive

All configuration lives in `~/.haute/config.yaml`. Environment variables override config file values. Run `ht config init` to create a starter config.

Minimal Config

```
# ~/.haute/config.yaml (Vituity Graph defaults are pre-filled)
auth:
  tenant_id: "56b24b68-..." # Vituity default
  client_id: "ff4acc71-..." # Vituity default

neo4j:
  uri: bolt://localhost:7687
  username: neo4j
  password: "your-password"

jira:
  instance: your-org.atlassian.net
  email: you@example.com
  api_token: "your-api-token"
```

Smart Init

`ht config init` auto-detects existing credentials from MCP configs (`~/.claude.json`, `~/.cursor/mcp.json`) and Obsidian vault locations. Detected values are pre-filled automatically.

Environment Variable Overrides

Microsoft 365

`HT_AUTH_CLIENT_ID`
`HT_AUTH_TENANT_ID`
`AZURE_CLIENT_ID` (alt)

Neo4j

`HT_NEO4J_URI`
`HT_NEO4J_USERNAME`
`HT_NEO4J_PASSWORD`

Search & AI APIs

`BRAVE_API_KEY`
`EXA_API_KEY`
`TAVILY_API_KEY`
`OPENAI_API_KEY`

ServiceNow

`SNOW_CLIENT_ID`
`SNOW_INSTANCE`

Config Commands

```
ht config init      # Create default config
ht config show      # Display current config
ht config set KEY VALUE  # Set a value
ht config check      # Validate configuration
ht config resolve PROVIDER # Show resolved values
```

Config Hygiene

All config access goes through `ConfigProvider` + `ConfigValues`. Direct `std::env::var` calls are prohibited and caught by CI. New fields must declare name, env_var, required, secret, default, and description.



Quick Reference

The essential command cheat sheet

Getting Started

```
ht --help
ht config init
ht auth login
ht auth status
ht doctor
```

Email

```
ht email list --unread
ht email search "query"
ht email draft --subject --body --to
ht email send DRAFT_ID
ht email reply --email_id --body
```

Calendar

```
ht cal today
ht cal range --from --to
ht cal create --title --start --end
ht cal available --start --end --emails
ht cal respond EVT_ID --action
```

Teams Chat

```
ht chat send --message --recipient
ht chat messages --recipient
ht chat update MSG_ID --message
ht chat delete MSG_ID
ht chat list
ht chat presence --user
```

Tasks

```
ht tasks list # Beads
ht tasks --jira list # JIRA
ht tasks --jira create "..." # new
ht tasks --snow list # SNOW
ht tasks mine
ht tasks show ID
```

UMA (Archive)

```
ht uma sync # email
ht uma sync-teams # chat
ht uma sync-transcripts # meetings
ht uma sync-jira # JIRA tickets
ht uma sync-snow # SNOW tickets
ht uma search "query"
ht uma stats
```

Memory (Neo4j)

```
ht mem find "entity"
ht mem search "query"
ht mem observe "entity" "fact"
ht mem neighbors "entity"
ht mem rag "question"
```

RIVER

```
ht river create --title --tier
ht river status ID
ht river advance ID
ht river ship ID
ht river list
```

Browser

```
ht browse navigate URL
ht browse screenshot out.png
ht browse click "selector"
ht browse type "selector" "text"
ht browse eval "js"
```

Search & AI

```
ht search web "query"           # Brave
ht search ask "query"           # Perplexity
ht search code "query"          # Exa
ht search oai "query"           # GPT-4.1
ht search oai-reason "query"     # o4-mini
ht search oai-image "prompt"     # Images 2.0
ht search transcribe audio.mp3  # Whisper
```

Backup & Restore

```
ht backup create --target vault
ht backup create --target neo4j
ht backup list
ht backup status
ht backup prune --keep 10
ht backup restore SNAPSHOT
ht backup enable           # schedule daily
ht backup disable
```

Health & Daemon

```
ht doctor
ht health status
ht health run CHECK_NAME
ht daemon run --port 3210
ht daemon schedules
```

Output Formats

```
# All commands support:
--format markdown # default
--format json     # full detail
--format oneline  # compact
```

You're ready to go!

Run `ht --help` for the full command list • `ht doctor` to verify your setup • Happy hacking!