UHB — Manuel d'utilisation

Version 0.1.0 · Universal Harness Backend
Ce manuel décrit l'installation, les commandes et les garde-fous de l'implémentation de référence. Il est embarqué dans le paquet et servi par `uhb run` sur `/docs/manuel.pdf`.

---

1. Ce qu'est UHB, en trois phrases

UHB est le **backend d'exécution** posé sous un plan de contrôle (Harness Maxxing, OpenClaw, Hermes Agent, ou votre CLI). Il ne raisonne pas à votre place : il décide **ce qu'un tour a le droit de faire** (modes), **quel modèle répond** (routeur) et **où les connaissances sont écrites** (wiki compilé), et il refuse par défaut ce qui n'est pas explicitement autorisé.

---

2. Installation

Prérequis : Python 3.11 ou plus. Une seule dépendance d'exécution, `httpx`.

2.1 Installateur

```
git clone https://github.com/Antoine-Thomas/UHB-Universal-Harness-Backend.git
cd UHB-Universal-Harness-Backend
./install.sh                # crée .venv, installe, vérifie uhb --version
./install.sh --with-yaml    # ajoute pyyaml : nécessaire à `uhb config set`
```

Options utiles :

Option	Effet
`--dry-run`	affiche les commandes, ne modifie rien
`--with-yaml`	installe pyyaml (écriture de `config.yaml`)
`--help`	rappelle ces options

L'installateur n'utilise jamais `sudo`, n'écrit jamais hors du dépôt, et relancé deux fois il ne casse rien.

2.2 À la main

```
python -m venv .venv && . .venv/bin/activate
pip install -e ".[yaml]"
uhb --version
```

2.3 Vérifier

```
uhb status
```

Doit afficher la version, le commit git, les fournisseurs avec `key_env` et l'état `loaded` / `MISSING`, la politique de routage et les modes disponibles.

---

3. Les commandes

3.1 Lecture seule

Commande	Rôle
`uhb status`	version, dépôt, fournisseurs, politique, modes
`uhb status --json`	le même instantané, en JSON (pour un script ou une UI)
`uhb modes list`	nom, type, palier, portée d'écriture, budget de tours
`uhb modes show <nom>`	le profil complet d'un mode
`uhb config list`	toutes les clés effectives, défauts compris
`uhb config get <clé>`	une valeur, chemin pointé (`routing.default`)
`uhb providers`	table des fournisseurs (protocole, URL, **nom** de variable)
`uhb health`	sonde de santé du backend
`uhb docs`	chemin du manuel PDF et URL de téléchargement
`uhb ui`	ce qu'il faut coller dans une interface, et où est le guide
`uhb brain status`	RAG, SiYuan, MARI : en ligne ou non, avec le détail

3.2 Actions

Commande	Rôle
`uhb run [--mode chat] [--port 8101]`	démarre le point d'entrée HTTP
`uhb execute "<prompt>" --mode dev`	un tour complet via le harness configuré
`uhb config set <clé> <valeur>`	écrit une clé de configuration
`uhb update [--dry-run]`	avance en fast-forward depuis le dépôt distant

`uhb run` sert trois surfaces :

- `POST /v1/chat/completions` — forme OpenAI (streaming accepté)

- `POST /v1/messages` — forme Anthropic
- `GET /` — panneau de contrôle UHB (état, accès rapide, manuel)
- `GET /status` — état en JSON
- `GET /docs/manuel.pdf` — ce manuel

---

4. Les modes

Un mode est un fichier JSON : portée de lecture, portée d'écriture, outils autorisés, prompt système, palier de modèle, budget de tours.

Mode	Lit	Écrit	Palier
`chat`	`wiki/**`, `raw/**`	rien	eco
`dev`	tout	`uhb/**`, `tests/**`, `docs/**`, fichiers mémoire	eco
`review`	tout	rien	eco
`observe`	`wiki/**`, `logs/**`, `data/**`, `*.log`	`wiki/log.md`, `reports/**`	free

Règles non négociables :

1. `deny` gagne toujours sur `allow`.
2. Une clé inconnue dans un profil est une erreur au chargement, pas un silence.
3. Un chemin absolu ou un `..` dans un motif est refusé au chargement.
4. Toute écriture passe par `check_write` **avant** d'ouvrir le fichier.
5. Un chemin d'écriture sans mode lève `ModeRequired` : il n'existe pas d'« enveloppe implicite ».

Ajouter un mode = déposer un fichier dans `uhb/hub/modes/profiles/`, ou dans le dossier pointé par `UHB_MODES_DIR` (qui remplace par nom et ajoute les nouveaux).

---

5. Le routeur

```
uhb status   # puis, pour la décision détaillée, en Python :
```

```
from uhb.hub.modes import get_mode
from uhb.hub.router import route
decision = route("Refactor le routeur et ajoute des tests", get_mode("dev"))
print(decision.reason)
```

Trois temps : **classifier** (longueur, blocs de code, verbes multi-étapes, chemins, questions multiples), **planifier** la chaîne (palier du mode, puis `eco` → `free` → `paid`, filtré par la complexité déclarée), **exécuter** (premier candidat qui répond).

Repli, dans l'ordre : un fournisseur distant sans variable de clé est **sauté** (pas appelé) ; un fournisseur qui lève est **noté puis sauté** (la classe d'exception est conservée, jamais le contenu) ; si tout échoue, `AllProvidersFailed` est levée avec les tentatives — jamais une chaîne vide qui ressemble à une réponse.

`paid` n'est atteignable que si le mode porte `allow_paid: true`. Aucun profil livré ne l'autorise : un modèle cher reste une décision manuelle.

---

6. Le wiki compilé

Trois couches : `raw/notes/` (sources immuables), `wiki/` (pages compilées), `schema.md` (les conventions). Trois opérations :

Opération	Ce qu'elle fait
`ingest`	une source → N pages, **un seul appel** au modèle, écriture atomique
`query`	lit l'index et renvoie des pages candidates ; ne répond jamais à votre place
`lint`	orphelins, liens morts, trous d'index, frontmatter invalide, contradictions

L'ingestion est idempotente : l'empreinte sha256 de la source est stockée dans le frontmatter de chaque page ; réingérer une source inchangée n'écrit rien.

---

7. Configuration

Fichier : `uhb/config.yaml` (surchargeable par la variable `UHB_CONFIG`). Absent, les défauts intégrés s'appliquent — c'est pourquoi `uhb config list` fonctionne sur un dépôt neuf.

```
uhb config get routing.default
uhb config set routing.default deepseek/deepseek-flash
uhb config set routing.retries 3
```

Deux garde-fous : seules les sections `providers` et `routing` sont modifiables, et toute clé dont le dernier segment ressemble à un secret (`api_key` , `token` , `password` , `secret`) est **refusée**. Une valeur de credential ne va jamais dans un fichier : la configuration porte le **nom** de la variable, la valeur reste dans l'environnement.

Variables reconnues :

Variable	Effet
`UHB_CONFIG`	chemin du fichier de configuration
`UHB_LLM_BASE_URL`	endpoint compatible OpenAI pour les appels directs (défaut : Ollama local)
`UHB_LLM_MODEL`	modèle de ces appels directs
`UHB_LLM_TOKEN`	jeton éventuel (nommé, jamais écrit)
`UHB_LLM_REASONING`	`none` par défaut : une extraction n'a pas besoin de délibérer
`UHB_MODES_DIR`	dossier de profils de modes supplémentaire
`UHB_ROUTER_CATALOG`	catalogue de routage alternatif
`UHB_HERMES_BIN`	binaire du harness (nom, chemin, ou préfixe argv)

---

8. Le second cerveau (Hermes Home Vision)

UHB sait lire l'infrastructure locale quand elle existe. Trois services, quatre appels, aucune dépendance nouvelle :

Service	Appel	Réponse mesurée
RAG (8200)	`GET /sante` · `POST /search {"question","k"}`	4 434 fragments, `intfloat/multilingual-e5-base` · passages classés
SiYuan (6806)	`POST /api/system/version` · `POST /api/export/exportMdContent {"id"}`	v3.8.6 · markdown d'un document
MARI (55553)	`GET /status`	`{"status":"ok","signalsEmitted":2}`

```
uhb brain status            # ● / ○ par service, avec le détail et la latence
uhb brain status --strict   # code 1 si un service est tombé (utile en cron)
uhb brain search "Comment fonctionne le Bridge MARI ?" -k 3
uhb brain read 20261109131511-donr63p   # exporte un document SiYuan en markdown
```

Règles de conduite :

- `brain status` ne lève jamais : un service éteint est un **état** (○), pas une panne de la commande.
- `brain search` et la lecture SiYuan **lèvent** (`SecondBrainError`) au lieu de renvoyer une liste vide — une liste vide se lirait « ton second cerveau n'a rien », ce qui est faux quand le service est simplement indisponible.
- Chaque appel a son timeout (2 s pour une sonde, 30 s pour une recherche) : un service figé ne fige pas l'appelant.

- SiYuan exige un jeton pour lire du contenu : variable `SIYUAN_TOKEN`, lue par son nom, envoyée en en-tête, jamais journalisée ni conservée sur l'objet.
- Les chemins renvoyés par le RAG sont **bruts** (ils viennent de son index) ; UHB les raccourcit en `~` pour l'affichage, mais ne les normalise pas. À traiter à la source le jour où un index serait publié.

Installer les services

Voie	Commande	Ce qui est récupéré
Docker	`docker compose up -d`	images `uhb`, `rag`, `b3log/siyuan:v3.8.10`
Script	`./install_services.sh` (ou `install_services.ps1`)	SiYuan, sha256 vérifié contre `SHA256SUMS.txt`
Manuel	`python -m uhb.services.install --with rag --yes`	dépendances du RAG (~1,5 Go : torch, faiss, flashrank)
MARI	`… --with mari`	rien à télécharger : le bridge est dans `uhb/services/mari/bridge`, son framework est un submodule (MIT) — `git submodule update --init`

Le dépôt est un monorepo : le code du RAG et le bridge MARI sont dedans ; le framework MARI et SiYuan ne le sont pas (submodule MIT et téléchargement vérifié).

`uhb brain start --dry-run` montre ce qui serait lancé et d'où ; `uhb brain start` lance ce qui est installé et affiche les pid. Rien de téléchargé n'entre dans git (`uhb/bin/` est ignoré, jamais empaqueté), et l'index du RAG n'est pas livré : il est fait de vos notes.

Pour poser le bouton « 9+ UHB » dans une interface, `uhb ui` donne le markup et le chemin de `UI_INTEGRATION.md` : CSS préfixé, aucune origine en dur, CORS local.

9. Mise à jour

```
uhb update --dry-run   # récupère et rapporte, ne déplace rien
uhb update             # git pull --ff-only, puis prouve le résultat
```

`uhb update` refuse un arbre sale, n'a ni `--force` ni `reset`, et s'arrête si la branche a divergé : il ne fusionne pas, ne rebase pas. Il rapporte le commit avant et après.

Il **ne redémarre aucun service** : UHB n'embarque pas de superviseur. systemd, le Planificateur de tâches ou votre terminal possède le processus.

---

10. Dépannage

`uhb status` affiche `MISSING` pour un fournisseur

La variable d'environnement nommée n'est pas chargée dans ce shell. Rien n'est cassé : le routeur sautera ce fournisseur et continuera sa chaîne.

`AllProvidersFailed` pendant une compilation

Tous les candidats ont échoué. Lisez les tentatives : `skipped` signifie « clé absente », `failed` porte le nom de la classe d'exception. Sans clé configurée, la chaîne se termine sur le modèle local.

`ModeRequired`

Un chemin d'écriture a été appelé sans mode. Passez-en un explicitement (`get_mode("dev")`) : c'est la garantie qu'aucune écriture ne se produit sans politique.

`ModeViolation: read-only`

Le mode choisi n'écrit pas. Utilisez `dev` pour compiler, ou un profil que vous avez écrit.

• `siyuan` mais la lecture d'un document échoue

SiYuan répond à `/api/system/version` sans authentification, mais refuse le contenu : `SIYUAN_TOKEN` manque dans ce shell. La sonde reste verte parce qu'elle ne teste que la version ; `uhb brain read` échoue alors avec `Auth failed [session]` .

Le PDF ne se télécharge pas

`GET /docs/manuel.pdf` n'existe que si le paquet a été installé avec ses données (`pip install -e .` depuis un checkout les inclut). `uhb docs` affiche le chemin local réel.

---

11. Limites connues

- La compilation a été mesurée sur une note de trois phrases, pas sur un corpus.
- Seul DeepSeek est vérifié contre le catalogue réel de son fournisseur ; Kimi, OpenAI et Anthropic restent déclarés, non prouvés.
- Les adaptateurs Harness Maxxing et OpenClaw déclarent le contrat ; ils n'ont pas été exécutés contre ces produits.
- Un harness lancé en sous-processus sort du périmètre de la gate : UHB protège ses propres écritures, pas celles d'un processus tiers.

---

12. Licence

MIT. Auteur : Thomas Leroyer (Searching Murphy).