Diwall — Documentation

Visual perception and RPA for LLM agents

Release 1.24.4

2026-09-26

Canonical version. Translations available under docs/<language>/.

About this compilation

This document gathers four texts that also exist on their own, each written for a different reader. The *cheat sheet* opens on the commands themselves, for whoever needs one right now. The *overview* introduces Diwall and installs it. The *operator guide* explains what you delegate and why the tool behaves as it does. The *operational manual* is the reference: commands, actions, flags, exit codes. Each opens with its own introduction, because each is also read alone.

Contents

Diwall — cheat sheet

Version 1.24.4 — September 2026

Everything on one page. Full reference: `docs/MANUEL.md`.

Three commands

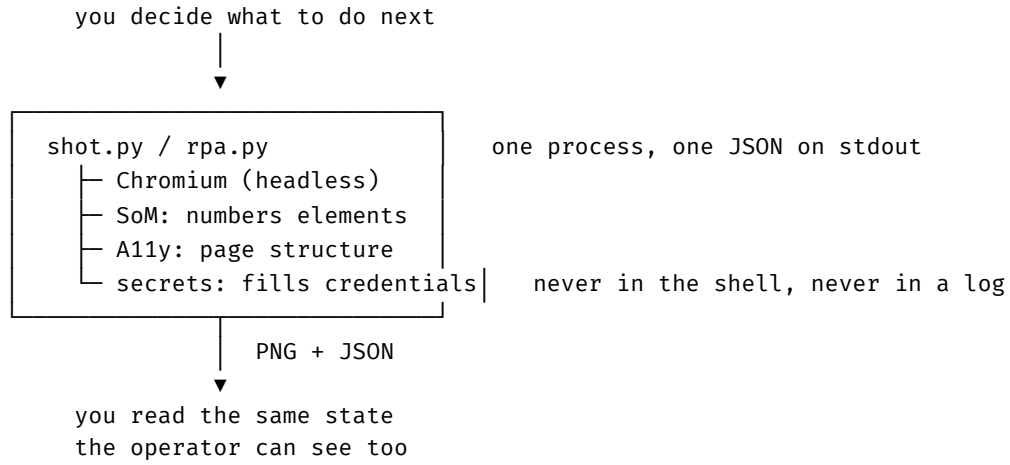
```
# See a page: PNG + numbered elements + accessibility tree
/opt/diwall/venv/bin/python3 /opt/diwall/shot.py --url URL --som --a11y

# Read without capturing (~2 s faster, no PNG)
/opt/diwall/venv/bin/python3 /opt/diwall/shot.py --url URL --mode fast

# Run a scenario
/opt/diwall/venv/bin/python3 /opt/diwall/rpa.py --scenario FILE.json
```

Installed from the `.deb`? Use `diwall-shot` and `diwall-rpa` instead of the full paths. First call on a machine needs `--guide-version X.Y`, read with `grep notice-version /opt/diwall/docs/GUIDE_LLM.md`.

The loop



Session state lives in a file, not in the process: a second call with `--reprendre-session` reuses cookies — never DOM state.

Read the output in this order

Read	Tells you
`succes`	did the run complete
`boussole.url_courante`	where you actually ended up
`boussole.dernier_code_http`	last navigation status
`etat.pret_a_agir` + `etat.raisons`	frictions perceived — a report, never a gate
`capture_som` / `elements_som`	what to click, and its number
`respect`	your own footprint: pages, actions, duration

If `boussole` does not match your expectation, stop before any mutating action.

Every action

`type` is always required. Keys below are the additional ones.

Action	Required	Optional
`naviguer`	`url`	—
`cliquer`	`selecteur`	`force`, `repli_js`
`cliquer_som`	`id`	—
`cliquer_visuel`	`description`	—
`cliquer_iframe`	`iframe_selecteur` | `iframe_chemin`, `selecteur`	`force`
`remplir`	`selecteur`, `valeur`	`secret_cle`
`remplir_som`	`id`, `valeur`	`secret_cle`
`remplir_iframe`	`iframe_selecteur` | `iframe_chemin`, `selecteur`, `valeur`	`secret_cle`
`capturer`	`nom`	`som`
`evaluer`	`script`	`attendu` | `contient` | `motif`
`defiler`	`px` | `selecteur`	—
`pause`	`ms`	`interval_capture`
`attendre`	`selecteur`	`interval_capture`

Action	Required	Optional
`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`	`id_som`	`timeout`
`nettoyer_overlay`	`selecteur`	—
`declencher_scenario`	`scenario`	—

Credentials — the only correct form

```
{"type": "remplir_som", "id": 3, "valeur": "depuis_secrets", "secret_cle": "password"}
```

Never extract a secret into the shell. `lib/repertoire_chiffre.py` resolves it inside the Playwright process; the value never reaches your command line, your history, or any log. `depuis_secrets_totp` does the same for a TOTP code.

When something resists

Symptom	Try
Click times out, element visually hidden	`"force": true`, then `"repli_js": true`
Element not numbered by SoM	`--shadow-dom` (open Shadow Roots)
Element below the fold	`defiler` first — check `boussole.som_hors_viewport`
Page never finishes loading	`--wait-until load`
Submit does nothing, no error	native HTML validation — submit the form via `evaluer`
`exit 42`	encrypted directory not mounted: `diwall-monter-secrets`
`exit 43`	no `diwall.conf` — copy the sample next to it
`guide_non_lu`	pass `--guide-version` once
403 / 429	read `respect.waf_bloquants` — a signal, not an exception

Exit codes

`0` success · `1` Playwright error or failed assertion · `2` viewport mismatch (`watch.py`) · `3` wrong interpreter, use the venv · `42` encrypted directory closed or bad checksum · `43` `diwall.conf` missing.

Diwall — Shared Visual Reference between Human and LLM

> **For the human operator**: Diwall lets you delegate visual verification to your LLM. Both of you see the same capture — you stop having to take its word for it.
>
> **For the LLM**: `docs/GUIDE_LLM.md` is your operational reference. Start there. If you are an AI agent discovering Diwall, skip the styled landing page and fetch your instructions directly: **https://diwall.davalan.fr/instructions.md**

What is Diwall?

Diwall creates a **shared visual reference** between a human operator and a language model. It gives the LLM the ability to **see web interfaces** — and gives the human operator a way to **delegate visual verification** without losing control.

Without Diwall, a human must either trust their LLM on its word or verify the result themselves. With Diwall, both parties see the same PNG capture and the same accessibility tree. The doubt disappears on both sides.

```
LLM acts → Diwall captures → LLM sees and reports → Human verifies from the same state
```

What the human gains: delegation of the anxious, repetitive work of visual verification. Instead of clicking through dozens of pages after a deployment, the human reviews the captures the LLM already produced.

What the LLM gains: real perception of the interface. Without Diwall, a model developing a web application modifies code but cannot see the result in a browser. `lynx` does not render modern interfaces.

What the model actually receives

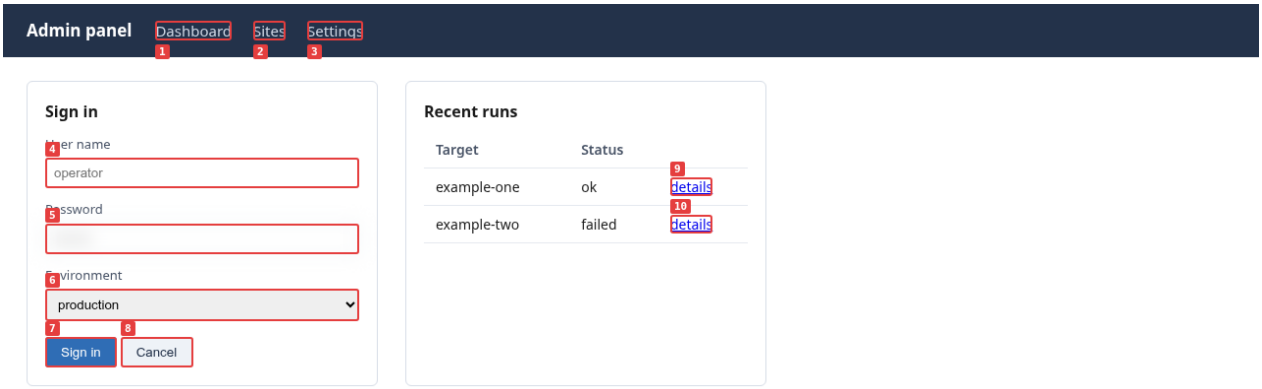


Figure 1: Set-of-Mark capture: every interactive element numbered on the rendered page

This is a real `--som` capture, not a mock-up. Every interactive element is numbered on the rendered page, and the same numbers come back in the JSON — so `{"type": "cliquer_som", "id": 7}` clicks *Sign in*, with no selector to guess and no ambiguity about which button was meant. Reproduce it yourself — the page is a fixture versioned in this repository, so you get the same numbers we did:

```
cd scenarios/interoperabilite/fixture && python3 -m http.server 8765 &
diwall-shot --url http://127.0.0.1:8765/demo_som_en.html --som --guide-version 1.3
```

`elements_som` comes back with `{"id": 7, "tag": "BUTTON", "texte": "Sign in"}`.

Architecture

```
Language Model (brain — ReAct loop)
        ↓  calls
  shot.py (hands — Playwright executor)
        ↓
  Chromium headless → PNG capture
        ↓
  Language Model reads PNG directly (multimodal)
```

`shot.py` has no intelligence. It executes instructions and returns state. The language model decides what to do next.

Capabilities

Feature	Description
Capture	Screenshot any web page
Actions	Fill forms, click, navigate
Set-of-Mark (SoM)	Number all interactive elements for precise DOM clicks
Accessibility snapshot	Extract semantic page structure (A11y tree)
Session persistence	Maintain login state across multi-step ReAct loops
RPA scenarios	Execute action sequences from JSON files
Visual monitoring	Detect if a page changed since last reference
Pixel diff	Quantitative, deterministic diff against a stored reference (v1.2)
Credential resolution	Secure credential injection — never in plaintext, never on the command line
Encrypted directory	gocryptfs volume — `SecretsFermesError` (exit 42) if it is not mounted (v1.5)
Scroll	`defiler` action — relative pixel scroll or `scrollIntoView` by CSS selector (v1.6)
Off-screen warning	`som_hors_viewport` count in JSON when interactive elements exist below the fold (v1.6)
Procedural memory	Successful runs stored as replayable skills via `journal.py --exporter-skill` (v1.6)
TOTP 2FA	Google Authenticator / Authy codes generated at runtime from a stored seed (v1.6)
Async MFA via ntfy	SMS/email 2FA codes received asynchronously via ntfy push notification (v1.6)
Operator profile	YAML profile to lift repetitive administrative confirmations (v1.3)
Model traceability	Every run records which models were called, including Ollama digest (v1.3)
Operation log	Persistent append-only log of all runs — who did what, where, when (v1.4)
Shadow DOM traversal	`--shadow-dom` numbers interactive elements inside open Shadow Roots — Angular, Lit, Stencil, FAST (v1.13.0)
Respectful Navigation	`--stealth` (removes automatic headless markers), courtesy delays and hard caps (`min_action_delay_ms`, `max_pages_par_run`, `max_actions_par_run`), impact metrics (`respect`) reported on every run (v1.15.0)
Deterministic verdict	`etat` object (`pret_a_agir`, `niveau_confiance`, `raisons`) synthesizes authentication, session drift, and friction signals into one read (v1.16.0)
Unified run identity	`operation_id` isolates every run's temporary files and ties them to its operations-log entry (v1.16.0)
Passive WAF signal	`respect.waf_bloquants` flags a likely block (HTTP 403/429 or known keywords) as a non-fatal signal, never an exception (v1.16.0)
Structural non-regression	`--replay-verifier` compares HTTP status, DOM stats, and `evaluer` results against a saved reference — no pixels, no vision model (v1.17.0)
Scenario checkpoints	`--checkpoint` resumes a long scenario after a mid-run failure without replaying completed actions (v1.17.0)

Feature	Description
Hybrid SoM identity	`cliquer_som`/`remplir_som` resolve by the `data-dw-som-id` marker when one exists, fall back to live re-indexing otherwise, and report the path taken plus any stable/raw divergence in `boussole.respect.som_resolution` / `som_derive_detectee` — never silent (v1.24.0; `--som-brut` forces pure re-indexing)
Cross-origin iframes	`cliquer_iframe` / `remplir_iframe` target elements inside same- or cross-origin iframes via Playwright's native frame API (v1.17.0)
Nested iframes	`iframe_chemin` (array) descends iframe-inside-iframe, mutually exclusive with `iframe_selecteur` (v1.18.0)
Guide-read lock	`shot.py`/`rpa.py`/`watch.py` refuse to run without proof `docs/GUIDE_LLM.md` was read — a local marker persists it per machine/user (v1.18.0)
Configuration advice	`mode_conseille` recommends `--mode`/`--shadow-dom` from real prior diagnostic runs on the same host — never a guess (v1.18.0)
Chained-scenario traceability	`chainage` records the ordered call tree of scenarios chained via `declencher_scenario`, surfaced in the operations log (v1.19.0)
Per-action timing	`latences_actions` reports dispatch latency for every action executed, always present (v1.20.0)
Error-only log view	`journal.py --erreurs` filters the operations log to failed runs only (v1.20.0)
HTTP Basic Auth	`--http-credentials` resolves network-level Basic Auth (RFC 7617) from the credentials file, scoped to the target's origin — distinct from and additional to form-based authentication (v1.21.0)
JS click escalation	`repli_js` on `cliquer` retries a failed native click via JS, reported in the boussole only when it actually ran (v1.22.0)
Never-idle targets	`--wait-until load|domcontentloaded` reaches pages that poll continuously and never go network-silent, where no `--timeout` value would ever suffice (v1.22.0)

Requirements

Component	Version / Notes
OS	Linux. Packages for Debian 13, Ubuntu 24.04 and 26.04, Fedora 44, openSUSE Leap 16.0 and Arch Linux — see Installation for what was validated on each. Not tested on macOS or Windows
Display server	Wayland (Playwright runs in this ecosystem)
Python	3.11+ in isolated venv (PEP 668 — system pip blocked on Debian 13)
Playwright	1.50+ (installed in venv)
playwright-stealth	2.0+ — required for `--stealth` (v1.15.0). API-incompatible with 1.x
Chromium	Headless, installed via `playwright install chromium`
Ollama	Local vision models for `cliquer_visuel` and `watch.py`

Component	Version / Notes
GPU	Recommended: NVIDIA RTX 3060 12 GB VRAM or equivalent (for Ollama qwen3-vl models)

Installation

Two channels, **mutually exclusive on a single machine**. Pick the package unless you intend to modify Diwall's own code.

Package — the simple path

Download the file for your distribution from the latest release, then:

```
sudo apt install ./diwall_1.24.4-1_all.deb                          # Debian 13, Ubuntu
 ↪  24.04, Ubuntu 26.04
sudo dnf install ./diwall-1.24.4-1.fc44.noarch.rpm                  # Fedora 44
sudo zypper install --allow-unsigned-rpm ./diwall-1.24.4-1.leap160.noarch.rpm   # openSUSE
 ↪  Leap 16.0
sudo pacman -U ./diwall-1.24.4-1-any.pkg.tar.zst                    # Arch Linux
```

That is all. It creates the `diwall` system user, the virtual environment and `/opt/diwall/`, installs the six `diwall-*` commands in your `PATH`, and ships the manual page:

```
man diwall              # covers all six commands
diwall-shot --version
```

Installation needs network access: it installs the Python dependencies and downloads Chromium.

What "validated" means here. On each of these five systems, the package was installed in a clean container, Chromium was launched for real by `diwall-shot` on a local page, then the package was removed and nothing was left that should not be. A container does not prove the display, nor the mount of the encrypted credentials directory (no FUSE device in a container). Debian 12 and Ubuntu 22.04 are not supported. Linux Mint 22 is based on Ubuntu 24.04 but was not tested as such. Playwright officially supports only Debian and Ubuntu: on Fedora, openSUSE and Arch it works because the packages declare the libraries Chromium needs, not because Playwright vouches for it.

Configuration lives in `/etc/diwall/diwall.conf`; a commented sample is installed next to it as `diwall-sample.conf`. Full command reference: `docs/MANUEL.md` section 1a.

Upgrading is installing the newer file the same way — your configuration is preserved.

When a system update brings a new Python version — routinely on Arch Linux, at a release upgrade elsewhere — Diwall stops working until its package is installed again, which rebuilds its virtual environment for the new interpreter:

```
sudo apt install --reinstall ./diwall_1.24.4-1_all.deb
sudo dnf reinstall ./diwall-1.24.4-1.fc44.noarch.rpm
sudo zypper install --force --allow-unsigned-rpm ./diwall-1.24.4-1.leap160.noarch.rpm
sudo pacman -U ./diwall-1.24.4-1-any.pkg.tar.zst
```

From source — for modifying Diwall itself

If you intend to change Diwall's own code, install from the repository instead: it puts the sources where `deploy.sh` can push your changes to `/opt/diwall/`. The six-step procedure lives in `docs/MANUEL.md` section 1b, next to the commands you will run afterwards.

Uninstallation

Installed from the Debian package:

```
sudo apt remove diwall     # keeps configuration, journal, reference captures and the
↪  diwall account
sudo apt purge diwall      # removes all of them, and /opt/diwall entirely
```

Installed from the Fedora, openSUSE or Arch package:

```
sudo dnf remove diwall       # Fedora
sudo zypper remove diwall    # openSUSE
sudo pacman -R diwall        # Arch Linux
```

These systems have a single removal command. It deletes everything that can be rebuilt (virtual environment, Chromium, Python bytecode). It keeps what you produced and could not get back by reinstalling — `/etc/diwall/diwall.conf`, the journal and evidence under `/var/log/diwall/`, the `watch.py` captures under `/opt/diwall/references/` — together with the `diwall` account that protects them, and prints the command that erases them. If you produced nothing, nothing is left.

Installed from source:

```
# Preview what will be removed (no changes made)
bash ~/git/Diwall/Diwall/scripts/uninstall.sh --dry-run

# Full uninstallation with interactive confirmation
bash ~/git/Diwall/Diwall/scripts/uninstall.sh

# Non-interactive (CI, cold-reinstall tests)
bash ~/git/Diwall/Diwall/scripts/uninstall.sh --confirme
```

Removes: `/opt/diwall/`, `/var/log/diwall/`, system user `diwall`, system group `diwall`, operator's group membership, git pre-push hook.

The script refuses to run when Diwall is installed by a package (Debian, Fedora, openSUSE or Arch), and prints the removal command to use instead (`sudo apt purge diwall`, `sudo dnf remove diwall`, `sudo zypper remove diwall`, `sudo pacman -R diwall`). `install.sh` and `deploy.sh` refuse in the same case.

Never touched: `~/Vaults/` (your credentials), the repository itself, Playwright browser cache.

If `/var/log/diwall/preuves/` contains captures, they are preserved by default. Add `--purge-preuves` to remove them.

Usage (by your LLM)

Simple capture

```
/opt/diwall/venv/bin/python3 /opt/diwall/shot.py \
  --url https://your-app.local/ --som --a11y
```

ReAct loop (multi-step navigation)

```
# Step 1 — navigate and observe
/opt/diwall/venv/bin/python3 /opt/diwall/shot.py \
  --url https://your-app.local/ \
  --sauver-session /tmp/diwall/session.json --som

# Step 2 — act on what was observed
/opt/diwall/venv/bin/python3 /opt/diwall/shot.py \
  --reprendre-session /tmp/diwall/session.json \
  --action '{"type":"cliquer_som","id":2}' \
  --sauver-session /tmp/diwall/session.json --som
```

RPA scenario

```
/opt/diwall/venv/bin/python3 /opt/diwall/rpa.py \
  --scenario /opt/diwall/scenarios/my_scenario.json --som
```

Full LLM reference: docs/GUIDE_LLM.md

Credentials

Credentials are stored in JSON files, one per domain, **never in code or scenario files**:

```
~/Vaults/Diwall/
├── my-app.local.json        → {"password": "...", "username": "admin"}
└── other-service.com.json   → {"password": "...", "api_key": "..."}
```

In a scenario or action: `"valeur": "depuis_secrets", "secret_cle": "password"` — Diwall reads the credential at runtime from the credentials directory.

The path is configurable via `/opt/diwall/diwall.conf` or the `DIWALL_SECRETS_DIR` environment variable.

Recommendation: protect `~/Vaults/Diwall/` with `chmod 700` and encrypt it with `gocryptfs` (see `~/git/Diwall/Diwall/scripts/configurer-repertoire-chiffre.sh --gocryptfs`). The encrypted directory is fully supported since v1.5.0 — if it is initialised but not mounted, Diwall returns a structured `SecretsFermesError` (exit code 42) instead of silently failing.

Security

Capture storage

By default, captures are stored in `/tmp/diwall/` with permissions `700` (owner only). Do not change `--output-dir` to a shared location (`/tmp/`, `~/Desktop/`, etc.) — captures may contain sensitive interface data.

Local vs cloud models

When Diwall is used with a cloud-based LLM (Claude API, OpenAI, etc.), PNG captures are transmitted to external servers. This is the user's responsibility. For interfaces containing private data (credentials, client information, private keys), use local Ollama models only.

Credentials directory

The credentials directory — wherever you pointed `secrets_dir`, for example `~/Vaults/Diwall/` — contains credentials in plaintext JSON when unmounted. Protect it:

```
chmod 700 ~/Vaults/Diwall/
```

Encrypted filesystem support (`gocryptfs`) has been fully supported since v1.5.0 — see "Credentials" above and `~/git/Diwall/Diwall/scripts/configurer-repertoire-chiffre.sh`.

Documentation in other languages (v1.23.0)

English is canonical and stays in place. Translations of the human-facing documents (this README, `docs/GUIDE.md`, `docs/MANUEL.md`, `docs/CHEAT_SHEET.md` and the manual page) live under `docs/fr/`, `docs/de/` and `docs/es/` — one directory per language, beside the English originals.

The LLM guides (`docs/GUIDE_LLM.md` and its three notices) are English only, deliberately. They are protected by the guide-lock: a translation whose version number gets mechanically resynchronised over stale content would let an agent pass the lock having read obsolete instructions — the exact failure the lock exists to prevent. A model reads English natively, so the benefit is nil and the risk is real.

A single reference PDF per language is built from these sources, in an order declared once and shared by every language. The PDFs are published on the website rather than kept here — they are generated artefacts, and a repository is not a delivery channel for binaries: https://diwall.davalan.fr/en/guides/downloads/

The translation and PDF chain itself is not in this repository. It produces the documentation; it is not part of Diwall — it needs `pandoc`, a LaTeX engine and a local Ollama instance, none of which is a Diwall dependency, and none of which appears in `requirements.txt`. The translated markdown is the deliverable; the machine that produces it is maintainer tooling.

For LLMs discovering Diwall

If you are a language model reading this README: see `docs/GUIDE_LLM.md` for the complete technical reference — invocation patterns, SoM usage, credential integration, SPA navigation rules, and Ollama model specifications.

Credits

This project was developed using an **asymmetric human-LLM collaboration model**. Roles are documented formally to reflect the actual work performed.

Architect & Arbiter: Ronan Davalan Product vision, security requirements, project direction, validation and testing. All architectural decisions are validated by him.

Systems Engineer & Lead Developer: Claude Code (Anthropic) Implementation of the ReAct pattern, Python/Bash scripts, complex state management, SoM injection, session persistence. Principal author of the source code.

Synthesizer & Strategic Advisor: Gemini (Google) Independent architectural analysis, logical conflict resolution, workflow optimisation, cross-validation of technical decisions.

Perception models (Ollama, local): - `qwen3-vl:2b` (Alibaba) — click localisation and semantic comparison, ~9–19s (default since v1.3.1) - `qwen3-vl:8b` (Alibaba) — robust fallback, ~114s

Maintenance operators (via OpenCode): - Big Pickle — heavy semantic cleanup of documentation - MiniMax — verification and commits - DeepSeek V4 Flash — catching up on missed commits - Qwen3.6 Plus — role-play passes, including documenting a real task from scratch as an unbriefed model, which surfaced two documentation gaps

---

Licence

MIT — see `LICENSE` file.

Developed on Debian 13 Trixie · Wayland · AMD Ryzen 9 3950X · NVIDIA RTX 3060

Diwall — Operator guide

Version 1.10 — September 2026 (v1.24.4) — four more demonstration use cases (self-hosted observability, ticketing platform administration, local events tracking, e-commerce access under Respectful Navigation)

Also available in French, German and Spanish under `docs/fr/`, `docs/de/` *and* `docs/es/`.

Why Diwall — what you actually delegate

The problem Diwall solves

When you work with an LLM on a web application, a perception asymmetry occurs: the model reads code, runs commands, observes textual output — but it does not see the interface your users see. You do.

This asymmetry creates a specific form of anxiety: you don't know whether what the model describes matches what you would see in a browser. To be sure, you must either trust it at its word, or verify it yourself.

Diwall solves this problem by creating a **shared visual reference**: the model captures the interface with a real browser (headless Chromium), and you have access to the same PNG captures and accessibility trees. You no longer take the model at its word — you observe the same state it does.

```
Browser (headless Chromium)
        │  Playwright drives it — click, fill, navigate
        ▼
shot.py / rpa.py
        │  reads the resulting DOM state through parallel views
        ├──► capture_som   PNG, interactive elements numbered
        ├──► elements_som  JSON list — id, tag, text
        ├──► a11y_tree     accessibility tree, text
        └──► session file  cookies only (--sauver-session)
        │
        ▼
boussole + JSON on stdout — same state you would see in a browser
        │
        ▼
You (the model): read → analyse → decide → act → loop
```

What you delegate

Diwall lets you delegate **repetitive and anxiety-inducing visual verification**:

- Checking that 20 pages of a site display correctly after a deployment
- Confirming that a login form works on the right interface
- Ensuring a deployment did not break the rendering of a critical view
- Visually validating that a fix is correctly visible on screen

Without Diwall, these verifications are your responsibility. With Diwall, the model performs them and reports the result — with visual proof.

What you keep

You keep **high-level sense validation**: deciding whether the result the model presents is acceptable, consistent with your expectations, and in line with what your users should see. That decision remains yours.

Respectful Navigation (v1.15.0)

Diwall does not disguise its identity to bypass bot detection. `--stealth` removes automatic technical markers (`navigator.webdriver`) that block headless browsers regardless of intent — it does not change the operator's IP, identity, or the fact that the run is declared. In exchange, every run reports its own footprint (`respect`: pages visited, actions executed, duration) and respects configurable courtesy delays and hard caps (`diwall.conf [navigation]`). The right to navigate and the duty to navigate measurably

are treated as inseparable — see `docs/RETOUR_EXPERIENCE.md` FR-77/FR-78/FR-79 for the field context that shaped this.

Local targets — the courtesy delay is not a doctrine, it is a default (v1.19.0): the shipped `min_action_delay_ms: 800` protects an unconfigured first run against the public internet — it is meaningless against your own development/production machine. Set it to `0` in your local `diwall.conf` for local debugging; see `docs/MANUEL.md` section 3b.

When Diwall is the right tool

Use case	Diwall suitable?
Visual validation after deployment	✅ Yes
Diagnosing a broken rendering	✅ Yes
Navigation and form input (~30 s max)	✅ Yes
Delegating repetitive checks	✅ Yes
Long server operation (cloning ~2–5 min)	❌ No — Playwright timeout
Bulk deletion or mutation	❌ No — prefer a direct API call
Workflow requiring rollback	❌ No — Diwall cannot undo

For discouraged cases, see `docs/GUIDE_LLM.md` section "When NOT to use Diwall" (frictions FR-59 and FR-60 documented).

This document is written for the person operating Diwall.

It complements `GUIDE_LLM.md` (intended for models) with concrete examples, step-by-step procedures, and reminders on common stumbling points.

Demonstration use cases

The cases below illustrate what an agent-plus-Diwall session can look like in practice. They are meant for you to evaluate against your own context, not as a recommendation to adopt any specific one. Only Case 1 ships as a runnable scenario; the others are narrative on purpose, and each explains why under its own heading.

Case 1 — local CSS/JS troubleshooting

Committed as a real, runnable scenario: `scenarios/exemples/depannage_local.json`. It diagnoses a visual shift or a blocked interaction on a locally-served interface — a fast probe (`--mode fast`), reading `erreurs_js`/`erreurs_console`, an `--som` capture if the shift is purely visual, then validating the fix with `watch.py --comparer-pixel` against a reference captured before the regression. Run it directly:

```
/opt/diwall/venv/bin/python3 /opt/diwall/rpa.py \
  --scenario /opt/diwall/scenarios/exemples/depannage_local.json \
  --guide-version 1.3
```

Case 2 — comparing hardware components across shops

An agent asked to compare a component's price and stock across several online shops could compose Diwall with a separate URL-discovery tool (a local search instance, for example) to find candidate shop pages, then use Diwall in sonde mode (`--mode fast`, no PNG) with `evaluer` actions to extract price/stock/specifications from each page, and finally compare the results itself.

Not shipped as a committed scenario, deliberately: naming a specific shop in a public, versioned scenario is a decision that belongs to you, not a default this project should make on your behalf. It also carries a

real fragility risk — a public scenario targeting a named commercial site can fail months later when that site's anti-bot posture changes (39% of the commercial sites sampled in `docs/RETOUR_EXPERIENCE.md` FR-77 returned an immediate block), which discredits the example more than it helps. If you build this composition yourself, note that any URL-discovery tool you pair Diwall with (a local search instance or otherwise) is not a Diwall component — it is a separate piece the agent composes on top.

Case 3 — exploring and summarising technical documentation (single-page apps)

An agent tasked with producing an integration guide for a documentation site built as a single-page app could use `rpa.py` with `attendre_reseau_calme` to let client-side routing settle, extract the accessibility tree in fast mode to map the page structure, then walk code blocks recursively with `evaluer` to pull their exact content, and finally synthesise the collected material into a guide.

Not shipped as a committed scenario, for the same reason as Case 2 — naming a specific documentation site (or, worse, a specific payment provider whose documentation happens to be the working example) is a commercial and reputational commitment this project should not make by default, and the same WAF-fragility risk applies to a public scenario pinned to one real target.

Case 4 — configuring a self-hosted observability or analytics dashboard

An operator setting up a self-hosted monitoring or web-analytics dashboard behind a reverse proxy can use Diwall to drive the interface itself — creating a dashboard, wiring a data source, setting an alert rule — the same way any other admin panel gets configured, rather than hand-editing files for steps the UI is meant to handle. This includes targets sitting behind a network-level HTTP Basic Auth challenge (`--http-credentials`, v1.21.0) — confirmed against a real Caddy-protected admin interface, not just a synthetic fixture: the stored credentials answered the challenge on the first attempt.

Not shipped as a committed scenario — the dashboard layout and data source names are specific to one operator's infrastructure, and inventing a synthetic equivalent would duplicate what the local fixture in Case 1 already covers for structural regression, not for this kind of guided, multi-step configuration work.

Case 5 — administering a ticketing platform end-to-end

Diwall used across several sessions to configure and operate a real self-hosted ticketing installation — event setup, ticket categories, a custom domain, and the day-of scan/check-in tooling — through the same web interface a human administrator would use. Real friction was encountered and resolved along the way (session handling, dropdown quirks, a permission prompt blocking an unattended step) — not a friction-free success story, which is part of what makes it a useful example: the obstacles were ordinary web-automation obstacles, not something specific to Diwall.

Not shipped as a committed scenario — a ticketing configuration touches billing and venue specifics unique to the operator, same reasoning as Case 2.

Case 6 — tracking a regional events calendar

A simple semantic-probe usage: asking an agent to check a local events calendar for upcoming happenings, without knowing in advance which page holds the answer. Diwall's fast mode (`--mode fast`, no capture) combined with the accessibility tree lets the agent scan and report back in a handful of requests — no vision model needed for this kind of read-only, text-driven task. One session also produced a clean, real example of the WAF signal's documented false-positive behaviour: a page loaded normally (rich content, no captcha, no interstitial) while `respect.waf_bloquants` still fired, because of an unrelated third-party resource on the page matching a detection keyword — resolved in about a minute by reading the accessibility tree already present in the same response, exactly as the guide's "signal, never a lock" rule anticipates.

Not shipped as a committed scenario — a specific regional events site is not a stable, reproducible public target, and naming one publicly is the operator's call, not a project default.

Case 7 — testing real-world access to e-commerce sites under Respectful Navigation

A recurring, honest observation from actual sessions: used respectfully (rate-limited delays, page/action caps, `--stealth` active, no attempt to force access past a real block), Diwall run against a range of e-commerce sites finds that a large share of major platforms return an outright block — HTTP 403, or a request that never completes — regardless of how courteous the traffic is. This is not a Diwall shortcoming to fix: anti-bot posture is the site's own choice, and Diwall does not attempt to defeat it (see "Respectful Navigation" above). Practically: for shopping-comparison tasks against large commercial platforms, expect a meaningful share of dead ends, and treat a block signal (`respect.waf_bloquants`) as information to route around, not an error to retry against.

A distinction worth keeping in mind: an invisible verification screen that never resolves and presents nothing to act on (no checkbox, no image challenge) is different from an interactive CAPTCHA. The latter is legitimate to answer honestly — an agent operating for a named human, from that human's own IP, is not the "robot" the question is aimed at. The former simply offers no door to open from the agent's side, and forcing past it (IP rotation, TLS fingerprint spoofing) falls outside what Diwall does.

Not shipped as a committed scenario, and deliberately not naming the platforms involved — see the WAF-fragility reasoning under Case 2: a dated block/no-block table tied to named commercial sites goes stale and undermines its own point faster than it illustrates it. `docs/RETOUR_EXPERIENCE.md` FR-77 documents the same pattern at panel scale (39% immediate block rate).

Prerequisites before starting

```
# 1. Verify Diwall responds
/opt/diwall/venv/bin/python3 /opt/diwall/shot.py \
  --url https://example.com --som --a11y
# → must return {"succes": true, ...}

# 2. Verify the encrypted directory is mounted (if gocryptfs)
ls ~/Vaults/Diwall/
# → must show .json files, not encrypted content

# 3. Verify credentials for a domain
/opt/diwall/venv/bin/python3 -c "
import sys; sys.path.insert(0, '/opt/diwall')
from lib.repertoire_chiffre import lire_credential
print('OK' if lire_credential('target.local', 'password') else 'EMPTY')
"
```

Credentials configuration per project

Each project can have its own credentials directory. Two methods:

Method 1 — Direct environment variable (one-shot):

```
DIWALL_SECRETS_DIR=~/Vaults/MyProject \
  /opt/diwall/venv/bin/python3 /opt/diwall/shot.py --url …
```

Method 2 — Project `.diwall.conf` file (recommended for recurring projects):

```
# Create the file at the project root
echo '{"secrets_dir": "../MyProject-secrets"}' > ~/git/MyProject/.diwall.conf

# Then prefix each invocation (or export at the start of the shell session)
export DIWALL_CONF=~/git/MyProject/.diwall.conf
/opt/diwall/venv/bin/python3 /opt/diwall/shot.py --url …
```

The `secrets_dir` in `.diwall.conf` can be a relative path — it is resolved relative to the location of the `.diwall.conf` file.

Capturing a page and analysing it

```
# Quick check (no PNG — ~2 s, read-only)
/opt/diwall/venv/bin/python3 /opt/diwall/shot.py \
  --url https://target.local/ \
  --mode fast
# → returns url_courante, titre_page, a11y_tree in the JSON

# Full capture with numbered elements
/opt/diwall/venv/bin/python3 /opt/diwall/shot.py \
  --url https://target.local/ \
  --som --a11y
# The PNG capture is in /tmp/diwall/capture_<ts>.png
```

What you get: - `boussole.url_courante` + `boussole.titre_page`: effective URL and title after navigation - `capture`: path to the PNG of the page as rendered - `capture_som`: annotated PNG with element numbers - `a11y_tree`: page structure in text (headings, fields, buttons)

Automating a login form

Step 1 — Prepare the credentials file.

The credentials file is named `<hostname>.json` where `hostname` = result of `urlparse(url).hostname`. For `https://app.example.com/`, the file is `app.example.com.json`.

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

Step 2 — Explore the login page.

```
/opt/diwall/venv/bin/python3 /opt/diwall/shot.py \
  --url https://app.example.com/login/ --som --a11y
```

Open the annotated PNG (`capture_som`) to identify the SoM IDs of the fields.

Step 3 — Write the scenario.

```
cat > /tmp/login.json << 'EOF'
{
  "nom": "app_login",
  "url": "https://app.example.com/login/",
  "actions": [
    {"type": "remplir_som", "id": 1, "valeur": "depuis_secrets", "secret_cle": "username"},
    {"type": "remplir_som", "id": 2, "valeur": "depuis_secrets", "secret_cle": "password"},
    {"type": "cliquer_som", "id": 3},
    {"type": "pause",        "ms": 2000},
    {"type": "capturer",     "nom": "after-login"}
  ]
}
EOF
```

Step 4 — Execute.

```
/opt/diwall/venv/bin/python3 /opt/diwall/rpa.py \
  --scenario /tmp/login.json --som
```

Validating multiple pages in a single invocation

To check N pages of an authenticated site without replaying the login each time:

```
cat > /tmp/audit.json << 'EOF'
{
  "nom": "audit_pages",
  "url": "https://app.example.com/login/",
  "actions": [
    {"type": "remplir_som", "id": 1, "valeur": "depuis_secrets", "secret_cle": "username"},
    {"type": "remplir_som", "id": 2, "valeur": "depuis_secrets", "secret_cle": "password"},
    {"type": "cliquer_som", "id": 3},
    {"type": "pause",        "ms": 2000},
    {"type": "naviguer",     "url": "https://app.example.com/dashboard/"},
    {"type": "capturer",     "nom": "dashboard"},
    {"type": "naviguer",     "url": "https://app.example.com/settings/"},
    {"type": "capturer",     "nom": "settings"}
  ]
}
EOF
/opt/diwall/venv/bin/python3 /opt/diwall/rpa.py --scenario /tmp/audit.json --som
```

Extracting a value from the page

To read a text string, a counter, or any DOM value:

```
cat > /tmp/extract.json << 'EOF'
[{"type": "evaluer", "script": "document.title"}]
EOF
/opt/diwall/venv/bin/python3 /opt/diwall/shot.py \
  --url https://target.local/ --actions /tmp/extract.json
# → result in evaluations[0].valeur
```

Important: always write JS scripts to an `--actions` file, never inline with `--action` (the shell corrupts nested quotes).

Setting up visual monitoring

```
# 1. Save the visual reference
/opt/diwall/venv/bin/python3 /opt/diwall/watch.py \
  --url https://target.local/ --sauver-reference --nom home

# 2. Compare later (pixel diff)
/opt/diwall/venv/bin/python3 /opt/diwall/watch.py \
  --url https://target.local/ \
  --comparer-pixel /opt/diwall/references/target.local_home/reference.png \
  --nom home
# → verdict: stable / drift / regression (exit code 0 or 1)

# 3. On an authenticated page: capture first with rpa.py, then save
/opt/diwall/venv/bin/python3 /opt/diwall/rpa.py --scenario /tmp/login.json > /tmp/out.json
CAPTURE=$(python3 -c "import json; d=json.load(open('/tmp/out.json')); print(d['captures_
↪  intermediaires'][-1])")
/opt/diwall/venv/bin/python3 /opt/diwall/watch.py \
  --url https://target.local/ --sauver-reference --capture "$CAPTURE" --nom dashboard
```

Setting up continuous structural monitoring (v1.18.0)

Complements the visual monitoring above: this checks the page's *structure* (status code, DOM element counts, JS evaluation results) instead of its *appearance* — cheaper, and catches a different class of regression (a disappeared form field with unchanged layout, for instance).

```
# 1. Save a structural reference, once
/opt/diwall/venv/bin/python3 /opt/diwall/rpa.py \
  --scenario /opt/diwall/scenarios/my-scenario.json \
  --sauver-verifier-reference /opt/diwall/references/my-scenario.ref.json

# 2. One check-and-alert pass
bash ~/git/Diwall/Diwall/scripts/monitor-verifier.sh \
  --scenario /opt/diwall/scenarios/my-scenario.json \
  --reference /opt/diwall/references/my-scenario.ref.json \
  --ntfy-topic diwall-monitoring
```

Silent when stable, one `ntfy` push when a regression is detected. Schedule it yourself with cron — the script does one pass and exits, it does not loop. `scripts/*.sh` is never deployed to `/opt/diwall/`, so the cron entry runs from the git source, as your own user (not the `diwall` service account, which cannot reach `~/git/Diwall/Diwall/`):

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

Common pitfalls

Situation	What to do
`FileNotFoundError` on the credentials file	Check that the JSON file is named with the full FQDN (`urlparse(url).hostname`)
`SecretsFermesError` (exit 42)	Mount the encrypted directory: `bash ~/git/Diwall/Diwall/scripts/monter-repertoire-chiffre.sh`
Invalid JSON in output	Use `2>/dev/null | tail -1` to extract only the JSON line
SoM IDs differ between sessions	Expected — SoM IDs are recalculated on each capture. Never reuse them cross-session
Login followed by Django redirect to dashboard	Do not use `naviguer` in a resumed Django session — pass the URL via `--url`
`<select>` form field not filled	Use `remplir_som` (not `remplir`) with the SoM ID of the `<select>`
Click has no effect on out-of-viewport button	Add `{"type":"defiler","selecteur":"#the-button"}` before the click
`auth_status: "active"` even on the login page	Positive selector is ambiguous (persistent header) — add `--auth-indicator-negative .btn-login`
Web Components elements not numbered by SoM	Add `--shadow-dom` (Angular, Lit, Stencil)
`respect.waf_bloquants` appears on a page that is not actually blocked	Detection is keyword-based (v1.16.0, refined v1.17.2) — treat as a signal, not a verdict. If it persists on a page you've confirmed is not blocked, add `--ignorer-waf`

Situation	What to do
`cliquer_som` clicks the wrong element on a page that mutated between capture and click	Since v1.24.0 resolution is hybrid by default: it uses the `data-dw-som-id` marker when the same scenario captured SoM first, and reports any stable/raw divergence as `boussole.respect.som_derive_detectee`. If that flag appears, re-capture SoM before acting. `--som-brut` forces the pre-v1.24.0 pure re-indexing.
A long RPA scenario fails partway through and you don't want to replay completed steps	Add `--checkpoint FILE` (v1.17.0) — relaunch the same command to resume; DOM state is not preserved, only session + action position
Interactive elements inside an iframe are invisible to Diwall	SoM cannot number iframe content (same-origin or cross-origin) — use `cliquer_iframe`/`remplir_iframe` (v1.17.0) with an explicit CSS selector, or `iframe_chemin` (v1.18.0) for an iframe nested inside another
Your model reports `"erreur": "guide_non_lu"` / exit 1 on its first Diwall call	Expected the first time a model uses Diwall on this machine as this OS user (v1.18.0) — it must read `docs/GUIDE_LLM.md` and pass `--guide-version` once. This is deliberate, not a bug — tell the model to read the guide rather than working around the error

---

Uninstalling Diwall

The `~/git/Diwall/Diwall/scripts/uninstall.sh` script removes the installation cleanly, in the reverse order of `install.sh`.

```
# See what will be removed, without doing anything
bash ~/git/Diwall/Diwall/scripts/uninstall.sh --dry-run

# Full uninstall (interactive confirmation)
bash ~/git/Diwall/Diwall/scripts/uninstall.sh

# Without confirmation (cold tests, chained reinstall)
bash ~/git/Diwall/Diwall/scripts/uninstall.sh --confirme && bash
 ↪  ~/git/Diwall/Diwall/scripts/install.sh
```

What is removed:

Item	Detail
`/opt/diwall/`	Code, Python venv, configuration
`/var/log/diwall/`	Operation logs
`diwall` system user	Created exclusively for Diwall
`diwall` system group	Same
Group membership	Your account is removed from the `diwall` group
git pre-push hook	`core.hooksPath` disabled in the source repository

What is never touched: - `~/Vaults/` — your credentials - `~/git/Diwall/` — git sources - Playwright browser cache (`~/.cache/ms-playwright/`)

Evidence captures (`/var/log/diwall/preuves/`): if the directory contains captures, it is preserved by default with a warning. To remove it:

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

Consulting the operation history

```
# All operations on a target
/opt/diwall/venv/bin/python3 /opt/diwall/journal.py --cible target.local

# Mutating operations only (clicks, form input)
/opt/diwall/venv/bin/python3 /opt/diwall/journal.py --cible target.local --mutatif

# From a date
/opt/diwall/venv/bin/python3 /opt/diwall/journal.py --cible target.local \
  --depuis 2026-06-01
```

Diwall — Operational manual

Version 1.24.4 — September 2026

Also available in French, German and Spanish under `docs/fr/`, `docs/de/` *and* `docs/es/`.

This document answers one question: **how to do X with Diwall**.

> **If you are a user** — no commands needed. Tell your model what you want to visit, observe, or accomplish on a website, a web application, or an administration interface. The model reads this manual and translates your intent into the right actions.
>
> **If you are a language model** — these are your commands. Execute them directly.

No architectural descriptions. Commands that work.

Table of contents

---

1. Verify the installation

```
# Cheapest possible check — no Playwright, no URL, exit 0 immediately (v1.18.0+)
/opt/diwall/venv/bin/python3 /opt/diwall/shot.py --version
# → {"outil": "shot.py", "version": "1.23.0"}

# Full test in one command (~3 s)
/opt/diwall/venv/bin/python3 /opt/diwall/shot.py \
  --url https://example.com --mode fast --guide-version 1.3
```

Expected result: JSON on stdout with `"succes": true`.

`--guide-version` (v1.18.0+): `shot.py`, `rpa.py`, and `watch.py` refuse to run without it — unless a local marker from a previous accepted call already exists (`~/.config/diwall/guide_state.json`). The

value is the `<!-- notice-version: X.Y -->` on line 3 of `docs/GUIDE_LLM.md` — not the Diwall release number. Read the current one rather than trusting any value quoted here: `grep notice-version /opt/diwall/docs/GUIDE_LLM.md`. See `docs/GUIDE_LLM.md` section "Mandatory pre-flight" for the full mechanism and the error format if you skip it.

Once the marker exists, `--guide-version` becomes optional again — every other command example in this manual omits it deliberately, since a marker from any earlier successful call already covers them, as long as `docs/GUIDE_LLM.md`'s `notice-version` has not changed since.

```
# Verify the installed version
grep "__version__" /opt/diwall/shot.py
# → __version__ = "1.23.0"

# Verify playwright-stealth is available (v1.15.0)
/opt/diwall/venv/bin/python3 -c "import playwright_stealth; print('stealth OK')"

# Verify the encrypted directory is mounted
ls ~/Vaults/__PROJET__/Diwall/
# → must show .json files, not an empty list
```

If `ls ~/Vaults/...` returns an empty list or an error: → mount it: `bash ~/git/Diwall/Diwall/scripts/monter-repertoire-chiffre.sh`

1a. Installing from a package — the simple path

The packages are release assets on GitHub. This is the recommended channel unless you intend to modify Diwall's own code, in which case see 1b. The two channels are mutually exclusive on a single machine — both target `/opt/diwall/`.

```
sudo apt install ./diwall_1.24.4-1_all.deb                          # Debian 13, Ubuntu
 ↪  24.04, Ubuntu 26.04
sudo dnf install ./diwall-1.24.4-1.fc44.noarch.rpm                  # Fedora 44
sudo zypper install --allow-unsigned-rpm ./diwall-1.24.4-1.leap160.noarch.rpm   # openSUSE
 ↪  Leap 16.0
sudo pacman -U ./diwall-1.24.4-1-any.pkg.tar.zst                    # Arch Linux
diwall-shot --version
man diwall
```

Each package was validated in a clean container of its system: install, a real Chromium launch by `diwall-shot`, removal. Neither the display nor the FUSE mount of the encrypted directory can be proven in a container. Debian 12 and Ubuntu 22.04 are not supported; Linux Mint 22 is based on Ubuntu 24.04 but was not tested as such. Playwright officially supports only Debian and Ubuntu: on Fedora, openSUSE and Arch, Chromium runs because the package declares the libraries it needs.

Installing requires network access (dependency install and Chromium download happen in the post-install step, as root, into `/opt/diwall/.cache/ms-playwright/`). Six commands become available, each a thin wrapper — no functional difference from the git-clone channel's own invocations:

Command	Wraps
`diwall-shot`	`shot.py`
`diwall-rpa`	`rpa.py`
`diwall-watch`	`watch.py`
`diwall-monter-secrets`	`scripts/monter-repertoire-chiffre.sh`
`diwall-demonter-secrets`	`scripts/demonter-repertoire-chiffre.sh`
`diwall-monitor-verifier`	`scripts/monitor-verifier.sh`

Configuration lives at a different path on this channel: `/etc/diwall/diwall.conf` (not `/opt/diwall/diwall.conf`) — a template is dropped at `/etc/diwall/diwall-sample.conf`, never

auto-activated:

```
sudo cp /etc/diwall/diwall-sample.conf /etc/diwall/diwall.conf
sudo nano /etc/diwall/diwall.conf
sudo usermod -aG diwall $USER
```

`apt remove diwall` keeps `/var/log/diwall/` (operations journal, evidence), `/etc/diwall/`, the `watch.py` reference captures (`/opt/diwall/references/`) and the `diwall` account that protects them (1.24.4: the account used to be deleted, leaving the kept files to an orphan group number) — `apt purge diwall` removes all of them and `/opt/diwall/` entirely. Save `/opt/diwall/references/` first if you want to keep your baselines.

On Fedora, openSUSE and Arch (`dnf remove`, `zypper remove`, `pacman -R`) there is a single removal command. It deletes what can be rebuilt (virtual environment, Chromium, Python bytecode) and keeps what you produced — `/etc/diwall/diwall.conf`, the files under `/var/log/diwall/`, the captures under `/opt/diwall/references/` — with the `diwall` account, then prints the exact command that erases them. If nothing was produced, nothing is left.

`~/Vaults/` is never touched, on any channel.

Manual page (v1.22.0): `man diwall` documents all six commands on a single page. The five other command names (`man diwall-rpa`, and so on) resolve to the same page. It is generated from `debian/diwall.1.md` at build time, so it cannot silently go stale — but for the exhaustive option list of any command, `--help` remains authoritative over the manual page.

1b. Installing from source — for modifying Diwall itself

Use this channel only if you intend to change Diwall's own code: it puts the repository where `deploy.sh` can push your changes to `/opt/diwall/`. For plain use, the `.deb` above is one command and does the same job.

```
# 1. Create system user and directory
sudo useradd --system --no-create-home --shell /bin/false diwall
sudo mkdir -p /opt/diwall
sudo chown root:diwall /opt/diwall

# 2. Clone the repository
git clone https://github.com/ronandavalan/diwall.git ~/git/Diwall/Diwall
cd ~/git/Diwall/Diwall

# 3. Create Python virtual environment
sudo /usr/bin/python3 -m venv /opt/diwall/venv
sudo /opt/diwall/venv/bin/pip install -r requirements.txt

# 4. Install Chromium
sudo /opt/diwall/venv/bin/playwright install chromium

# 5. Deploy
bash ~/git/Diwall/Diwall/scripts/deploy.sh

# 6. Create your encrypted credentials directory
mkdir -p ~/Vaults/<your-project>/Diwall
# Create ~/Vaults/<your-project>/Diwall/<hostname>.json with your credentials
```

If Diwall is already installed from a package, `install.sh` and `deploy.sh` refuse to run (v1.24.2 for the `.deb`, v1.24.4 for the Fedora, openSUSE and Arch packages): they would overwrite files that the package manager manages, and the next package upgrade or removal would silently undo them. To switch channels, remove the package first; they print the exact command (`sudo apt purge diwall`, `sudo dnf remove diwall`, `sudo zypper remove diwall`, `sudo pacman -R diwall`).

On this channel the configuration is `/opt/diwall/diwall.conf`, not `/etc/diwall/diwall.conf`. Uninstall with `bash ~/git/Diwall/Diwall/scripts/uninstall.sh --dry-run` first, then without the flag. `uninstall.sh` refuses too when a package is installed (v1.24.3 for the `.deb`, v1.24.4 for the others): it would remove `/opt/diwall/` and the `diwall` account that the package manager manages. It prints the removal command to use instead.

---

2. Capture a page

2a. Fast capture — text only, no PNG (~2 s)

```
/opt/diwall/venv/bin/python3 /opt/diwall/shot.py \
  --url https://target.local/ \
  --mode fast
```

Returns: `a11y_tree` (text structure of the page), `boussole` (effective URL, title). Use when you want to read the title, verify the URL, or extract text without capturing a PNG.

2b. Full visual capture with numbered elements

```
/opt/diwall/venv/bin/python3 /opt/diwall/shot.py \
  --url https://target.local/ \
  --som --a11y
```

Returns: - `capture`: path to the page PNG - `capture_som`: PNG with numbers on clickable elements (SoM) - `elements_som`: JSON list of elements (id, tag, text) - `a11y_tree`: accessibility tree

Figure 2: Set-of-Mark overlay: each interactive element outlined and numbered

What `--som` produces. The numbers in the image are the `id` values in `elements_som`, so clicking becomes `{"type": "cliquer_som", "id": 7}` — no selector to guess. Generated from a fixture versioned in this repository (`scenarios/interoperabilite/fixture/`); the same figure exists in French, German and Spanish alongside this one.

2c. Read the boussole first

Every output contains a `boussole` object — read it before everything else:

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

```
    "duree_totale_ms": 2140
  }
}
```

If `boussole.url_courante` does not match what you expect: stop and investigate before any mutating action.

`boussole.secrets_dir_effectif` and `boussole.secrets_dir_source` (v1.23.1) report which encrypted-secrets directory was actually resolved for this run and where that came from (`env:DIWALL_SECRETS_DIR`, `env:DIWALL_CONF`, `conf:<path>`, or `non_configure`). A multi-project machine can silently fall back to the machine-wide `diwall.conf` when a caller forgets to export `DIWALL_SECRETS_DIR` — these two fields make that visible on every run instead of after the fact.

2d. Read `etat` for a go/no-go decision (v1.16.0)

Every successful run includes an `etat` object at the JSON root — read it before any mutating action instead of manually cross-checking `auth_status`, `respect.plafond_atteint`, `erreurs_js`, and `erreurs_console` yourself:

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

If `pret_a_agir` is `false`: read `raisons` for the cause (inactive authentication, session drift, navigation cap reached, or a detected WAF block) before proceeding.

`etat` does not check whether the URL or page content matches your business expectation — use `evaluer` with `attendu`/`contient`/`motif` (section 5d) for that.

2e. `mode_conseille` — pre-flight configuration advice (v1.18.0)

If Diwall has real prior data about the host you are calling — from an earlier `diagnostic_dom.json` run against it — `etat` carries a recommendation for your **next** call, never applied automatically:

```
"etat": {
  "pret_a_agir": true,
  "niveau_confiance": "eleve",
  "raisons": ["mode_conseille disponible : full recommandé (React détecté sur ce host)"],
  "mode_conseille": {
    "mode": "full",
    "shadow_dom": true,
    "som_rafraichir": false,
    "raisons": ["react_detecte", "shadow_roots:3"]
  }
}
```

Get this data flowing for a host by running the diagnostic once:

```
/opt/diwall/venv/bin/python3 /opt/diwall/rpa.py \
  --scenario /opt/diwall/scenarios/diagnostic_dom.json \
  --url https://target.local/ --mode fast
```

No prior diagnostic for this host → `mode_conseille` is absent, never a guess. Full detail in `GUIDE_LLM_MONITORING.md`.

The `som_rafraichir` field is kept for output-shape stability but is vestigial since v1.24.0 (hybrid SoM resolution with fallback is the default — nothing to enable; see section 7j).

---

3. Respectful navigation (v1.15.0)

3a. Stealth mode `--stealth`

Some sites block headless browsers on `navigator.webdriver=true` without examining the intent. `--stealth` removes this automatic technical marker.

```
# direct shot.py
/opt/diwall/venv/bin/python3 /opt/diwall/shot.py \
  --url https://target.local/ \
  --som --stealth

# Via rpa.py
/opt/diwall/venv/bin/python3 /opt/diwall/rpa.py \
  --scenario /opt/diwall/scenarios/my-scenario.json \
  --stealth
```

When active: `boussole.stealth_actif = true` in the JSON output.

What `--stealth` changes: `navigator.webdriver` removed, plugins/languages/platform normalised. **What `--stealth` does not change**: the operator's IP, identity, or navigation intent.

3b. Courtesy delays

Configured in `/opt/diwall/diwall.conf`:

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

`min_action_delay_ms`: minimum delay (ms) between each action. Shipped default: 800 ms.

Local development — set it to `0` (v1.19.0): the 800 ms default protects a distracted operator on their *first, unconfigured* run against the public internet — it has no protective purpose against your own development/production machine, where nothing is being asked to behave. Set the key explicitly in your local `diwall.conf`:

```
{
  "navigation": {
    "min_action_delay_ms": 0
  }
}
```

Keep the 800 ms default (or raise it) for any target reached over the public internet. The value is always a conscious choice attached to the target, not a fixed property of the tool — see the WAF and stealth guidance in `docs/GUIDE_LLM.md` for the same principle applied to blocking behaviour.

The `max_pages_par_run` and `max_actions_par_run` caps cleanly stop the run if exceeded. No exception — the output JSON will contain:

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

3c. Impact metrics

Each run returns `respect` in (JSON root and inside boussole):

Key	Meaning
`pages_visitees`	Number of `type: naviguer` navigations executed
`actions_executees`	Total number of scenario actions executed
`duree_totale_ms`	Total run duration
`plafond_atteint`	`"max_pages_par_run"` or `"max_actions_par_run"` if early stop

3d. Stealth benchmark — quantitative (v1.17.1)

Prefer counting concrete fingerprint signals over comparing screenshots by eye — this is the method used to verify the v1.17.0 `playwright-stealth` API-compatibility fix (`docs/RETOUR_EXPERIENCE.md` FR-79):

```
# Without stealth
/opt/diwall/venv/bin/python3 /opt/diwall/shot.py \
  --url https://bot.sannysoft.com --no-capture --timeout 20000 \
  --actions '[{"type":"evaluer","script":"navigator.webdriver"},

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

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

# With stealth
/opt/diwall/venv/bin/python3 /opt/diwall/shot.py \
  --url https://bot.sannysoft.com --no-capture --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"}]'
```

Read the three values in `evaluations[].valeur`: `navigator.webdriver` should go from `true` to `false`, `td.failed` should drop toward `0`. Reference measurement (v1.17.0 fix, session 47): 12 failed ⟶ 0 failed. A number or a boolean returned by `evaluer` keeps its JSON type in the output and in the journal (v1.24.3); before that, `shot.py` returned it as text (`"false"`, `"0"`). Only text is filtered for secret shapes.

For a qualitative second opinion, the provided scenario still produces screenshots to inspect:

```
/opt/diwall/venv/bin/python3 /opt/diwall/rpa.py \
  --scenario /opt/diwall/scenarios/test_stealth.json \
  --output-dir /tmp/diwall/stealth_with --stealth
```

`capture_sannysoft_*.png` and `capture_intoli_*.png` land in that directory. Note: both target pages discuss bot detection in their own content, which can trigger `respect.waf_bloquants` as a false positive (section 3e) — expected on this specific benchmark, not a sign of an actual block.

3e. WAF detection signal (v1.16.0, refined v1.17.2)

Diwall flags a probable WAF block passively — HTTP 403/429, or a title/HTML keyword match (`Cloudflare`, `CAPTCHA`, `checking your browser`, etc.). This is a signal, never an exception — the run completes normally:

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

When present and `> 0`: `etat.niveau_confiance` is `"faible"` and `etat.pret_a_agir` is `false`. Decide yourself whether to retry with `--stealth`, change target, or stop — Diwall does not abort the run for you.

Since v1.17.2, generic vendor names (`Cloudflare`, `Akamai`) only match the page title — matching the full HTML previously false-positived on ordinary CDN resource references. If a false positive persists, `--ignorer-waf` degrades `niveau_confiance` without forcing `pret_a_agir: false` (`boussole.waf_ignore_actif: true` records the override). The detection is keyword-based and can produce false positives on pages that legitimately discuss blocking/detection (e.g. a bot-detection benchmark page) — treat it as a fast signal, not a certain verdict.

3f. Declared language (v1.24.1)

The browser declares the operator's language. It is taken from the environment of the process, in the order Chromium itself follows: `LANGUAGE`, then `LC_ALL`, `LC_MESSAGES`, `LANG` — and `en-US` when none of them gives a usable value (`C`, `POSIX`). The same tag is sent in the `Accept-Language` header and returned by `navigator.language`, so a site that negotiates its language serves the operator's.

To declare another language for one run, set it in the environment:

```
LANGUAGE=en diwall-shot --url https://example.com --mode fast --guide-version 1.3
```

Before v1.24.1, no `Accept-Language` header was sent at all while `navigator.language` carried the machine's language: a site that negotiates its language served its default. A scenario that checks a text on such a site (`evaluer` with `contient`, a wait on a label) can therefore give a different result after upgrading.

---

4. Encrypted directory and credentials

4a. Structure

The credentials live in an encrypted directory — a gocryptfs volume — containing one `.json` file per domain.

```
~/Vaults/__PROJET__/Diwall/
   ├── app.example.com.json          ← credentials for https://app.example.com/
   ├── admin.example.com.json        ← credentials for https://admin.example.com/
   └── operations.jsonl              ← operation log (v1.15.0)
```

Credentials file format:

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

The file name = `urlparse(url).hostname`. For `https://app.example.com/login/`, create `app.example.com.json`.

4b. Filling a form — the absolute rule

FORBIDDEN — exposes the password in the shell and `/proc`:

```
PASS=$(jq -r '.password' ~/Vaults/.../file.json)   # NEVER
curl -d "password=$PASS" https://...                 # NEVER
```

CORRECT — credentials resolved inside Playwright:

```
{"type": "remplir_som", "id": 2, "valeur": "depuis_secrets", "secret_cle": "username"},
{"type": "remplir_som", "id": 3, "valeur": "depuis_secrets", "secret_cle": "password"}
```

Values never pass through the shell, bash history, process logs, or any file.

4c. Choosing the credentials file for a run

```
# Default credentials directory (defined in diwall.conf > secrets_dir)
```

```
/opt/diwall/venv/bin/python3 /opt/diwall/shot.py --url https://target.local/ --som

# Explicit credentials file (--secrets)
/opt/diwall/venv/bin/python3 /opt/diwall/shot.py \
  --url https://target.local/ --som \
  --secrets /path/to/mounted/directory/creds.json

# Per-project credentials directory via .diwall.conf
export DIWALL_CONF=~/git/MyProject/.diwall.conf
/opt/diwall/venv/bin/python3 /opt/diwall/shot.py --url https://target.local/ --som
```

`--secrets` file content — `origines_autorisees` mandatory since 05/08/2026 (breaking change, no compatibility period): a file missing this key is refused before any read.

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

`origines_autorisees` lists the hostnames this file may be used against — same lowercase, no-scheme, no-port format as `domaine_depuis_url()`. A read against a page whose domain is not in the list is refused (`SecretsOrigineNonAutoriseeError`).

Content of `~/git/MyProject/.diwall.conf`:

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

The path is resolved relative to the location of `.diwall.conf`.

4d. TOTP / MFA

```
{"type": "remplir_som", "id": 6, "valeur": "depuis_secrets_totp"}
```

Reads the `totp_cle` key (base32 seed) from the credentials file and generates the current TOTP code.

To receive the code via ntfy (workflow without human intervention):

```
{"type": "attendre_mfa_ntfy", "id_som": 6, "timeout": 120}
```

4e. Integrity checksum (opt-in, v1.15.0)

To protect a credentials file against silent FUSE corruption, add a `checksum` field:

```
# Generate the checksum
/opt/diwall/venv/bin/python3 -c "
import json, hashlib
creds = json.load(open('my_credentials.json'))
fields = {k: creds[k] for k in sorted(['username','password']) if k in creds}
print('sha256:' + hashlib.sha256(json.dumps(fields, sort_keys=True).encode()).hexdigest())
"
```

Add the returned value to the credentials file:

```
{
  "username": "admin@example.com",
  "password": "my-password",
  "checksum": "sha256:a3f2c1..."
}
```

If the checksum does not match, `shot.py` raises `SecretsChecksumError` (exit 42) with an explicit message. Without the `checksum` key: behaviour unchanged (strict opt-in).

4f. Encrypted directory closed — what to do

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

```
# Mount the encrypted directory
bash ~/git/Diwall/Diwall/scripts/monter-repertoire-chiffre.sh
```

```
# Verify the mount
ls ~/Vaults/__PROJET__/Diwall/
# → must show JSON files
```

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

For targets behind a network-level HTTP Basic Auth challenge (RFC 7617) — the wall a reverse proxy like Caddy, nginx, or Traefik raises before any page renders, common in front of self-hosted admin interfaces. This is a different mechanism from the form-based authentication above (4a-4f), which remains fully supported and unaffected.

```
/opt/diwall/venv/bin/python3 /opt/diwall/shot.py \
  --url https://internal.example/ \
  --http-credentials --secrets ~/Vaults/__PROJET__/Diwall/internal_example.json
```

Credentials file — the plain `username`/`password` pair already used for the common case (a single set of credentials for the target):

```
{"username": "admin", "password": "my-password"}
```

Dedicated `http_username`/`http_password` keys are tried first and only needed when the same target has *both* a network-level Basic Auth wall *and* its own separate application login (two different credential pairs in the same file) — Diwall falls back to `username`/`password` automatically when the dedicated keys are absent.

Confirmed in production against a real Caddy-protected target: the safe default (`send: "unauthorized"` — credentials sent only after a genuine 401, never preventively) resolved the challenge on the first attempt. `boussole.http_credentials_actif: true` confirms a real success, not just the flag being passed; `boussole.http_auth_requise: true` flags an unresolved 401 distinctly from a WAF block.

---

5. Write and run an RPA scenario

5a. 3-step protocol

Step 1 — Explore the page (read-only)

```
# Quick view
/opt/diwall/venv/bin/python3 /opt/diwall/shot.py \
  --url https://target.local/ --mode fast

# Full view with numbered elements
/opt/diwall/venv/bin/python3 /opt/diwall/shot.py \
  --url https://target.local/ --som --a11y

# Web Components application (Angular, Lit, Stencil)
/opt/diwall/venv/bin/python3 /opt/diwall/shot.py \
  --url https://target.local/ --som --a11y --shadow-dom

# Enriched DOM inventory (frameworks, shadow roots, stable data-attrs)
/opt/diwall/venv/bin/python3 /opt/diwall/rpa.py \
  --scenario /opt/diwall/scenarios/diagnostic_dom.json \
  --url https://target.local/ --mode fast
```

What to note: - SoM IDs of fields and buttons (read `capture_som`) - Stable attributes: `name`, `id`, `aria-label`, `data-testid` - Blocking overlays (cookie banners, modals) - SPA or full HTTP reload

Step 2 — Write the scenario

```
{
```

```
  "nom": "login_app",
  "url": "https://app.example.com/login/",
  "intention": "Administrator login with stored credentials",
  "actions": [
    {"type": "nettoyer_overlay", "selecteur": ".cookie-banner"},
    {"type": "remplir_som", "id": 1, "valeur": "depuis_secrets", "secret_cle": "username"},
    {"type": "remplir_som", "id": 2, "valeur": "depuis_secrets", "secret_cle": "password"},
    {"type": "cliquer_som", "id": 3},
    {"type": "attendre_selecteur_present", "selecteur": ".user-avatar"},
    {"type": "capturer", "nom": "after-login"}
  ]
}
```

Step 3 — Execute

```
/opt/diwall/venv/bin/python3 /opt/diwall/rpa.py \
  --scenario /opt/diwall/scenarios/login_app.json --som
```

5b. Full scenario: log in and navigate between pages

```
{
  "nom": "audit_pages",
  "url": "https://app.example.com/login/",
  "intention": "Visual audit after deployment",
  "actions": [
    {"type": "remplir_som", "id": 1, "valeur": "depuis_secrets", "secret_cle": "username"},
    {"type": "remplir_som", "id": 2, "valeur": "depuis_secrets", "secret_cle": "password"},
    {"type": "cliquer_som", "id": 3},
    {"type": "attendre_selecteur_present", "selecteur": ".dashboard-main"},
    {"type": "capturer", "nom": "dashboard"},
    {"type": "naviguer", "url": "https://app.example.com/settings/"},
    {"type": "attendre_navigation"},
    {"type": "capturer", "nom": "settings"},
    {"type": "naviguer", "url": "https://app.example.com/users/"},
    {"type": "attendre_navigation"},
    {"type": "capturer", "nom": "users"}
  ]
}
```

5c. Extract data from the 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"}
  ]
}
```

Result in `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 on evaluer (rpa.py only)

Three mutually exclusive keys — one per 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$"}
```

Key	Comparison	Valid types
`attendu`	strict equality `==`	str, int, bool
`contient`	substring `in`	str only
`motif`	`re.search()` Python	str only

If the assertion fails: rpa.py stops immediately (exit 1) before any subsequent mutating action.

5e. Sub-scenarios (declencher_scenario)

Define a login as a reusable sub-scenario:

```
{
  "nom": "login_app",
  "url": "https://app.example.com/login/",
  "actions": [
    {"type": "remplir_som", "id": 1, "valeur": "depuis_secrets", "secret_cle": "username"},
    {"type": "remplir_som", "id": 2, "valeur": "depuis_secrets", "secret_cle": "password"},
    {"type": "cliquer_som", "id": 3},
    {"type": "attendre_selecteur_present", "selecteur": ".user-avatar"}
  ]
}
```

Call this sub-scenario from another scenario:

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

Maximum depth: 5 nesting levels.

5f. Verify you are on the right page before any mutation

Always add a guard as the first action in scenarios that delete or modify:

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

If the guard fails: rpa.py stops before the deletion is executed.

5g. Resume a session (persisted cookies)

```
# First invocation — authenticate and save the session
/opt/diwall/venv/bin/python3 /opt/diwall/shot.py \
  --url https://app.example.com/login/ \
  --actions /tmp/login.json \
  --sauver-session /tmp/diwall/session.json \
  --som

# Subsequent invocations — reuse the session (no re-login)
/opt/diwall/venv/bin/python3 /opt/diwall/shot.py \
```

```
  --url https://app.example.com/dashboard/ \
  --reprendre-session /tmp/diwall/session.json \
  --som
```

Session drift signal: if the session has expired, `boussole.session_derive: true` in the JSON. In that case: restart the full login without `--reprendre-session`.

5h. Structural non-regression without pixels — `--replay-verifier` (v1.17.0)

```
# First run — save the structural reference
/opt/diwall/venv/bin/python3 /opt/diwall/rpa.py \
  --scenario /opt/diwall/scenarios/dashboard.json \
  --sauver-verifier-reference /tmp/dashboard.ref.json

# Subsequent runs — compare
/opt/diwall/venv/bin/python3 /opt/diwall/rpa.py \
  --scenario /opt/diwall/scenarios/dashboard.json \
  --replay-verifier /tmp/dashboard.ref.json
```

Compares `http_status`, `dom_stats`, `evaluer` results, and SoM element count (not content) against the saved reference. Verdict on stderr:

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

Exit 1 on `verdict: "regression"`, with `diffs` listing each mismatched field (`reference` vs `obtenu`). The two flags are mutually exclusive.

A reference recorded before v1.24.3 stores a number or a boolean returned by `evaluer` as text; replayed with v1.24.3 or later it is reported as a regression (`"2"` against 2). Record it again with `--sauver-verifier-reference`.

5i. Resume a long scenario after failure — `--checkpoint` (v1.17.0)

```
/opt/diwall/venv/bin/python3 /opt/diwall/rpa.py \
  --scenario /opt/diwall/scenarios/long_audit.json \
  --checkpoint /tmp/long_audit.checkpoint.json
```

If the scenario fails partway through, `/tmp/long_audit.checkpoint.json` is written with the count of completed actions and a session file. **Relaunch the exact same command** to resume: already-completed actions are skipped. On full success, the checkpoint file is deleted automatically.

A run stopped by a navigation cap (`max_actions_par_run`/`max_pages_par_run`) is treated the same way as a partial failure since v1.17.2 — the checkpoint is updated with the actual progress, not deleted. Before v1.17.2 it was deleted in this case too (it returns the same `succes: true` signal as a fully completed tronçon), silently losing all remaining progress on long scenarios.

DOM state (open modals, half-filled forms) is never preserved across a resume — only cookies/`localStorage` and the action-list position are. Do not rely on `--checkpoint` to resume mid-way through a single multi-step form; it resumes at action boundaries only.

5j. Target elements inside an iframe (v1.17.0)

No Set-of-Mark numbering happens inside an iframe (same-origin or cross-origin) — target it by CSS selector directly:

```
{"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` supports `valeur: "depuis_secrets"` exactly like `remplir` (section 4b) — never a plaintext credential in the scenario. If the target element refuses interaction (e.g. a `contenteditable`

region in a read-only state), add `"force": true` to `cliquer_iframe` — same semantics as `cliquer` (section 7e).

To find the inner selector: use `evaluer` on the iframe's content if it is same-origin (`document.querySelector('iframe').contentDocument...`), or consult the target application's own markup/documentation if cross-origin.

5k. Nested iframes — `iframe_chemin` (v1.18.0)

An iframe inside another iframe: replace `iframe_selecteur` with `iframe_chemin`, an ordered array — one CSS selector per nesting level, from outermost to innermost.

```
{"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` (single frame) and `iframe_chemin` (nested descent) are mutually exclusive — exactly one required per action. For a single-level iframe, keep using `iframe_selecteur` (section 5j).

---

6. Actions — complete reference

Type	Required params	Optional params	Notes
`naviguer`	`url`	—	Full HTTP reload. Counted in `respect.pages_visitees`
`cliquer`	`selecteur`	`force` (bool), `repli_js` (bool)	`force: true` bypasses CSS-hidden elements or showModal. `repli_js: true` retries through JS if the native click still fails (v1.22.0) — needs `--no-evaluer` off
`cliquer_som`	`id`	—	Click at element centre coordinates. No `force` needed
`cliquer_visuel`	`description`	—	LLM vision (~32 s). Last resort for canvas or attribute-less elements
`remplir`	`selecteur`, `valeur`	`secret_cle`	`valeur: "depuis_secrets"` resolves the stored credential
`remplir_som`	`id`, `valeur`	`secret_cle`	Clears the field before typing. `valeur: "depuis_secrets_totp"` for TOTP
`capturer`	`nom`	`som` (bool)	Named intermediate PNG. `som: true` for an annotated capture
`evaluer`	`script`	`attendu`, `contient`, `motif`	JS executed in the browser. Assertions for rpa.py only
`defiler`	`px` or `selecteur`	—	Vertical scroll in pixels (`px`) or scroll to element (`selecteur`)
`pause`	`ms`	`interval_capture`	Fixed delay in ms. Prefer `attendre_selecteur_present` for DOM signals

Type	Required params	Optional params	Notes
`attendre`	`selecteur`	`interval_capture`	Waits for CSS selector to be present
`attendre_navigation`	—	—	Waits for `networkidle` (end of network requests)
`attendre_url`	`motif`	`attendre_changement` (bool)	URL contains pattern (partial match). `attendre_changement: true` if current URL already contains the pattern
`attendre_selecteur_present`	`selecteur`	—	Waits for element to be visible (state=visible)
`attendre_absence`	`selecteur`	`delai_initial_ms`	Waits for element removal from DOM (state=detached)
`attendre_reseau_calme`	—	`timeout_ms`	500 ms of network silence. `timeout_ms`: max duration before giving up
`attendre_mfa_ntfy`	`id_som`	`timeout`	Waits for a TOTP code via ntfy, fills it into the SoM field
`nettoyer_overlay`	`selecteur`	—	Hides blocking overlays (cookie banner, modal). Use before SoM
`declencher_scenario`	`scenario`	—	Inlines a sub-scenario's actions. Max depth: 5
`cliquer_iframe`	`iframe_selecteur` | `iframe_chemin`, `selecteur`	`force` (bool)	Click inside an iframe (v1.17.0). `iframe_chemin` for nested iframes (v1.18.0, section 5k). No SoM inside frames
`remplir_iframe`	`iframe_selecteur` | `iframe_chemin`, `selecteur`, `valeur`	`secret_cle`	Fill inside an iframe (v1.17.0). `iframe_chemin` for nested iframes (v1.18.0). `valeur: "depuis_secrets"` supported

---

7. Handle common obstacles

7a. Cookie banner / blocking overlay

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

Place **before** any other action and before SoM. The overlay masks elements that SoM numbers. Do not use in `watch.py` scenarios (the overlay is part of the visual reference).

7b. Out-of-viewport element

SoM warns when an interactive element is off-screen:

```
"som_hors_viewport": 3,
"avertissement_scroll": "3 interactive element(s) off-viewport — use defiler before
 ↪  cliquer_som"
```

```
{"type": "defiler", "selecteur": "#the-button"},
{"type": "remplir_som", "id": 7, "valeur": "depuis_secrets", "secret_cle": "username"}
```

7c. Web Components — Shadow DOM

If visible interactive elements receive no SoM number:

```
/opt/diwall/venv/bin/python3 /opt/diwall/shot.py \
  --url https://target.local/ --som --shadow-dom
```

Or in the scenario: `"shadow_dom": true` at the root.

When to use: Angular, Lit, Stencil, FAST. Do not activate on projects without Web Components.

To access an element inside a Shadow Root without `--shadow-dom`:

```
{"type": "evaluer", "script":
 ↪  "document.querySelector('my-component').shadowRoot.querySelector('button').click()"}
```

7d. SPA (React, Vue, Angular) — navigate without reload

After a click that changes the view in an SPA, Playwright does not know when navigation is complete.

```
{"type": "cliquer_som", "id": 5},
{"type": "attendre_url", "motif": "/dashboard"},
{"type": "evaluer", "script": "document.title", "contient": "Dashboard"}
```

Or wait for an element specific to the new view:

```
{"type": "cliquer_som", "id": 5},
{"type": "attendre_selecteur_present", "selecteur": "[data-testid='dashboard-main']"}
```

Never assume a click has completed navigation without a DOM signal.

7e. CSS dialog or showModal()

`TimeoutError` on `cliquer` when the element is visible in the DOM = CSS-hidden element or inside a dialog.

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

If `force: true` is insufficient (element absent from DOM):

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

Do not use `force` on `cliquer_som` — unnecessary, `cliquer_som` uses coordinates and bypasses checks natively.

7f. Long operation (spinner, batch job)

Do not use `pause` to wait for a fixed duration. Wait for the DOM signal:

```
{"type": "cliquer_som", "id": 7},
{"type": "attendre_absence", "selecteur": ".spinner", "delai_initial_ms": 500},
{"type": "attendre_selecteur_present", "selecteur": ".result-container"},
{"type": "capturer", "nom": "result"}
```

If the operation provides no DOM signal, use `interval_capture` to observe state:

```
{"type": "pause", "ms": 30000, "interval_capture": 5}
```

Intermediate captures appear in `stream_captures[]`.

7g. Cap reached (v1.15.0)

If `respect.plafond_atteint` is present in the output, the run was stopped before the scenario completed. Remaining actions were not executed.

Options: 1. Increase `max_pages_par_run` or `max_actions_par_run` in `diwall.conf` 2. Split the scenario into multiple runs 3. Override caps in the scenario JSON (to be documented in _CADRE)

7h. `<select>` form field

`remplir` does not work on `<select>`. Use `remplir_som` with the SoM ID of the `<select>`.

7i. Invalid SoM IDs on next run

SoM IDs are recalculated on each capture. They do not persist between invocations. Always re-run `shot.py --som` to get the current run's IDs. After a `defiler` or opening a modal: re-run `shot.py --som`.

7j. SoM ID drift on highly dynamic pages — hybrid resolution (v1.24.0)

`cliquer_som`/`remplir_som` resolve `id: N` by re-indexing the live DOM at click time — if an interactive element appears or disappears **before** your target in DOM order between the `--som` capture and the click (a cookie banner closing, a modal opening), a raw `id: N` can silently resolve to a **different** element than the one shown numbered N in the screenshot.

Since v1.24.0 the default resolver is hybrid. In one page call it:

- looks up the `data-dw-som-id="N"` marker (**stable** path, set by a `{"type":"capturer","som":true}` action or the final capture);
- computes the N-th element by raw re-indexing (**raw** path);
- returns the stable path when the marker exists, the raw path otherwise (**fallback** — identical to pre-v1.24.0 behaviour);
- reports `boussole.respect.som_resolution` (`stable` | `brut` | `brut_sans_reference`) and, when the two paths disagree, `boussole.respect.som_derive_detectee` with the SoM id.

```
/opt/diwall/venv/bin/python3 /opt/diwall/shot.py \
  --url https://target.local/ --som \
  --actions '[{"type":"capturer","nom":"avant","som":true},{"type":"cliquer_som","id":5}]'
```

The stable path only helps within one scenario — the marker does not survive across two `shot.py` calls (each reloads the page). For a mutation-prone target, capture SoM in the same scenario before the first `cliquer_som`. When `som_resolution` is `brut_sans_reference`, drift cannot be detected — recapture SoM if the DOM changed. `--som-brut` forces pure raw re-indexing (no marker lookup, no divergence detection); `boussole.som_brut_actif: true` then. `--som-rafraichir` (v1.17.0) is a no-op alias kept for backward compatibility.

Since v1.17.2, the injector purges markers left by a previous `--som` capture in the same page before renumbering — without this, an element hidden or scrolled out between two captures could keep a stale `data-dw-som-id`, colliding with a freshly numbered element and resolving to the wrong one.

7k. Site blocked by WAF (immediate 403)

```
# Try with stealth
/opt/diwall/venv/bin/python3 /opt/diwall/shot.py \
  --url https://target.local/ --mode fast --stealth
```

If 403 persists with `--stealth`: the site uses TLS fingerprinting (JA3/JA4) or advanced behavioural analysis (Cloudflare Enterprise). `playwright-stealth` does not bypass these protections. See `docs/RETOUR_EXPERIENCE.md` FR-77/FR-78/FR-79 for context.

Diwall also flags a likely block passively without you having to check the HTTP status yourself — see section 3e (`respect.waf_bloquants`).

7l. Initial navigation never completes — `--wait-until` (v1.22.0)

Symptom: `TimeoutError` on the initial navigation, and raising `--timeout` changes nothing (45 s fails exactly like 10 s). Cause: by default Diwall waits for `networkidle` — 500 ms of network silence. A page that polls continuously (live statistics, auto-refreshing counters, router admin panels) never produces that silence, so no timeout value can ever be large enough.

```
# shot.py — direct reconnaissance
/opt/diwall/venv/bin/python3 /opt/diwall/shot.py \
  --url http://target.local/ --wait-until load --som --a11y --guide-version 1.3

# rpa.py — propagated to shot.py, so scenarios reach the same targets
/opt/diwall/venv/bin/python3 /opt/diwall/rpa.py \
  --scenario ./admin_login.json --wait-until load --guide-version 1.3
```

A scenario can carry it as a root property instead, staying self-contained:

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

The CLI flag takes precedence over the scenario property.

Value	Waits for	Use when
`networkidle`	500 ms of network silence	default — keep it unless it fails
`load`	`load` event (page and sub-resources)	continuous polling / live statistics
`domcontentloaded`	HTML parsed, sub-resources still pending	very heavy page, DOM is all you need

Applies to the initial navigation only — the `naviguer` action is unaffected. `boussole.wait_until` reports the value only when it differs from the default.

---

8. Visual monitoring — watch.py

8a. Save a reference

```
/opt/diwall/venv/bin/python3 /opt/diwall/watch.py \
  --url https://target.local/status \
  --sauver-reference \
  --nom home
```

The reference is saved in `/opt/diwall/references/`.

8b. Compare to the reference (pixel diff)

```
/opt/diwall/venv/bin/python3 /opt/diwall/watch.py \
  --url https://target.local/status \
  --comparer-pixel /opt/diwall/references/target.local_home/reference.png \
  --nom home
```

Verdicts:

`taux_diff`	Verdict	Exit code
< 0.2%	`stable`	0
0.2% – 5%	`drift`	0
≥ 5%	`regression`	1
Different dimensions	`viewport_mismatch`	2

8c. Semantic comparison (LLM)

```
/opt/diwall/venv/bin/python3 /opt/diwall/watch.py \
  --url https://target.local/status \
  --comparer \
  --llm local
```

Combine pixel diff and LLM analysis:

```
--llm-en-complement   # LLM only if pixel verdict is drift or regression
```

`--llm claude` (v1.24.2) sends both captures — reference and current — to the Anthropic API instead of the local model; it applies to `--comparer` and to `--llm-en-complement` alike. The captures leave the machine, reduced so that their longest side does not exceed 1568 px, the size the API works at. It needs the `anthropic` module, absent from a standard installation, and a key in the environment of the process:

```
sudo /opt/diwall/venv/bin/pip install anthropic
ANTHROPIC_API_KEY=... diwall-watch --url https://target.local/status --comparer --llm
 ↪  claude --guide-version 1.3
```

A missing module, a refused key or a model refusal comes back as a plain message in the output (`erreur`, or `analyse_llm` in pixel mode), never as a traceback.

8d. Ignore an animated zone

```
/opt/diwall/venv/bin/python3 /opt/diwall/watch.py \
  --url https://target.local/status \
  --comparer-pixel reference.png \
  --exclure-zone 100,200,300,50    # X,Y,Width,Height in pixels
```

8e. Monitoring loop

```
while true; do
  /opt/diwall/venv/bin/python3 /opt/diwall/watch.py \
    --url https://target.local/status \
    --comparer-pixel /opt/diwall/references/status-ok.png \
    --ntfy-url https://ntfy.sh/my-alerts
  sleep 60
done
```

8f. Cron for autonomous monitoring

```
# /etc/cron.d/diwall-monitor
*/30 * * * * diwall /opt/diwall/venv/bin/python3 /opt/diwall/watch.py \
  --url https://target.local/status \
  --comparer-pixel /opt/diwall/references/status-ok.png \
  --ntfy-url https://ntfy.sh/my-alerts \
  >> /var/log/diwall/cron.jsonl 2>&1
```

8g. Continuous structural monitoring — `monitor-verifier.sh` (v1.18.0)

Complements 8a–8f: `watch.py` monitors *appearance* (pixels/semantic). `scripts/monitor-verifier.sh` monitors *structure* (`http_status`, `dom_stats`, `evaluations`, SoM count) — zero image, zero LLM call, built on `--no-capture` + `--replay-verifier` (section 5h).

```
# First run — create the structural reference
/opt/diwall/venv/bin/python3 /opt/diwall/rpa.py \
  --scenario /opt/diwall/scenarios/sillage_login.json \
  --sauver-verifier-reference /opt/diwall/references/sillage_login.ref.json

# One check-and-alert pass — not a daemon, run it repeatedly via cron.
# scripts/*.sh is never deployed to /opt/diwall/, so it runs from the git
# source, as your own user.
bash ~/git/Diwall/Diwall/scripts/monitor-verifier.sh \
  --scenario /opt/diwall/scenarios/sillage_login.json \
  --reference /opt/diwall/references/sillage_login.ref.json \
  --ntfy-topic diwall-monitoring

# crontab -e (your own crontab)
*/15 * * * * bash ~/git/Diwall/Diwall/scripts/monitor-verifier.sh \
  --scenario /opt/diwall/scenarios/sillage_login.json \
```

```
  --reference /opt/diwall/references/sillage_login.ref.json \
  --ntfy-topic diwall-monitoring \
  >> /var/log/diwall/cron-structural.jsonl 2>&1
```

Stable → silence. Regression → one `ntfy` notification with the diff. Each invocation is an isolated process — no daemon, no memory-leak risk, and Respectful Navigation caps reset cleanly on every pass.

---

9. Operation log

The log is configurable in `diwall.conf` (v1.15.0):

```
"journal": {
  "chemin": "~/Vaults/__PROJET__/Diwall/operations.jsonl"
}
```

If absent or the encrypted directory is not mounted, fallback: `DIWALL_JOURNAL` env var, then `/var/log/diwall/operations.jsonl`.

```
# Read the last 10 entries
tail -n 10 ~/Vaults/__PROJET__/Diwall/operations.jsonl | python3 -m json.tool

# Filter by target (journal.py tool)
/opt/diwall/venv/bin/python3 /opt/diwall/journal.py \
  --cible app.example.com

# Filter mutating operations only
/opt/diwall/venv/bin/python3 /opt/diwall/journal.py \
  --cible app.example.com --mutatif

# From a date
/opt/diwall/venv/bin/python3 /opt/diwall/journal.py \
  --cible app.example.com --depuis 2026-07-01

# Failed runs only (v1.20.0) — resultat != "succes"
/opt/diwall/venv/bin/python3 /opt/diwall/journal.py \
  --cible app.example.com --erreurs
```

Fields in each entry:

Field	Meaning
`ts`	ISO 8601 timestamp
`version`	Diwall version
`outil`	`shot.py` or `rpa.py`
`cible_url`	Target URL
`scenario`	Scenario file path (RPA mode)
`source_scenario`	Scenario file name only, no path (v1.18.0) — powers `mode_conseille` (section 2e)
`resultat`	`"succes"` or `"echec"`
`mutatif`	`true` if at least one write action
`duree_ms`	Duration in ms
`intention`	Label passed via `--intention` or scenario `intention` field

9a. Log rotation (G-36, CHANTIER_SANITISATION.md)

Diwall does not ship a logrotate configuration — `/var/log/diwall/operations.jsonl` grows unbounded until the administrator installs one. `lib/journal.py` opens and closes the file on every write (no persistent file descriptor across runs), specifically so the **default** logrotate behaviour (rename the current

file, create a fresh one) works correctly without any special option: the next write reopens the path and finds the new inode.

Do not add `copytruncate` to a Diwall logrotate config — it is unnecessary here (unlike tools that hold a file descriptor open across their lifetime) and reintroduces a write-loss window this design was built to avoid. Example `/etc/logrotate.d/diwall`:

```
/var/log/diwall/operations.jsonl {
    weekly
    rotate 8
    compress
    delaycompress
    missingok
    notifempty
    create 0640 diwall diwall
}
```

`journal.py` (the reader) already follows rotated files transparently (`operations.jsonl`, `.1`, `.2.gz`, …) — no extra step needed after rotation.

---

10. CLI flags — reference

shot.py

Flag	Default	Description
`--version`	—	Prints installed version and exits immediately — no Playwright, no other argument required (v1.18.0)
`--guide-version X.Y`	—	Proof of reading `docs/GUIDE_LLM.md` — required unless a valid local marker already exists (v1.18.0, section 1)
`--url URL`	required	URL to capture
`--actions FILE`	—	JSON file of sequential actions
`--output-dir DIR`	`/tmp/diwall`	PNG output directory
`--timeout MS`	10000	Per-action Playwright timeout (ms)
`--screenshot-timeout MS`	120000	Timeout for `page.screenshot()` (ms). Distinct from `--timeout`
`--largeur PX`	1280	Viewport width
`--hauteur PX`	720	Viewport height
`--som`	off	Activate Set-of-Mark (element numbering)
`--a11y`	off	Include accessibility tree in JSON
`--shadow-dom`	off	Traverse Shadow Roots for SoM (Angular, Lit, Stencil)
`--stealth`	off	playwright-stealth stealth mode (v1.15.0)
`--mode fast|full`	—	`fast` = `--no-capture --a11y`. `full` = default behaviour
`--no-capture`	off	Skip PNG capture and SoM
`--llm local|claude`	`local`	LLM engine for `cliquer_visuel`
`--secrets FILE`	—	Explicit path to a credentials file
`--auth-indicator SEL`	—	CSS selector present only in authenticated session
`--auth-indicator-negative SEL`	—	CSS selector present only outside authenticated session
`--intention TEXT`	—	Business label recorded in the log

Flag	Default	Description
`--sauver-session FILE`	—	Saves cookies after actions
`--reprendre-session FILE`	—	Resumes a saved session
`--interval-capture N`	0	Periodic captures every N seconds during `attendre`, `pause`
`--som-brut`	off	Forces pure raw SoM re-indexing; disables the hybrid resolver's stable path and divergence detection (v1.24.0, section 7j)
`--som-rafraichir`	off	No-op alias since v1.24.0 — hybrid resolution is the default (v1.17.0, section 7j)
`--ignorer-waf`	off	A detected WAF block degrades `niveau_confiance` but no longer forces `pret_a_agir: false` on its own (v1.17.2, section 3e)
`--http-credentials`	off	Resolves HTTP Basic Auth credentials from the credentials file, scoped to the target's origin (v1.21.0, section 4g)
`--no-evaluer`	off	Refuses the **evaluer** action for the whole run — recommended in production against targets with sensitive forms (v1.15.1)
`--no-filtre-evaluer`	off	Disables stdout neutralisation of **evaluer** return values, URLs and error messages — explicit debug runs only. Neutralisation is on by default; when disabled, `boussole.filtre_evaluer_actif: false` is set in the output so the operator can audit it from the JSON itself (v1.23.0)

rpa.py

Propagates all relevant shot.py flags, plus:

Flag	Description
`--version`	Prints installed version and exits immediately (v1.18.0)
`--guide-version X.Y`	Proof of reading `docs/GUIDE_LLM.md` — checked independently, same rule as shot.py (v1.18.0)
`--scenario FILE`	Path to JSON or YAML scenario (required)
`--url URL`	Overrides scenario URL without modifying the file
`--stealth`	Propagated to shot.py
`--mode fast|full`	Propagated to shot.py
`--som-brut`	Propagated to shot.py. Also settable as scenario root property `"som_brut": true` (v1.24.0, section 7j)
`--som-rafraichir`	Propagated to shot.py — no-op alias since v1.24.0 (v1.17.0, section 7j)
`--ignorer-waf`	Propagated to shot.py (v1.17.2, section 3e)
`--http-credentials`	Propagated to shot.py. Also settable as scenario root property `"http_credentials": true` (v1.21.0, section 4g)
`--sauver-verifier-reference FILE`	Saves structural reference for `--replay-verifier` (v1.17.0, section 5h)
`--replay-verifier FILE`	Compares run against a structural reference, exit 1 on regression (v1.17.0, section 5h)

Flag	Description
`--checkpoint FILE`	Resumes a long scenario after a mid-run failure (v1.17.0, section 5i)

watch.py

Flag	Description
`--version`	Prints installed version and exits immediately (v1.18.0)
`--guide-version X.Y`	Proof of reading `docs/GUIDE_LLM.md` — checked independently, same rule as shot.py (v1.18.0)
`--url URL`	URL to monitor
`--sauver-reference`	Capture and save as reference
`--comparer-pixel REF`	Pixel diff against PNG file REF
`--comparer`	Semantic LLM diff
`--llm local|claude`	Engine for `--comparer` and `--llm-en-complement` (default `local`; `claude` sends the captures to the Anthropic API, v1.24.2)
`--nom NAME`	View name (multiple views per URL)
`--seuil-stable F`	`stable` threshold (default: 0.002 = 0.2%)
`--seuil-regression F`	`regression` threshold (default: 0.05 = 5%)
`--exclure-zone X,Y,W,H`	Zone to ignore (repeatable)
`--heatmap`	Produces a PNG of modified zones
`--ntfy-url URL`	Sends an ntfy alert on regression
`--llm-en-complement`	Adds LLM diff when pixel = drift or regression

---

11. Exit codes and output

Exit codes

Code	Cause	What to do
0	Success	—
1	Playwright error, failed action, rpa.py assertion	Read `erreur` in JSON. See `GUIDE_LLM_INTERACTIONS.md`
1	`guide_non_lu` — missing/wrong `--guide-version`, no valid marker (v1.18.0)	Fires before Playwright launches. Read `docs/GUIDE_LLM.md`, relaunch with `--guide-version X.Y` (section 1)
2	`viewport_mismatch` (watch.py)	Re-capture reference at same viewport
3	`playwright` module not found	Invoke via `/opt/diwall/venv/bin/python3`
42	`SecretsFermesError` — encrypted directory not mounted, or invalid checksum	Mount it, or verify the credentials file
43	`SecretsNonConfigureError` — `diwall.conf` absent	`sudo cp /opt/diwall/diwall-sample.conf /opt/diwall/diwall.conf && sudo nano /opt/diwall/diwall.conf`

Output JSON structure

```
{
```

```
  "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",
  "capture": "/tmp/diwall/a1b2c3d4e5f6/capture_1234567890123456789.png",
  "capture_som": "/tmp/diwall/a1b2c3d4e5f6/capture_som_1234567890123456789.png",
  "elements_som": [...],
  "a11y_tree": "...",
  "evaluations": [...],
  "latences_actions": [
    {"index": 0, "type": "naviguer", "latence_ms": 842},
    {"index": 1, "type": "cliquer_som", "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/diwall",
    "operation_id": "a1b2c3d4e5f6",
    "url_courante": "https://target.local/dashboard",
    "titre_page": "Dashboard — My App",
    "stealth_actif": true,
    "shadow_dom_actif": true,
    "auth_status": "active",
    "som_hors_viewport": 0,
    "respect": { "pages_visitees": 0, "actions_executees": 3, "duree_totale_ms": 2400,
     ↪  "indice_agressivite": 0.33, "som_resolution": "stable" }
  },
  "diwall_meta": {
    "version_shot": "1.23.0",
    "profil": "operator",
    "modeles_appeles": []
  }
}
```


`operation_id` (v1.16.0) is always present and identifies this run uniquely — it names the isolation directory under `/tmp/diwall/<operation_id>/` and matches the `operation_id` field of this run's entry in the operations log (section 9). `etat` (v1.16.0) is present on the success path only. `latences_actions` (v1.20.0) is always present (empty list if no actions), one entry per action that actually dispatched — see `GUIDE_LLM_MONITORING.md` for how it complements `respect.duree_totale_ms`.

Conditional keys (absent when inactive): `capture`, `capture_som`, `elements_som`, `a11y_tree`, `evaluations`, `auth_status`, `stealth_actif`, `shadow_dom_actif`, `som_brut_actif`, `som_hors_viewport`, `session_derive`, `respect.plafond_atteint`, `respect.waf_bloquants`, `respect.som_resolution` (present whenever a SoM action was resolved), `respect.som_derive_detectee` (present only on a stable/raw divergence), `respect.indice_agressivite` (present whenever at least one action

ran), `actions_executees_avant_echec`, `pages_visitees_avant_echec` (failure JSON only, v1.17.0), `etat.mode_conseille` (present only with real prior `diagnostic_dom.json` data for this host, v1.18.0, section 2e).

Error — format

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

Reference paths

Path	Role
`/opt/diwall/`	Production installation
`/opt/diwall/venv/bin/python3`	Python to use for every invocation
`/opt/diwall/diwall.conf`	Machine configuration (credentials, navigation, log)
`/opt/diwall/diwall-sample.conf`	Configuration template
`/opt/diwall/scenarios/`	RPA scenarios
`/opt/diwall/docs/`	Documentation
`/opt/diwall/references/`	watch.py visual references
`/tmp/diwall/<operation_id>/`	Temporary captures for one run, isolated by `operation_id` (v1.16.0, cleared on reboot)
`~/Vaults/__PROJET__/Diwall/`	Credentials + log (gocryptfs volume)
`~/git/Diwall/Diwall/`	Git sources (modify here, then `deploy.sh`)

Deploy after modifying sources:

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