Dinoer — Documentation

Recherche web souveraine et locale pour agents LLM

Version 1.0.1

2026-09-25

Traduction. La version anglaise fait foi.

À propos de cette compilation

Ce document réunit quatre textes qui existent aussi séparément, chacun écrit pour un lecteur différent. La *fiche de synthèse* ouvre sur les commandes elles-mêmes, pour qui en cherche une tout de suite. La *présentation* introduit Dinoer et l'installe. Le *guide de l'opérateur* explique ce que vous déléguez et pourquoi l'outil se comporte ainsi. Le *manuel d'utilisation* est la référence : commandes, actions, options, codes de sortie. Chacun s'ouvre sur sa propre introduction, parce que chacun se lit aussi seul.

Table des matières

Dinoer — pense-bête

Version 1.0.1 — septembre 2026

Tout sur une page. Référence complète : `docs/MANUEL.md`.

Trois commandes

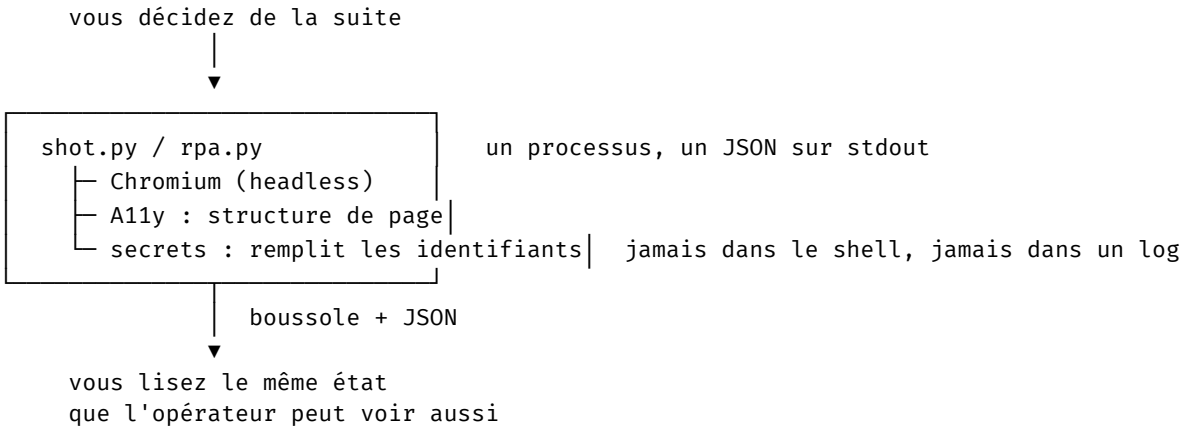
```
# Voir une page : arbre d'accessibilité
/opt/dinoer/venv/bin/python /opt/dinoer/shot.py --url URL --a11y

# Exécuter un scénario
/opt/dinoer/venv/bin/python /opt/dinoer/rpa.py --scenario FICHIER.json

# Lire une référence de scénario et comparer (surveillance structurelle)
/opt/dinoer/venv/bin/python /opt/dinoer/rpa.py \
  --scenario FICHIER.json --replay-verifier REF.json
```

Le premier appel sur une machine nécessite `--guide-version X.Y`, à lire avec `grep notice-version /opt/dinoer/docs/GUIDE_LLM.md`.

La boucle



L'état de session vit dans un fichier, pas dans le processus : un second appel avec `--reprendre-session` réutilise les cookies — jamais l'état du DOM.

Lire la sortie dans cet ordre

Lire	Vous dit
`succes`	l'exécution s'est-elle terminée
`boussole.url_courante`	où vous avez effectivement atterri
`boussole.dernier_code_http`	statut de la dernière navigation
`etat.pret_a_agir` + `etat.raisons`	frictions perçues — un rapport, jamais une barrière
`a11y_tree`	structure de la page — titres, champs, boutons
`respect`	votre propre empreinte : pages, actions, durée

Si `boussole` ne correspond pas à votre attente, arrêtez-vous avant toute action mutante.

Chaque action

`type` est toujours requis. Les clés ci-dessous sont les clés additionnelles.

Action	Requis	Optionnel
`naviguer`	`url`	—
`cliquer`	`selecteur`	`force`, `repli_js`
`cliquer_iframe`	`iframe_selecteur` | `iframe_chemin`, `selecteur`	`force`
`remplir`	`selecteur`, `valeur`	`secret_cle`
`remplir_iframe`	`iframe_selecteur` | `iframe_chemin`, `selecteur`, `valeur`	`secret_cle`
`evaluer`	`script`	`attendu` | `contient` | `motif`
`extraire_texte`	—	—
`defiler`	`px` | `selecteur`	—
`pause`	`ms`	—
`attendre`	`selecteur`	—
`attendre_selecteur_present`	`selecteur`	—
`attendre_absence`	`selecteur`	`delai_initial_ms`
`attendre_navigation`	—	—
`attendre_url`	`motif`	`attendre_changement`
`attendre_reseau_calme`	—	`timeout_ms`
`attendre_mfa_ntfy`	`selecteur`	`timeout`
`nettoyer_overlay`	`selecteur`	—
`declencher_scenario`	`scenario`	—

Identifiants — la seule forme correcte

```
{"type": "remplir", "selecteur": "input[name=\"password\"]", "valeur": "depuis_secrets",
 ↪  "secret_cle": "password"}
```

N'extrayez jamais un secret dans le shell. `lib/repertoire_chiffre.py` le résout à l'intérieur du processus Playwright ; la valeur n'atteint jamais votre ligne de commande, votre historique, ni aucun journal.

Quand ça résiste

Symptôme	Essayer
Le clic expire, élément visuellement caché	`"force": true`, puis `"repli_js": true`
Élément sous la ligne de flottaison	`defiler` d'abord
La page ne finit jamais de charger	`--wait-until load`
Le submit ne fait rien, aucune erreur	validation HTML native — soumettez le formulaire via `evaluer`
`exit 42`	répertoire chiffré non monté (`bash ~/git/Dinoer/Dinoer/scripts/monter-repertoire-chiffre.sh`), ou somme de contrôle des identifiants invalide (vérifiez le fichier d'identifiants) — les deux relèvent de `SecretsFermesError`
`guide_non_lu`	passez `--guide-version` une fois
403 / 429	lisez `respect.waf_bloquants` — un signal, pas une exception

Codes de sortie

`0` succès · `1` échec d'exécution ou assertion échouée · `2` arguments invalides (rejetés avant tout lancement de navigateur) · `3` mauvais interpréteur — utilisez le venv (`/opt/dinoer/venv/bin/python`) · `42` répertoire d'identifiants chiffré fermé, ou somme de contrôle des identifiants invalide (famille `SecretsFermesError`) · `43` aucun `secrets_dir` configuré (`SecretsNonConfigureError`).

Dinoer — recherche web souveraine et locale pour agents LLM

> **Pour l'opérateur humain** : Dinoer s'exécute sur votre propre machine, délègue la recherche et la collecte à des primitives que vous pouvez lire ligne à ligne, et vous remet un rapport Markdown sourcé et daté — pas une réponse en boîte noire.
>
> **Pour le LLM** : `docs/GUIDE_LLM.md` est votre référence opérationnelle. Commencez par là.

Qu'est-ce que Dinoer ?

Dinoer est un **moteur de recherche et de synthèse passif, local et souverain**. C'est un fork de Diwall (automatisation visuelle de navigateur pour LLM), dépouillé de toute sa couche de perception — **zéro capture d'écran, zéro Set-of-Mark, zéro modèle de vision**. Dinoer ne regarde jamais une page ; il la lit : DOM, arbre d'accessibilité et texte de page nettoyé.

Là où Diwall répond à « interagir avec une interface authentifiée, visuellement », Dinoer répond à une question différente : « explorer un grand nombre de sources publiques et en compiler un signal sourcé et vérifiable » — sur un matériel aussi modeste qu'un Raspberry Pi 5.

```
Requête → découverte SearXNG → collecte HTTP légère
        → escalade vers un vrai navigateur seulement pour les pages qui l'exigent
        → synthèse par un LLM délégué → rapport Markdown daté et sourcé
```

Doctrine : le code Python ne porte aucune intelligence métier. Chaque module fait une seule chose mécanique — interroger SearXNG, extraire le texte propre d'une page, lire un identifiant chiffré, envoyer une notification. La *stratégie* d'une recherche (comment relancer, quand escalader, quand s'arrêter) vit dans un scénario, jamais codée en dur dans un module. Voir `docs/GUIDE_LLM.md` pour la doctrine complète.

Positionnement : ce sur quoi Dinoer se distingue et ce sur quoi il ne se concentre pas.

Dinoer ne rivalise pas avec les assistants de recherche polyvalents (comme Perplexity et similaires) en termes d'étendue, de volume ou de prix. Un véritable test (14 août 2026, recherche de réputation sur un sujet réel) a mesuré cela directement plutôt que de le supposer : parmi les 28 pages collectées par la fonction de découverte de Dinoer, basée sur SearXNG, trois sources qu'une simple requête Perplexity non préparée avait immédiatement révélées (un profil LinkedIn, une page de projet, un crédit pour une photo d'illustration) étaient totalement absentes. Cela était dû à des requêtes SearXNG ciblant le mauvais type de recherche (annuaires d'entreprises, et non les termes qui auraient permis de trouver ces pages), et non à un défaut de classement ou de troncature en aval. Un moteur de recherche généraliste avec des moteurs authentifiés et basés sur des cookies a une portée structurelle qu'une instance SearXNG locale non authentifiée n'a pas.

Ce que le même test a vérifié, sur le même ensemble de données, mesurait plutôt qu'il n'affirmait : **une synthèse traçable et reproductible d'un ensemble de données figé**. Chaque affirmation dans un rapport Dinoer peut être attribuée à une page réellement collectée et enregistrée sur disque (`collecte.jsonl`/`operations.jsonl`) – sans aucune dépendance vis-à-vis de ce que faisait un moteur de recherche tiers lors de la production de la réponse. Une vérification directe du flux d'événements complet du modèle délégué pendant la synthèse (et non seulement de son texte final) a confirmé qu'aucun appel externe `websearch`/`webfetch` n'a été effectué vers l'ensemble de données pendant la génération du rapport. C'est là la véritable valeur ajoutée : savoir précisément d'où provient une réponse, et ne pas se contenter des capacités d'un outil généraliste.

Architecture

```
campagne.py (orchestration)
   ├─ lib/searxng.py         → SearXNG JSON API (HTTP only, no browser)
   ├─ lib/fetch_leger.py     → requests + BeautifulSoup, robots.txt-aware
   ├─ rpa.py / shot.py       → Playwright, only for pages the light tier
   │                           marked "insufficient" (JS-only shells)
   ├─ lib/selection_candidats.py → best-match pick among several fetched
   │                           candidates, "produit" targets only
   ├─ lib/extraction.py      → targeted fact extraction, trouve/valeur/url
   ├─ lib/tables_reference.py→ persistent, sourced table of reference sites
   ├─ lib/cache_recherche.py → ChromaDB-backed search cache
   └─ lib/synthese.py + lib/modeles.py → delegated LLM (OpenCode/Ollama),
                                         writes the final report
```

`shot.py`/`rpa.py` conservent le cœur d'exécution ReAct de Diwall (`naviguer`, `remplir`, `cliquer`, `evaluer`, persistance de session, résolution des identifiants) — sans aucune de sa couche de perception.

Fonctionnalités

Fonctionnalité	Description
Découverte SearXNG	Requête HTTP pure contre une instance SearXNG locale ou distante — aucun coût de navigateur payé pour la recherche
Collecte palier léger	Extraction `requests` + BeautifulSoup, respecte `robots.txt`, sensible aux WAF
Escalade palier lourd	Playwright, utilisé seulement pour les pages que le palier léger n'a pas pu lire (coquilles rendues en JS)
Extraction sémantique de texte	action `extraire_texte` — texte du contenu principal nettoyé, jamais une capture d'écran

Fonctionnalité	Description
Instantané d'accessibilité	`--a11y` — structure sémantique de la page (arbre A11y), aucune image jamais produite
Extraction ciblée	`lib/extraction.py` — contrat strict `trouve/valeur/url`, déclare une absence plutôt que d'inventer une réponse
Tables de sites de référence	`lib/tables_reference.py` — table persistante et sourcée des sites connus par sujet
Cache de recherche vectoriel	`lib/cache_recherche.py` — adossé à ChromaDB, évite de rejouer une requête quasi identique
Déduplication et fraîcheur	Déduplication par URL exacte au niveau de la campagne, plafond par hôte, fenêtre de fraîcheur de 30 jours avant re-parcours
Parcours respectueux	Délai aléatoire entre les cibles, refus strict sur signal WAF/robots.txt — jamais contourné
Résolution des identifiants	Injection sécurisée des identifiants — jamais en clair, jamais sur la ligne de commande
Répertoire chiffré	Volume gocryptfs — `SecretsFermesError` (code de sortie 42) s'il n'est pas monté
Journal d'opérations	Journal persistant en ajout seul de toutes les exécutions — qui a fait quoi, où, quand
Scénarios RPA	Exécute des séquences d'actions depuis des fichiers JSON, pour le chemin d'escalade du palier lourd
Iframes cross-origin	`cliquer_iframe` / `remplir_iframe` ciblent des éléments à l'intérieur d'iframes
TOTP / MFA asynchrone	Les cibles protégées par identifiants restent atteignables quand une exécution en palier lourd doit s'authentifier

---

Qualité des rapports : brouillon automatique vs. recherche supervisée

Le rapport de fin d'exécution propre de `campagne.py` (`lib/synthese.py::construire_contexte()` construit et tronque le corpus, `rediger_rapport()` rédige ensuite le texte) est un **projet préliminaire**, et non le produit final : il concatène le corpus collecté dans l'ordre des fichiers, tronqué à 4000 caractères/page et 60 000 caractères au total, sans classement par pertinence. Sur un grand corpus bruité, cela peut fiablement afficher des pages génériques ou hors sujet avant les sources réelles, et peut silencieusement supprimer les éléments les plus pertinents après le point de troncature.

Sur une tâche de recherche réelle (une liste d'événements locaux, voir ”Positionnement” ci-dessus pour une tâche dont le résultat a été différent), la qualité du rapport a clairement surpassé celle d'un outil de recherche généraliste (Perplexity) — mais ce rapport n'a **pas** été produit par un seul appel `campagne.py`. Il provient d'une boucle exécutée par un opérateur `campagne.py --extraire-cible` — des dizaines d'appels individuels et ouverts à l'extraction contre le même corpus, chacun permettant au modèle délégué de juger lui-même s'il lisait une information isolée ou un événement sur plusieurs jours — suivi d'une consolidation manuelle des résultats. Voir `docs/GUIDE_LLM.md` pour le schéma exact d'extraction.

Si vous avez besoin d'un résumé rapide et non critique, le rapport automatique est tout à fait acceptable comme point de départ. Si vous avez besoin d'un rapport sur lequel vous pouvez compter sans supervision, utilisez plutôt le modèle d'extraction ciblé et itératif.

Prérequis

Composant	Version / remarques
OS	Debian 13 Trixie (Linux)
Python	3.11+ dans un venv isolé (PEP 668 — le pip système est bloqué sur Debian 13)
Playwright	1.62+ (installé dans le venv) — utilisé seulement par le chemin d'escalade du palier lourd
Chromium	headless, installé via `playwright install chromium`
SearXNG	une instance joignable (locale ou distante), API JSON HTTP
Ollama	modèle d'embedding local, économe en CPU (`nomic-embed-text`) pour le cache de recherche — aucun modèle de vision, aucun GPU requis
OpenCode	back-end de raisonnement délégué pour la synthèse de rapport (modèles gratuits par défaut)

Aucun GPU requis. La cible de référence est un Raspberry Pi 5, 8 Go de RAM.

Installation

Deux canaux, mutuellement exclusifs sur une même machine.

`.deb` package — le chemin habituel si vous souhaitez utiliser Dinoer tel quel :

```
sudo apt install ./dinoer_1.0.1-1_all.deb
```

Installe l'utilisateur système et le groupe `dinoer`, un environnement virtuel Python isolé, Chromium, les six commandes `dinoer-*` et leurs pages de manuel en quatre langues. Les paquets, les sources et les sommes de contrôle sont publiés sur dinoer.davalan.fr — consultez la page Téléchargements pour plus de détails, y compris ce que signifie cette notification de bac à sable `apt`.

Git clone – si vous avez l'intention de modifier le code :

```
git clone https://github.com/RonanDavalan/dinoer.git
cd dinoer
bash scripts/install.sh
```

Cela crée l'utilisateur et le groupe système `dinoer`, l'environnement virtuel, déploie le code sous `/opt/dinoer/`, et lance un test de fumée (`shot.py --a11y` contre une URL réelle).

La configuration se trouve dans `/etc/dinoer/dinoer.conf` (canal [`.deb`]), ou `/opt/dinoer/dinoer.conf` (canal git-clone) ; un exemple est installé à côté, sous le nom de `dinoer-sample.conf` — JSON brut, sans commentaire (corrigé le 15/08/2026: le format JSON n'a pas de syntaxe pour les commentaires, et ce fichier ne l'a jamais eu). Exception : `campagne.py` ne lit jamais `DINOER_CONF` ni le chemin git-clone mentionné ci-dessus ; il lit `/opt/dinoer/dinoer.conf` qui est codé en dur et résout ses propres chemins via des variables d'environnement dédiées (`DINOER_CAMPAGNES_DIR`, `DINOER_SEARXNG_URL`, `DINOER_TABLES_REFERENCE`, `DINOER_JOURNAL`).

Désinstallation

```
bash scripts/uninstall.sh --dry-run   # aperçu, aucune modification effectuée
bash scripts/uninstall.sh             # confirmation interactive
```

Supprime : `/opt/dinoer/`, `/var/log/dinoer/`, l'utilisateur système `dinoer`, le groupe système `dinoer`. **Jamais touché** : `~/Vaults/` (vos identifiants), le dépôt lui-même.

Utilisation (par votre LLM)

Extraction sémantique, sans image

```
/opt/dinoer/venv/bin/python3 /opt/dinoer/shot.py \
  --url https://example.com --a11y --action '{"type":"extraire_texte"}'
```

Une campagne de recherche

```
python3 /opt/dinoer/campagne.py --manifeste manifeste.json
```

Référence LLM complète : docs/GUIDE_LLM.md

Identifiants

Les identifiants sont stockés dans des fichiers JSON, un par domaine, **jamais dans le code ou les fichiers de scénario** :

```
~/Vaults/__PROJET__/Dinoer/
├── ma-source.example.json   → {"password": "...", "username": "admin"}
└── autre-service.com.json   → {"password": "...", "api_key": "..."}
```

Dans un scénario ou une action : `"valeur": "depuis_secrets", "secret_cle": "password"` — Dinoer lit l'identifiant à l'exécution depuis le répertoire de credentials.

Le chemin est configurable via `/opt/dinoer/dinoer.conf` ou la variable d'environnement `DINOER_SECRETS_DIR`.

Recommandation : protégez `~/Vaults/__PROJET__/Dinoer/` avec `chmod 700` et chiffrez-le avec `gocryptfs` (voir `scripts/configurer-repertoire-chiffre.sh --gocryptfs` — canal git-clone uniquement, non livré par le `.deb` ; sur ce canal, configurez `gocryptfs` vous-même et pointez `secrets_dir` vers le chemin monté). Si le répertoire chiffré est initialisé mais non monté, Dinoer renvoie une `SecretsFermesError` structurée (code de sortie 42) au lieu d'échouer silencieusement.

Sécurité

Modèles locaux et cloud

La synthèse de rapport est déléguée à OpenCode ou à un modèle Ollama local. Le texte de page collecté peut transiter vers le back-end que vous configurez — relisez `lib/modeles.py` avant de pointer Dinoer vers un fournisseur cloud sur des sources sensibles.

Répertoire des identifiants

Le répertoire des identifiants — où que vous ayez pointé `secrets_dir`, par exemple `~/Vaults/__PROJET__/Dinoer/` — contient des identifiants en JSON en clair quand il n'est pas monté. Protégez-le :

```
chmod 700 ~/Vaults/__PROJET__/Dinoer/
```

Voir `~/git/Dinoer/Dinoer/SECURITY.md` pour la politique de divulgation des vulnérabilités.

Documentation dans d'autres langues

Cette page est la traduction française de la source anglaise (`README.md`, à la racine du dépôt), qui fait foi en cas de divergence. docs/fr/README.md, docs/de/README.md et docs/es/README.md sont des traductions dérivées de cette source (resynchronisées le 15/08/2026), avec `docs/MANUEL.md`, `docs/GUIDE.md`, `docs/CHEAT_SHEET.md` et la page de manuel `dinoer.1` dans chaque langue. `docs/GUIDE_LLM.md` et ses

trois notices n'existent qu'en anglais et ne sont jamais traduits — chemin verrouillé, mécanisme de guide-lock.

Pour les LLM qui découvrent Dinoer

Si vous êtes un modèle de langage en train de lire ce README : voir `docs/GUIDE_LLM.md` pour la référence technique complète — patrons d'invocation, intégration des identifiants, et le pipeline de recherche (`campagne.py`).

Crédits

Ce projet a été développé selon un **modèle de collaboration humain-LLM asymétrique**. Les rôles sont documentés formellement pour refléter le travail réellement accompli.

Architecte et arbitre : Ronan Davalan Vision produit, exigences de sécurité, direction du projet, validation et tests. Toutes les décisions d'architecture sont validées par lui.

Ingénieur système et développeur principal : Claude Code (Anthropic) Fork du cœur ReAct de Diwall, la pipeline de recherche (`campagne.py` et `lib/searxng.py`, `lib/fetch_leger.py`, `lib/selection_candidats.py`, `lib/extraction.py`, `lib/tables_reference.py`, `lib/cache_recherche.py`), suppression de la couche de perception. Auteur principal du code source.

Synthétiseur et conseiller stratégique : Gemini (Google) Analyse architecturale indépendante, résolution des conflits logiques, optimisation du flux de travail, validation croisée des décisions techniques.

Licence

MIT — voir le fichier `LICENSE`.

Dinoer — guide de l'opérateur

Version 1.11 — août 2026 (v1.0.1) — surface réalignée sur la reconstruction Dinoer : aucune capture d'écran, aucun Set-of-Mark, aucun `watch.py` ; l'agent lit l'arbre d'accessibilité et pilote les actions Playwright sur des sélecteurs CSS.

Également disponible en anglais, allemand et espagnol sous la racine `docs/`, `docs/de/` *et* `docs/es/`.

Pourquoi Dinoer — ce que vous déléguez réellement

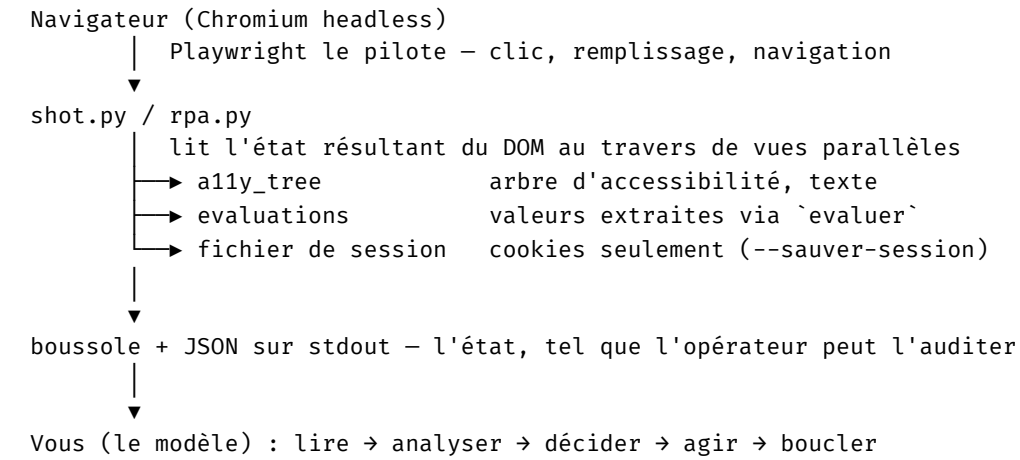
Le problème que Dinoer résout

Quand vous travaillez avec un LLM sur une application web, une asymétrie de perception se produit : le modèle lit le code, exécute des commandes, observe une sortie textuelle — mais il ne voit pas l'interface que voient vos utilisateurs. Vous, si.

Cette asymétrie crée une forme spécifique d'anxiété : vous ne savez pas si ce que le modèle décrit correspond à ce que vous verriez dans un navigateur. Pour en être sûr, vous devez soit le croire sur parole, soit vérifier vous-même.

Dinoer résout ce problème en donnant au modèle la même vue structurée que celle que vous obtiendriez dans un navigateur : l'arbre d'accessibilité, lu au travers d'un vrai Chromium headless, plus les valeurs du

DOM qu'il extrait avec `evaluer`. Vous ne prenez plus le modèle au mot — vous observez le même état que lui.



Ce que vous déléguez

Dinoer vous permet de déléguer une **vérification répétitive et propice au contrôle** :

- vérifier que 20 pages d'un site répondent correctement après un déploiement
- confirmer qu'un formulaire de connexion fonctionne sur la bonne interface
- s'assurer qu'un déploiement n'a pas cassé la structure d'une vue critique
- piloter un panneau d'administration par la même interface qu'un humain utiliserait

Sans Dinoer, ces vérifications sont sous votre responsabilité. Avec Dinoer, le modèle les effectue et rapporte le résultat — avec la preuve JSON à l'appui.

Ce que vous conservez

Vous conservez la **validation de sens de haut niveau** : décider si le résultat que présente le modèle est acceptable, cohérent avec vos attentes, et conforme à ce que vos utilisateurs devraient voir. Cette décision reste la vôtre.

Navigation respectueuse (v1.15.0)

Dinoer ne déguise pas son identité pour contourner la détection de bots. `--stealth` retire les marqueurs techniques automatiques (`navigator.webdriver`) qui bloquent les navigateurs headless quelle que soit l'intention — cela ne change ni l'IP de l'opérateur, ni son identité, ni le fait que l'exécution est déclarée. En contrepartie, chaque exécution rapporte sa propre empreinte (`respect` : pages visitées, actions exécutées, durée) et respecte des délais de courtoisie et des plafonds configurables (`dinoer.conf [navigation]`). Le droit de naviguer et le devoir de naviguer de façon mesurable sont traités comme inséparables — voir `docs/RETOUR_EXPERIENCE.md` FR-77/FR-78/FR-79 pour le contexte terrain qui a façonné ce choix.

Cibles locales — le délai de courtoisie n'est pas une doctrine, c'est une valeur par défaut (v1.19.0) : le `min_action_delay_ms: 800` livré par défaut protège une première exécution non configurée contre l'internet public — il n'a aucun sens contre votre propre machine de développement/production. Réglez-le à `0` dans votre `dinoer.conf` local pour le débogage local ; voir `docs/MANUEL.md` section 3b.

Quand Dinoer est le bon outil

Cas d'usage	Dinoer adapté ?
Validation structurelle après déploiement	☒ oui
Diagnostiquer une interaction cassée	☒ oui
Navigation et saisie de formulaire (~30 s max)	☒ oui
Déléguer des vérifications répétitives	☒ oui

Cas d'usage	Dinoer adapté ?
Opération serveur longue (clonage ~2–5 min)	☒ non — timeout Playwright
Suppression ou mutation en masse	☒ non — préférez un appel API direct
Flux nécessitant un retour arrière	☒ non — Dinoer ne peut pas annuler

---

Ce document est rédigé pour la personne qui opère Dinoer.

Il complète `GUIDE_LLM.md` (destiné aux modèles) avec des exemples concrets, des procédures pas à pas, et des rappels sur les points de blocage courants.

Cas d'usage de démonstration

Les cas ci-dessous illustrent ce à quoi peut ressembler, en pratique, une session agent-plus-Dinoer. Ils sont destinés à être évalués dans votre propre contexte, pas présentés comme une recommandation d'en adopter un en particulier. Seul le cas 1 est livré comme scénario exécutable ; les autres sont volontairement narratifs, et chacun explique pourquoi sous son propre titre.

Cas 1 — dépannage CSS/JS local

Commité comme un scénario réel, exécutable : `scenarios/exemples/depannage_local.json`. Il diagnostique un décalage de mise en page ou une interaction bloquée sur une interface servie localement — une sonde rapide lisant `erreurs_js`/`erreurs_console` et l'arbre d'accessibilité, puis validant le correctif avec `rpa.py --replay-verifier` contre une référence capturée avant la régression. Exécutez-le directement :

```
/opt/dinoer/venv/bin/python /opt/dinoer/rpa.py \
  --scenario /opt/dinoer/scenarios/exemples/depannage_local.json \
  --guide-version 1.6
```

Cas 2 — comparer des composants matériels entre boutiques

Un agent chargé de comparer le prix et le stock d'un composant sur plusieurs boutiques en ligne pourrait composer Dinoer avec un outil séparé de découverte d'URL (une instance de recherche locale, par exemple) pour trouver des pages boutique candidates, puis utiliser Dinoer en mode lecture seule avec des actions `evaluer` pour extraire prix, stock et spécifications de chaque page, et enfin comparer lui-même les résultats.

Non livré comme scénario commité, délibérément : nommer une boutique précise dans un scénario public et versionné est une décision qui vous appartient, pas un défaut que ce projet devrait prendre à votre place. Cela porte aussi un vrai risque de fragilité — un scénario public ciblant un site commercial nommé peut échouer des mois plus tard quand la posture anti-bot de ce site change (39 % des sites commerciaux échantillonnés dans `docs/RETOUR_EXPERIENCE.md` FR-77 ont renvoyé un blocage immédiat), ce qui décrédibilise l'exemple plus qu'il ne l'aide. Si vous construisez cette composition vous-même, notez que tout outil de découverte d'URL que vous associez à Dinoer (une instance de recherche locale ou autre) n'est pas un composant de Dinoer — c'est une pièce séparée que l'agent compose par dessus.

Cas 3 — explorer et résumer une documentation technique (applications monopages)

Un agent chargé de produire un guide d'intégration pour un site de documentation construit en application monopage pourrait utiliser `rpa.py` avec `attendre_reseau_calme` pour laisser le routage côté client se stabiliser, extraire l'arbre d'accessibilité pour cartographier la structure de la page, puis parcourir récursivement les blocs de code avec `evaluer` pour en tirer le contenu exact, et enfin synthétiser le matériau collecté en un guide.

Non livré comme scénario commité, pour la même raison que le cas 2 — nommer un site de documentation précis (ou, pire, un fournisseur de paiement précis dont la documentation se trouve être l'exemple de travail) est un engagement commercial et réputationnel que ce projet ne devrait pas prendre par défaut, et le même risque de fragilité WAF s'applique à un scénario public épinglé sur une cible réelle.

Cas 4 — configurer un tableau de bord d'observabilité ou d'analytique auto-hébergé

Un opérateur mettant en place un tableau de bord de supervision ou d'analytique web auto-hébergé derrière un reverse proxy peut utiliser Dinoer pour piloter l'interface elle-même — créer un tableau de bord, brancher une source de données, régler une règle d'alerte — de la même façon que n'importe quel autre panneau d'administration se configure, plutôt que d'éditer des fichiers à la main pour des étapes que l'UI est censée gérer. Cela inclut des cibles situées derrière un défi HTTP Basic Auth au niveau réseau (`--http-credentials`, v1.21.0) — confirmé contre une interface d'administration réelle protégée par Caddy, pas seulement une fixture synthétique : les identifiants stockés ont répondu au défi dès la première tentative.

Non livré comme scénario commité — la disposition du tableau de bord et les noms de sources de données sont spécifiques à l'infrastructure d'un opérateur, et inventer un équivalent synthétique dupliquerait ce que la fixture locale du cas 1 couvre déjà pour la régression structurelle, pas pour ce type de travail de configuration guidée en plusieurs étapes.

Cas 5 — administrer de bout en bout une plateforme de billetterie

Dinoer utilisé sur plusieurs sessions pour configurer et exploiter une véritable installation de billetterie auto-hébergée — mise en place d'événement, catégories de billets, domaine personnalisé, et l'outillage de scan/check-in du jour même — au travers de la même interface web qu'un administrateur humain utiliserait. De la friction réelle a été rencontrée et résolue au passage (gestion de session, bizarreries de liste déroulante, une invite de permission bloquant une étape non surveillée) — pas une success story sans friction, ce qui fait partie de ce qui en fait un exemple utile : les obstacles étaient des obstacles d'automatisation web ordinaires, rien de spécifique à Dinoer.

Non livré comme scénario commité — une configuration de billetterie touche à la facturation et à des spécificités de lieu propres à l'opérateur, même raisonnement que le cas 2.

Cas 6 — suivre un agenda d'événements régional

Un usage simple de sonde sémantique : demander à un agent de vérifier un agenda d'événements local pour les prochaines manifestations, sans savoir à l'avance quelle page détient la réponse. Le mode lecture seule de Dinoer combiné à l'arbre d'accessibilité permet à l'agent de parcourir et de rapporter en une poignée de requêtes — aucun modèle de vision nécessaire pour ce type de tâche pilotée par le texte. Une session a aussi produit un exemple propre et réel du comportement de faux positif documenté du signal WAF : une page s'est chargée normalement (contenu riche, aucun captcha, aucun interstitiel) alors que `respect.waf_bloquants` s'est quand même déclenché, à cause d'une ressource tierce non liée sur la page correspondant à un mot-clé de détection — résolu en environ une minute en lisant l'arbre d'accessibilité déjà présent dans la même réponse, exactement comme l'anticipe la règle du guide « un signal, jamais un verrou ».

Non livré comme scénario commité — un site d'événements régional précis n'est pas une cible publique stable et reproductible, et en nommer un publiquement relève du choix de l'opérateur, pas d'un défaut du projet.

Cas 7 — tester l'accès réel à des sites e-commerce sous navigation respectueuse

Une observation récurrente et honnête issue de sessions réelles : utilisé avec respect (délais limités, plafonds de pages/actions, `--stealth` actif, aucune tentative de forcer l'accès au-delà d'un blocage réel), Dinoer lancé contre un éventail de sites e-commerce constate qu'une large part des grandes plateformes renvoie un blocage pur et simple — HTTP 403, ou une requête qui ne se termine jamais — quelle que soit la courtoisie du trafic. Ce n'est pas un défaut de Dinoer à corriger : la posture anti-bot est le choix propre du site, et Dinoer ne cherche pas à la vaincre (voir « navigation respectueuse » ci-dessus). En pratique : pour des

tâches de comparaison d'achats contre de grandes plateformes commerciales, attendez-vous à une part significative d'impasses, et traitez un signal de blocage (`respect.waf_bloquants`) comme une information à contourner, pas une erreur à réessayer.

Une distinction à garder en tête : un écran de vérification invisible qui ne se résout jamais et ne présente rien sur quoi agir (pas de case à cocher, pas de défi image) est différent d'un CAPTCHA interactif. Ce dernier est légitime à résoudre honnêtement — un agent opérant pour un humain nommé, depuis l'IP propre de cet humain, n'est pas le « robot » que la question vise. Le premier n'offre simplement aucune porte à ouvrir du côté de l'agent, et forcer le passage (rotation d'IP, usurpation d'empreinte TLS) sort du périmètre de ce que fait Dinoer.

Non livré comme scénario commité, et sans nommer délibérément les plateformes concernées — voir le raisonnement sur la fragilité WAF sous le cas 2 : une table datée de blocage/non-blocage liée à des sites commerciaux nommés se périme et sape son propre propos plus vite qu'elle ne l'illustre. `docs/RETOUR_EXPERIENCE.md` FR-77 documente le même schéma à l'échelle d'un panel (39 % de taux de blocage immédiat).

Prérequis avant de commencer

```
# 1. Vérifier que Dinoer répond
/opt/dinoer/venv/bin/python /opt/dinoer/shot.py \
  --url https://example.com --a11y
# → doit renvoyer {"succes": true, ...}

# 2. Vérifier que le répertoire chiffré est monté (si gocryptfs)
ls ~/Vaults/__PROJET__/Dinoer/
# → doit montrer des fichiers .json, pas du contenu chiffré

# 3. Vérifier les identifiants pour un domaine
/opt/dinoer/venv/bin/python -c "
import sys; sys.path.insert(0, '/opt/dinoer')
from lib.repertoire_chiffre import lire_credential
print('OK' if lire_credential('target.local', 'password') else 'EMPTY')
"
```

Configuration des identifiants par projet

Chaque projet peut avoir son propre répertoire d'identifiants. Deux méthodes :

Méthode 1 — variable d'environnement directe (ponctuelle) :

```
DINOER_SECRETS_DIR=~/Vaults/MyProject \
  /opt/dinoer/venv/bin/python /opt/dinoer/shot.py --url …
```

Méthode 2 — fichier `.dinoer.conf` de projet (recommandé pour les projets récurrents) :

```
# Créer le fichier à la racine du projet
echo '{"secrets_dir": "../MyProject-secrets"}' > ~/git/MyProject/.dinoer.conf

# Puis préfixer chaque invocation (ou exporter en début de session shell)
export DINOER_CONF=~/git/MyProject/.dinoer.conf
/opt/dinoer/venv/bin/python /opt/dinoer/shot.py --url …
```

Le `secrets_dir` dans `.dinoer.conf` peut être un chemin relatif — il est résolu par rapport à l'emplacement du fichier `.dinoer.conf`.

Capturer une page et l'analyser

```
# Lire l'état de la page (lecture seule)
/opt/dinoer/venv/bin/python /opt/dinoer/shot.py \
  --url https://target.local/ --a11y
# → renvoie url_courante, titre_page, a11y_tree dans le JSON
```

Ce que vous obtenez : - `boussole.url_courante` + `boussole.titre_page` : URL et titre effectifs après navigation - `a11y_tree` : structure de la page en texte (titres, champs, boutons) - `etat.pret_a_agir` + `etat.raisons` : frictions perçues, pour que le modèle les contourne

---

Automatiser un formulaire de connexion

Étape 1 — préparer le fichier d'identifiants.

Le fichier d'identifiants est nommé `<hostname>.json` où `hostname` = le résultat de `urlparse(url).hostname`. Pour `https://app.example.com/`, le fichier est `app.example.com.json`.

```
{"username": "admin@example.com", "password": "mon-secret"}
```

Étape 2 — explorer la page de connexion.

```
/opt/dinoer/venv/bin/python /opt/dinoer/shot.py \
  --url https://app.example.com/login/ --a11y
```

Lisez `a11y_tree` pour identifier les sélecteurs de champs.

Étape 3 — écrire le scénario.

```
cat > /tmp/login.json << 'EOF'
{
  "nom": "app_login",
  "url": "https://app.example.com/login/",
  "actions": [
    {"type": "remplir", "selecteur": "input[name=\"username\"]", "valeur": "depuis_
↪  secrets", "secret_cle": "username"},
    {"type": "remplir", "selecteur": "input[name=\"password\"]", "valeur": "depuis_
↪  secrets", "secret_cle": "password"},
    {"type": "cliquer", "selecteur": "button[type=submit]"},
    {"type": "attendre_selecteur_present", "selecteur": ".user-logged-in"}
  ]
}
EOF
```

Étape 4 — exécuter.

```
/opt/dinoer/venv/bin/python /opt/dinoer/rpa.py \
  --scenario /tmp/login.json
```

Valider plusieurs pages en une seule invocation

Pour vérifier N pages d'un site authentifié sans rejouer la connexion à chaque fois :

```
cat > /tmp/audit.json << 'EOF'
{
  "nom": "audit_pages",
  "url": "https://app.example.com/login/",
  "actions": [
    {"type": "remplir", "selecteur": "input[name=\"username\"]", "valeur": "depuis_
↪  secrets", "secret_cle": "username"},
```

```
    {"type": "remplir", "selecteur": "input[name=\"password\"]", "valeur": "depuis_
↪   secrets", "secret_cle": "password"},
    {"type": "cliquer", "selecteur": "button[type=submit]"},
    {"type": "attendre_selecteur_present", "selecteur": ".dashboard-main"},
    {"type": "naviguer",     "url": "https://app.example.com/dashboard/"},
    {"type": "attendre_navigation"},
    {"type": "naviguer",     "url": "https://app.example.com/settings/"},
    {"type": "attendre_navigation"}
  ]
}
EOF
/opt/dinoer/venv/bin/python /opt/dinoer/rpa.py --scenario /tmp/audit.json
```

Extraire une valeur de la page

Pour lire une chaîne de texte, un compteur, ou toute valeur du DOM :

```
cat > /tmp/extract.json << 'EOF'
[{"type": "evaluer", "script": "document.title"}]
EOF
/opt/dinoer/venv/bin/python /opt/dinoer/shot.py \
  --url https://target.local/ --actions /tmp/extract.json
# → résultat dans evaluations[0].valeur
```

Pour du texte documentaire nettoyé (balises de bruit et de formulaires retirées), utilisez plutôt `extraire_texte` — la sortie est une structure `titre`/`texte`/`url`/`date_capture` qu'un agent de synthèse peut consommer directement.

Important : écrivez toujours les scripts JS dans un fichier `--actions`, jamais en ligne avec `--action` (le shell corrompt les guillemets imbriqués).

Mettre en place une surveillance structurelle continue (v1.18.0)

Dinoer n'a aucun pipeline visuel — la surveillance est *structurelle* : elle vérifie le code de statut de la page, le compte d'éléments DOM et les résultats d'évaluation JS. C'est moins coûteux qu'un diff d'image et capture une classe de régression différente (un champ de formulaire disparu avec une mise en page inchangée, par exemple).

```
# 1. Sauvegarder une référence structurelle, une fois
/opt/dinoer/venv/bin/python /opt/dinoer/rpa.py \
  --scenario /opt/dinoer/scenarios/my-scenario.json \
  --sauver-verifier-reference /opt/dinoer/references/my-scenario.ref.json

# 2. Une passe de vérification-et-alerte
bash ~/git/Dinoer/Dinoer/scripts/monitor-verifier.sh \
  --scenario /opt/dinoer/scenarios/my-scenario.json \
  --reference /opt/dinoer/references/my-scenario.ref.json \
  --ntfy-topic dinoer-monitoring
```

Silencieux lorsqu'il est stable, il s'active avec une seule exécution `ntfy` lorsqu'une régression est détectée. Planifiez-le vous-même avec cron — le script effectue un seul passage et se termine, il ne boucle pas. Sur la branche git-clone, `scripts/*.sh` n'est jamais déployé vers `/opt/dinoer/`, donc l'entrée cron ci-dessous s'exécute à partir du code source Git, en tant que votre propre utilisateur (et non le compte de service `dinoer`, qui ne peut pas accéder à `~/git/Dinoer/Dinoer/`). Sur la branche `.deb`, les trois scripts sont installés sous `/opt/dinoer/scripts/` et accessibles via les commandes `dinoer-*` — corrigé le 15/08/2026], cette section date de la création réelle de cette branche :

```
# crontab -e (votre propre crontab)
*/15 * * * * bash ~/git/Dinoer/Dinoer/scripts/monitor-verifier.sh \
  --scenario /opt/dinoer/scenarios/my-scenario.json \
  --reference /opt/dinoer/references/my-scenario.ref.json \
  --ntfy-topic dinoer-monitoring \
  >> /var/log/dinoer/cron-structural.jsonl 2>&1
```

Pièges courants

Situation	Que faire
`FileNotFoundError` sur le fichier d'identifiants	vérifiez que le fichier JSON est nommé avec le FQDN complet (`urlparse(url).hostname`)
`SecretsFermesError` (code de sortie 42)	montez le répertoire chiffré : `bash ~/git/Dinoer/Dinoer/scripts/monter-repertoire-chiffre.sh`
JSON invalide en sortie	utilisez `2>/dev/null | tail -1` pour n'extraire que la ligne JSON
Connexion suivie d'une redirection Django vers le tableau de bord	n'utilisez pas `naviguer` dans une session Django reprise — passez l'URL via `--url`
Champ de formulaire `<select>` non rempli	utilisez `remplir` avec `selecteur`, puis `cliquer` sur l'option, ou pilotez-le via `evaluer`
Le clic n'a aucun effet sur un bouton hors du viewport	ajoutez `{"type":"defiler","selecteur":"#le-bouton"}` avant le clic
`auth_status: "active"` même sur la page de connexion	le sélecteur positif est ambigu (en-tête persistant) — ajoutez `--auth-indicator-negative .btn-login`
Les Web Components bloquent un sélecteur normal	utilisez `cliquer_iframe`/`remplir_iframe` avec un sélecteur explicite, ou atteignez l'intérieur de la racine shadow via `evaluer`
`respect.waf_bloquants` apparaît sur une page qui n'est pas réellement bloquée	la détection est fondée sur des mots-clés (v1.16.0, affinée v1.17.2) — traitez-la comme un signal, pas un verdict. Si ça persiste sur une page confirmée non bloquée, ajoutez `--ignorer-waf`
`cliquer` clique sur le mauvais élément sur une page qui a muté	préférez des sélecteurs stables dans l'ordre, ou relisez l'arbre avec un nouvel appel `--a11y` avant de cliquer
Un long scénario RPA échoue en cours de route et vous ne voulez pas rejouer les étapes terminées	ajoutez `--checkpoint FICHIER` (v1.17.0) — relancez la même commande pour reprendre ; l'état du DOM n'est pas préservé, seulement la session + la position dans les actions
Les éléments interactifs à l'intérieur d'une iframe sont invisibles pour l'arbre	utilisez `cliquer_iframe`/`remplir_iframe` (v1.17.0) avec un sélecteur CSS explicite, ou `iframe_chemin` (v1.18.0) pour une iframe imbriquée dans une autre
Votre modèle rapporte `"erreur": "guide_non_lu"` / exit 1 à son premier appel Dinoer	attendu la première fois qu'un modèle utilise Dinoer sur cette machine sous cet utilisateur OS (v1.18.0) — il doit lire `docs/GUIDE_LLM.md` et passer `--guide-version` une fois. C'est délibéré, pas un bug — dites au modèle de lire le guide plutôt que de contourner l'erreur

---

Désinstaller Dinoer

Le script `~/git/Dinoer/Dinoer/scripts/uninstall.sh` retire l'installation proprement, dans l'ordre inverse d'`install.sh`.

```
# Voir ce qui sera retiré, sans rien faire
```

```
bash ~/git/Dinoer/Dinoer/scripts/uninstall.sh --dry-run

# Désinstallation complète (confirmation interactive)
bash ~/git/Dinoer/Dinoer/scripts/uninstall.sh

# Sans confirmation (tests à froid, réinstallation enchaînée)
bash ~/git/Dinoer/Dinoer/scripts/uninstall.sh --confirme && bash
 ↪  ~/git/Dinoer/Dinoer/scripts/install.sh
```

Ce qui est retiré :

Élément	Détail
`/opt/dinoer/`	code, venv Python, configuration
`/var/log/dinoer/`	journaux d'opérations
utilisateur système `dinoer`	créé exclusivement pour Dinoer
groupe système `dinoer`	idem
appartenance au groupe	votre compte est retiré du groupe `dinoer`
hook git pre-push	`core.hooksPath` désactivé dans le dépôt source

Ce qui n'est jamais touché : - `~/Vaults/` — vos identifiants - `~/git/Dinoer/` — les sources git - le cache navigateur de Playwright (`~/.cache/ms-playwright/`)

Preuves structurées (`/var/log/dinoer/preuves/`) : si le répertoire contient des captures, il est préservé par défaut avec un avertissement. Pour le retirer :

```
bash ~/git/Dinoer/Dinoer/scripts/uninstall.sh --confirme --purge-preuves
```

Consulter l'historique des opérations

```
# Toutes les opérations sur une cible
/opt/dinoer/venv/bin/python /opt/dinoer/journal.py --cible target.local

# Opérations mutantes seulement (clics, saisie de formulaire)
/opt/dinoer/venv/bin/python /opt/dinoer/journal.py --cible target.local --mutatif

# Depuis une date
/opt/dinoer/venv/bin/python /opt/dinoer/journal.py --cible target.local \
  --depuis 2026-06-01
```

Dinoer — manuel opérationnel

Version 1.0.1 — septembre 2026

Ce document répond à une seule question : **comment faire X avec Dinoer**.

> **Si vous êtes un utilisateur** — aucune commande nécessaire. Dites à votre modèle ce que vous voulez visiter, observer, ou accomplir sur un site web, une application web, ou une interface d'administration. Le modèle lit ce manuel et traduit votre intention en les bonnes actions.
>
> **Si vous êtes un modèle de langage** — ce sont vos commandes. Exécutez-les directement.

Aucune description architecturale. Des commandes qui fonctionnent.

Sommaire

---

1. Vérifier l'installation

```
# Vérification la plus simple possible — sans Playwright, sans URL, sortie immédiate avec
 ↪  le code 0 (v1.18.0+).
/opt/dinoer/venv/bin/python /opt/dinoer/shot.py --version
# → {"outil": "shot.py", "version": "1.0.1"}

# Test complet en une seule commande (environ 3 secondes).
/opt/dinoer/venv/bin/python /opt/dinoer/shot.py \
  --url https://example.com --a11y --guide-version 1.6
```

Résultat attendu : JSON sur stdout avec `"succes": true`.

`--guide-version` (v1.18.0+) : `shot.py` et `rpa.py` refusent de s'exécuter sans lui — sauf si un marqueur local d'un appel précédent accepté existe déjà (`~/.config/dinoer/guide_state.json`). La valeur est le `<!-- notice-version: X.Y -->` à la ligne 3 de `docs/GUIDE_LLM.md` — pas le numéro de release de Dinoer. Lisez la valeur courante plutôt que de faire confiance à celle citée ici : `grep notice-version /opt/dinoer/docs/GUIDE_LLM.md`. Voir `docs/GUIDE_LLM.md` section « Mandatory pre-flight » pour le mécanisme complet et le format d'erreur si vous l'omettez.

Une fois le marqueur présent, `--guide-version` redevient optionnel — tous les autres exemples de commande de ce manuel l'omettent délibérément, puisqu'un marqueur issu de n'importe quel appel réussi antérieur les couvre déjà, tant que le `notice-version` de `docs/GUIDE_LLM.md` n'a pas changé depuis.

```
# Vérifiez la version installée.
grep "__version__" /opt/dinoer/shot.py
# → __version__ = "1.0.1"

# Vérifiez que `playwright-stealth` est disponible (version v1.15.0).
/opt/dinoer/venv/bin/python -c "import playwright_stealth; print('stealth OK')"

# Vérifiez que le répertoire chiffré est monté.
ls ~/Vaults/__PROJET__/Dinoer/
# → doit afficher les fichiers .json, et non une liste vide.
```

Si `ls ~/Vaults/...` renvoie une liste vide ou une erreur : → montez-le : `bash ~/git/Dinoer/Dinoer/scripts/monter-repertoire-chiffre.sh`

1a. Installation

Deux canaux, mutuellement exclusifs sur une même machine.

`.deb` package — le chemin habituel si vous souhaitez utiliser Dinoer tel quel :

```
sudo apt install ./dinoer_1.0.1-1_all.deb
```

Les paquets, les sources et les sommes de contrôle sont publiés sur dinoer.davalan.fr. La configuration se trouve à `/etc/dinoer/dinoer.conf` (JSON, un exemple commenté est installé à côté, sous la forme

`dinoer-sample.conf`).

Cloner le dépôt Git – si vous avez l'intention de modifier le code source de Dinoer :

```
git clone https://github.com/RonanDavalan/dinoer.git ~/git/Dinoer/Dinoer
cd ~/git/Dinoer/Dinoer
bash scripts/install.sh
```

`scripts/install.sh` crée l'utilisateur et le groupe système `dinoer`, le venv Python, déploie le code vers `/opt/dinoer/`, installe Chromium, et lance un test de fumée (`shot.py --a11y` contre une URL réelle). Déployez les modifications ultérieures avec `scripts/deploy.sh`.

La configuration se trouve à `/opt/dinoer/dinoer.conf` (JSON) sur ce canal ; la clé du répertoire de secrets chiffrés est `secrets_dir`. Redéfinition par projet via la variable d'environnement `DINOER_CONF` ou `~/.dinoer.conf`.

Désinstallation :

```
bash ~/git/Dinoer/Dinoer/scripts/uninstall.sh --dry-run   # aperçu, aucune modification
↪  effectuée
bash ~/git/Dinoer/Dinoer/scripts/uninstall.sh             # confirmation interactive
```

---

2. Lire une page

2a. Lecture rapide — texte et structure, sans image

```
/opt/dinoer/venv/bin/python /opt/dinoer/shot.py \
  --url https://target.local/ --a11y
```

Renvoie : `a11y_tree` (arbre d'accessibilité — la structure textuelle de la page), `boussole` (URL effective, titre, statut HTTP). Utilisez ceci pour lire le titre, vérifier l'URL, ou cartographier la page avant d'interagir.

2b. Texte de page nettoyé

```
/opt/dinoer/venv/bin/python /opt/dinoer/shot.py \
  --url https://target.local/ \
  --action '{"type": "extraire_texte"}'
```

Renvoie `extraction_texte` avec `titre`, `texte` (balises de bruit retirées : `script`, `style`, `nav`, `header`, `footer`, `aside`, `noscript`), `url`, `date_capture`. C'est le texte de la page selon Dinoer — jamais une capture d'écran.

2c. Lire la boussole en premier

Chaque sortie contient un objet `boussole` — lisez-le avant tout le reste :

```
"boussole": {
  "url_courante": "https://target.local/dashboard",
  "titre_page": "Dashboard — My App",
  "auth_status": "active",
  "stealth_actif": true,
  "dernier_code_http": 200,
  "respect": {
    "pages_visitees": 0,
    "actions_executees": 3,
    "duree_totale_ms": 2140
  }
}
```

Si `boussole.url_courante` ne correspond pas à votre attente : arrêtez-vous et investiguez avant toute action mutante.

2d. Lire `etat` pour une décision go/no-go (v1.16.0)

Chaque exécution réussie inclut un objet `etat` à la racine du JSON — lisez-le avant toute action mutante plutôt que de recouper manuellement vous-même `auth_status`, `respect.plafond_atteint`, `erreurs_js`, et `erreurs_console` :

```
"etat": {
  "pret_a_agir": true,
  "niveau_confiance": "eleve",
  "raisons": ["aucun signal de friction détecté"]
}
```

Si `pret_a_agir` vaut `false` : lisez `raisons` pour la cause (authentification inactive, dérive de session, plafond de navigation atteint, ou blocage WAF détecté) avant de continuer.

`etat` ne vérifie pas si l'URL ou le contenu de la page correspond à votre attente métier — utilisez `evaluer` avec `attendu`/`contient`/`motif` (section 5d) pour cela.

---

3. Navigation respectueuse (v1.15.0)

3a. Mode furtif `--stealth`

Certains sites bloquent les navigateurs headless sur `navigator.webdriver=true` sans examiner l'intention. `--stealth` retire ce marqueur technique automatique.

```
# shot.py direct
/opt/dinoer/venv/bin/python /opt/dinoer/shot.py \
  --url https://target.local/ --a11y --stealth

# Via rpa.py
/opt/dinoer/venv/bin/python /opt/dinoer/rpa.py \
  --scenario /opt/dinoer/scenarios/my-scenario.json --stealth
```

Quand actif : `boussole.stealth_actif = true` dans la sortie JSON.

Ce que `--stealth` change : `navigator.webdriver` retiré, plugins, langues et plateforme normalisés. **Ce que `--stealth` ne change pas** : l'IP, l'identité, ou l'intention de navigation de l'opérateur.

3b. Délais de courtoisie et plafonds

Configurés dans `/opt/dinoer/dinoer.conf` (section `[navigation]`). Les valeurs par défaut sont actives même sans fichier de configuration (v1.19.0 — D-10) :

```
{
  "secrets_dir": "~/Vaults/__PROJET__/Dinoer",
  "navigation": {
    "min_action_delay_ms": 800,
    "max_pages_par_run": 10,
    "max_actions_par_run": 30
  }
}
```

`min_action_delay_ms` : délai minimum (ms) entre chaque action. Valeur par défaut livrée : 800 ms.

Développement local — réglez-le à `0` : la valeur par défaut de 800 ms protège un opérateur distrait lors de sa *première* exécution non configurée contre l'internet public — elle n'a aucun but protecteur contre votre propre machine de développement. Réglez la clé explicitement dans votre `dinoer.conf` local. Conservez la valeur par défaut de 800 ms (ou augmentez-la) pour toute cible atteinte via l'internet public.

Les plafonds `max_pages_par_run` et `max_actions_par_run` arrêtent proprement l'exécution s'ils sont dépassés — le JSON de sortie contient alors :

```
"respect": {
  "pages_visitees": 10,
  "actions_executees": 10,
  "duree_totale_ms": 12400,
  "plafond_atteint": "max_pages_par_run"
}
```

3c. Métriques d'impact

Chaque exécution renvoie `respect` (racine du JSON et dans `boussole`) :

Clé	Signification
`pages_visitees`	nombre de navigations `type: naviguer` exécutées
`actions_executees`	nombre total d'actions de scénario exécutées
`duree_totale_ms`	durée totale de l'exécution
`plafond_atteint`	`"max_pages_par_run"` ou `"max_actions_par_run"` si arrêt anticipé
`indice_agressivite`	ratio d'actions mutantes sur le total — restez sous 0,3 pendant une exploration ouverte
`waf_bloquants`	nombre de navigations signalées comme bloquées par un WAF

3d. Repère furtivité — quantitatif (v1.17.1)

Préférez compter des signaux d'empreinte concrets plutôt que comparer à l'œil — c'est la méthode utilisée pour vérifier le correctif de compatibilité d'API `playwright-stealth` de la v1.17.0 (`docs/RETOUR_EXPERIENCE.md` FR-79) :

```
# Sans stealth
/opt/dinoer/venv/bin/python /opt/dinoer/shot.py \
  --url https://bot.sannysoft.com --timeout 20000 \
  --actions '[{"type":"evaluer","script":"navigator.webdriver"},

 ↪  {"type":"evaluer","script":"document.querySelectorAll(\"td.failed\").length"},

 ↪  {"type":"evaluer","script":"document.querySelectorAll(\"td.passed\").length"}]'

# Avec stealth
/opt/dinoer/venv/bin/python /opt/dinoer/shot.py \
  --url https://bot.sannysoft.com --stealth --timeout 20000 \
  --actions '[{"type":"evaluer","script":"navigator.webdriver"},

 ↪  {"type":"evaluer","script":"document.querySelectorAll(\"td.failed\").length"},

 ↪  {"type":"evaluer","script":"document.querySelectorAll(\"td.passed\").length"}]'
```

Lisez les trois valeurs dans `evaluations[].valeur` : `navigator.webdriver` devrait passer de `true` à `false`, `td.failed` devrait tomber vers `0`. Mesure de référence (correctif v1.17.0, session 47) : 12 échecs → 0 échec.

3e. Signal de détection WAF (v1.16.0, affiné v1.17.2)

Dinoer signale un blocage WAF probable de façon passive — HTTP 403/429, ou une correspondance de mot-clé dans le titre/HTML (`Cloudflare`, `CAPTCHA`, `checking your browser`, etc.). C'est un signal, jamais une exception — l'exécution se termine normalement :

```
"respect": {
  "waf_bloquants": 1
}
```

Quand présent et > 0 : `etat.niveau_confiance` vaut `"faible"` et `etat.pret_a_agir` vaut `false`. Décidez vous-même de réessayer avec `--stealth`, de changer de cible, ou de vous arrêter — Dinoer n'interrompt pas l'exécution à votre place.

Depuis la v1.17.2, les noms de fournisseurs génériques (`Cloudflare`, `Akamai`) ne correspondent qu'au titre de la page — faire correspondre le HTML complet produisait auparavant des faux positifs sur de simples références de ressource CDN. Si un faux positif persiste, `--ignorer-waf` dégrade `niveau_confiance` sans forcer `pret_a_agir: false` (`boussole.waf_ignore_actif: true` enregistre la dérogation). La détection est fondée sur des mots-clés et peut produire des faux positifs sur des pages qui discutent légitimement de blocage/détection — traitez-la comme un signal rapide, pas un verdict certain.

---

4. Répertoire chiffré et identifiants

4a. Structure

Les identifiants vivent dans un répertoire chiffré — un volume gocryptfs — contenant un fichier `.json` par domaine.

```
~/Vaults/__PROJET__/Dinoer/
  ├── app.example.com.json         ← identifiants pour https://app.example.com/
  ├── admin.example.com.json       ← identifiants pour https://admin.example.com/
  └── operations.jsonl             ← journal d'opérations (v1.15.0)
```

Format du fichier d'identifiants :

```
{
  "username": "admin@example.com",
  "password": "mon-mot-de-passe"
}
```

Le nom du fichier = `urlparse(url).hostname`. Pour `https://app.example.com/login/`, créez `app.example.com.json`. Le répertoire provient de la clé `secrets_dir` dans le fichier de configuration nommé par `DINOER_CONF` (par défaut `/opt/dinoer/dinoer.conf`) — corrigé le 15/08/2026: `~/.dinoer.conf` est une convention de nommage pour l'endroit où `DINOER_CONF` pointe généralement, et non une étape de repli automatique distincte.

4b. Remplir un formulaire — la règle absolue

INTERDIT — expose le mot de passe dans le shell et `/proc` :

```
PASS=$(jq -r '.password' ~/Vaults/.../file.json)   # JAMAIS
curl -d "password=$PASS" https://...                 # JAMAIS
```

CORRECT — identifiants résolus à l'intérieur de Playwright :

```
{"type": "remplir", "selecteur": "input[name=\"username\"]", "valeur": "depuis_secrets",
 ↪  "secret_cle": "username"},
{"type": "remplir", "selecteur": "input[name=\"password\"]", "valeur": "depuis_secrets",
 ↪  "secret_cle": "password"}
```

Les valeurs ne transitent jamais par le shell, l'historique bash, les journaux de processus, ni aucun fichier.

4c. Choisir le fichier d'identifiants pour une exécution

```
# Répertoire d'identifiants par défaut (défini dans dinoer.conf > secrets_dir)
/opt/dinoer/venv/bin/python /opt/dinoer/shot.py --url https://target.local/ --a11y

# Fichier d'identifiants explicite (--secrets)
/opt/dinoer/venv/bin/python /opt/dinoer/shot.py \
  --url https://target.local/ --a11y \
```

```
  --secrets /path/to/mounted/directory/creds.json

# Répertoire d'identifiants par projet via .dinoer.conf
export DINOER_CONF=~/git/MyProject/.dinoer.conf
/opt/dinoer/venv/bin/python /opt/dinoer/shot.py --url https://target.local/ --a11y
```

Contenu de `~/git/MyProject/.dinoer.conf` :

```
{"secrets_dir": "../MyProject-secrets"}
```

Le chemin est résolu par rapport à l'emplacement de `.dinoer.conf`.

Contenu du fichier `--secrets` — `origines_autorisees` obligatoire depuis le 05/08/2026 (changement cassant, sans période de compatibilité) : un fichier auquel il manque cette clé est refusé avant toute lecture.

```
{"username": "operator", "password": "secret", "origines_autorisees": ["target.local"]}
```

`origines_autorisees` liste les hostnames contre lesquels ce fichier peut être utilisé — même format en minuscules, sans schéma, sans port que `domaine_depuis_url()`. Une lecture contre une page dont le domaine n'est pas dans la liste est refusée (`SecretsOrigineNonAutoriseeError`).

4d. TOTP / MFA

Deux chemins effectifs, tous deux résolus à l'intérieur de Playwright (jamais un code saisi) :

```
{"type": "remplir", "selecteur": "input[name=otp]", "valeur": "depuis_secrets_totp"}
```

Lit la clé `totp_cle` (graine base32) du fichier d'identifiants et calcule le code TOTP courant.

Pour recevoir le code via ntfy (flux sans intervention humaine) :

```
{"type": "attendre_mfa_ntfy", "selecteur": "input[name=otp]", "timeout": 120}
```

`selecteur` est le sélecteur CSS du champ OTP. L'URL de base ntfy vient de `DINOER_NTFY_URL` (environnement) ou de la clé `ntfy.url` de `dinoer.conf`.

4e. Somme de contrôle d'intégrité (opt-in, v1.15.0)

Pour protéger un fichier d'identifiants contre une corruption FUSE silencieuse, ajoutez un champ `checksum` :

```
# Générer la somme de contrôle — corrigé le 15/08/2026, vérifié contre
# lib/repertoire_chiffre.py:32 (_CHAMPS_CHECKSUM) : la somme couvre les
# quatre champs présents, pas seulement username/password.
/opt/dinoer/venv/bin/python -c "
import json, hashlib
creds = json.load(open('mes_identifiants.json'))
champs = ('username', 'password', 'totp_cle', 'origines_autorisees')
fields = {k: creds[k] for k in sorted(champs) if k in creds}
print('sha256:' + hashlib.sha256(json.dumps(fields, sort_keys=True).encode()).hexdigest())
"
```

Ajoutez la valeur renvoyée au fichier d'identifiants :

```
{
  "username": "admin@example.com",
  "password": "mon-mot-de-passe",
  "checksum": "sha256:a3f2c1..."
}
```

Si la somme de contrôle ne correspond pas, `shot.py` lève une `SecretsChecksumError` (code de sortie 42) avec un message explicite. Sans la clé `checksum` : comportement inchangé (opt-in strict).

4f. Répertoire chiffré fermé — que faire

```
SecretsFermesError: Le répertoire chiffré Dinoer est initialisé mais non monté.
```

```
# Monter le répertoire chiffré
bash ~/git/Dinoer/Dinoer/scripts/monter-repertoire-chiffre.sh

# Vérifier le montage
ls ~/Vaults/__PROJET__/Dinoer/
# → doit montrer des fichiers JSON
```

4g. HTTP Basic Auth — `--http-credentials` (v1.21.0)

Pour les cibles derrière un défi HTTP Basic Auth au niveau réseau (RFC 7617) — le mur qu'un reverse proxy comme Caddy, nginx, ou Traefik dresse avant qu'aucune page ne se rende, courant devant les interfaces d'administration auto-hébergées. C'est un mécanisme différent de l'authentification par formulaire décrite ci-dessus (4a-4f), qui reste entièrement supportée et non affectée.

```
/opt/dinoer/venv/bin/python /opt/dinoer/shot.py \
  --url https://internal.example/ \
  --http-credentials --secrets ~/Vaults/__PROJET__/Dinoer/internal_example.json
```

Fichier d'identifiants — la paire `username`/`password` déjà utilisée pour le cas courant (un seul jeu d'identifiants pour la cible) :

```
{"username": "admin", "password": "mon-mot-de-passe"}
```

Les clés dédiées `http_username`/`http_password` sont essayées en premier et ne sont nécessaires que quand la même cible a *à la fois* un mur Basic Auth au niveau réseau *et* sa propre connexion applicative séparée (deux paires d'identifiants différentes dans le même fichier) — Dinoer se replie automatiquement sur `username`/`password` quand les clés dédiées sont absentes.

Confirmé en production contre une cible réelle protégée par Caddy : le comportement par défaut sûr (`send: "unauthorized"` — les identifiants ne sont envoyés qu'après un vrai 401, jamais préventivement) a résolu le défi dès la première tentative. `boussole.http_credentials_actif: true` confirme un vrai succès, pas seulement le fait que le drapeau a été passé ; `boussole.http_auth_requise: true` signale un 401 non résolu, distinct d'un blocage WAF.

---

5. Écrire et exécuter un scénario RPA

5a. Protocole en 3 étapes

Étape 1 — explorer la page (lecture seule)

```
# Vue rapide — arbre d'accessibilité
/opt/dinoer/venv/bin/python /opt/dinoer/shot.py \
  --url https://target.local/ --a11y

# Lecture complète — arbre + texte nettoyé
/opt/dinoer/venv/bin/python /opt/dinoer/shot.py \
  --url https://target.local/ --a11y \
  --action '{"type": "extraire_texte"}'

# Inventaire DOM enrichi (frameworks, attributs data stables)
/opt/dinoer/venv/bin/python /opt/dinoer/rpa.py \
  --scenario /opt/dinoer/scenarios/diagnostic_dom.json \
  --url https://target.local/
```

Ce qu'il faut noter : - attributs stables : `name`, `id`, `aria-label`, `data-testid` - overlays bloquants (bannières cookies, modales) - SPA ou rechargement HTTP complet

Étape 2 — écrire le scénario

```
{
```

```
  "nom": "login_app",
  "url": "https://app.example.com/login/",
  "intention": "Connexion administrateur avec identifiants stockés",
  "actions": [
    {"type": "nettoyer_overlay", "selecteur": ".cookie-banner"},
    {"type": "remplir", "selecteur": "input[name=\"username\"]", "valeur": "depuis_
    ↪  secrets", "secret_cle": "username"},
    {"type": "remplir", "selecteur": "input[name=\"password\"]", "valeur": "depuis_
    ↪  secrets", "secret_cle": "password"},
    {"type": "cliquer", "selecteur": "button[type=submit]"},
    {"type": "attendre_selecteur_present", "selecteur": ".user-avatar"}
  ]
}
```

Étape 3 — exécuter

```
/opt/dinoer/venv/bin/python /opt/dinoer/rpa.py \
  --scenario /opt/dinoer/scenarios/login_app.json
```

5b. Scénario complet : se connecter et naviguer entre les pages

```
{
  "nom": "audit_pages",
  "url": "https://app.example.com/login/",
  "intention": "Lecture après déploiement",
  "actions": [
    {"type": "remplir", "selecteur": "input[name=\"username\"]", "valeur": "depuis_
    ↪  secrets", "secret_cle": "username"},
    {"type": "remplir", "selecteur": "input[name=\"password\"]", "valeur": "depuis_
    ↪  secrets", "secret_cle": "password"},
    {"type": "cliquer", "selecteur": "button[type=submit]"},
    {"type": "attendre_selecteur_present", "selecteur": ".dashboard-main"},
    {"type": "naviguer", "url": "https://app.example.com/settings/"},
    {"type": "attendre_navigation"},
    {"type": "evaluer", "script": "document.title", "contient": "Settings"},
    {"type": "naviguer", "url": "https://app.example.com/users/"},
    {"type": "attendre_navigation"},
    {"type": "evaluer", "script": "document.querySelectorAll('.user-row').length",
    ↪  "attendu": 12}
  ]
}
```

5c. Extraire des données du DOM

```
{
  "nom": "extract_counters",
  "url": "https://app.example.com/dashboard/",
  "actions": [
    {"type": "evaluer", "script": "document.title"},
    {"type": "evaluer", "script": "document.querySelectorAll('.user-row').length"},
    {"type": "evaluer", "script": "window.location.href"}
  ]
}
```

Résultat dans `evaluations[]` :

```
"evaluations": [
  {"index": 0, "script": "document.title", "valeur": "Dashboard — My App"},
  {"index": 1, "script": "...", "valeur": 42},
  {"index": 2, "script": "...", "valeur": "https://app.example.com/dashboard/"}
]
```

5d. Assertions sur evaluer (rpa.py uniquement)

Trois clés mutuellement exclusives — une par action :

```
{"type": "evaluer", "script": "document.querySelectorAll('.row').length", "attendu": 3}
{"type": "evaluer", "script": "document.title", "contient": "Dashboard"}
{"type": "evaluer", "script": "window.location.href", "motif": "/dashboard$"}
```

Clé	Comparaison	Types valides
`attendu`	égalité stricte `==`	str, int, bool
`contient`	sous-chaîne `in`	str seulement
`motif`	`re.search()` Python	str seulement

Si l'assertion échoue : rpa.py s'arrête immédiatement (exit 1) avant toute action mutante ultérieure.

5e. Sous-scénarios (declencher_scenario)

Définir une connexion comme un sous-scénario réutilisable :

```
{
  "nom": "login_app",
  "url": "https://app.example.com/login/",
  "actions": [
    {"type": "remplir", "selecteur": "input[name=\"username\"]", "valeur": "depuis_
    ↪  secrets", "secret_cle": "username"},
    {"type": "remplir", "selecteur": "input[name=\"password\"]", "valeur": "depuis_
    ↪  secrets", "secret_cle": "password"},
    {"type": "cliquer", "selecteur": "button[type=submit]"},
    {"type": "attendre_selecteur_present", "selecteur": ".user-avatar"}
  ]
}
```

Appeler ce sous-scénario depuis un autre scénario :

```
{
  "nom": "full_audit",
  "url": "https://app.example.com/login/",
  "actions": [
    {"type": "declencher_scenario", "scenario": "login_app"},
    {"type": "naviguer", "url": "https://app.example.com/report/"}
  ]
}
```

Profondeur maximale : 5 niveaux d'imbrication. `declencher_scenario` est aplati par `rpa.py` avant que les actions n'atteignent `shot.py`.

5f. Vérifier que vous êtes sur la bonne page avant toute mutation

Ajoutez toujours une garde comme première action dans les scénarios qui suppriment ou modifient :

```
{"type": "evaluer", "script": "window.location.href", "contient": "/dashboard"},
{"type": "evaluer", "script": "document.querySelector('.alert-danger')?.textContent ??
↪  null", "attendu": null}
```

Si la garde échoue : rpa.py s'arrête avant que la suppression ne soit exécutée.

5g. Reprendre une session (cookies persistés)

```
# Première invocation — authentifier et sauvegarder la session
/opt/dinoer/venv/bin/python /opt/dinoer/shot.py \
  --url https://app.example.com/login/ \
  --actions /tmp/login.json \
```

```
  --sauver-session /tmp/dinoer/session.json

# Invocations suivantes — réutiliser la session (pas de reconnexion)
/opt/dinoer/venv/bin/python /opt/dinoer/shot.py \
  --url https://app.example.com/dashboard/ \
  --reprendre-session /tmp/dinoer/session.json
```

Signal de dérive de session : si la session a expiré, `boussole.session_derive: true` dans le JSON. Dans ce cas : relancez la connexion complète sans `--reprendre-session`.

5h. Non-régression structurelle — `--replay-verifier` (v1.17.0)

```
# Première exécution — sauvegarder la référence structurelle
/opt/dinoer/venv/bin/python /opt/dinoer/rpa.py \
  --scenario /opt/dinoer/scenarios/dashboard.json \
  --sauver-verifier-reference /tmp/dashboard.ref.json

# Exécutions suivantes — comparer
/opt/dinoer/venv/bin/python /opt/dinoer/rpa.py \
  --scenario /opt/dinoer/scenarios/dashboard.json \
  --replay-verifier /tmp/dashboard.ref.json
```

Compare `http_status`, `dom_stats`, et les résultats d'`evaluer` à la référence sauvegardée. Verdict sur stderr :

```
{"type_comparaison": "replay_verifier", "verdict": "stable", "diffs": []}
```

Exit 1 sur `verdict: "regression"`, avec `diffs` listant chaque champ divergent (`reference` vs `obtenu`). Les deux drapeaux sont mutuellement exclusifs.

5i. Reprendre un long scénario après un échec — `--checkpoint` (v1.17.0)

```
/opt/dinoer/venv/bin/python /opt/dinoer/rpa.py \
  --scenario /opt/dinoer/scenarios/long_audit.json \
  --checkpoint /tmp/long_audit.checkpoint.json
```

Si le scénario échoue en cours de route, le fichier de checkpoint est écrit avec le compte d'actions terminées et un fichier de session. **Relancez exactement la même commande** pour reprendre : les actions déjà terminées sont sautées. En cas de succès complet, le fichier de checkpoint est supprimé automatiquement.

Une exécution arrêtée par un plafond de navigation (`max_actions_par_run`/`max_pages_par_run`) est traitée de la même façon qu'un échec partiel depuis la v1.17.2 — le checkpoint est mis à jour avec la progression réelle, pas supprimé.

L'état du DOM (modales ouvertes, formulaires à moitié remplis) n'est jamais préservé lors d'une reprise — seuls les cookies/`localStorage` et la position dans la liste d'actions le sont. Ne comptez pas sur `--checkpoint` pour reprendre au milieu d'un formulaire multi-étapes unique ; il ne reprend qu'aux frontières entre actions.

5j. Cibler des éléments à l'intérieur d'une iframe (v1.17.0)

Aucune numérotation d'éléments n'existe à l'intérieur d'une iframe (même origine ou cross-origin) — ciblez-la directement par sélecteur CSS :

```
{"type": "cliquer_iframe", "iframe_selecteur": "iframe#paiement", "selecteur":
 ↪  "button.valider"},
{"type": "remplir_iframe", "iframe_selecteur": "iframe#paiement", "selecteur":
 ↪  "input[name=cvv]", "valeur": "depuis_secrets", "secret_cle": "cvv"}
```

`remplir_iframe` prend en charge `valeur: "depuis_secrets"` exactement comme `remplir` (section 4b) — jamais un identifiant en clair dans le scénario. Si l'élément cible refuse l'interaction (par exemple une

région `contenteditable` en état lecture seule), ajoutez `"force": true` à `cliquer_iframe` — mêmes sémantiques que `cliquer` (section 7e).

Pour trouver le sélecteur interne : utilisez `evaluer` sur le contenu de l'iframe s'il est de même origine (`document.querySelector('iframe').contentDocument...`), ou consultez le balisage/la documentation propre de l'application cible si cross-origin.

5k. Iframes imbriquées — `iframe_chemin` (v1.18.0)

Une iframe à l'intérieur d'une autre iframe : remplacez `iframe_selecteur` par `iframe_chemin`, un tableau ordonné — un sélecteur CSS par niveau d'imbrication, du plus externe au plus interne.

```
{"type": "cliquer_iframe", "iframe_chemin": ["iframe#wrapper", "iframe#paiement"],
 ↪  "selecteur": "button.valider"},
{"type": "remplir_iframe", "iframe_chemin": ["iframe#wrapper", "iframe#paiement"],
 ↪  "selecteur": "input[name=cvv]", "valeur": "depuis_secrets", "secret_cle": "cvv"}
```

`iframe_selecteur` (une seule frame) et `iframe_chemin` (descente imbriquée) sont mutuellement exclusifs — exactement un requis par action. Pour une iframe à un seul niveau, continuez d'utiliser `iframe_selecteur` (section 5j).

---

6. Actions — référence complète

Type	Paramètres requis	Paramètres optionnels	Remarques
`naviguer`	`url`	—	rechargement HTTP complet. Compté dans `respect.pages_visitees`
`cliquer`	`selecteur`	`force` (bool), `repli_js` (bool)	`force: true` contourne les éléments cachés en CSS ou un showModal. `repli_js: true` retente via JS si le clic natif échoue encore (v1.22.0) — rejeté avec `--no-evaluer` (exit 2, avant le lancement)
`remplir`	`selecteur`, `valeur`	`secret_cle`	`valeur: "depuis_secrets"` requiert `secret_cle` ; `"depuis_secrets_totp"` pour le TOTP
`evaluer`	`script`	`attendu`, `contient`, `motif`	JS exécuté dans le navigateur. Assertions pour rpa.py uniquement
`defiler`	`px` ou `selecteur`	—	défilement vertical en pixels (`px`) ou jusqu'à un élément (`selecteur`)
`pause`	`ms`	—	délai fixe en ms. Préférez `attendre_selecteur_present` pour les signaux DOM

Type	Paramètres requis	Paramètres optionnels	Remarques
`attendre`	`selecteur`	—	attend que le sélecteur CSS devienne visible (`state=visible`, défaut Playwright — corrigé le 16/08/2026, identique à `attendre_selecteur_present`)
`attendre_navigation`	—	—	attend `networkidle` (fin des requêtes réseau)
`attendre_url`	`motif`	`attendre_changement` (bool)	correspondance de sous-chaîne d'URL. `attendre_changement: true` attend d'abord une vraie navigation (voir le piège FR-55)
`attendre_selecteur_present`	`selecteur`	—	attend que l'élément soit visible (`state=visible`)
`attendre_absence`	`selecteur`	`delai_initial_ms`	attend la disparition de l'élément du DOM (`state=detached`)
`attendre_reseau_calme`	—	`timeout_ms`	500 ms de silence réseau. `timeout_ms` : durée maximale avant d'abandonner
`attendre_mfa_ntfy`	`selecteur`	`timeout`	attend un code TOTP via ntfy, le remplit dans le champ
`nettoyer_overlay`	`selecteur`	—	masque les overlays bloquants (bannière cookies, modale) — sélecteur explicite, aucune détection automatique
`declencher_scenario`	`scenario`	—	intègre les actions d'un sous-scénario. Profondeur max : 5 (rpa.py)
`extraire_texte`	—	—	texte de page nettoyé depuis le DOM rendu — `extraction_texte` (`titre`, `texte`, `url`, `date_capture`)
`cliquer_iframe`	`iframe_selecteur` | `iframe_chemin`, `selecteur`	`force` (bool)	clic à l'intérieur d'une iframe (v1.17.0). `iframe_chemin` pour les iframes imbriquées (v1.18.0, section 5k)
`remplir_iframe`	`iframe_selecteur` | `iframe_chemin`, `selecteur`, `valeur`	`secret_cle`	remplissage à l'intérieur d'une iframe (v1.17.0). `valeur: "depuis_secrets"` pris en charge

---

7. Gérer les obstacles courants

7a. Bannière de cookies / overlay bloquant

```
{"type": "nettoyer_overlay", "selecteur": ".cookie-consent-banner, #gdpr-overlay"}
```

Placez-la **avant** toute autre action de lecture/interaction. L'overlay masque des éléments dans l'arbre d'accessibilité.

7b. Élément hors du viewport

Défilez jusqu'à lui (par quantité ou par sélecteur), puis agissez :

```
{"type": "defiler", "selecteur": "#le-bouton"},
{"type": "cliquer", "selecteur": "#le-bouton"}
```

ou

```
{"type": "defiler", "px": 600},
{"type": "cliquer", "selecteur": "button[data-testid='load-more']"}
```

7c. SPA (React, Vue, Angular) — naviguer sans rechargement

Après un clic qui change la vue dans une SPA, Playwright ne sait pas quand la navigation est terminée.

```
{"type": "cliquer", "selecteur": "a[href*='/dashboard']"},
{"type": "attendre_url", "motif": "/dashboard"},
{"type": "evaluer", "script": "document.title", "contient": "Dashboard"}
```

Ne présumez jamais qu'un clic a terminé une navigation sans signal DOM. Après un submit, associez `attendre_url` avec `attendre_selecteur_present` (piège de correspondance partielle, voir `docs/GUIDE_LLM_INTERACTIONS.md`).

7d. Dialogue CSS ou showModal()

`TimeoutError` sur `cliquer` alors que l'élément est visible dans le DOM = élément caché en CSS ou à l'intérieur d'un dialogue.

```
{"type": "cliquer", "selecteur": "#dialog-confirm button[type=submit]", "force": true}
```

Si `force: true` est insuffisant (erreur d'interactivité/obstruction) : ajoutez `repli_js: true` à la même action (v1.22.0), ou repliez-vous sur du JS :

```
{"type": "evaluer", "script": "document.querySelector('#dialog-confirm
↪  button[type=submit]').click()"}
```

7e. Opération longue (spinner, tâche par lot)

N'utilisez pas `pause` pour attendre une durée fixe. Attendez le signal DOM :

```
{"type": "cliquer", "selecteur": "button[data-testid='run-job']"},
{"type": "attendre_absence", "selecteur": ".spinner", "delai_initial_ms": 500},
{"type": "attendre_selecteur_present", "selecteur": ".result-container"}
```

Si l'opération ne fournit aucun signal DOM, sondez l'état avec `evaluer` et continuez quand la preuve est présente.

7f. Plafond atteint (v1.15.0)

Si `respect.plafond_atteint` est présent dans la sortie, l'exécution s'est arrêtée avant que le scénario ne se termine. Les actions restantes n'ont pas été exécutées.

Options : 1. augmenter `max_pages_par_run` ou `max_actions_par_run` dans `dinoer.conf` 2. découper le scénario en plusieurs exécutions 3. reprendre une section partielle avec `--checkpoint`

7g. Champ de formulaire `<select>`

`remplir` (`.fill()`) ne fonctionne pas sur `<select>`. Utilisez un setter JS via `evaluer` :

```
{"type": "evaluer", "script": "(() => { const s =
↪  document.querySelector('select[name=role]'); s.value='admin'; s.dispatchEvent(new
↪  Event('change',{bubbles:true})); })()"}
```

7h. Site bloqué par un WAF (403 immédiat)

```
# Essayer avec stealth
/opt/dinoer/venv/bin/python /opt/dinoer/shot.py \
  --url https://target.local/ --a11y --stealth
```

Si le 403 persiste avec `--stealth` : le site utilise du fingerprinting TLS (JA3/JA4) ou une analyse comportementale avancée (Cloudflare Enterprise). `playwright-stealth` ne contourne pas ces protections. Voir `docs/RETOUR_EXPERIENCE.md` FR-77/FR-78/FR-79 pour le contexte.

Dinoer signale aussi un blocage probable de façon passive — voir la section 3e (`respect.waf_bloquants`).

7i. La navigation initiale ne se termine jamais — `--wait-until` (v1.22.0)

Symptôme : `TimeoutError` sur la navigation initiale, et augmenter `--timeout` ne change rien (45 s échoue exactement comme 10 s). Cause : par défaut Dinoer attend `networkidle` — 500 ms de silence réseau. Une page qui interroge en continu (statistiques en direct, compteurs auto-rafraîchis, panneaux d'administration de routeur) ne produit jamais ce silence, donc aucune valeur de timeout ne peut jamais être assez grande.

```
# shot.py — reconnaissance directe
/opt/dinoer/venv/bin/python /opt/dinoer/shot.py \
  --url http://target.local/ --wait-until load --a11y

# rpa.py — propagé à shot.py, donc les scénarios atteignent les mêmes cibles
/opt/dinoer/venv/bin/python /opt/dinoer/rpa.py \
  --scenario ./admin_login.json --wait-until load
```

Un scénario peut le porter comme propriété racine à la place, en restant autonome :

```
{"url": "http://target.local/", "wait_until": "load", "actions": [...]}
```

Le drapeau CLI a priorité sur la propriété de scénario.

Valeur	Attend	À utiliser quand
`networkidle`	500 ms de silence réseau	par défaut — conservez-la sauf échec
`load`	événement `load` (page et sous-ressources)	interrogation continue / statistiques en direct
`domcontentloaded`	HTML parsé, sous-ressources encore en attente	page très lourde, le DOM suffit

S'applique uniquement à la navigation initiale — l'action `naviguer` n'est pas affectée. `boussole.wait_until` rapporte la valeur seulement quand elle diffère de la valeur par défaut.

---

8. Surveillance — vérifications structurelles

Aucune surveillance basée sur l'image n'existe dans Dinoer (aucun diff visuel). Les vérifications structurelles sont textuelles et adaptées à la CI.

8a. Surveillance structurelle continue — `scripts/monitor-verifier.sh` (v1.18.0)

Surveille la *structure* (`http_status`, `dom_stats`, `evaluations`) — aucune image, aucun appel de LLM, construit sur `--replay-verifier` (section 5h).

```
# Premier lancement : créer la référence structurelle.
/opt/dinoer/venv/bin/python /opt/dinoer/rpa.py \
  --scenario /opt/dinoer/scenarios/example_login.json \
  --sauver-verifier-reference /opt/dinoer/references/example_login.ref.json
```

```
# Un script de vérification et d'alerte — ce n'est pas un démon, exécutez-le à plusieurs
 ↪  reprises via cron.
# Sur le canal git-clone, les fichiers scripts/*.sh ne sont jamais déployés vers
 ↪  /opt/dinoer/.
# de sorte qu'il s'exécute à partir du code source Git, en tant que votre propre
 ↪  utilisateur. Sur le canal .deb,
# les trois scripts compressés (monter/démonter-répertoire-chiffré,
# monitor-vérificateur) SONT installés sous /opt/dinoer/scripts/ — corrigé.
# 15/08/2026, ce commentaire date de l'époque précédant la véritable création de cette
 ↪  chaîne.
bash ~/git/Dinoer/Dinoer/scripts/monitor-verifier.sh \
  --scenario /opt/dinoer/scenarios/example_login.json \
  --reference /tmp/ref_sillage.json \
  --ntfy-topic dinoer-monitoring

# crontab -e (votre propre crontab)
*/15 * * * * bash ~/git/Dinoer/Dinoer/scripts/monitor-verifier.sh \
  --scenario /opt/dinoer/scenarios/example_login.json \
  --reference /opt/dinoer/references/example_login.ref.json \
  --ntfy-topic dinoer-monitoring \
  >> /var/log/dinoer/cron-structural.jsonl 2>&1
```

Stable ⟶ silence. Régression ⟶ une notification `ntfy` avec le diff. Chaque invocation est un processus isolé — aucun démon, aucun risque de fuite mémoire, et les plafonds de navigation respectueuse se réinitialisent proprement à chaque passe.

Corrigé le 16/08/2026 : le script appelait auparavant `rpa.py --no-capture --replay-verifier`, mais `--no-capture` n'était plus un drapeau de `rpa.py` — chaque exécution réelle échouait au niveau d'argparse. Le drapeau mort a été retiré (Dinoer n'a aucun chemin image par défaut, le retirer ne change rien d'autre).

Nuance sur le verrou de lecture du guide : si invoqué sous un utilisateur OS distinct (par exemple un compte de service système), cet utilisateur a besoin que `--guide-version` soit validé une fois (`~<home>/.config/dinoer/guide_state.json`).

---

9. Journal d'opérations

Le journal est `/var/log/dinoer/operations.jsonl`. Si le chemin de journal configuré se trouve à l'intérieur du répertoire chiffré et qu'il n'est pas monté, les entrées sont redirigées vers un repli local (écriture dégradée, 700/600) plutôt qu'écrites en clair sur l'hôte brut.

```
# Lire les 10 dernières entrées
tail -n 10 /var/log/dinoer/operations.jsonl | python3 -m json.tool

# Filtrer par cible (outil journal.py)
/opt/dinoer/venv/bin/python /opt/dinoer/journal.py \
  --cible app.example.com

# Filtrer les opérations mutantes seulement
/opt/dinoer/venv/bin/python /opt/dinoer/journal.py \
  --cible app.example.com --mutatif

# Depuis une date
/opt/dinoer/venv/bin/python /opt/dinoer/journal.py \
  --cible app.example.com --depuis 2026-07-01

# Exécutions en échec seulement (v1.20.0) — resultat != succes
/opt/dinoer/venv/bin/python /opt/dinoer/journal.py \
  --cible app.example.com --erreurs
```

Champs de chaque entrée :

Champ	Signification
`ts`	Horodatage ISO 8601
`operation_id`	Identifiant de session unique (v1.16.0), nomme le répertoire du fichier temporaire
`outil`	`"shot.py"`, `"rpa.py"` ou `"campagne.py"` — corrigé le 15/08/2026, était documenté comme `mode` (`shot.py`/`rpa.py`), un champ que le code n'a jamais écrit
`version`	Version de Dinoer
`cible_url`	URL cible
`resultat`	`"succes"` ou `"echec"`
`mutatif`	`true` si au moins une action d'écriture a été effectuée
`hostname_executant` / `utilisateur_executant` / `profil_actif`	Boussole d'exécution (hôte, utilisateur du système d'exploitation, profil d'opérateur actif)
`intention`	Étiquette transmise via le champ `--intention` ou le scénario `intention` — présente uniquement si définie
`source_scenario`	Nom du fichier de scénario uniquement, sans chemin (v1.18.0) — présent uniquement en mode RPA. Corrigé le 15/08/2026: précédemment, le tableau listait également un champ `scenario` (chemin complet); le code n'a jamais écrit celui-ci
`chainage`	Liste de `{scenario, profondeur, action_debut, action_fin}` — présente uniquement lorsque `declencher_scenario` a été utilisé
`actions`	Liste d'actions résumée — présente lorsque la session contenait des actions
`actions_raw`	Liste complète d'actions neutralisée — présente uniquement lors d'une exécution réussie avec des actions
`captures`	Référence(s) aux captures structurelles — présente uniquement lorsque l'exécution a produit quelque chose
`erreur`	Présent uniquement en cas d'échec
`respect`	Journal de navigation de la session — présent uniquement si défini
`evaluations`	Valeurs de `{script, valeur_retournee}` nettoyées — présent uniquement si les actions `evaluer` ont été exécutées

Aucun champ `duree_ms` n'existe dans le journal — corrigé le 15/08/2026, cette table en listait un que le code n'a jamais écrit. La mesure du temps par action est `latences_actions`, dans la sortie JSON d'un run, pas dans l'entrée du journal.

9a. Rotation du journal (G-36)

Dinoer ne livre pas de configuration logrotate — `/var/log/dinoer/operations.jsonl` croît sans limite jusqu'à ce que l'administrateur en installe une. `lib/journal.py` ouvre et ferme le fichier à chaque écriture (aucun descripteur de fichier persistant entre les exécutions), spécifiquement pour que le comportement **par défaut** de logrotate (renommer le fichier courant, en créer un nouveau) fonctionne correctement sans aucune option spéciale : la prochaine écriture rouvre le chemin et trouve le nouvel inode.

N'ajoutez pas `copytruncate` à une configuration logrotate pour Dinoer — c'est inutile ici et cela réintroduit une fenêtre de perte d'écriture. Exemple `/etc/logrotate.d/dinoer` :

```
/var/log/dinoer/operations.jsonl {
    weekly
    rotate 8
```

```
    compress
    delaycompress
    missingok
    notifempty
    create 0640 dinoer dinoer
}
```

`journal.py` (le lecteur) suit déjà les fichiers pivotés de façon transparente (`operations.jsonl`, `.1`, `.2.gz`, …) — aucune étape supplémentaire nécessaire après la rotation.

---

10. Options CLI — référence

shot.py

Option	Défaut	Description
`--version`	—	affiche la version installée et quitte immédiatement — pas de Playwright, aucun autre argument requis (v1.18.0)
`--guide-version X.Y`	—	preuve de lecture de `docs/GUIDE_LLM.md` — requis sauf si un marqueur local valide existe déjà (v1.18.0, section 1)
`--url URL`	requis	URL à capturer
`--actions FICHIER`	—	fichier JSON d'actions séquentielles
`--action JSON`	—	action unique en JSON en ligne — à échapper avec précaution, préférez `--actions FICHIER` pour les actions riches en JS
`--attendre-selecteur SEL`	—	attend un sélecteur avant de terminer l'exécution
`--timeout MS`	10000	timeout Playwright par action (ms)
`--wait-until VALEUR`	`networkidle`	`networkidle`|`load`|`domcontentloaded` — navigation initiale seulement (v1.22.0, section 7i)
`--largeur PX`	1280	largeur du viewport
`--hauteur PX`	720	hauteur du viewport
`--a11y`	désactivé	inclut l'arbre d'accessibilité dans le JSON
`--stealth`	désactivé	mode furtif playwright-stealth (v1.15.0)
`--secrets FICHIER`	—	chemin explicite vers un fichier d'identifiants
`--auth-indicator SEL`	—	sélecteur CSS présent seulement en session authentifiée
`--auth-indicator-negative SEL`	—	requiert `--auth-indicator` ; sélecteur CSS présent seulement hors session authentifiée
`--ignorer-waf`	désactivé	un blocage WAF détecté dégrade `niveau_confiance` mais ne force plus `pret_a_agir: false` à lui seul (v1.17.2, section 3e)
`--http-credentials`	désactivé	résout les identifiants HTTP Basic Auth depuis le fichier d'identifiants, cantonné à l'origine de la cible (v1.21.0, section 4g)

Option	Défaut	Description
`--ignore-tls-errors`	désactivé	accepte un TLS invalide sur des cibles LAN/dev contrôlées — jamais sur l'internet public (v1.15.1)
`--no-evaluer`	désactivé	refuse l'action **evaluer** (et `repli_js`) pour toute l'exécution — recommandé contre les formulaires sensibles (v1.15.1)
`--no-filtre-evaluer`	désactivé	désactive la neutralisation sur stdout des valeurs de retour d'**evaluer**, des URLs et des messages d'erreur — exécutions de débogage explicites seulement ; quand désactivé, `boussole.filtre_evaluer_actif: false` est posé (v1.23.0)
`--intention TEXTE`	—	libellé métier enregistré dans le journal
`--sauver-session FICHIER`	—	sauvegarde les cookies après les actions
`--reprendre-session FICHIER`	—	reprend une session sauvegardée
`--source-scenario NOM`	—	interne (plomberie rpa.py pour le journal — pas pour des appels directs)
`--chainage JSON`	—	interne (plomberie rpa.py pour le journal — pas pour des appels directs)

rpa.py

Propage tous les drapeaux pertinents de shot.py, plus :

Option	Description
`--version`	affiche la version installée et quitte immédiatement (v1.18.0)
`--guide-version X.Y`	preuve de lecture de `docs/GUIDE_LLM.md` — vérifiée indépendamment, même règle que shot.py (v1.18.0)
`--scenario FICHIER`	chemin vers un scénario JSON ou YAML (requis)
`--url URL`	surcharge l'URL du scénario sans modifier le fichier
`--stealth`	propagé à shot.py
`--wait-until`	propagé à shot.py (v1.22.0, section 7i)
`--ignorer-waf`	propagé à shot.py (v1.17.2, section 3e)
`--http-credentials`	propagé à shot.py ; également réglable comme propriété racine de scénario `"http_credentials": true` (v1.21.0, section 4g)
`--auth-indicator-negative`	requiert un `auth_indicator` (CLI ou propriété racine de scénario)
`--sauver-verifier-reference FICHIER`	sauvegarde la référence structurelle pour `--replay-verifier` (v1.17.0, section 5h)
`--replay-verifier FICHIER`	compare l'exécution à une référence structurelle, exit 1 en cas de régression (v1.17.0, section 5h)
`--checkpoint FICHIER`	reprend un long scénario après un échec en cours d'exécution (v1.17.0, section 5i)

campagne.py (pipeline de recherche)

Option	Description
`--manifeste FICHIER`	manifeste de campagne (JSON) — requiert `id_campagne` + `cibles`
`--id-campagne ID`	identifiant de campagne (utilisé dans le manifeste et l'extraction)
`--extraire-cible DEMANDE`	extraction ciblée sur un corpus déjà collecté, sans synthèse. Requiert `--id-campagne` ou `--corpus`
`--corpus FICHIER`	chemin direct vers un `collecte.jsonl` — alternative à `--id-campagne` pour `--extraire-cible`
`--format-extraction {json,markdown,html}`	format de sortie de `--extraire-cible` (par défaut : `json`) — ajouté le 15/08/2026
`--desactiver-cache`	contourne le cache de recherche
`--purger-cache`	purge l'intégralité du cache de recherche
`--purger-cache-avant-jours N`	purge les entrées de cache plus anciennes que N jours

Types de cible dans le manifeste : `query`, `url`, `produit`, `table_reference`. Artefacts : le `/var/log/dinoer/operations.jsonl` partagé + un `collecte.jsonl` par campagne. Détail complet : `campagne.py --help`.

---

11. Codes de sortie et sortie JSON

Codes de sortie

Code	Cause	Que faire
0	succès	—
1	erreur Playwright, action échouée, assertion rpa.py, `action_secret_en_clair`	lisez `erreur` dans le JSON. Voir `GUIDE_LLM_INTERACTIONS.md`
1	`guide_non_lu` — `--guide-version` manquant/faux, aucun marqueur valide (v1.18.0)	se déclenche avant le lancement de Playwright. Lisez `docs/GUIDE_LLM.md`, relancez avec `--guide-version X.Y` (section 1)
2	arguments incompatibles, `arguments_incompatibles`, `url_scheme_interdit`, `chemin_sensible_refuse`	lisez `message` — il nomme le conflit
3	module `playwright` introuvable	invoquez via `/opt/dinoer/venv/bin/python`
42	`SecretsFermesError` — répertoire chiffré non monté, ou somme de contrôle invalide	montez-le, ou vérifiez le fichier d'identifiants
43	`SecretsNonConfigureError` — aucun `secrets_dir` configuré	configurez `secrets_dir` dans `dinoer.conf` (pallier un exemple manquant : créez `/opt/dinoer/dinoer.conf`)

Structure de la sortie JSON

```
{
  "succes": true,
  "http_status": 200,
  "url_finale": "https://target.local/dashboard",
  "erreurs_js": [],
  "erreurs_console": [],
```

```
  "duree_ms": 2400,
  "horodatage": "2026-07-01T12:00:00+02:00",
  "dom_stats": {"boutons": 14, "inputs": 9, "listes_deroulantes": 2, "formulaires": 1,
   ↪  "liens": 41, "dialogues": 0},
  "a11y_tree": "...",
  "evaluations": [],
  "extraction_texte": null,
  "latences_actions": [
    {"index": 0, "type": "naviguer", "latence_ms": 842},
    {"index": 1, "type": "cliquer", "latence_ms": 63}
  ],
  "respect": {
    "pages_visitees": 0,
    "actions_executees": 3,
    "duree_totale_ms": 2400,
    "indice_agressivite": 0.33
  },
  "etat": {
    "pret_a_agir": true,
    "niveau_confiance": "eleve",
    "raisons": ["aucun signal de friction détecté"]
  },
  "boussole": {
    "utilisateur": "operator",
    "ip_locale": "__IP_LAN__",
    "repertoire": "/opt/dinoer",
    "operation_id": "a1b2c3d4e5f6",
    "url_courante": "https://target.local/dashboard",
    "titre_page": "Dashboard — My App",
    "dernier_code_http": 200,
    "stealth_actif": true,
    "auth_status": "active",
    "respect": { "pages_visitees": 0, "actions_executees": 3, "duree_totale_ms": 2400,
     ↪  "indice_agressivite": 0.33 }
  },
  "dinoer_meta": {
    "version_shot": "1.0.1",
    "horodatage_iso": "2026-08-12T14:23:11+02:00",
    "hostname_executant": "operator-host",
    "utilisateur_executant": "operator",
    "profil_actif": "operateur.exemple.yaml",
    "url_au_moment_capture": "https://target.local/dashboard"
  }
}
```


`operation_id` (v1.16.0) est toujours présent et identifie cette exécution de façon unique — il correspond au champ `operation_id` de l'entrée de cette exécution dans le journal d'opérations (section 9), et nomme le répertoire de preuves `preuves/<AAAA-MM>/<operation_id>/` quand des captures y sont archivées (corrigé le 15/08/2026, vérifié contre `lib/journal.py` : aucun répertoire `/tmp/dinoer/<operation_id>/` n'existe nulle part dans le code — `/tmp/dinoer/` ne contient jamais que le fichier de repli du journal d'opérations). `etat` (v1.16.0) est présent uniquement sur le chemin de succès. `latences_actions` (v1.20.0) est toujours présent (liste vide si aucune action), une entrée par action réellement exécutée — voir `GUIDE_LLM_MONITORING.md` pour la façon dont il complète `respect.duree_totale_ms`.

Clés conditionnelles (absentes si inactives) : `dom_stats`, `a11y_tree`, `evaluations`, `extraction_texte`, `auth_status`, `stealth_actif`, `session_derive`, `respect.plafond_atteint`, `respect.waf_bloquants`, `respect.indice_agressivite` (présent dès qu'au moins une action a été exécutée), `boussole.repli_js_utilise`, `boussole.wait_until`, `boussole.http_credentials_actif`, `boussole.http_auth_requise`, `boussole.tls_errors_ignored`, `boussole.waf_ignore_actif`, `bous-`

`sole.filtre_evaluer_actif`, `boussole.champs_rediges`, `actions_executees_avant_echec`, `pages_visitees_avant_echec` (JSON d'échec seulement, v1.17.0). Voir `GUIDE_LLM_MONITORING.md` pour la table d'activation exhaustive.

Erreur — format

```
{
  "succes": false,
  "erreur": "secrets_fermes",
  "message": "Le répertoire chiffré Dinoer est initialisé mais non monté.",
  "code_sortie_recommande": 42,
  "boussole": { "url_courante": "", "titre_page": "" }
}
```

Chemins de référence

Chemin	Rôle
`/opt/dinoer/`	installation de production
`/opt/dinoer/venv/bin/python`	Python à utiliser pour chaque invocation
`/opt/dinoer/dinoer.conf`	configuration machine (secrets_dir, navigation, ntfy)
`/opt/dinoer/scenarios/`	scénarios RPA (dont `diagnostic_dom.json`)
`/opt/dinoer/docs/`	documentation
`/opt/dinoer/references/`	références `--sauver-verifier-reference` / replay
`/tmp/dinoer/`	fichiers de session (`--sauver-session`/`--reprendre-session`) et fichier de repli du journal — corrigé le 15/08/2026 : aucun sous-répertoire par `operation_id` n'existe ici, voir la ligne suivante
`/var/log/dinoer/preuves/<AAAA-MM>/<operation_id>/`	répertoire de preuves pour une exécution, isolé par `operation_id` (v1.16.0), créé uniquement quand des captures sont archivées
`~/Vaults/__PROJET__/Dinoer/`	identifiants + journal (volume gocryptfs)
`~/git/Dinoer/Dinoer/`	sources git (éditez ici, puis `deploy.sh`)
`/var/log/dinoer/operations.jsonl`	journal d'opérations persistant (`journal.py`)

Déployer après modification des sources :

```
bash ~/git/Dinoer/Dinoer/scripts/deploy.sh
```